Recommendations for increased efficiency of Web services in the tactical domain

Frank T. Johnsen and Trude H. Bloebaum
Norwegian Defence Research Establishment (FFI)
Kjeller, Norway

Kristoffer R. Karud
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

***Abstract*—The C3 Board has chosen Service Oriented Architecture (SOA) realized using Web services to achieve interoperable software in NATO. The NATO research group IST-118 "SOA recommendations for disadvantaged grids in the tactical domain" aims to provide guidance on how to make SOA applicable at the tactical level. The focus of the group is to investigate how Core Enterprise Services can be applied in the tactical domain, as these form a basis one can use to build other services on. In this paper, we investigate how tuning application server parameters affects the performance of Web services in different emulated tactical networks.**

I. Introduction

In this paper we investigate how tuning server application parameters affects the performance of Web services. We use Commercial Off The Shelf (COTS) software, along with the *Netem* network emulator to introduce limitations in the communication between client and server. This work was performed in the context of the NATO Task Group IST-118, "SOA Recommendations for Disadvantaged Grids in the Tactical Domain" (see [10] for more on the group and its work), and it is this task group that has defined the network metrics that were emulated: A satellite link, a line-of-sight link, two different WIFI links (one short-distance and one long-distance) and a combat net radio link. To emulate these networks a Graphical User Interface (GUI) was created, that allowed the user to emulate different network parameters on a router between the service and the client. The GUI does this by using the tool Netem to emulate the network parameters. As for the Web services, we evaluated both W3C's SOAP Web services as well as Representational State Transfer (REST) Web services. The main differences between these technologies are explained below.

A. W3C's Web services – NATO's choice

In [3], Web services were identified as the key enabling technology for realizing NATO Network Enabled Capability (NNEC). Web services technology consists of a family of standards, where XML, WSDL, and SOAP constitute the core, as defined by the World Wide Web Consortium (W3C) [8]:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web services are currently identified as the technology that should be used to achieve interoperability with respect to machine-to-machine message-oriented information exchange in NATO, both for request/response and publish/subscribe. SOAP-based Web services constitute the foundation for an interoperable message-oriented middleware, and NATO's Core Enterprise Services are to a large extent based on Web services technology (as defined by the W3C). This approach and the relevant standards are further discussed in the NATO C3 Board's SOA Baseline [5]. The SOA baseline is being further refined in NATO's current work on defining Service Interface Profiles (SIPs). The SIPs vary in maturity, but their development is foundational for the Connected Forces Initiative (CFI) and Federated Mission Networking (FMN). A few SIPs are a part of the NATO FMN Implementation Plan (NFIP) already, with more to come in future NFIP development spirals.

TABLE I. W3C'S SOAP-BASED WEB SERVICES COMPARED TO RESTFUL WEB SERVICES

SOAP	REST
Can use almost any transport	Uses HTTP/HTTPS exclusively
Somewhat complex	Very easy to understand
Industry standard	Lacking in standardization
Based on XML	Can use XML, JSON, etc.
The foundation of a complete message-oriented middleware	Good for simplistic point-to-point connections
Identified as the key enabling technology for NNEC	Some interest in NATO for certain applications
We refer to this technology as SOAP Web services in this paper.	We refer to this technology as RESTful services in this paper.

B. RESTful Web services – increasingly popular for certain applications

The term *Web services* is also sometimes used to refer to a technology belonging to a completely different paradigm than SOA, the so-called Representational State Transfer (REST), also known as RESTful Web services. REST is an architectural approach that differs from SOA (and then also SOAP Web services) in that it places constraints on connector semantics rather than component semantics. REST employs a pull-based interaction style (i.e., the client requests data from servers as and when needed). REST does not support publish/subscribe. Also, REST is not standardized in any way — the approach stems from Roy Fielding's doctoral dissertation [6]. Fielding was also one of the principal authors of the HTTP specification. Hence, REST is tightly coupled with HTTP, and uses that protocol's defined operations as its "uniform interface" for accessing services. REST is often favored in communities focusing on data orientation (e.g., for semantic applications) rather than service orientation, and has in recent years been identified as a technology of interest by certain groupings in NATO. For example, in ACT's venue for Technology for Information, Decision and Execution superiority (TIDE) there is some experimentation involving REST in addition to SOAP Web services. Also, with the increasing focus on applying civil smart-devices at the tactical edge (see e.g., [1]) there is focus on REST as, e.g., the iOS and Android platforms natively support REST but not SOAP. For a comparison of the key features of SOAP and RESTful services, see Table I.

C. Towards SOA recommendations

As mentioned above, IST-118 aims to provide "SOA Recommendations for Disadvantaged Grids in the Tactical Domain". In this paper we discuss an optimization that is orthogonal to previous work, in that tuning server parameters can be performed independently of other measures like, e.g., payload compression. The remainder of this paper is organized as follows: For a discussion of related work, see Section II. Our experiment test setup is explained in Section III. The tests and results pertaining to SOAP Web services are presented in Section IV. During these tests we discovered that tuning certain server-side parameters had a measurable impact on performance. Hence, we continued with a comparative study of SOAP and REST services, as discussed in Section V. Section VI concludes the paper.

II. RELATED WORK

Previously, a lot of efforts have been made to improve the efficiency of Web services communication. For example, non-lossy compression has been identified as one of the most important techniques that can be applied to reduce the overhead of XML. Augeri et al. [2] showed that in most instances a general-purpose compressor should be used, although if maximum parsing and compression speed was needed an XML-specific compressor might be useful. The results indicated that binary format was best applied to small files. A more recent study by Teixeira et al. [15] considered the metrics compression rate and compression time, and concluded that the XML-specific W3C Efficient XML Interchange (EXI) format specification reached the best compression rate. Indeed, this algorithm has also been investigated for compressing XML in military Web services, namely in a study by Podlasek et al. [13] focusing on blue force tracking, where EXI was identified as the best algorithm to use. As compression affects the payload size but not the protocol behavior, it is a technique that can be applied in addition to other optimization techniques.

Another axis of optimization that has been pursued is that of optimizing the transport layer mechanism of Web services. SOAP is inherently transport layer agnostic, meaning that it is possible to deviate from the much-used and COTS tool-supported HTTP/TCP transport combination and attempt to use other protocols instead. Previously we have pursued this path in IST-118, a work that has been described in [9] where we investigated using such protocols as UDP, AMQP and SCTP as alternatives to TCP. Though under certain network conditions as tested in the study it would make sense to consider using another protocol like, e.g., UDP or SCTP, it would break compatibility between standard COTS services and clients which may not support bindings beyond HTTP/TCP. Hence, optimizing the transport layer should be seen as a technique that can be used in

addition to other techniques, more specifically it should be implemented in proxies as suggested by IST-090 [11].

Proxies are intermediary nodes that can be put to diverse use, for example for implementing optimizations in the network between service and consumer. The concept is well understood and has been studied extensively in IST-118's predecessor group, IST-090, as well as by others. In IST-090, proxies were used as test beds for experimental optimizations, and were put to such diverse use as providing delay-tolerance, multiple transport protocol support, and compression between services and clients in a network while retaining backward-compatibility with COTS software used in the applications and servers [12]. Also, the edge-proxy concept has been investigated, notably by Sliwa et al. [14], who present a proxy on the tactical edge that can be used for content adaptation (e.g., introducing lossy compression as opposed to non-lossy) to provide data better suited in resource-constrained networks.

Combining the above mentioned techniques has been shown to be fruitful in practice. For example, in the final demonstration by IST-090 [11], compression, filtering and proxies were employed. Also, in the Coalition for Secure Information Sharing (CoNSIS) experiment [4], the above techniques were employed in an actual multi-national heterogeneous tactical radio environment. In both these events SOA principles were realized using (optimized) SOAP Web services, effectively demonstrating the viability of the approaches in actual use.

However, all the approaches above target optimizing the payload or the transport mechanism. Another axis to optimize towards is the tuning of application server parameters. Little has been done so far regarding this. To the best of our knowledge, the only study including server side optimizations is the one by Fiske et al. [7], where the focus was on efficiently supporting HTTP by looking into configuring HTTP continue, HTTP sockets (number of parallel connections), and increasing the HTTP timeout. In this paper we go beyond what was done in that study, as we investigate a wider array of server parameters (TCP tuning, HTTP tuning, and so on) and evaluate the impact on both SOAP and REST Web services. This work is orthogonal to previous work on optimizations, and can be performed in addition to leveraging any or all of the above mentioned approaches.

III. Test setup

At the start of the project, three virtual machines were set up. Two Windows machines (Windows 7) on different subnets acted as service and client, while a Linux machine (Ubuntu 14.04) connected to both subnets acted as a router. After turning on IP forwarding on the Linux machine and setting up routes between the Windows machines through the Linux machine, it was possible to send a ping from one Windows machine to the other.

Netem commands could easily be written into the console on the Linux machine to emulate network parameters like delay, packet loss, packet duplication, packet corruption, packet reordering and data rates, to affect communication between the two Windows machines. However, running these commands through a Java program (Java version 7 update 67) proved to be more difficult. All the Netem commands are sudo commands, and thus require a user password to run. This was solved by letting the Java program ask the user for a password, and then using "echo" to pipe in the password together with the Netem command through a shell. The user password is validated by the Java program by trying to run a Netem command, and if the resulting message is an "incorrect password attempt" message, the program will prompt the user for a new password.

The LogIn class prompts the user for a password, and once a correct password is given, it starts up the GUI. The GUI class handles input from the user through the user interface, and passes the input to the Netem class. All the parameters are stored in the Netem class, which also translates these to Linux commands. These commands are then executed through the Command class, which sends in the commands along with the user password to a runtime instance, through a shell. The IO class on the other hand, handles saving and loading configuration sets to and from files. The program was tested by sending pings from one Windows machine to the other, and observing that the program emulated the network parameters correctly. Changes in the network bandwidth were tested with the tool iperf (iperf 2.0.5-2), as a regular ping does not measure this.

The Netem class uses an array of Doubles to hold all the network parameters, where each parameter has its own index in the array as described in comments in the code. An array was used for easy instantiating, saving and loading. When the user wishes to save a set of configuration parameters, the list is written to a

.txt file, where the name of the file is given through the user interface. The parameters are written to the file in order, with one parameter per line. All file names are written to a separate file, "filelist.txt", that the program uses to let the user choose which files to load back into the program. The GUI class prevents the user from overwriting this file, and issues an error if the user attempts to do this. Input from the user is allowed through JformattedTextFields that are formatted to accept only numeric values. All other text that is entered is discarded by the program. Input for distribution values is through a JComboBox, because the user only has three different options for delay distribution.

To control if a parameter is set to a correct value, the GUI class uses response messages from the Netem class. If a parameter is set to a correct value, the response is of the form "[parameter] set to [value]". The GUI class therefore interprets all responses that does not contain the String "set to" as negative responses, and prints out the response message to the user. When the Netem class receives an incorrect value, it discards the change and returns a message that does not contain the words "set to". Because the simulated network has to be symmetrical for testing purposes, Netem set up equal emulations on both subnets (that is, configuring the "eth0" and "eth1" Ethernet adapters). This was done simply by calling each command twice, once for each subnet.

The service was implemented in three different ways. The first service used the SOAP protocol, while the two others were RESTful. All services were implemented with a method that retrieves one of 22 different NFFI[1] tracks, saved as XML and JSON files. One of the RESTful services as well as the SOAP service sent the XML version of the tracks, while the other RESTful service sent the JSON version. The test client calls this method to receive these files over the network. The RESTful services were both implemented so that each file was a separate resource that could be fetched with a simple HTTP GET command. Before the client starts timing each file transfer, it fetches all 22 files once to warm up the service. The client then fetches all the 22 files again, this time measuring the average transfer time. The client also keeps track of how many errors that occurred during the test (how many files that could not be read on arrival).

[1]NFFI, short for NATO Friendly Force Information — STANAG 5527. The STANAG includes an XML schema that NFFI XML documents must adhere to. There is no JSON format in the STANAG, so for the sake of the tests in this paper we created JSON files containing the exact same data elements as the XML documents.

TABLE II. THE NETWORKS IDENTIFIED BY IST-118: PARAMETERS EMULATED USING NETEM

Network	Bandwidth	Delay	Packet error rate
SATCOM	250 Kbps	550 ms	0.00%
LOS	2 Mbps	5 ms	0,00%
WIFI1	2 Mbps	100 ms	1.00%
WIFI2	2 Mbps	100 ms	20.00%
CNR	9.6 Kbps	100 ms	1.00%

The test service was deployed with a GlassFish application server (GlassFish version 4), where changes could be made to the server parameters.

IV. SOAP WEB SERVICES PERFORMANCE TESTING

The performance testing used five different simulated networks, defined by the NATO IST-118 Task Group, with the parameters shown in Table II.

Here, a typical satellite link is represented (SATCOM), a line-of-sight (LOS) link, combat network radio (CNR), as well as wifi. The wifi network is divided into two instances, one indicating operation in the "sweet spot" (less than 100m range, called WIFI1) and one indicating operating near the edge of network reach (more than 100m range, called WIFI2). These networks can be encountered at the tactical edge, or they can be used as transit networks from the tactical edge and back to infrastructure as illustrated in Figure 1.

We identified the following application server parameters for testing:

- HTTP Timeout: Controls how long a HTTP connection can be deemed as idle and kept in the "keep-alive" state.
- HTTP Compression: Can enable or force HTTP/1.1 GZIP compression.
- HTTP Chunking: Allows the server to send data in dynamic chunks.
- HTTP Header and Send Buffer Sizes: Vary the size of the buffers that hold the request and send data, respectively.
- TCP Idle Key Timeout: Sets the time before an idle TCP channel closes.
- TCP Read and Write Timeouts: Set the timeout for TCP read and write operations, respectively.
- TCP Selector Poll Timeout: Sets the time a NIO selector will block waiting for user requests.

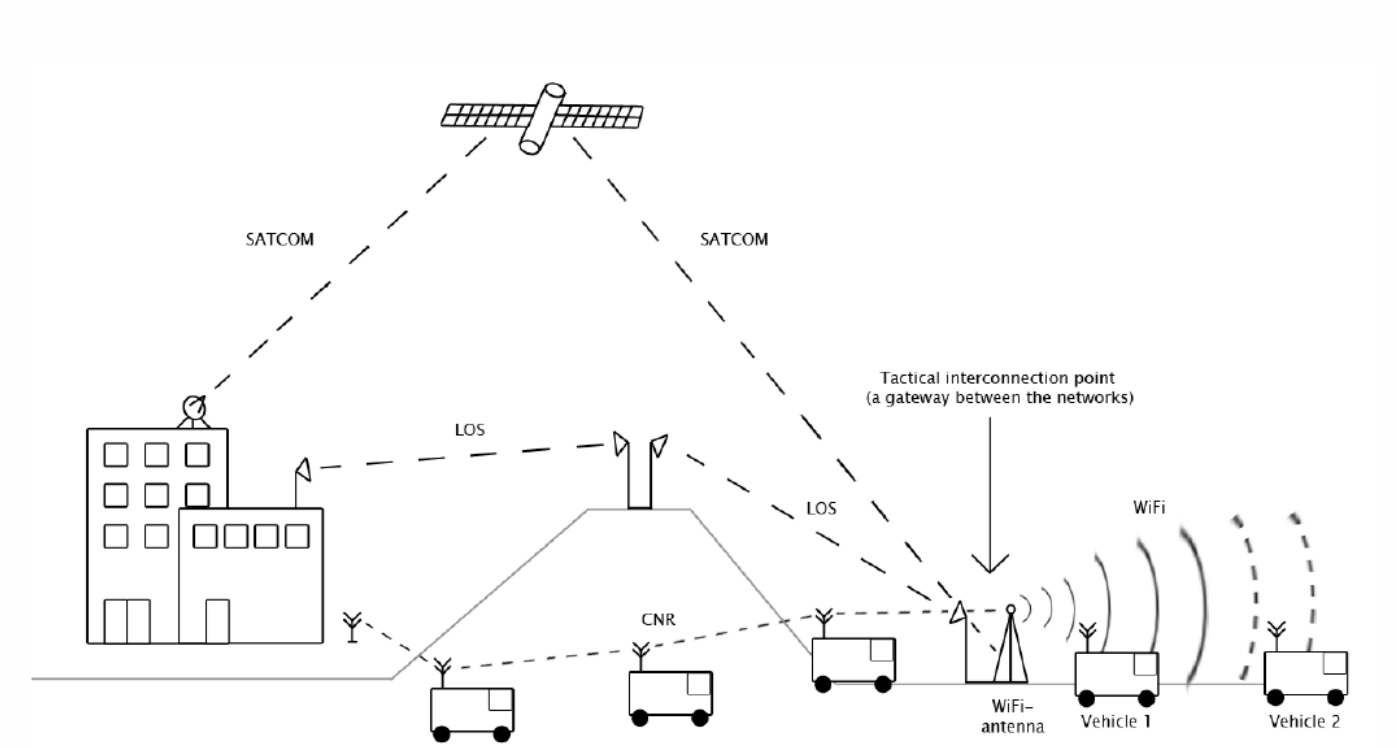

Fig. 1. The networks identified by IST-118: Usage example

TABLE III. GLASSFISH PARAMETER VALUES USED FOR TESTING

Parameter	Default value	Test values
HTTP Timeout	180s	90s, 30s, 15s
HTTP Compression	Off	On, Forced
HTTP Chunking	Disabled	Enabled
HTTP Send Buffer	32768 bytes	8192 bytes, 16384 bytes
HTTP Header Buffer	32768 bytes	8192 bytes, 16384 bytes
TCP Idle Key Timeout	120s	30s, 60s
TCP Read Timeout	120s	30s, 60s
TCP Write Timeout	120s	30s, 60s
TCP Selector Poll Timeout	120s	30s, 60s
TCP Buffer	32768 bytes	8192 bytes, 16384 bytes
TCP Batching (NoDelay)	Enabled	Disabled

- TCP Buffer Size: Sets the size of the buffer that holds input streams created by the network listener.
- TCP Batching/TCP NO_DELAY (also called Nagle's algorithm): Batches together small TCP packets into larger packets.

The default and additional test values for each parameter are shown in Table III.

Note that default values we set in our experiments are not necessarily the GlassFish default values from a fresh installation. Rather, these values are the ones we used as defaults across experiments where a given parameter was not explicitly being varied through the tests. In this way tests are less likely to fail because of a low timeout or a too small buffer, unless we are testing that parameter. All other parameters in GlassFish are set to default values, except for the Java Virtual Machine (JVM), which was set to use the "-server" parameter instead of "-client" to optimize it for running a server.

We proceeded to test each separate parameter in Table III in the initial round of testing. Here we tested with the given test values in the table, with all the other parameters set to default. The most promising parameters were then tested further in-depth. For all the tests the success criterion was that a client could complete a successful request/response message exchange with the server without timing out or otherwise failing. The metric we focused on was delay, because given a successful service invocation the delay is the metric directly affecting application performance and thus the user's experience.

A. *Initial tests*

In the initial tests we sought to identify the impact tuning individual server parameters had. We evaluated each parameter in turn, executing the tests for each parameter in all the different emulated network settings.

1) Timeouts: Neither HTTP Timeout nor any of the TCP Timeouts had much impact on the performance of the SATCOM network. On average, the delay experienced was 6500ms, and the same across all different parameter settings here. This indicates that none of the test values were low enough to actually cause timeouts. However, as the performance seems stable, then it might not be necessary to tune these parameters further, as this would not yield any performance gain.

For the LOS network the performance with different timeout values varied more than for the SATCOM network, but the results were still fairly consistent as the variance in delay was in the order of +/- 50ms, with the average being 260ms. However, it does not seem to be any clear consistency to this variation, and it is possible that the variation is caused by the fact that the network is very fast, and therefore more prone to slight random variations. It is also not very likely that timeout values as high as those tested would have any major impact on performance, because the tests finished quicker than any of the timeouts.

Just as with the LOS network, the WIFI1 network showed inconsistent variations for the different TCP timeout values, here the variations were +/- 100ms with the average being 1500ms. Here, this increased variance was likely caused by the network's 1% packet error rate. However, just as with the LOS network, the timeout values have little impact on the performance.

The results for the WIFI2 network were inconclusive. Here the TCP Write and Read timeout tests did not finish for the lowest test values, while the TCP Selector Poll and Idle Key timeout tests did not finish for the middle test values. Apart from this, the TCP timeout tests seemed stable, except from the Read middle value timeout test which had significantly worse performance than the other tests. The Write and Read tests also both had errors occurring for the middle value tests. The HTTP timeout tests, on the other hand, showed worse performance and more errors with lower timeout values. Here, the decrease in performance seems quite stable. Both the HTTP timeout and the TCP timeouts were subject to further in-depth testing due to the behavior identified here. Overall poor performance was expected, though, due to the very high error rate of this network.

The CNR network seemed to have a slight performance gain with low TCP Timeout values, dropping from 35000ms down to approximately 32500ms. This was consistent for all the TCP Timeout tests. The HTTP tests on the other hand, showed that a low HTTP Timeout caused errors to occur. This makes sense, as the network operation would be interrupted if the HTTP connection timed out during transfer, as it seems to have done here. Further in-depth testing was scheduled to investigate this further.

2) Buffers: Varying the size of the different buffers had no measurable impact on the performance for the SATCOM network, where the delay showed a consistent average of 6500ms.

The tests for the LOS and WIFI1 networks showed some variation in performance for different buffer sizes. However, this variation was inconsistent, just as in the timeout testing. For LOS the average delay was 275ms, varying with +/- 25ms independently of the buffer parameter values set. For WIFI1 the average delay was 1500ms, varying with +/- 100ms. Again, these inconsistencies indicate that the buffer sizes do not impact the performance over either of these networks to any significant degree.

The WIFI2 buffer size tests were inconclusive, due to several of the test runs failing. However, the general trend seemed to be that the biggest buffer performed best. These findings warranted further in-depth testing.

The buffer tests for the CNR network showed some variation, but this did not seem to be a result of the different buffer sizes, as the change in performance was not consistent with increasing buffer size. Here, the average delay was 34000ms, with a delay variance of +/- 1000ms.

3) Compression, chunking, and batching: HTTP Header Compression and TCP Batching had no measurable impact on the performance of the SATCOM network, as we experienced an average delay of 6500ms for all tests. HTTP Chunking, on the other hand, showed promising results. With HTTP Chunking enabled the experienced delay decreased significantly, exhibiting an average of 3500ms. These results warranted further in-depth testing.

Like for SATCOM, the LOS network exhibited similar behavior for the respective parameters. Header Compression and TCP Batching seemed to have little effect, while HTTP Chunking increased performance. Here the average delay measured was 275ms, dropping to 250ms with HTTP Chunking enabled, thus showing this parameters positive effect on overall delay.

Just as with the other networks, the WIFI1 network responded well to HTTP Chunking with a doubling in

performance as the average delay dropped from 1600ms to 800ms. The results for HTTP Header Compression and TCP Batching, on the other hand, showed only minor variations that could be attributed to the same random variation that this network has shown in previous tests.

In the WIFI2 network, enabling HTTP Chunking increased performance (average delay dropped from around 18000ms to approximately 11000ms), while TCP batching reduced performance (average delay increased from 18000ms to about 28000ms). The tests involving HTTP Header Compression, on the other hand, showed no consistent results and did not complete fully. Thus, all of these parameters were candidates for further in-depth testing.

The CNR network also showed a performance increase with HTTP Chunking enabled, since the average delay dropped from 35000ms to 30000ms. HTTP Header Compression and TCP Batching also seemed to have a positive effect on performance, but the performance gain was very slight and may be attributed to the performance variation in this network. As with the other networks, these parameters warranted more in-depth testing in combination with HTTP chunking.

B. In-depth tests

The initial testing described above revealed that HTTP Chunking was the single most promising parameter, and in this section we take a closer look at how well chunking combines with other parameters, as well as finding out how great a performance boost the different networks receive from it. We also explore further many of the parameters for the WIFI2 and CNR networks, as the initial testing yielded somewhat unclear results. We can already draw the conclusion that we cannot use SOAP Web services in WIFI2, i.e., at the very edge of WIFI reach, because the high error rates cause the communication to fail, and thus also our tests to yield inconclusive results. Thus, we introduce a third WIFI network at this point that we call WIFI3, which is still outside the "sweet spot" but not completely at the edge of radio reach. This additional emulated network was given the characteristics shown in Table IV.

The new WIFI3 network has all the same values as WIFI2, except a packet error rate of 5% instead of 20%. This was done to overcome the issues we experienced with the underlying TCP communication more or less breaking down when the error rate was that high, and thus better see the impact of tuning the server parameters in a network with high error rates. Using this network we re-visited the above tests, and were able to draw further conclusions.

TABLE IV. WIFI3 NETWORK – BETWEEN THE "SWEET SPOT" AND "THE EDGE"

Network	Bandwidth	Delay	Packet error rate
WIFI3	2 Mbps	100 ms	5.00%

1) Timeout: We performed tests comparing the TCP Read Timeout using the WIFI3 network. When using WIFI2 the tests were inconclusive, but the WIFI3 tests indicate that TCP Read Timeout does not impact the system performance. Further, results from the re-testing TCP Write Timeout indicates that this parameter also has little impact on the performance for WIFI networks as well. Just as with the TCP Read Timeout, the test results for WIFI2 were inconclusive, while the results for WIFI3 were stable. The TCP Idle Key Timeout results showed that it too, just like the previous timeouts, had little impact on performance over the WIFI networks. The results for TCP Selector Poll Timeout show, just like the other timeout tests, that the TCP Selector Poll Timeout has little impact on performance. Again the WIFI2 results are inconsistent, while the WIFI3 results indicate that the timeout does not change the performance of the system. Finally, for the HTTP Timeout tests we found that that when the HTTP Timeout is too low, errors will start occurring. However, when we used a HTTP Timeout of 40s or above, the tests were stable and without errors.

2) Buffers: The HTTP Send Buffer tests indicate that the buffer size does not impact the performance. Further, the HTTP Header Buffer also has little impact on the WIFI performance. The TCP Buffer test differs from the others in one aspect. From the WIFI3 results, it seems like a very small TCP Buffer results in worse performance than a larger one. The performance then seems to stabilize around 8192 bytes.

3) HTTP Chunking, HTTP Header Compression and TCP Batching: HTTP Chunking improves the performance significantly for the SATCOM network. In addition, the performance is increased even more when combining HTTP Chunking with TCP Batching. HTTP Header Compression on the other hand, does not affect the performance in any way for this network.

The results for the LOS network were similar to the SATCOM results: HTTP Chunking improves the performance, and combined with TCP Batching, the performance is increased even further. However, here it

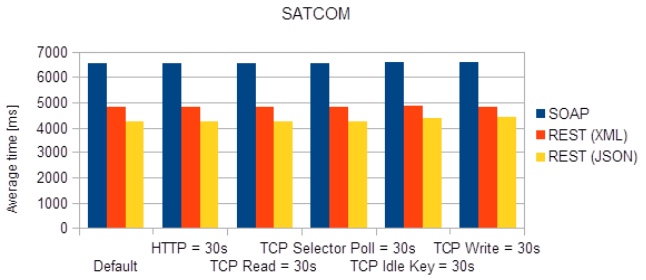

Fig. 2. SATCOM Timeout tests

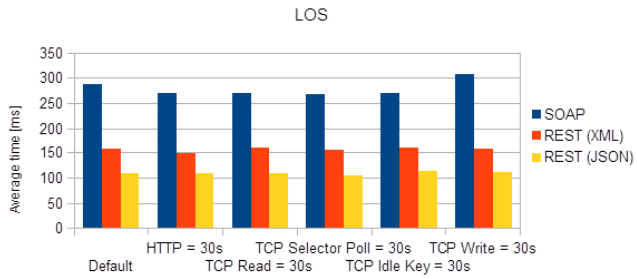

Fig. 3. LOS Timeout tests

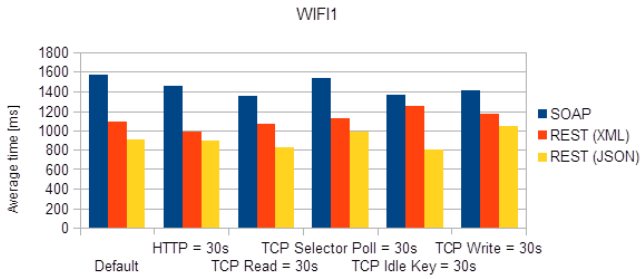

Fig. 4. WIFI1 Timeout tests

seems like HTTP Header Compression actually makes the performance worse.

The CNR test results showed that for the CNR network, HTTP Chunking was the only parameter that increases performance. TCP Batching on the other hand reduced performance, while HTTP Header Compression had little impact.

The test results for the WIFI1 network indicate that HTTP Chunking is the only parameter that impacts performance for this network. While HTTP Chunking gives a significant performance increase, none of the other parameters affect the performance. The WIFI2 results were inconclusive, while the WIFI3 results indicated that only HTTP Chunking affected the system performance. Because the WIFI1 results also indicate this, it seems plausible that HTTP Chunking is the only parameter that increases performance over WIFI networks.

V. SOAP AND REST COMPARISON

To Compare the SOAP based service that we have studied so far to a RESTful one, some of the key tests above were repeated using two RESTful services: One based on XML and one based on JSON.

A. Timeouts

As Figure 2 shows, both RESTful services outperform the SOAP service over the SATCOM network. We can also see that the RESTful service with JSON has the best performance of the two RESTful services. When it comes to timeouts, none of the services seem affected by the different timeout settings. This indicates that the RESTful services behave similarly to the SOAP service in this regard.

Just as with the SATCOM network, the RESTful service with JSON performs best over the LOS network (see Figure 3), followed by the RESTful service with XML. The timeout settings also seem to have little impact on the performance over this network.

Figure 4 shows the performance over the WIFI1 network, and the RESTful services perform better than the SOAP service over this network as well. When it comes to the timeout parameters there are some variation in the results here, but as we have seen before the WIFI networks show some random inconsistency caused by the network's packet error rate.

The results for the WIFI2 network shown in Figure 5 are inconclusive. Hence, only data for the test runs that could be completed are included in the figure. It is, however, clear that the RESTful services perform better than the SOAP service, and the RESTful service with JSON performs better than the XML service. It is also worth noticing that both of the RESTful services show less variation in performance than the SOAP service.

The CNR network results (see Figure 6) are consistent, and show that once again the RESTful services outperform the SOAP service, with the JSON service coming out on top. Just as for the other networks, the RESTful services seem to behave just like the SOAP service when it comes to different timeout values over the CNR network as well.

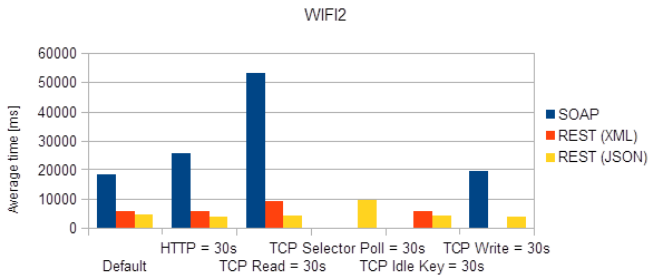

Fig. 5. WIFI2 Timeout tests

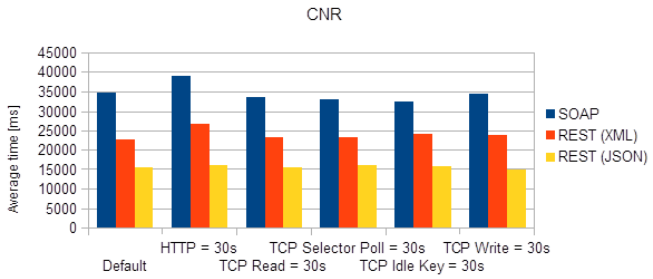

Fig. 6. CNR Timeout tests

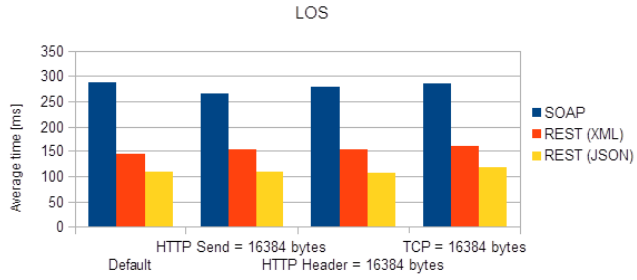

Fig. 8. LOS buffer tests

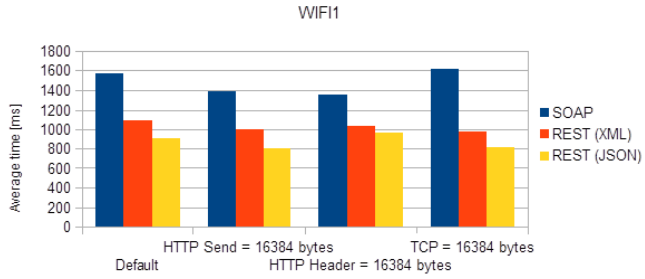

Fig. 9. WIFI1 buffer tests

B. *Buffers*

Just as in the timeout tests above, the RESTful services behave in the same way as the SOAP service for different buffer sizes over the SATCOM network. This is evident in Figure 7, where we see that the RESTful services perform consistently better than the SOAP service in these tests as well.

Figure 8 shows the results for the LOS tests, which exhibit similar characteristics as before: The RESTful services exhibit better performance than the SOAP service.

As we can see in Figure 9, there are some variances in the results of the WIFI1 buffer tests. However, this is most likely not caused by the buffer sizes, but rather the network's packet error rate as we have seen in previous tests. Apart from this slight variation, the RESTful services seem to behave similar to the SOAP service. The results for the WIFI2 tests were too inconclusive to display graphically. However, this network has proved to be very unstable due to its high error rate.

Finally, the results for the CNR network (see Figure 10) clearly show that the RESTful services perform better than the SOAP service, while the RESTful service using JSON performs better than the one using XML.

C. *Compression, Chunking and Batching*

In the SATCOM test (see Figure 11) we found that HTTP Chunking increases performance for the RESTful services as well as the SOAP service. Just as with

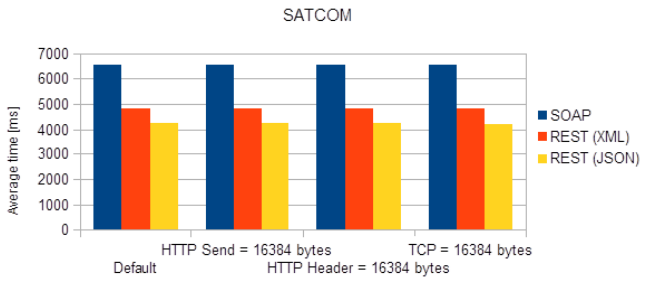

Fig. 7. SATCOM buffer tests

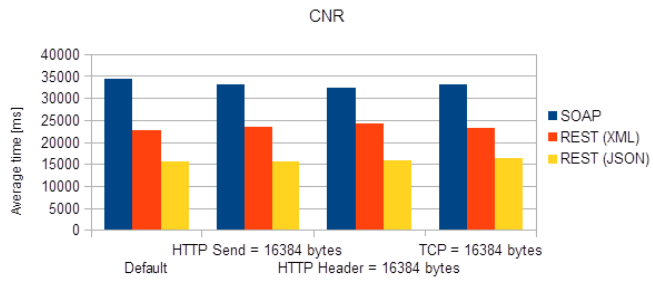

Fig. 10. CNR buffer tests

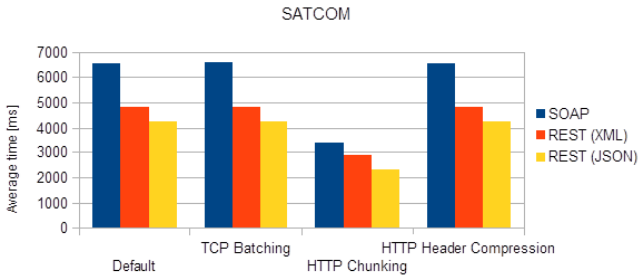

Fig. 11. SATCOM compression, chunking and batching tests

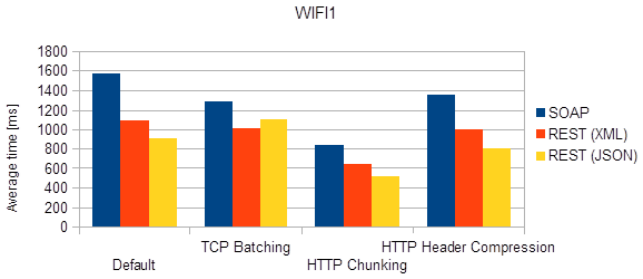

Fig. 12. WIFI1 compression, chunking and batching tests

the previous tests, the RESTful services outperform the SOAP service here as well. The LOS tests (graph omitted for brevity) showed that the SOAP service and the JSON RESTful service both got a slight increase in performance from HTTP Chunking.

For the WIFI1 network, shown in Figure 12, all of the services respond well to HTTP Chunking. For TCP Batching on the other hand, we can see that the RESTful service with JSON performs worse than the RESTful service with XML. Just like in the SATCOM test, it is strange that the two almost identical services should behave differently, and this may be caused by the WIFI1 network's packet error rate. Apart from that, the three services behave similarly. As before, the WIFI2 tests were inconclusive, and the graph is omitted.

The CNR network, on the other hand, shows stable results (see Figure 13). Here, all of the services behave similarly, and get a slight performance increase from HTTP Chunking. Just as with the other tests, the RESTful services perform better than the SOAP service, with the JSON service performing better than the XML service.

VI. Conclusion

In this paper we examined how tuning the application server parameters affected Web services performance

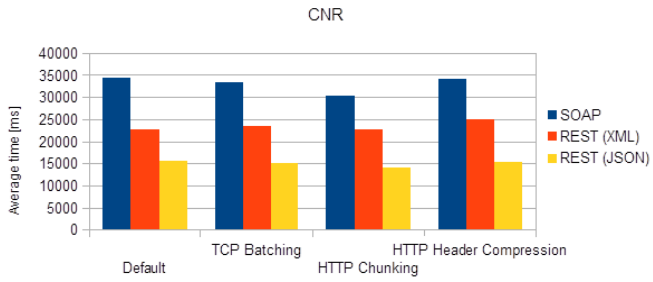

Fig. 13. CNR compression, chunking and batching tests

for different networks defined by IST-118. As we have seen in the SOAP in-depth testing, few of the network parameters actually had any significant impact on the performance of the test system. Some of the timeout and buffer parameters caused errors in some of the networks when set very low, but as the performance was stable for both higher timeout and buffer values, all of these values can safely be set high to prevent premature timeouts and full buffers.

HTTP Chunking on the other hand, proved to be the parameter with the most impact on performance. HTTP Chunking alone significantly improved performance for all of the networks, and combining HTTP Chunking with other parameters improved performance even further for some of the networks. For the SATCOM network, combining HTTP Chunking with TCP Batching gave a slightly better result than HTTP Chunking alone. Combining HTTP Chunking with TCP Batching also gave a better result for the LOS network, while HTTP Header Compression on the other hand reduced performance for this network. For all of the WIFI networks only HTTP Chunking affected the performance, while HTTP Header Compression and TCP Bathing had little effect. Last but not least, for the CNR network HTTP Chunking was the only parameter that improved performance, while TCP Batching reduced performance.

The tests with the RESTful services showed that both of the RESTful services behaved in the same way as the SOAP service. Enabling HTTP Chunking in the application service also increases performance, while the effect of enabling HTTP Header Compression and TCP Batching depends on the network. Out of the two RESTful services the one using JSON came out on top, performing better than the one using XML. This indicates that RESTful services could be an interesting alternative to SOAP for certain limited applications where performance is more important than interoperability concerns.

TABLE V. APPLICATION SERVER PARAMETER TUNING RECOMMENDATIONS BY NETWORK TYPE

Networking environment	Recommendation
SATCOM	Use HTTP chunking, TCP batching
LOS	Use HTTP chunking, TCP batching Don't use HTTP Header compression
WIFI (all conditions)	Use HTTP chunking
CNR	Use HTTP chunking Don't use TCP batching

In conclusion, we have summarized our recommendations in Table V.

ACKNOWLEDGMENT

The authors would like to thank Ketil Lund for facilitating the lab experiments.

REFERENCES

[1] J. R. Agre, K. D. Gordon, and M. S. Vassiliou. Practical considerations for use of mobile apps at the tactical edge. 19th International Command and Control Research and Technology Symposium (ICCRTS), Alexandria, VA, USA, June 16-19, 2014.

[2] Augeri, C.J., et al. An Analysis of XML Compression Efficiency. Workshop on Experimental Computer Science (ExpCS). New York, NY, USA., 2007.

[3] P. Bartolomasi, T. Buckman, A. Campbell, J. Grainger, J. Mahaffey, R. Marchand, O. Kruidhof, C. Shawcross, and K. Veum. NATO network enabled capability feasibility study. Version 2.0, October 2005.

[4] T. H. Bloebaum and K. Lund. CoNSIS: Demonstration of SOA Interoperability in Heterogeneous Tactical Networks. Military Communications and Information Systems Conference (MCC), Gdansk, Poland, October 2012.

[5] Consultation, Command and Control Board (C3B). Core enterprise services standards recommendations: The soa baseline profile version 1.7. Enclosure 1 to AC/322-N(2011)0205, NATO Unclassified releasable to EAPC/PFP, 11 November 2011.

[6] R. T. Fielding. Architectural styles and the design of network-based software architectures. Doctoral dissertation, Univ. of California, Irvine, USA, 2000.

[7] R. Fiske, T. Rogula, and L. Schenkels. Mediation of Network Load over Disadvantaged Grids Using Enterprise Service Bus (ESB) Technology. Military Communications and Information Systems Conference (MCC) , Amsterdam, Nederlands, October 2011.

[8] Hugo Haas and Allen Brown (eds.). Web services glossary. W3C working group note, http://www.w3.org/TR/ws-gloss/, 11 February 2004.

[9] F. T. Johnsen, T. H. Bloebaum, M. Avlesen, S. Spjelkavik, and B. Vik. Evaluation of transport protocols for web services. Military Communications and Information Systems Conference (MCC 2013), 7-8 October, Saint-Malo, France, 2013.

[10] F. T. Johnsen, T. H. Bloebaum, P.-P. Meiler, I. Owens, C. Barz, and N. Jansen. IST-118 — SOA recommendations for Disadvantaged Grids in the Tactical Domain. 18th International Command and Control Research and Technology Symposium (ICCRTS), Alexandria, VA, USA, June 19-21, 2013.

[11] F. T. Johnsen, T. H. Bloebaum, L. Schenkels, R. Fiske, M. van Zelm, V. de Sortis, A. van der Zanden, J. Sliwa, and P. Caban. SOA over disadvantaged grids experiment and demonstrator. *Military Communications and Information Systems Conference (MCC 2012), 8-9 October, Gdansk, Poland*, 2012.

[12] K. Lund, E. Skjervold, F. T. Johnsen, T. Hafsøe, and A. Eggen. Robust web services in heterogeneous military networks. IEEE Communications Magazine, Vol. 48, No. 10, October 2010.

[13] T. Podlasek, J. Sliwa, and M. Amanowicz. Efficiency of compression techniques in SOAP. Military Communications and Information Systems Conference (MCC), Wroclaw, Poland, September 2010.

[14] J. Sliwa and B. Jasiul. Efficiency of dynamic content adaptation based on semantic description of web service call context. IEEE Military Communications Conference (MILCOM), Orlando, FL, USA, October 2012.

[15] Teixeira, M.A., et al. New Approaches for XML Data Compression. in proceedings of International Conference on Web Information Systems and Technologies (WEBIST 2012), pp. 233–237., 2012.