Bandwidth optimizations for standards-based publish/subscribe in disadvantaged grids

Espen Skjervold, Ketil Lund, Trude H. Bloebaum, Frank T. Johnsen
Norwegian Defence Research Establishment (FFI)
Kjeller, Norway
E-Mail: {espen.skjervold, ketil.lund, trude-hafsoe.bloebaum, frank-trethan.johnsen}@ffi.no

***Abstract*— NATO has identified Web services as a key enabler for its network enabled capability. Web services facilitate interoperability, easy integration and use of commercial off-the-shelf components, and while request/response-based schemes have hitherto been predominant, publish/subscribe-based services are gaining ground. SOAP-based Web services, however, introduce considerable communication overhead, and optimization must be done to enable use on the tactical level. Data compression is one such optimization, and it works well for large messages. We claim that the inherent characteristics of publish/subscribe-based Web services are such that using difference-based compression will allow effective compression also for small messages.**

In this paper we present the design and implementation of a proof-of-concept mechanism called ZDiff, which we have tested on several types of military data formats. Together with our SOAP-based proxy system it can be used together with commercial off-the-shelf Web services software. The results show that difference-based compression outperforms traditional compression for small messages, at the same time as it never performs worse than traditional compression for larger messages.

***Index terms* – Compression, Web services, publish/subscribe, WS-Notification**

I. Introduction

One of the main challenges of realizing NATO network enabled capability (NNEC) is enabling users to exchange information with each other at all operational levels, even down to the tactical level. There, one frequently encounters so-called disadvantaged grids, which are characterized by low and variable throughput, unreliable connectivity, and energy constraints imposed by the wireless communications grid that links the nodes [1]. NATO has identified Web services as a key enabler for NNEC [2], and we have previously shown that the use of Web services is possible across heterogeneous networks, even down to the tactical level [3].

So far, Commercial Off-The-Shelf (COTS) Web services infrastructure has primarily been used within the request/response paradigm. However, the publish/subscribe paradigm is making headway due to favorable characteristics such as loose and asynchronous coupling between information producers and consumers, as well as push-based data dissemination. The latter improves latency/liveness and eliminates wasted requests, which encumbers request/response mechanisms, since consumers have to poll for new information (see Fig. 1). Furthermore, with the emergence of mobile ad-hoc networks (MANETs) on the tactical level, communication systems have to deal with new challenges such as changing network topology and network fragmentation. Indeed, [4] argues that the decoupling properties of publish/subscribe makes it especially suited for MANET environments.

Web services, whether it is request/response or publish/subscribe, are based on SOAP and eXtensible Markup Language (XML). Standardizing on these formats facilitates interoperability, easy integration and the use of COTS components [3]. On the other side, the verbosity of XML introduces a lot message overhead, making information exchange particularly challenging in low bandwidth environments like disadvantaged grids. As we have shown in previous work [5], compression techniques enable the use of Web services on low bandwidths, but the degree of compression varies greatly with the volumes of data being transferred within a single message. In general, large messages will achieve a better compression ratio than small messages. Also, the data type will influence the achievable compression ratio. For instance, lossless compression algorithms such as Zlib Deflate [6] may compress a 10 kB XML message with a factor of 1:15. A 0.8 kB XML message of the same structure and format on the other hand, only achieves a compression ratio of 1:2 [5].

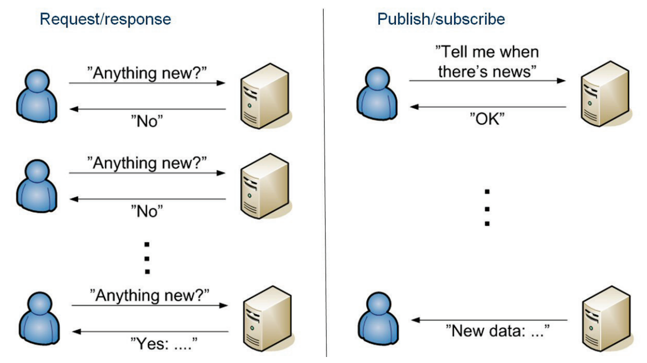

Figure 1. Request/response vs publish/subscribe.

This can be a problem on the tactical level, because much of the network traffic on this level typically consists of data like position updates and chat messages, producing small messages. By aggregating several small messages before sending them, larger messages and thereby better compression ratios, can be achieved. This, however, means that the individual messages must be delayed until the aggregate message is large enough, which in turn means increased and highly variable latency. In situations where, e.g., up-to-date position information is critical, this may not be acceptable.

On the other hand, much of the content of SOAP messages is information that does not change between messages, and this is especially so for publish/subscribe. By taking advantage of the similarity between messages and only transmitting the differences from messages sent earlier, it is possible to achieve a much better compression ratio for small messages than ordinary compression does.

We therefore propose to utilize such difference-based compression for publish/subscribe based on Web services, and in this paper we present a proof-of-concept implementation, called ZDiff. It is an important premise that unmodified COTS software should be able to utilize the compression mechanism, and we have therefore integrated ZDiff in our DSProxy software [3]. This allows the client and application software to be unaware of the existence of the compression mechanism at the same time as low latency information exchange over disadvantaged grids is made possible.

Our measurements show that difference-based compression performs considerably better than Zlib for small messages, while at the same time never doing worse than Zlib for large messages. It is worth noting that, because plain Zlib compression is used as a fallback, ZDiff will never perform worse than Zlib, independent of data type. In addition, contrary to some compression methods, the messages are not altered in any way prior to compression, meaning that digital signatures will still be valid after compression and decompression.

It should also be noted that, similar to other compression mechanisms, the use of ZDiff requires access to the non-encrypted message. This is because encryption removes all visible structure from the message, and thus renders all types of compression impossible. Therefore, the compression component must reside within the trusted domain, and securing the message exchange must be done on lower layers. Consequently, application-level encryption is not feasible unless compression is also done at this level.

The remainder of this paper is structured as follows: We first present related work in Section II, before discussing the background for our work in Section III. Next, we address the concept and implementation in Section IV and present the evaluation in Section V. Finally, Section VI concludes the paper.

II. Related Work

It should be noted that ZDiff is only intended as a proof-of-concept implementation. Our main focus is on showing the feasibility and benefits of using diff-based compression for publish/subscribe in disadvantaged grids. There exist several algorithms and implementations of diff-based compression, and some of these may prove to be at least as efficient as ZDiff. On the other hand, ZDiff demonstrates how existing, well-proven compression software (Zlib) can be reused to achieve differential compression.

RFC 3284 [7] - VCDIFF and the xdelta open source implementation of this diff-based algorithm is an example of previous work using diff for compression purposes. It is targeted towards storage and transmission of files and file versions. It has status as "Proposed standard", and has had this since 2002.

RFC 3229 [8] - Delta encoding in HTTP defines the use of delta encoding for HTTP 1.1. Though this is an old specification (2002), it has not been widely adopted by neither server vendors nor browser developers. Delta coding for HTTP did not see widespread use, because it required modifying servers and browsers in order to add support for the technique. Hence, this inspired our idea in this paper to add delta coding to Web services proxies. This allows us to get the benefits of delta coding for our Web services traffic, while at the same time retaining compatibility with COTS services and clients.

ZDelta [10] is a general purpose, lossless compression library that uses a modified version of Zlib to generate deltas of HTML pages, although it can also be used on other data types. There has been little activity around ZDelta after 2004, but it could potentially be considered as an alternative to ZDiff.

In [9], an edge proxy between tactical network and disadvantaged grids is proposed. This proxy adapts the content passing through it in a way that reduces overhead. This adaptation is an informed content filtering which removes optional fields from XML formatted messages and increases the compression rate of images that pass through it. This does reduce overhead, but in a destructive manner where information is lost, as the content that was sent does not match the content that is received on the other side of the proxy. This requires both clients and services to be aware of the existence of the proxy, as they must be able to cope with the possibility that in-transport modification of messages may occur. This is particularly challenging if one attempts to leverage Web services security standards, because modifying messages will cause digital signature verification to fail. Hence our orthogonal approach in this paper - we developed a proxy which can function with COTS clients and services. As a result, we get the benefit of reduced network resource usage without the penalty of having to tailor clients and services to our solution.

III. Background

A. Publish/subscribe

In publish/subscribe systems, information consumers need to be able to express which types of information they are interested in. This can be done in one of two ways, either by the use of topics, or through content-based filtering. The NATO Core Enterprise Services Working Group (CESWG) has chosen WS-Notification for its publish/subscribe services. WS-Notification is an OASIS-approved standard for Web services publish/subscribe, and comprises three parts, - WS-BaseNotification, WS-BrokeredNotification and WS-Topics. Together, they enable consumers (or some other node on behalf of a consumer) to subscribe to a broker (or some other node acting as a subscription manager) for information published under a specific topic. When producers publish information to the broker under a specific topic, the broker then pushes a notification message to all subscribing consumers. This is illustrated in Fig. 2, where a producer publishes notifications on two different topics, "NFFI" and "NVG".

This scheme allows for the decoupling of consumers and publishers, and enables consumers to receive only what they are interested in. As with all publish/subscribe-based mechanisms, new data are pushed to the consumer after being published, eliminating the need for consumers to poll for new data. This enables instant data dissemination, and eliminates the risk of the request returning empty handed, which happens when requests are made when no new data is available. Fewer wasted requests means reduced network traffic.

A consumer that has expressed interest in a specific topic will need to have the notifications published on that topic delivered in a data format it understands. In a Web services context, this often means that all information published on a specific topic is of the same type, i.e., uses the same XML Schema definition. This in turn means that consecutive messages exchanged between any pair of nodes are likely to be very similar in structure and format. In many cases, much of the content will also be repeated from message to message.

This high degree of similarity between consecutive messages makes compression techniques based on message differencing an obvious choice. In other words, even in the case of small messages that normally compress poorly, there is a potential for significant message size reduction by utilizing knowledge about previous message content in the compression process.

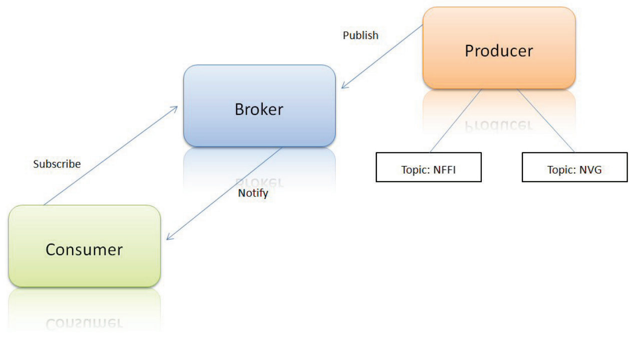

Figure 2. WS-Notification components.

B. *DSProxy*

While compression enables the use of XML and SOAP in disadvantaged grids, frequent delay and disruptions make end-to-end connections difficult to initiate and maintain. Connection-oriented transport protocols such as TCP break down when the delays get too long, or when disruptions occur. In practice, TCP is non-functional in many wireless tactical environments [11]. Standard Web services and publish/subscribe-based schemes such as WS-Notification rely on TCP-based end-to-end connections for communications between consumers, producers and brokers. In [3] we describe how the need for persistent end-to-end connections can be eliminated, using the Delay and disruption tolerant SOAP Proxy (DSProxy).

We designed the DSProxy in order to extend the reach of Web services from networks with infrastructure into the tactical domain [3]. It is a self-organizing proxy-based network overlay enabling robust Web services across heterogeneous networks. By presenting a standard HTTP/TCP interface to the applications, COTS Web services as well as WS-notification brokers and consumers are able to send and receive SOAP messages through disadvantaged grids. This is illustrated in Fig. 3.

Each DSProxy node throughout the overlay provides store-and-forward functionality, adding robustness to the network. The Web services client opens an ordinary TCP connection to its nearest DSProxy node that is held open for the duration of the call. The Web services invocation SOAP message is then routed between DSProxy nodes across the overlay, and once it arrives at the correct DSProxy node, the requested Web service is invoked. The Web services response SOAP message is then routed back to the first DSProxy, which in turn returns it to the Web services client, and then closes the TCP connection. Normally, a DSProxy is deployed on both the client machine and the server machine, meaning that the previously required end-to-end TCP connection is reduced to a local TCP connection between the client and its local DSProxy and between the service and its local DSProxy.

The DSProxy already uses standard compression (ZLib), but we have now further refined the overhead reducing capabilities by introducing difference-based compression, or "diffing", as described in this paper. The reason for integrating ZDiff in the DSProxy is to utlilize the standard HTTP/TCP-based interface of the DSProxy, allowing COTS software to take advantage of ZDiff.

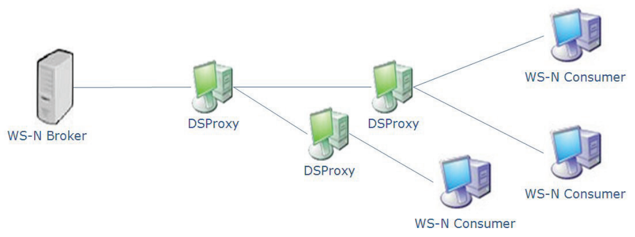

Figure 3. A simple network layout adding store-and-forward capability to a publish/subscribe system using DSProxy nodes.

C. *Compression*

When deployed in a publish/subscribe system, the DSProxy nodes will carry both subscription messages from the consumers to the producers (or notification brokers/subscription managers, depending on the setup), and notification messages from the producers (or notification brokers) to the consumers. In order to reduce bandwidth usage between DSProxy nodes, all messages are compressed using Zlib. SOAP messages, using XML, will usually see significant size reductions when compressed. Zlib employs two compression strategies, the Huffman encoding and LZ77 [12]. While Huffman encoding creates an alphabet allowing frequently used characters to be represented by very few bits, LZ77 is a lossless substitution coder algorithm that finds sequences of data that are repeated. The repeated data are then substituted with a reference to a previous occurrence of the same data, requiring only a few bytes to represent the starting position and length.

Because LZ77 substitutes recurring data sequences with references, its ability to compress data depends on the extent of repeating data within the message. While this may vary from message to message, large messages tend to contain more repeating data, yielding higher compression ratios. This is illustrated by [5], where NATO Friendly Force Information (NFFI) track messages containing a varying number of tracks are compressed using GZip (Zlib). While compressed messages containing a single track achieves a compression ratio of 1:2, messages containing 10 tracks achieve a ratio of 1:15. This is due to XML being verbose, and the fact that a lot of the XML that makes up a track is metadata in the form of tags, attributes and namespace references describing the data. Because this metadata are repeated for every track, compression ratios will improve with the number of tracks per message. This fact can be exploited by aggregating NFFI messages from multiple sources over a period of time, constructing aggregate messages containing several smaller messages, thus achieving better compression ratios. However, this leads to a compromise between bandwidth requirements and liveness/latency requirements and is clearly not ideal. To preserve the low latency/liveness property, aggregation should be avoided, and single tracks should be sent without any delay.

IV. Concept and Implementation

Even though the DSProxy already enables Web services and XML/SOAP-based publish/subscribe schemes in disadvantaged grids, bandwidth scarcity is still a factor and may become prohibitive when using a publish/subscribe system that produces many small notification messages frequently. To improve the compression ratios for small messages without compromising liveness/latency, we have devised a diff-based compression scheme to be used with the DSProxy network overlay system. Because DSProxy nodes can be deployed throughout the network, reliable exchange of diff-based messages can be performed between any two DSProxy nodes.

In order to achieve favorable compression ratios on small messages one can create a virtual aggregate message based on data previously exchanged with other nodes. By keeping track of previously exchanged messages (a local cache), and keeping track of which nodes they are exchanged with, a virtual message can be constructed by the sender, comprising one or more previously exchanged messages.

Once a node receives a new message M to forward, it first constructs the virtual message based on a subset of the previously exchanged messages, called message V_1. It then creates a new virtual message V_2, such that $V_2 = V_1 + M$. Both messages V_1 and V_2 are now compressed using Zlib. The resulting compressed messages are called $C(V_1)$ and $C(V_2)$, respectively.

Next, Diff = $C(V_2) - C(V_1)$ is computed. Diff is now a compressed, binary representation of the difference between M and V_1, and together with a few bytes of metadata (describing the IDs of the messages V_1 was constructed from), this constitutes the message (also known as a patch) sent to the receiving DSProxy. In Fig. 4, an example is shown where V_1 is constructed from messages 1, 2 and 4.

On the receiving side, V_1 and V_2 can be reconstructed from Diff, the message IDs and the local message cache. By decompressing both virtual messages, message M can be reconstructed as $M = V_2 - V_1$. The receiver will now add M to its message cache for future referencing. It is important to note that the reconstructed M on the receiving side is an exact copy of M on the sending side. This means that digital signatures will still be valid, as opposed to some compression schemes that remove unnecessary data (white spaces, prefixes, etc.) in order to improve compression.

The scheme described above is basically a difference-based compression scheme, which in effect allows a patch to be created from the differences between one target document and multiple source documents. The patch will contain both new data and references to series of shared sequences of data between the target document and the source documents. The scheme, which we have named ZDiff, was implemented as a software component using Java, and embedded into the DSProxy middleware system. When utilized by the DSProxy system, the source documents are the messages previously exchanged between any two DSProxys, and bandwidth usage is reduced by sending the patch instead of the entire compressed message. As discussed in the "Results and Discussion" section, this amounts to significant bandwidth savings while preserving the liveness/latency properties and providing exact/lossless message reconstruction.

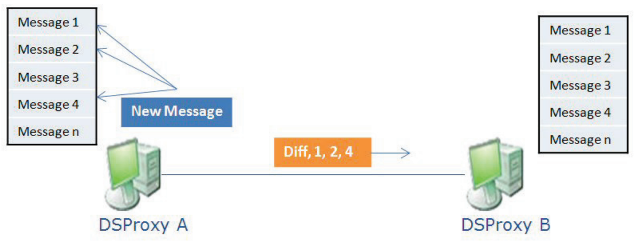

Figure 4. Example of diff-based message transfer.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2012.6415635

ZDiff was designed to produce the delta between multiple source messages (cached messages) and one target message (the new message), unlike most existing diff-based compression protocols, which takes one source and one target message. The ZDiff protocol was developed in Java, and uses JZlib [13] (version 1.1.1) internally. JZlib is an open source Java project designed to be a pure-Java re-implementation of the Zlib algorithms. It supports the originally specified Z_Partial_Flush mode, which for unclear reasons have been deprecated in later versions of Zlib. The partial flush mode ensures that when compressing data, hitherto buffered data are finalized and output, but allowing the next packet to continue using compression tables from the end of the previous packet.

This is key to the ZDiff protocol, because it allows the new message to be compressed and concatenated at the end of the virtual message, without altering the compressed binary data preceding it. This allows the receiver to recreate the compressed virtual message-based on its cache, and then concatenate the received diff, together forming a valid Zlib compressed binary set which can then be decompressed to restore the new message. The JZlib deflator was configured to use the highest compression level (Zlib level 9), ensuring the smallest possible compressed messages and diffs.

The patch messages produced by ZDiff contain, in addition to the actual set differences, a few bytes of metadata representing the number of referenced messages and their respective unique ids. The total metadata size amounts to 4 + (n x 4) bytes, where n is the number of referenced documents. A patch referencing 3 previously exchanged messages will thus contain 16 bytes of metadata.

V. Results and Discussion

The application specific efficiency of diff-based compression was measured by using ZDiff to compress NFFI messages, NVG messages (Nato Vector Graphics) and MIP messages (Multilateral Interoperability Programme, JC3IEDM), standardized XML formats used for military applications. Their compressed sizes were then compared to the uncompressed messages' sizes, their sizes when compressed with Zlib only, and their sizes when compressed with EXIficient [14] (version 0.8). The latter is an open source implementation of the W3C Efficient XML Interchange (EXI) [15] format specification written in the Java programming language, and is a protocol especially designed for high efficiency XML compression. EXIficient was used in both schema-less and schema-informed modes, the latter requiring the XML schemas for the message formats being compressed, yielding higher compression ratios. EXIficient was configured to use the Default fidelity options and the COMPRESSION coding mode. For all tests, Zlib was configured for best compression, using Zlib compression level 9.

Because ZDiff produces the difference between the new message and previously exchanged messages, at least one previously exchanged message needs to be present in the sender's and receiver's cache. The worst case situation for ZDiff, which occurs when no messages have yet been exchanged, produces a message compressed with Zlib only, with exactly 4 bytes of message overhead. For all the following tests, it's assumed that at least one message has already been exchanged. For the setup, the sender and receiver's cache was populated with one message of the same format as the one being sent. The new message was modified, so that all fields carrying format-specific information (the actual content, or payload) were altered using arbitrary values of the same length as in the cached message. For the NFFI messages, these were fields like system id, transponder id, datetime, latitude and longitude.

As shown in Fig. 5, ZDiff performs better than the other compression techniques in all cases. For the NFFI message, the Zlib compressed message size is 40.5% of the original message, while the ZDiff compressed message size is a mere 7.6%. EXIficient performed better than pure Zlib with 28.3% for schema-less mode, and 27.2% for schema informed mode. While the best EXIficient compression was 1.5 times better than Zlib alone, ZDiff was yet 3.6 times better than EXIficient, a significant improvement.

WS-Security is often used with SOAP-based services, and comprises a flexible and feature-rich SOAP extension providing end-to-end security. It can be used to add digital signatures, certificates and confidentiality labels to SOAP messages, facilitating authentication and authorization. While being very useful for providing security in systems using SOAP-based messaging, it may significantly add to the total message size. As shown in Fig. 5, the "NFFI" message increases from 1107 to 4560 bytes (labeled "NFFI w/ security") when signature, certificate, and confidentiality labels are added. While this is fine for internet-based systems, it may become a problem on the tactical level and in disadvantaged grids. Standard Zlib compression, as well as schema-less and schema-informed EXIficient compression may remedy the situation to some extent, yielding compressed message sizes of 2026, 1715 and 1576 bytes respectively. The compressed message produced with ZDiff however, is only 115 bytes large, a mere 2.5 % of the original message size.

The reason for the big differences in the compression results lies in the structure of the SOAP data associated with the WS-Security extension. In addition to being verbose and voluminous, there is little repeating data, making it less suitable for standard Zlib compression. Furthermore, the message contains a certificate requiring 890 bytes of Base64 encoded data, which compress poorly. However, this data remain unmodified between messages created by the same producer/system, and the ZDiff protocol can take advantage of this by replacing the certificate in its entirety with a reference, requiring only a few bytes. For messages carrying a unique certificate sent for the first time, which occurs once every time a WS-Security enabled producer sends its first message, ZDiff's efficiency decreases somewhat, as seen from the bar labeled "NFFI w/security new origin" in Fig. 5. While still half the size of EXIficient, the compressed message does increase from 2.5% to 16.9% of the uncompressed message size.

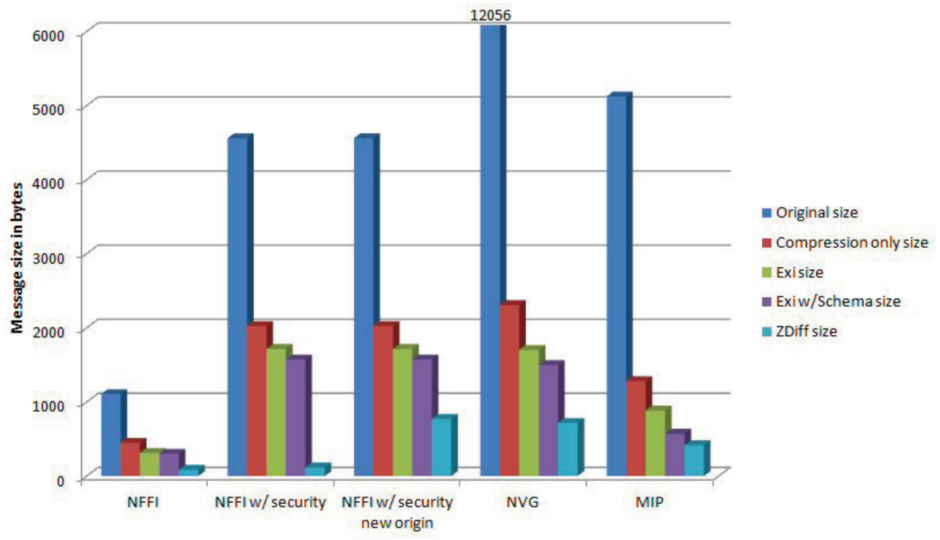

Figure 5. Comparison of message sizes for the different messages using the different compression techniques. Note that the blue NVG bar is truncated for readability.

The NVG message compression tests display yet a different pattern, in that all compression algorithms perform fairly well. This is due to the fact that the NVG message contains a lot of repeating data, in the form of recurring XML tags and namespace references. This allows Zlib to substitute a lot of data sequences with references pointing to previously occurrences of the same sequences, yielding high compression efficiency. ZDiff still performs 2.1 times better than EXIficient, producing a compressed message size that is only 6% of the uncompressed message.

Finally the MIP message achieves a compression rate of 1:12 using ZDiff, or 8.1% of the uncompressed image, making it 1.4 times more efficient than schema-informed EXIficient. As with the NVG message, the MIP message contains a lot of repeating data, enabling all algorithms to perform decently.

While the aforementioned results show significant improvements using ZDiff, they also demonstrate efficiency differences based on the content and structure of the XML formats. In order to present a more general performance evaluation of the ZDiff scheme, an experiment was performed using random data.

Fig. 6 shows compression efficiency when random changes are inserted into a message before performing ZDiff-based compression. The test is initiated with two identical 1000 byte large messages, one being the source message and one being the target message. The two messages are identical, constructed from random data. Then, the target message is modified, substituting parts of it with new, random data. Compression tests are performed 101 times (0%-100% changes), each time introducing one more percent of changes to the target message. The graph is based on the averages over 1000 runs. For the sequential placement scenario shown in Fig. 6, the random data modifications are all introduced in one continuous block at the beginning of the target message. This causes ZDiff to create a patch starting with the changed data, followed by exactly one reference for the remainder of the message, referring to one continuous block in the source document. As the portion of the target message which is being changed grows, the patch grows in a linear fashion, presented with the blue line in Fig. 6.

For the second test, the aforementioned setup is repeated, but now the random data are inserted (substituted) at random places within the message, one byte at a time.

Measures are taken to avoid two random insertions being placed in the same position. Because the modified data are no longer located within a continuous block, the references created by ZDiff will substitute progressively smaller sequences of data. Instead of one reference pointing to a large block, there will be more and more references in between short sequences of modified data. As seen from the red line in Fig. 6, the random placement generally produces significantly larger patches than sequential insertion, but the efficiency improves with the percentage of the message being modified. This is because while bytes are inserted at random locations, with the increasing number of insertions, an increasing number of changes will end up forming continuous sequences of changed data.

While ZDiff produces patches smaller than the uncompressed message for up to 91% changes using sequential placement, it is only beneficial for up to 54% changes when using random placement. Because randomized data are very different from SOAP and XML, these findings should not be used to infer efficiency when dealing with actual SOAP messages. They do however illustrate how compression efficiency changes with the distribution and amount of changes between messages. Note that it would not make sense comparing these results with Zlib, as random data compress very poorly (actually producing a message larger than the original for this test).

In order to compare CPU requirements for the different compression schemes, compression was performed 10,000 times using each solution. The message compressed was a single track NFFI message 1107 bytes large. The averages are presented in Table 1.

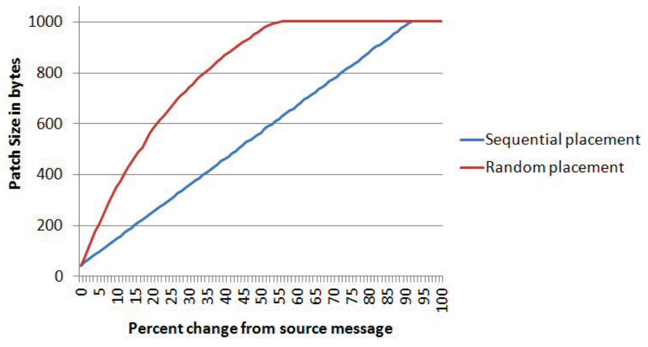

Figure 6. The blue line showing patch size as the amount of sequential random data increases, the red line showing patch size as the amount of randomly inserted random data increases.

As seen in Table 1, standard Zlib by far outperforms the other solutions in terms of processing time. ZDiff performs worst, taking 1.3 times longer than EXIficient w/Schema, and 12.2 times longer than pure Zlib. We argue that this is an acceptable tradeoff in order to significantly reduce network utilization, enabling instant dissemination of small, standards-based messages in publish/subscribe systems on the tactical level. It should also be noted that other diff-based algorithms may be faster than ZDiff. Our prototype implementation, however, shows that processing time is not a prohibitive factor with diff-based compression.

All the evaluations and comparisons above have been based on the premise that ZDiff is performed with one message of the same format present in the sender and the receiver's cache, serving as the source message. When using a DSProxy overlay, multiple messages of different formats would be cached, in order to be able to perform ZDiff compression regardless of the format of the received messages. In topic-based notification systems it makes sense to cache and group messages within a data structure according to their topics. Messages published under the same topic will most likely be of the same format, and should be diffed against other messages of the same format in order to achieve favorable compression rates.

Depending on the resource constraints for the nodes in a given network, the SOAP proxies should be configured to cache a given number of messages of each format, e.g., 10. When ZDiff on the sender side receives a message to compress, it simultaneously receives the 10 cached messages of the same format, and then performs diff-based compression against all ten, one by one. It then determines which cached message yielded the best compression ratio, and returns the corresponding patch to the SOAP middleware component for transmission. Because ZDiff performs the compression 10 times, the total processing time will increase with the same factor.

TABLE I. AVERAGE CPU TIME IN MILLISECONDS PER MESSAGE USING DIFFERENT COMPRESSION SCHEMES

CPU times				
Compression scheme	Zlib	EXIficient	EXIficient w/schema	ZDiff
Milliseconds per message	0.22	2.03	2.10	2.69

While ZDiff scales linearly with the number of messages to diff against and thus is predictable, a very large cache size together with large message sizes in a high throughput system could potentially put too high load on the CPU on limited devices. Therefore, ZDiff can be utilized in ratio-bounded mode and time-bounded mode. The former halts diffing towards cached messages when a threshold compression ratio is achieved, and the latter halts the diffing when the specified amount of milliseconds has been exhausted. Theoretically, ZDiff could allow a new message to be diffed against multiple messages simultaneously, referencing several messages in the same patch. In practice however, this has proven to yield insignificant gains for typical scenarios. ZDiff benefits by referencing shared data across messages, which is usually things like XML tags, attributes and namespace references, normally present in every message of the same format. One could, however, imagine situations where a new message also contains dynamic content, which have earlier been exchanged over several messages. Here, differencing based on multiple messages may prove beneficial.

Aside from ZDiff, schema-informed EXIficient achieves the highest compression rates for all tests. However, unlike both ZDiff and Zlib, it does not reconstruct an exact copy of the source message. This is due to some of its optimization strategies, such as removing whitespaces and reducing precision for floating-point numbers [15]. While this would be fine for most applications, it constitutes a problem when dealing with WS-Security mechanisms such as digital signatures. When signing a message, a hash value is calculated for the part of the message one wants to sign. Then, the cryptographic signature is produced based on the hash value. If the signed portion of the SOAP is altered ever so slightly, e.g. by removing a whitespace, the corresponding hash value will change, and the signature verification will fail. While the EXI specification describes a compression mode allowing exact reconstruction, all our attempts achieving this with EXIficient failed. It should also be noted that doing so would decrease the compression ratio achieved with EXIficient.

A requirement for using schema-informed EXIficient is for the XML schemas to be pre-distributed to all nodes sending and receiving the messages. With ZDiff, this is not necessary, potentially easing the deployment process and providing more flexibility. A catch with ZDiff however, is that the first message of any data format cannot be diffed, and will only be compressed with pure Zlib. While this means the first message of any format will be larger than if compressed with schema-informed EXIficient, one could just as easily pre-distribute the first message (a sample message of the specific format), and instruct ZDiff to include this in its list of source messages. However, we argue that although the first message will be somewhat larger, this is quickly compensated when receiving the subsequent diffed messages.

VI. Conclusion

Using ZDiff, our prototype implementation of a difference-based compression scheme, together with DSProxy, a SOAP proxy overlay, we demonstrated significant compression rate improvements for small SOAP messages. Tests using random generated messages show that ZDiff's compression efficiency changes with the distribution and amount of changes between messages. Our implementation performed better than standard lossless data compression (Zlib), as well as the specialized, high performance XML compression standard EXI, yielding significantly smaller message sizes for most tests. For very small messages (1100 bytes), ZDiff performed 5.3 times better than Zlib, and 3.6 times better than EXIficient. Because ZDiff restores an exact copy of the received message, it works well will security standards such as WS-Security and digital signatures. Such messages are also ideal for diff-based compression, achieving compression ratios as high as 1:40. While requiring slightly more processing time per message, we argue that this is an acceptable tradeoff for enabling standards-based publish/subscribe in disadvantaged grids on the tactical level.

References

[1] A. Gibb, H. Fassbender, M. Schmeing, J. Michalak, and J. E.Wieselthier. Information management over disadvantaged grids. Final report of the RTO Information Systems Technology Panel, Task Group IST-030 / RTG-012, RTO-TR-IST-030, 2007.

[2] 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.

[3] Lund, K., Skjervold, E., Johnsen, F. T., Hafsøe, T., Eggen, A., Robust Web Services in Heterogeneous Military Networks, IEEE Communications Magazine, Vol. 48, No. 10, October 2010, pp. 78-83.

[4] P. Costa et al., "Socially-Aware Routing for Publish-Subscribe in DelayTolerant Mobile Ad Hoc Networks," IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 26, NO. 5, JUNE 2008.

[5] F. T. Johnsen and T. Hafsøe, "Using NFFI Web Services on the tactical level: An evaluation of compression techniques", 13th ICCRTS, Seattle, USA, June 2008.

[6] [RFC-1951] Deutch, P: "DEFLATE Compressed Data Format Specification version 1.3", May 1996.

[7] The VCDIFF Generic Differencing and Compression Data Format, <http://tools.ietf.org/html/rfc3284, June 2002>.

[8] Delta Encoding in HTTP, <http://www.ietf.org/rfc/rfc3229.txt>, January 2002

[9] Śliwa J., Gleba K., Amanowicz M., Adaptation Framework foR web services prOvisionin tactical environment, MCC 2010: Military Communications and InformationSystems Conference, Wrocław, 27-28.09.2010; in: Concepts and Implementationsfor Innovative Military Communications and Information Technologies, Warszawa: Redakcja Wydawnictw Wojskowej Akademii Technicznej, 2010. ISBN 978-83-61486-70-1, s. 213-227(MK-301).

[10] Trendafilov, D., Memon, N., and Suel, T., "zdelta: An Efficient Delta Compression Tool", Technical report, TR-CIS-2002-02 (6/26/2002).

[11] A. Gibb. Challenges for middleware imposed by the tactical army communications environment. NATO IST-030/RTG-012 Workshop on 'Role of Middleware in Systems Functioning over Mobile Communication Networks', 2003.

[12] D. Salomon. Data Compression — The Complete Reference, 2nd edition. Springer, 2000

[13] JZlib - zlib in pure Java, <http://www.jcraft.com/jzlib/>

[14] EXIficient, <http://exificient.sourceforge.net/>.

[15] Kamiya, T. and J. Schneider, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.