19th ICCRTS

Title of Paper

Collaboration services: Enabling chat in disadvantaged grids

Topic

Topic 6: Cyberspace, Communications, and Information Networks

Name of Authors

F.T. Johnsen, T.H. Bloebaum,
Norwegian Defence Research Establishment (FFI)

K.M. Kittilsen and team*
Norwegian University of Science and Technology (NTNU)

Point of Contact

Frank T. Johnsen
Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller
Norway

E-mail: Frank-Trethan.Johnsen@ffi.no

(*the team: Luka Cetusic, Hans Kristian Flaatten, Karsten Kjensmo, Erik Lothe, Ole Johnny Pettersen, Thomas Martin Schmid and Bjørn Tungesvik)

This page is intentionally left blank

Collaboration services: Enabling chat in disadvantaged grids

Abstract

Instant messaging is an important aspect of collaboration. There are many different solutions for instant messaging or "chat" as it is often called. One of the most prominent solutions in recent years is the XMPP protocol, which is implemented in several instant messaging products, both servers and clients. This protocol has also been chosen for chat by NATO, as it is mentioned in the SOA baseline as one of the protocols to use when implementing the collaboration core services. NATO's JChat client implements XMPP, and has been used with success in many missions. However, the protocol is not well suited for use in highly dynamic environments. In this paper we present our approach to bringing chat into such environments. We build our chat solution on ACP142, a protocol developed for use in tactical radio networks that can cope with mobility and disruptions. Finally, we discuss how a gateway solution can be used to bridge our experimental chat with the standard XMPP which should be used in networks with infrastructure.

1. Introduction

This work has been performed in the context of NATO STO/IST-118 "SOA recommendations for disadvantaged grids in the tactical domain" [2]. The main focus of this group is to identify what we call *tactical SOA foundation services*. By this we mean which core enterprise services we need support for in the tactical domain. Examples so far include, but are not limited to, the messaging service, publish/subscribe service, collaboration, and service discovery service. In other words, we aim to investigate how services from the SOA baseline [3] can be extended for use in tactical networks. In this paper we focus on the *Instant messaging service,* which is a subset of *Collaboration* in the SOA baseline. The baseline recognizes that there are additional collaboration services and states that more services will be addressed in future versions, but currently only instant messaging/chat is identified with a standard to employ. For chat, the SOA baseline points to XMPP. XMPP is the most important chat protocol in use in NATO today, as it is the protocol being used in JChat. JChat is currently being evaluated for use in disadvantaged grids [4]. However, XMPP is intended for use in stable networks with high bit rate (e.g., LAN, Internet) and does not function well in military tactical networks where resources are scarce and disruptions are frequent, as our experiments in [1] have shown. As XMPP is not particularly well suited for use in disadvantaged grids, yet chat is important, it is in the IST-118 group's interest to attempt to extend this aspect of collaboration also to such environments.

Chat is becoming increasingly important in military operations, and such services have been used on several occasions (e.g., in Operation Enduring Freedom and Operation Iraqi Freedom [6], where chat was also used in special operations). In this paper we present our approach to enabling chat in disadvantaged grids by leveraging the ACP142 protocol for transporting the instant messages. This protocol has been designed for use in tactical networks, and can function in certain disadvantaged grids. Our contribution is twofold:

1. We have implemented ACP142 according to the specification and released the implementation as open source.

2. We have implemented a multiuser chat solution for disadvantaged grids on top of ACP142, also released as open source.

The remainder of this paper is organized as follows: In section 2 we discuss our approach in light of related work, and provide a brief introduction to ACP142. Section 3 presents details on our implementations, along with the URLs where the open source code can be obtained. Section 4 summarizes the tests performed, and Section 5 concludes the paper.

2. Background

2.1 Chat solutions for tactical networks

[7] presents an application suite built on top of DoDWAN, where one of the applications is a delay-tolerant text messaging application with group support: TextWAN. ChatWAN is another chat application built on top of DoDWAN [5]. ChatWAN implements a subset of the IRC protocol [8], enabling clients to connect using standard IRC clients.

NATO has chosen to standardize on the eXtensible Messaging and Presence Protocol (XMPP) [9,10] for instant messaging. XMPP is server-based making it ill-suited for use in disadvantaged grids where a central server constitutes a single point of failure. This has led to an increased research focus on developing chat solutions that are adapted to the tactical domain, as well as a shift from the previous focus on IRC to the current focus on XMPP.

Multicast is an efficient means of distributing one message to many recipients. This can be leveraged in order to decentralize a chat application and do away with the central server. For example, a multicast-based solution for secure group chat and XMPP compatibility is discussed in [11, 12]. The authors suggest using a proxy for compatibility with XMPP. Further, in [13, 14], Lass et al. present their design and evaluation of an XMPP overlay protocol for tactical chat based on multicast. By using gateways and proxies, the chat solution is compatible with XMPP clients.

When using multicast, the delivery success rate and bandwidth use depend on the efficiency of the protocol. Reliable multicast protocols can be leveraged, such as NACK-Oriented Reliable Multicast (NORM) [15]. Previously we have created a decentralized chat mechanism with its own reliable distribution mechanism that does not rely on underlying routing or multicast mechanisms, and that can achieve interoperability with XMPP through a gateway. Experiments have shown that this solution significantly outperforms XMPP in networks with disruptions due to high node mobility [1]. This solution (called Mist chat) was presented as input to NATO STO/IST-090 "SOA challenges for real-time and disadvantaged grids" and was a part of the group's final demonstration at MCC 2011 in Amsterdam, Netherlands.

However, if there is a need for a node to go into radio silence, then that solution will not necessarily work well. Ideally, we want a node to be able to receive messages even if it is currently in EMCON[1] and does not transmit anything. This led us to investigate other possible solutions as well. We have now chosen to focus on the ACP142 protocol for reliable multicast, because it has been designed specifically for use in tactical networks. It is a highly configurable protocol that, unlike other reliable

[1]EMCON is short for *emission control*, also known as radio silence, where a node has entered a state where it may receive but not transmit data.

multicast solutions we are aware of, can function under EMCON as well. In this paper we present our current approach to chat in disadvantaged grids, which we have built on top of ACP142. This current solution (called P_MUL Chat), as well as our previous Mist chat, have both been submitted to the NATO STO/IST-ET-070 exploratory team for tactical chat for consideration.

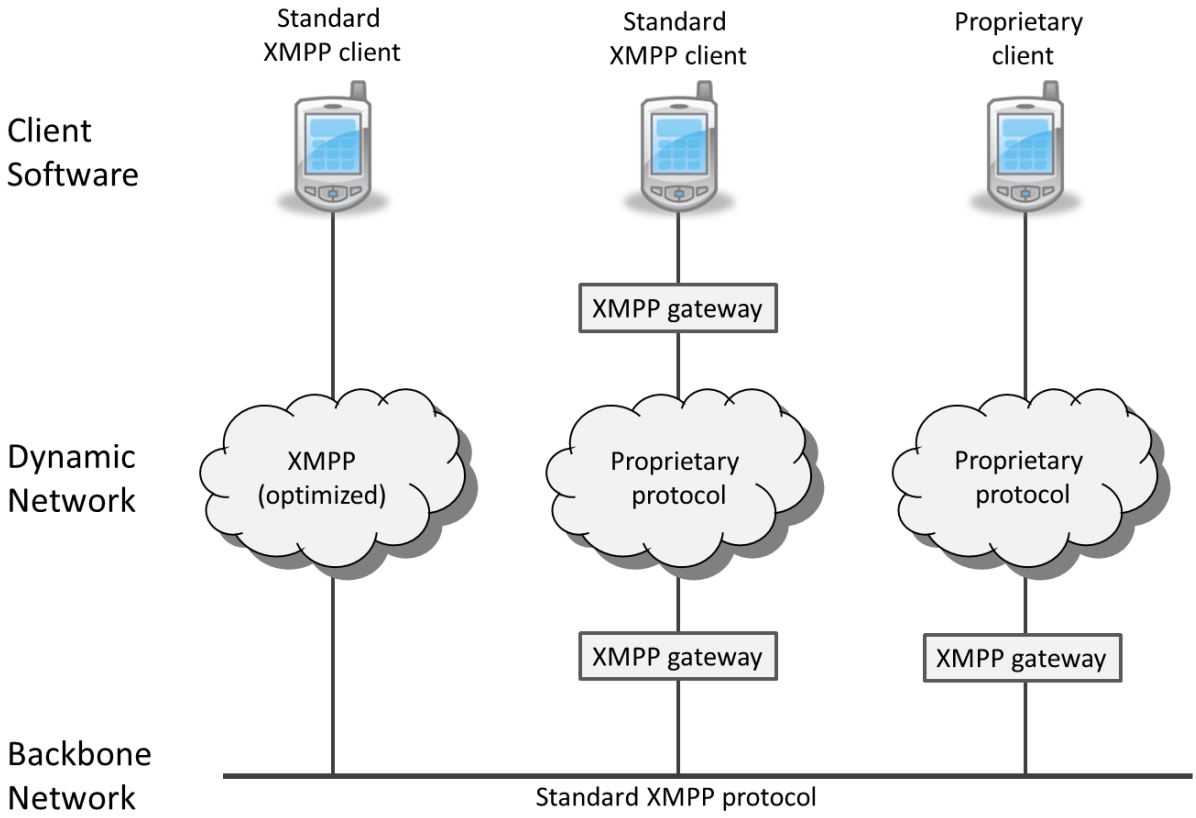

Figure 1: Approaches to implementing Chat solutions

From the literature overview provided above, we have identified three approaches that are commonly used when attempting to realize chat in tactical networks. Figure 1 illustrates these three approaches, from left to right: 1) Attempting to use XMPP directly, but with certain optimizations, 2) Using a proprietary solution in the dynamic environment, but using gateways to achieve interoperability with COTS XMPP clients and servers, and 3) Proprietary client and optimizations, but using a gateway for interoperability with an XMPP server in the backbone network.

2.2 ACP142

ACP 142 [16] provides a protocol definition for reliable multicast information transfer in bandwidth-constrained and delayed-acknowledgement environments to support efficient allied information transfer. The objective of ACP 142 is to specify and standardize the P_MUL protocol. P_MUL, as a reliable multicast protocol, requires an underlying connectionless network infrastructure with multicast routing functionality. Current P_MUL implementations often use the User Diagram Protocol (UDP) and Internet Protocol (IP) as illustrated in Figure 2. The protocol specification does not limit it to this protocol stack, but due to NATO's current "Everything over IP" mindset, we anticipate that this combination will be the most viable for interoperability in the coalition.

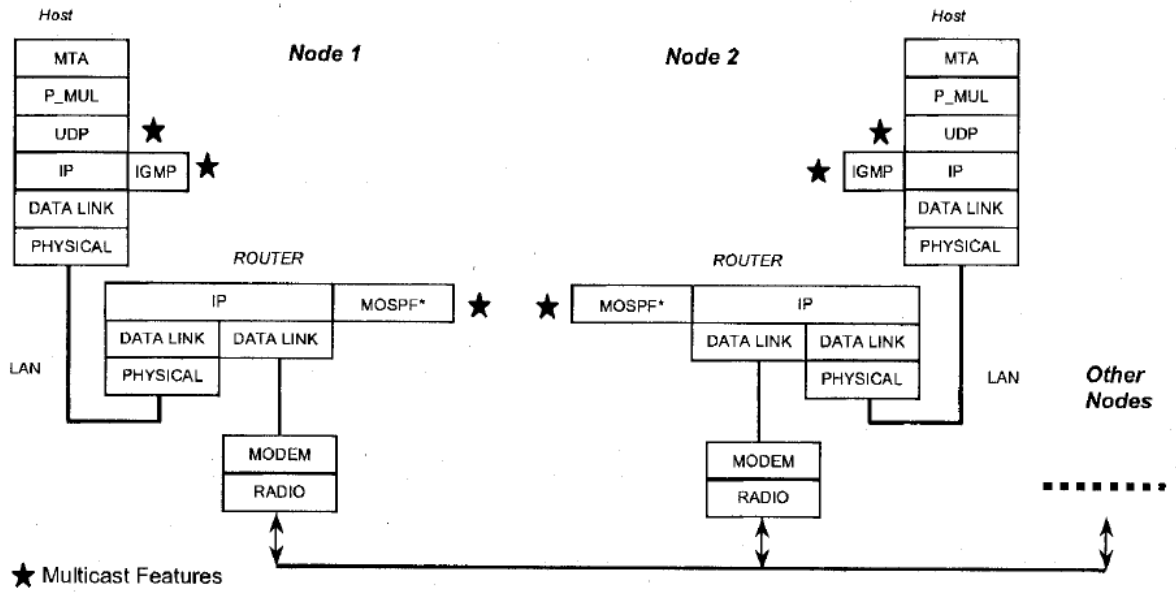

Figure 2: Implementation of P_MUL on an UDP/IP stack (taken from [16])

Although P_MUL is based on a connectionless transport protocol, it provides users with reliable connection-oriented multicast services. It enables the receivers to receive messages while being under EMCON restrictions. It ensures that the transmitter is informed about the timely completeness of the transmission of the messages after the receivers leave the EMCON status and, if required, enables the re-transmission of any messages that were not properly received. P_MUL requires that the router is able to support static multicast routing. P_MUL works for fixed multicast groups and can also work for dynamically formed ad hoc multicast groups. For fixed multicast groups each node has knowledge about the group memberships of one or more multicast groups. Dynamically formed ad hoc multicast groups may be used for transmitting single messages, but implementation of dynamic multicast group formation is an optional capability in the specification. For the complete protocol details, please refer to [16].

3. Design and implementation

For software development, the chosen Integrated Development Environment (IDE) was Eclipse. Eclipse is tightly integrated with Java and has functionalities such as real-time error and warning checking, automated code refactoring and powerful autocomplete capabilities. Both our P_MUL library and our chat solution (called P_MUL chat) were implemented in Java using Eclipse. The solution has been tested in emulated network conditions (see Section 4). As a contribution to the research community we have chosen to release our implementations online as open source at Github, where detailed documentation can also be found. Below we summarize the most important highlights of both implementations.

3.1 P_MUL

The complete P_MUL design and functionality is described in [16], so we will not go into those details here. Instead, we focus on where our implementation goes beyond the mandatory functions specified:

- We implement the optional support for dynamic multicast groups as well as the mandatory static groups.
- We implement support for IPv6 when using static multicast groups, in addition to IPv4 for both static and dynamic multicast groups.

The specification only requires one to implement support for static multicast groups. This implies that when only such groups are used, one must configure all relevant groups at deployment. In context of our intended application of the protocol, as a carrier for chat, it was desirable to also have support for dynamic multicast groups. The reason for this being that one could then map chat topics to multicast groups, and allow new topics to be introduced at run time. When leveraging only static groups, the chat topics to be used must be planned and configured pre-deployment. Thus, we chose to implement this optional functionality as well. Here, only IPv4 is supported, as differences relating to IPv4 and IPv6 addressing meant that this functionality was not easily extended to use IPv6 without going beyond compatibility with the specification. That being said, we still anticipate the most important mode to be the static multicast groups, since that functionality is mandatory and must thus also be present when encountering other implementations of the protocol. So, for complete compatibility with the specification, it is safest to rely on the pre-planned, static multicast group functionality for disseminating messages.

In recent years there has been an increasing interest in IPv6 also for defense use (see e.g., the recent CONSiS experiment [17] where IPv6 was employed). Inspired by this, we chose to go beyond the basic IPv4 addressing in the specification and implement support for both IPv4 and IPv6 when using static multicast groups. Our IPv4 implementation is fully compliant with the specification, whereas the IPv6 implementation uses our own interpretation of how the addressing scheme can be extended to function in an IPv6 based network. By using hashing, we are able to obtain the necessary identifiers of appropriate length and computational strength.

The implementation is free, released as open source, and can be obtained from https://github.com/libjpmul/libjpmul (P_MUL including the above mentioned enhancements). The protocol implementation has been subject to performance and specification compliance testing in an emulated network environment, ensuring that it fulfils the specification.

3.2 P_MUL Chat

The following explains the creation of the chat application design that is built upon the P_MUL protocol. The code has been released as open source at https://github.com/libjpmul/pmulchat where it can be downloaded free of charge.

Figure 3 illustrates the major functionality of the chat application's graphical user interface (GUI). Here, *Chat* indicates the window displaying the chat messages, *Comment field* is the window where you enter your messages, *topic* contains a list of chat topics that can be subscribed to, and the *clear* and *send* buttons do what the names imply. In addition, the actual GUI also has a couple of other functions as well, the most important being a means of configuring the parameters of the underlying P_MUL protocol.

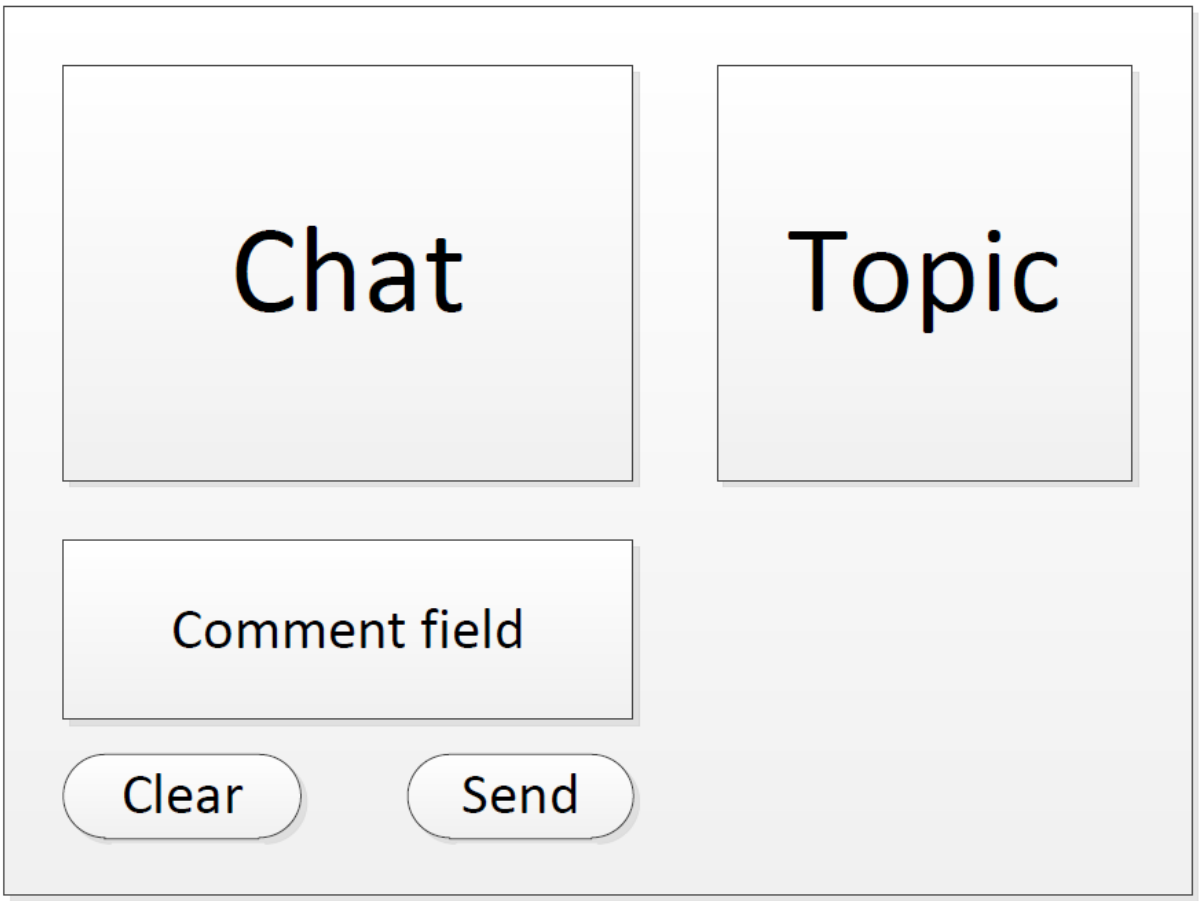

Figure 3: Chat application GUI outline

The chat application relies on a set of messages to function. All messages are used if dynamic multicast groups are leveraged. If static multicast groups are used, then only a subset of the messages is needed. The complete set of messages is:

- SEND_MESSAGE: Messages of this type transmit chat messages to all subscribers of the current topic.
- NODE_LIST: Upon starting the application, it transmits its destination ID over a configurable unreliable UDP Multicast. Any node that sees this will respond with a message of this type, containing the destination IDs of all nodes it knows are online.
- NODE_LEAVE: When the application is terminated, a message of this type is sent out so other nodes know not to transmit to this user any more. To make the quit action responsive, these messages have a very short time to live.
- GET_TOPICS: Asks for a list of all topics, triggers a TOPIC_LIST response.
- NEW_TOPIC: Announces the creation of a new topic. Should a topic with this name already exist, a SUBSCRIBER_LIST response is triggered.
- DELETE_TOPIC_ QUERY: Queries whether any clients object to the deletion of a topic. Any client, the user of which is currently subscribed to this topic, will respond with a TOPIC_IN_USE message.
- DELETE_TOPIC_SUCCESS: Follows a DELETE_TOPIC_QUERY message if no TOPIC_IN_USE message was received during the waiting period. Upon receipt, the topic will be deleted.

- JOIN_TOPIC: Sent when the user selects a new topic to subscribe to. Allows the other clients to know that a new subscriber is now a part of this topic.
- LEAVE_TOPIC: Sent when the user selects a new topic to subscribe to having previously been in another topic. Ensures the other clients in the old topic know that this subscriber is no longer in the topic.
- TOPIC_LIST: Sent upon receipt of a GET_TOPIC message. This is used to share a list of all topics the client is aware of.
- TOPIC_IN_USE: Triggered by a DELETE_TOPIC_QUERY message if the client objects to the deletion of the concerned topic.
- SUBSCRIBER_LIST: When a user creates a new topic that already exists, or joins a topic, this is sent as a response. It contains destination identifiers and user names for all clients currently subscribed to the topic.
- INVALID: Only used internally, this marks messages which are found to be invalid when parsed from raw bytes.

These messages constitute the chat solution's network behavior in the following manner:

3.2.1 Startup

When the application first starts, it transmits its unique identifier over unreliable UDP multicast to a predefined group that all clients listen to. This notifies the group members that this client is now online, and they will respond with a NODE_LIST message containing a list of all clients they know are online. The fact that all clients respond generates significant potentially unnecessary network traffic. Due to the unreliable nature of pure UDP multicast, this is however needed to maximize the chances that the identifiers of all online nodes are received. After receiving the first NODE_LIST response, the client sends out a GET_TOPICS message if dynamic topics are supported. If this is the case, then a node will respond with a TOPIC_LIST message containing all existing topics.

3.2.2 Changing topics

If the application is in static multicast mode, changing topics is not possible at runtime. Otherwise, this will first issue a LEAVE_TOPIC message, given that the client was previously in a different topic, to all subscribers of the topic it is leaving. It will then send out a JOIN_TOPIC message to all clients, to which a node currently subscribing to the joined topic will respond with a SUBSCRIBER_LIST message with all current subscribers to that topic. If no one is subscribed to the newly joined topic, no response is seen.

3.2.3 Sending messages

Messages are sent in a SEND_MESSAGE message to all clients subscribing to the topic the client is currently in.

3.2.4 Creating topics

When creating a topic, a NEW_TOPIC message is sent to all clients. This can only be done if no topic with that name already exists in the local topic list. All other clients will then add that topic to their list, unless they already have a topic with that name in their lists. If they do, then they will respond

with a SUBSCRIBER_LIST message for that topic. The client creating the topic will immediately join the newly created topic.

3.2.5 Deleting topics

A topic can only be deleted if no other client is subscribing to it. A DELETE_TOPIC_QUERY message is sent to all clients to check this. Any client subscribing to that topic will respond with a TOPIC_IN_USE message to indicate that it may not be deleted. If, after a configurable waiting period, the original client has not received a response, it will send out a DELETE_TOPIC_SUCCESS message to all clients. This message will in turn trigger the deletion of that topic.

3.2.6 Quitting

When the chat client is shut down, it sends a NODE_LEAVE message, with a very short time-to-live, to all clients. This notifies them to delete the destination identifier of this client from their recipients list. The short time to live is used so the client shuts down in a reasonably responsive fashion, while still leaving the maximum reach for this final message (to minimize the count of nodes that try to transmit to it after its disconnection).

4. Testing

Testing of the solution was performed in several stages: Code review, unit testing, conformance testing, and validation testing.

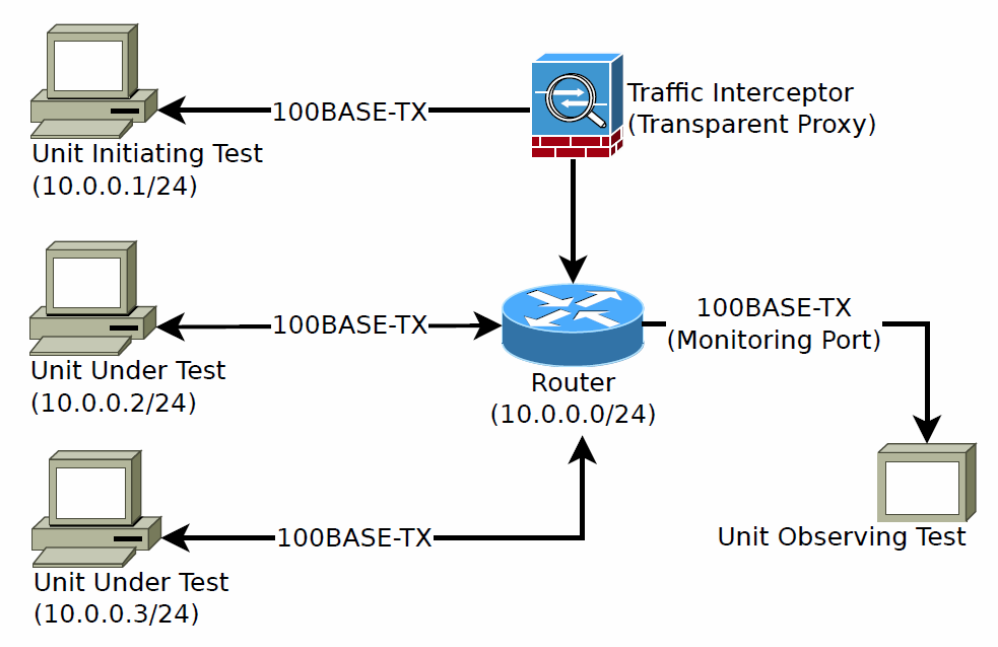

Figure 4: Test environment

For conformance and validation testing we used the setup shown in Figure 4. The router was multicast-enabled since multicast support is a requirement of ACP142. Further, we employed a traffic interceptor for those tests involving unreliable links. This was necessary due to the testbed being wired, so the proxy was used to introduce unreliable network behavior like packet loss. Testing involved the following four approaches (see [19, ch. 7] for complete details):

4.1 Code review

Reading through the implemented code can help identify a lot of potential problems before they arise. By continuously having code inspections on committed code, as well as a well drafted code convention, many problems were caught early on in the implementation phase.

4.2 Unit testing

Unit testing is the task of testing methods individually. This method focuses solely on an expected output given a specific input. It does not test whether the method actually conforms directly with protocol design (this is ensured through conformance testing).

Unit testing ensures that the method does what the programmer intended it to do; technically speaking. Unit testing is an excellent way to test whether newly introduced changes break existing functionality from a technical standpoint. We employed such tests as a means to allow developers to instantly test a subset of the system before committing the changes. We used JUnit, which is the most commonly used framework for unit testing in Java. JUnit is fairly simple to use and integrates nicely with the existing developer tools and environment already in place. Please refer to [19, annex D] for the complete unit test plan and [19, annex F] for the complete results.

4.3 Conformance testing

Protocol conformance testing aims to test to what degree the implemented protocol conforms to its specification. This stage involves using the implementation and sending real packets over a network while monitoring the connection. This process was performed by systematically selecting each major requirement in the protocol specification and then creating a single function test for it.

Depending on the size and complexity of the protocol specification, this type of testing can result in potentially thousands of different tests. It is often infeasible to create that many tests, hence only a subset of them can be selected for implementation – after careful consideration. Please refer to [19, annexes E and G] for further details on the conformance test plan and results.

4.4 Validation testing

When the software development is completed, tested, and assembled as a package, the final series of tests, Validation Testing, begin. Here testing succeeds when “software functions in a manner that can be reasonably expected” [18, p. 521]. These expectations were defined as the overall software requirements, that is the functionality described in Section 3. Software validation was achieved through a series of black-box tests that demonstrated conformity with the requirements. Please refer to [19, section 7.3.4], which describes how the functional requirements were validated in further detail.

5. Conclusion

In this paper we have presented our twofold contribution:

- A Java implementation of the tactical multicast protocol ACP142 that we have released as open source, and

- a solution for chat for use in disadvantaged grids leveraging the above protocol (also released as open source).

Chat is an important aspect of the NATO Core Enterprise Service for Collaboration, and the work presented in this paper has been performed in context of the currently active NATO STO/IST-118 "SOA recommendations for disadvantaged grids in the tactical domain" group.

Our chat solution, that we call P_MUL Chat, is a multiuser protocol for mobile ad-hoc networks based on the reliable tactical multicast protocol ACP142. P_MUL chat is fully decentralized, and leverages multicast groups in ACP142 to differentiate between chat rooms. We suggest that it can be used in combination with XMPP through the use of gateways, along the same lines as other experimental solutions (e.g., as described in [1]). The motivation for creating this chat solution was to be able to leverage the key properties of ACP142 [16] for instant messaging in disadvantaged grids:

- Reliable multicast messaging
- Designed for bandwidth-constrained networks
- Delayed acknowledgement for EMCON environments

Tests in an emulated networking environment have shown that P_MUL Chat is suitable for typical tactical, disadvantaged environments where disruptions occur: both planned phases where certain nodes enter radio silence, and unplanned disruptions due to node mobility and network partitioning. It should be noted that ACP142 provides several parameters for fine-tuning the protocol's behavior (e.g., MTU size, sending delay between packet fragments, etc.) and it is necessary to configure it to match the capabilities of each network before deployment. Default parameters will not function in all networks. Further, we have adopted NATO's "everything over IP" mindset, meaning that we have only implemented and tested P_MUL chat on ACP142's IP stack. This, in turn, puts a requirement on the network where the solution is deployed – it must support IP multicast. This implementation has also been contributed to the community as open source. We think it can be of interest to others, since it is, to the best of our knowledge, the only available ACP142 implementation that is open source and supports both IPv4 and IPv6.

For future work the IST-118 group plans to experiment with other Core Enterprise Services in disadvantaged grids in the tactical domain as well (such as the Publish/Subscribe service), possibly by extending the solutions to run on ACP142.

Acknowledgements

We would like to thank Magnus Skjegstad, Norway's representative to NATO STO/IST-ET-070 exploratory team for tactical chat, for fruitful discussions during this work. His valuable insights contributed to section 2.1. Also, we would like to acknowledge Professor Monica Divitini and Ph.D. candidate Meng Zhu at NTNU for their efforts as supervisors for the student team. Thanks to Torleiv Maseng for his constructive criticism, and to Johnny Johnsen for proofreading. Last but not least, thanks to Stig Bjørlykke at Thales Norway for providing valuable insights to the team when developing and testing the solution.

References

[1] Magnus Skjegstad, Ketil Lund, Espen Skjervold, Frank T. Johnsen, Distributed Chat in Dynamic Networks in Proceedings of IEEE Military Communications Conference (MILCOM) 2011, pp.1651,1657, 7-10 Nov. 2011, Baltimore, MD, USA

[2] 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 ICCRTS, Alexandria, VA, USA, June 19-21, 2013

[3] NATO C3 Board, Core Enterprise Services Standards Recommendations: The SOA baseline profile v.1.7, November 2011. Enclosure 1 to AC/322-N(2011)0205.

[4] NCIA. Joint Tactical Chat (JChat). http://tide.act.nato.int/tidepedia/index.php?title=Joint_Tactical_Chat_%28JChat%29 (access requires a Tidepedia account).

[5] ChatWAN. https://www-casa.irisa.fr/dodwan-expe/chatwan/

[6] B. A. Eovito, The impact of synchonous text-based chat on military command and control, in 11th ICCRTS Coalition Command and Control in the Networked Era, Cambridge, UK, Sept 2006.

[7] Y. Mahéo, N. L. Sommer, P. Launay, F. Guidec, and M. Dragone. Beyond Opportunistic Networking Protocols: a Disruption-Tolerant Application Suite for Disconnected MANETs. http://extremecom2012.ee.ethz.ch/papers/1-extremecom2012-Maheo.pdf, Extremecom, 2012.

[8] C. Kalt. Internet Relay Chat: Client Protocol. RFC 2812 (Informational), Apr. 2000.

[9] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), Mar. 2011.

[10] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 6121 (Proposed Standard), Mar. 2011.

[11] J. Tölle, T. Ginzler, and P. Steinmetz. Challenges of Instant Messaging in Tactical Environments - Concepts and Practical Implementation. In Proceedings of NATO IST-083 Symposium on Military Communications with a special focus on Tactical Communications for Network Centric Operations, Prague, Czech Republic, April 2008.

[12] T. Aurisch and P. Steinmetz. Securely connecting instant messaging systems for ad hoc networks to server based systems. The 16th International Command and Control Research and Technology Symposium (ICCRTS), Quebec City, Canada, 2011.

[13] R. Lass, J. Macker, D. Millar, C. William, and I. Taylor. XO: XMPP overlay service for distributed chat. In Military Communications Conference (MILCOM), IEEE, pages 1116–1121, San Jose, CA, USA, 2010.

[14] R. Lass, D. Nguyen, D. Millar, W. Regli, J. Macker, and R. Adamson. An evaluation of serverless group chat. In Military Communications Conference (MILCOM), pages 1639–1644, Baltimore, ML, USA, 2011.

[15] B. Adamson, C. Bormann, M. Handley, and J. Macker. NACK-Oriented Reliable Multicast (NORM) Transport Protocol. RFC 5740 (Proposed Standard), Nov. 2009.

[16] The Combined Communications-Electronics Board (CCEB), ACP142, P_MUL - A PROTOCOL FOR RELIABLE MULTICAST MESSAGING IN BANDWIDTH CONSTRAINED AND DELAYED ACKNOWLEDGEMENT (EMCON) ENVIRONMENTS, http://jcs.dtic.mil/j6/cceb/acps/acp142/ACP142.pdf

[17] Trude H. Bloebaum and Ketil Lund, CoNSIS: Demonstration of SOA Interoperability in Heterogeneous Tactical Networks, 2012 Military Communications and Information Systems Conference (MCC), Gdansk, Poland, 8-9 Oct 2012

[18] Roger S. Pressman. Software Engineering - A Practitioner's Approach. McGraw-Hill, 4. edition, 1997.

[19] L. Cetusic, H.K. Flaatten, K.M. Kittilsen, K. Kjensmo, E. Lothe, O.J. Pettersen, T.M. Schmid, B. Tungesvik, Report on acp142 (P_MUL) Java implementation , https://github.com/libjpmul/report