Distributed Chat in Dynamic Networks

Magnus Skjegstad
University of Oslo
Oslo, Norway
E-mail: magnus.skjegstad@ifi.uio.no

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

***Abstract*—Chat is becoming increasingly important in military operations. XMPP is the de facto chat protocol in use in NATO today. 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. In this paper, we present our novel decentralized solution specially tailored for multi user chat in tactical mobile ad-hoc networks. Our protocol is implemented in Java, and then tested in emulated network conditions. Compatibility with XMPP is achieved through the use of a gateway, so that compatibility with existing deployed chat infrastructure is ensured. Our tests show that our chat solution has a higher message delivery rate in poor conditions than XMPP.**

I. Introduction

Text based chat is a fast, efficient and non-intrusive way of communicating, it consumes little resources, and is easy to use. Chat also provides an important presence service, as well as simultaneous communication between many people. These characteristics have made chat services very popular on the Internet, and there exists a large number of different chat services. Examples are IRC (Internet Relay Chat), Google Talk, Jabber, Windows Live and ICQ.

Since communication is essential within C2 (Command and Control), chat services have also become popular in the military domain. Chat services have been used in military operations on several occasions. Two examples of this are Operation Enduring Freedom and Operation Iraqi Freedom [1], where chat was also used in special operations. The fact that NATO addresses chat in both the NATO NEC Feasibility Study and through the Core Enterprise Services Working Group (CESWG) shows that chat has become an essential C2 tool.

Common for popular chat services on the Internet is that they are server-based, i.e., all clients must connect to a server, which in turn relays the message to the recipient(s), possibly via other servers. For all such connections, TCP is the overriding protocol. This poses no problems in wired networks, which provide stable topologies and high bandwidths. For mobile ad-hoc networks (MANETs), on the other hand, end-to-end TCP connections can be difficult to establish and maintain.

Mist is an experimental protocol for publish/subscribe in MANETs. It leverages node mobility and application layer epidemic routing to achieve information dissemination without relying on a functioning routing layer or IP multicast support. Mist is resource efficient, and designed to overcome the limitations of networks with scarce resources. It can be used for any publish/subscribe application, and has earlier shown promise when applied as a service discovery protocol for Web services [2]. In this paper, we present a decentralized multi user chat solution for tactical MANETs where an improved version of Mist is used as the message dissemination protocol. Thus, by defining decentralized chat in terms of a publish/subscribe application, we are able to achieve high message delivery rates in disruptive environments.

The remainder of the paper is organized as follows: Section II introduces related work. In Section III we present the design of our chat solution, putting emphasis on the use of Mist, our special purpose chat client, and an XMPP gateway used for interoperability with deployed chat infrastructure. We evaluate our solution in IV. Section V concludes the paper.

II. Related work

In the military domain, IRC has been one of the most widely used chat protocols for C2. NATO has later chosen XMPP as their standard for chat/instant messaging (IM)[1], through a system called JChat. Both IRC and XMPP are server based, and require TCP connections between client and server to be able to transfer messages. As described above, this makes them less suitable for tactical networks, and consequently, there has been some research activity focusing on how to enable chat in MANETs.

In [3], the architecture of a mobile IM system based on XMPP is presented. Each mobile node runs a distributed server entity in addition to the IM client, and presence and information dissemination is achieved using a protocol called Passive Distributed Indexing. Only the presence dissemination has been implemented, and the communication between the servers uses TCP connections that are established using a (non-specified) state of the art MANET protocol.

[4] proposes a Jabber proxy for disruption tolerant networks (DTN), that enables standard Jabber clients to utilize DTN by redirecting the Jabber traffic through the DTN stack. In addition, it prevents TCP timeouts, and it allows group chat messages to utilize last mile multicast. The authors have performed a proof of concept evaluation, but no measurements are presented.

In [5], a solution based on an XMPP gateway is proposed by Tölle et al. A peer-to-peer based, server-less «tactical chat» for use in the tactical domain can connect through this

[1] In the remainder of this document we will use chat and IM interchangeably, although IM is sometimes viewed as a subtype of chat.

gateway and reach other, standard XMPP clients. The paper focuses on the requirements of «tactical chat», and a future implementation is outlined. No details are provided beyond the choice of hardware and programming language. We adopt their suggestion for a gateway approach in our work, thus enabling a connection from our experimental decentralized chat solution to interact with a deployed XMPP infrastructure.

The proposed solution in [5] has recently been implemented by Aurisch et al. as an experimental solution called «Chat and Instant Messaging for Tactical Environments» (CIM-TE) [6]. CIM-TE provides secure group chat to a set of connected devices.The CIM-TE protocol relies on IP multicast to function, meaning that it is limited to MANETs with IP multicast support in the routing protocol. Thus, the main difference to our solution is that while CIM-TE relies on underlying IP multicast, Mist solves multicast on the application layer and is thus suitable for use in any MANET, regardless of the capabilities of the underlying network protocol. On the other hand, CIM-TE addresses security issues explicitly, which Mist does not. Currently, Mist assumes the network is secured on lower layers (e.g., link cryptography). Thus, this work is complementary to ours in that it addresses different issues of chat.

The work in [7] and [8] presents a solution called XO, for using standard XMPP clients in server-less peer-to-peer networks, as well as a gateway for connecting the server-less network with standard XMPP networks. XO is realized as a plugin to the Generic Unicast-to-multicast proxy (GUMP), developed by the authors. The solution has been evaluated in an emulated mobile environment, with different degrees of connectivity. The results show that XO performs significantly better than standard XMPP, but still there is a message loss of 10 to 20% in medium and low connectivity scenarios. To the best of our knowledge, an implementation of XO was not available online for comparison at the time of writing.

[9] presents a server-less IM protocol for MANETs based on ring routing (Virtual Ring Routing and MADPastry are mentioned as examples) for message routing. In addition, presence dissemination is handled using distributed hash tables storing status information together with the user name. However, there is no information about implementations or evaluations of the suggested solution.

III. Design

Our chat implementation is written in Java and based on an improved version of the Mist protocol. Mist is a distributed publish/subscribe protocol for radio networks with mobility, as described in [2]. In the following we briefly describe the protocol, as well as our basic XMPP/Mist gateway. The gateway is used to bridge networks using XMPP and Mist-based chat.

A. The Mist protocol

In Mist, data is represented as «data elements». A data element can be seen as a single unit of data of any size, e.g. a file or a chat message. Compared to a network packet, the data element can be larger and have a longer lifetime, typically measured in minutes or hours. The data element is associated with a topic, which allows it to be forwarded to subscribers.

When a data element is published, the publisher broadcasts the data element along with the topic it is associated with to other nodes within radio range. Neighboring nodes may then forward the data element to nodes in other areas of the network that are subscribing to the same topic.

Each node running Mist periodically broadcasts what it subscribes to and which data elements it has seen. By keeping track of already forwarded messages, the bandwidth consumption is reduced.

The Mist protocol has two basic message types; the Subscription-message and the Advertisement-message.

The simplest message is the *Advertisement-message*, which is used to send the actual data element to other nodes. The Advertisement-message contains the data element and the topic it is associated with, as well as a few data fields describing the creator. These data fields include the unique identification number of the creator and a sequence number. If the length of the data element should exceed the maximum transmission unit (MTU) at the network layer, the data element can be split into fragments of suitable sizes. In these cases, the Advertisement-message also contains a fragment identifier.

The *Subscription-message* is broadcast within radio range to notify neighbors about subscriptions a node is interested in. It also doubles as a neighbor discovery mechanism and an acknowledgment message. The two most important fields of the Subscription-message is the list of subscriptions and the acknowledgment-field.

The list of subscriptions contains a list of topics the node subscribes to, as well as a distance counter for each topic. The distance counter allows Mist nodes to control how far in the network their subscriptions are propagated. When a node receives a new Subscription-message it goes through the list of subscriptions in the message and decrements each distance counter by 1. Topics that have a distance counter greater than 0 are then added to the node's own subscription list and included in its next Subscription broadcast. The distance counter is thus used to propagate subscription information outside the radio range of each node.

To save bandwidth, the topic identifier in the subscription list is stored as 32 bit checksums of a text string representing the topic. Namespaces may be specified by inserting «.»'s in the topic string, e.g. «no.ffi.messages». The topic is then stored as three 32 bit checksums, one for each part of the namespace. This storage method was chosen to allow for wildcard subscriptions, e.g. one can subscribe to «no.ffi.*» to receive all messages in the «no.ffi» namespace. This would not be possible if the whole namespace was stored as a single checksum.

Each node maintains a Bloom filter [10] with the identifiers of the data elements it is currently storing. The Bloom filter is included in the acknowledgment field of the Subscription-message. With the help of the Bloom filter, other nodes are able to determine with a known probability whether a node

has already received a given data element. As Bloom filters produce false positives, the mechanism may result in too few data elements being forwarded by other nodes. It will however never result in the transfer of a data element that has already been received. The false positive probability of the Bloom filter can be adjusted so that it is very unlikely that some of the data elements are never sent. Adjusting the false positive probability is not always an easy task, as the proper parameters depend on the number of new data elements that are to be exchanged. To avoid having to approximate the number of new data elements in advance, we make sure that each Subscription-message contains a new Bloom filter. Over time, the combined false positive probability of all the Bloom filters converges absolutely to 0. Given infinite time and full connectivity, all data elements are thus guaranteed to be delivered. This is an improvement over the original Mist-protocol, where the same Bloom filter was sent in each Subscription-message and the false positive probability remained constant. Further details about this mechanism can be found in [11].

As a further optimization, nodes send Bloom filters with very low false positive probability when there are no known neighbors. This ensures that when new nodes are discovered, as much information as possible will be transferred immediately. As nodes become aware of more neighbors, the false positive probability of the Bloom filters is increased. This is beneficial in dense networks, as it reduces the amount of bandwidth required by the Subscription-message. Additionally, when more nodes are sending Bloom filters in the same area it increases the probability of data elements being sent. As Mist uses opportunistic listening it does not matter if a data element is sent to a node directly or just forwarded to a neighbor as long as they both are within radio range of the sender.

B. Chat client

The chat client subscribes to two topics using Mist; a message topic («no.ffi.messages») and a control channel topic («no.ffi.control»). New chat messages are published on the message topic and distributed to all other nodes using the mechanism described in the previous section. The control channel topic is used for nick or name changes and to inform the XMPP gateway of login credentials, as is explained in the next section. Other topics could be added to support multiple chat rooms.

C. XMPP gateway

We have developed a software gateway component to bridge chat traffic between the static XMPP network and the the mobile network running Mist. The gateway is deployed on the same node as the XMPP server, and a TCP connection to it is initiated and held open. This configuration is used under the assumption that units in the field use (MANET) VHF/UHF radios with short range but relatively high bandwidth, while the reachback link is slow (e.g., an HF link). Work done by Isode show that for XMPP over slow links, server to server communication is the only viable solution [12]. Thus, internally the fielded units use Mist-based chat, which is able to handle both network partitions and continually changing network topology, and at the same time they have a reachback connection into an XMPP based chat room at, for instance, the deployed HQ.

The gateway is configured to monitor a specific XMPP multi user chat room (MUC) by logging on with usernames and passwords provided by the Mist clients. The gateway ensures that messages originated by clients in the static XMPP network are relayed into the Mist network, and vice versa. This deployment effectively merges the two chat networks, allowing all clients to communicate across heterogeneous networks. The gateway is implemented in Java, and utilizes Jive Software's Smack XMPP client API in order to integrate with the XMPP server.

In addition to listening for chat messages originated in the Mist network, the gateway also subscribes to the control channel topic, listening for commands sent from the Mist clients. Among the supported commands is a command for submitting XMPP user login information. When the gateway receives a chat message from a Mist-based client it performs an internal lookup, logs on to the XMPP server using the corresponding credentials, and posts the message on behalf of the client. If no credentials are received prior to receiving a message from a given client, the message is queued until the credentials can be obtained.

IV. Evaluation

The evaluation is performed using Linux containers (lxc) and the topology is emulated using the network simulator NS-3.

Linux containers can be seen as lightweight virtual machines. Each node runs within the same Linux kernel, but has its own network stack and process space. The network interface on each node is connected to the NS-3 driven topology. NS-3 emulates the topology and the physical network interface, i.e. the link layer and the physical layer. The setup enables us to run actual implementations of the protocols we evaluate, using the standard TCP/IP stack of the Linux kernel. NS-3 is configured to use IEEE 802.11a wireless links, running 6 mbps OFDM.

In the XMPP experiments, an XMPP server (Prosody, http://prosody.im/) is running on one of the nodes. Each client produces chat messages at a regular interval and tries to deliver them to a multi user chat room on the XMPP server. When the message has been received by the server and subsequently by all the other nodes, the message is considered to have been delivered. If the node is unable to deliver the messages immediately, it is stored in a local queue and delivered when a connection to the XMPP server is available. All communication with the XMPP server is compressed using zlib compression, as specified in XEP-0138.

In the Mist experiments, an XMPP gateway is running on one of the nodes. As with XMPP, each client produces chat messages at a regular interval and the message is considered to be delivered when it has been received by all the other nodes, including the XMPP gateway node. Mist is set to produce

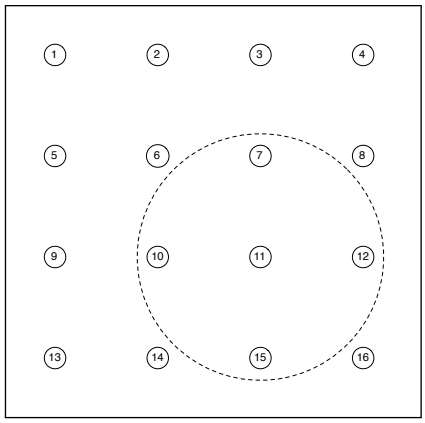

Fig. 1. Static mesh topology used in the experiment described in Section IV-A. Nodes are positioned 100 meters apart. The approximate radio range is shown for node 11.

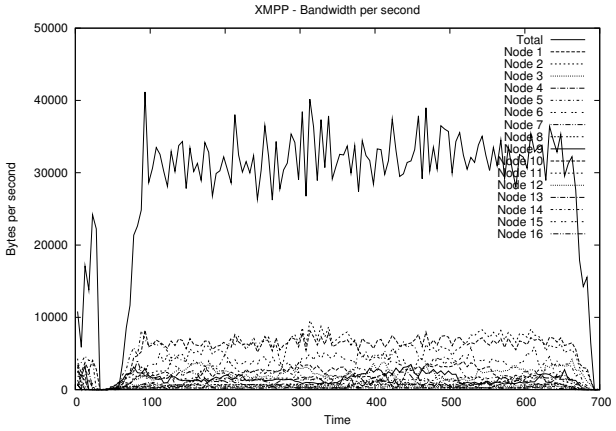

Fig. 2. Bandwidth consumed by XMPP in the static topology shown in Figure 1. Node 1 runs the XMPP server.

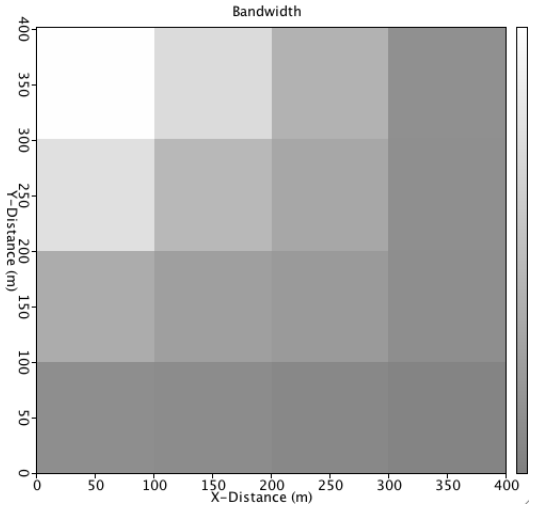

Fig. 3. Heat map showing the traffic distribution for the XMPP experiment. The XMPP server in the upper left corner consumes the most bandwidth.

periodic beacon messages at 15 second intervals and chat messages are published with a 2 minute timeout.

Each experiment is run for 660 seconds. During the first 600 seconds, chat messages of random lengths ranging from 1 to 30 characters are produced at a rate of 0.25 messages per second on each node. Each chat message is prepended a timestamp and a sequence number used for measurements. The full message length is therefore from 17 to 49 characters. The last 60 seconds of the experiment is a cool down period, allowing the protocols to attempt to deliver any remaining messages.

The XMPP experiments require a routing mechanism to enable multihop connections to the server. For this, we ran *babeld* (http://www.pps.jussieu.fr/ jch/software/babel/) on each node, which is a Linux implementation of the BABEL routing protocol. We chose BABEL over other protocols (e.g OLSR), as recent work [13] has shown it to have lower convergence time in mobile environments. Prior to starting the XMPP experiments, the routing protocol is given 120 seconds to converge. The Mist experiments are run without a routing protocol, as neither routing nor multicast is required.

Bandwidth usage is measured by capturing outgoing traffic from the network interface on each node and averaging the traffic over 5 second intervals. All traffic sent by the MAC address associated with the measured node and protocol is included. Additional traffic produced by the BABEL routing protocol is not included in the XMPP results, as this can vary depending on the chosen routing mechanism. We used the tools *tcpdump* (http://www.tcpdump.org), *tcpstat* (http://www.frenchfries.net/paul/tcpstat/) and *tcpslice* (ftp://ftp.ee.lbl.gov/tcpslice.tar.gz) to capture and analyze the results.

A. Static topology

In this experiment we test 16 nodes in a 4 x 4 mesh topology. The nodes are positioned 100 meters apart. The topology is depicted in Figure 1. The dotted circle around node 11 depicts the approximate radio range. The nodes can communicate directly with neighbors, except diagonally.

In Figure 2, the bandwidth usage is shown for XMPP. The total bandwidth used by the network averages around 32000 bytes/sec, with peaks above 40000 bytes/sec. The outgoing traffic on each node varies depending on the distance from the XMPP server. Nodes closer to the server have to relay more data on behalf of more distant nodes, thus increasing their bandwidth usage.

Figure 3 shows how the produced traffic is distributed among the nodes in one of the XMPP experiments. The upper left node is running the XMPP server and consumes the most bandwidth. As the distance form the server increases, the bandwidth consumption decreases. At node 1 (the server node) the bandwidth usage is 5700 bytes/sec on average. On node 16 (lower right), the bandwidth consumption is 214 bytes/sec.

The bandwidth results for Mist is shown in Figure 4. As we can see, the average total bandwidth is less than half of the total bandwidth required in the XMPP experiment. As Mist uses synchronization to distribute messages, all nodes forward approximately the same amount of traffic and there are no hot spots in the topology.

The total bandwidth for all nodes in the XMPP and Mist

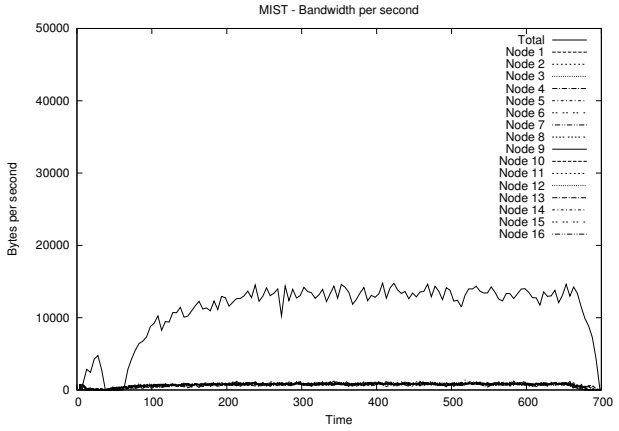

Fig. 4. Traffic produced by Mist in the static topology shown in Figure 1. Node 1 runs the XMPP gateway.

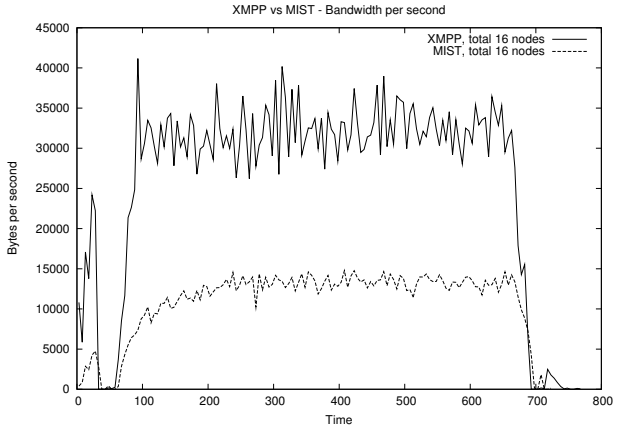

Fig. 5. Total bandwidth for all nodes shown for Mist and XMPP.

experiment are compared in Figure 5.

In the static topology both Mist and XMPP delivered all the chat messages in all of the experiments.

B. Mobility

We repeat the static experiment for different degrees of mobility. First off, we test the protocols with the nodes moving at 1 m/s, or 3.6 km/h - approximately slow walking speed. The mobility model is random walk, with no pauses. The experiment is repeated 10 times.

Table I shows the message delivery rate for both protocols. In the static experiment, both protocols delivered all the messages. As mobility increases, XMPP gradually delivers fewer messages, and at 4 to 6 m/s, XMPP delivers 86.2% of the messages. Mist delivers more than 99.5% of the messages in all experiments, but is not able to deliver messages when there are still partitions when the simulation ends. As mobility increases, partitions lasts for shorter time periods, leading

TABLE I
MESSAGES DELIVERED

Protocol	*0 m/s*	*1 m/s*	*2-4 m/s*	*4-6 m/s*
Mist	100%	99.54%	99.99%	100%
XMPP	100%	95.77%	90.50%	86.20%

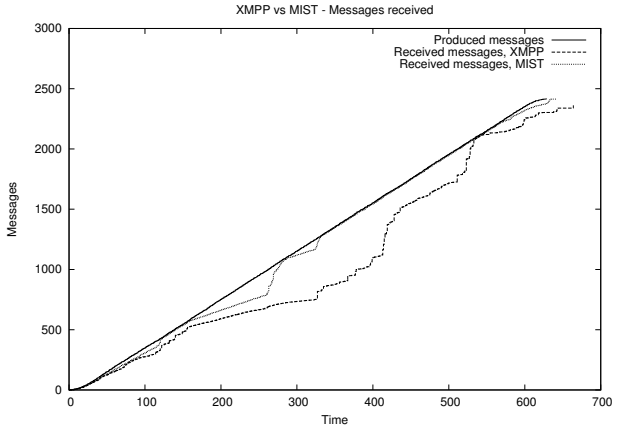

Fig. 6. Messages produced versus messages delivered in the mobile topology with nodes moving at 1 m/s. Mist is able to recover from partitioning and deliver all messages, while XMPP is unable to deliver all the messages within the simulation period.

to higher success rate. At 4 to 6 m/s, Mist achieves 100% message delivery.

In Figure 6, we can see the messages delivered over time from one of the experiments with 1 m/s movement speed. When the nodes move, partitions occur in the network. For XMPP, this reduces the nodes ability to reach the server in node 1. Although the XMPP clients are configured to store the messages if they are not able to deliver them to the server, some of the messages are too delayed for XMPP to be able to recover them during the simulation period. As this result shows, Mist recovers much faster than XMPP from network partitioning. Both protocols were given 60 seconds to deliver any remaining messages at the end of the experiment.

In Figure 7, the result from one of the experiments with nodes moving randomly from to 2 to 4 m/s (7.2 to 14.4 km/h) is shown. As we can see, XMPP struggles to deliver all messages at this speed. This is caused by the extra time it takes for the routing protocol to converge once the network has changed, as well as the stronger requirement for delivery in XMPP. As described earlier, the XMPP client must have a continuous route to the XMPP server to deliver the message and each node must have a continuous route to the server to receive it. Mist on the other hand, only requires a node to have met another node that carries the information it is interested in. The increased movement speeds helps Mist deliver messages faster, as the network is partitioned for shorter periods of time.

Figure 8 shows the bandwidth requirements for Mist when the nodes are moving from 2 to 4 m/s. The increase in bandwidth compared to Figure 4 is caused by retransmissions when nodes move out of radio range.

Table II shows the average message delivery delays for both protocols. In the static network, Mist is generally slower than XMPP, having delivery times at 1.32 seconds on average. However, as the nodes begin to move, XMPP's delivery times increases dramatically. Again, it can be seen how the higher mobility helps Mist to deliver messages faster. With 1 m/s movement the average delay of Mist is 6.44 seconds, but at 4

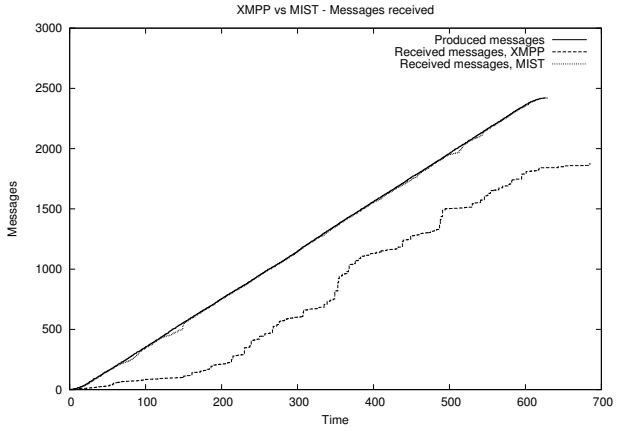

Fig. 7. Messages produced versus messages delivered in the mobile topology with nodes moving at 2 to 4 m/s. XMPP is unable to deliver all messages, while Mist delivers 100%.

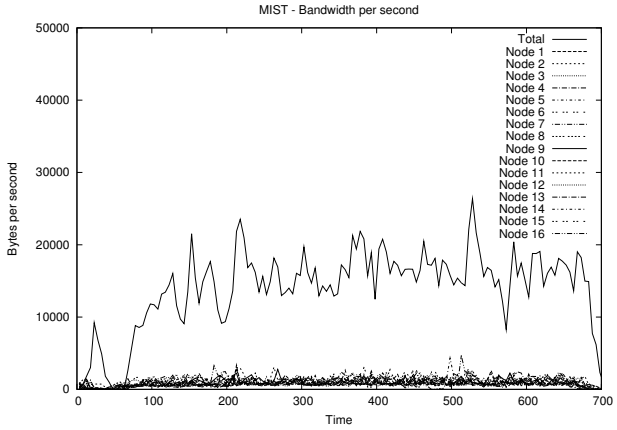

Fig. 8. Bandwidth used by the Mist network when the nodes move at 2 to 4 m/s. The increase in bandwidth compared to Figure 4 is caused by an increase in retransmissions due to nodes moving in and out of radio range.

to 6 m/s it is decreased to 2.39 seconds.

C. UAV message ferry

In this experiment we test how one fast moving node can be used to transport messages between network partitions. We deploy two groups, each having 6 nodes moving at 1 m/s in an area of 150 x 150 meters. The groups are positioned 500 meters apart, outside of each others radio range. A node representing a UAV moves between the groups. When the UAV node comes within radio range of the nodes of one of the groups, it sends any messages it has that the nodes in the group are missing. Similarly, the nodes in the group responds

TABLE II
MESSAGE DELAYS IN SECONDS

Protocol	*Speed*	*Avg*	*Std. dev*
Mist	0 m/s	1.324	0.788
XMPP	0 m/s	0.033	0.023
Mist	1 m/s	6.443	17.569
XMPP	1 m/s	29.285	60.922
Mist	2-4 m/s	2.771	5.690
XMPP	2-4 m/s	76.666	96.864
Mist	4-6 m/s	2.399	4.314
XMPP	4-6 m/s	101.008	109.905

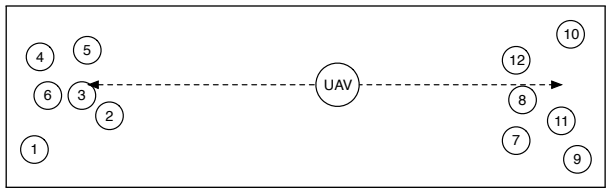

Fig. 9. Topology with two groups with nodes moving at 1 m/s and a UAV moving between them. The nodes are never able to connect directly to nodes in the other group.

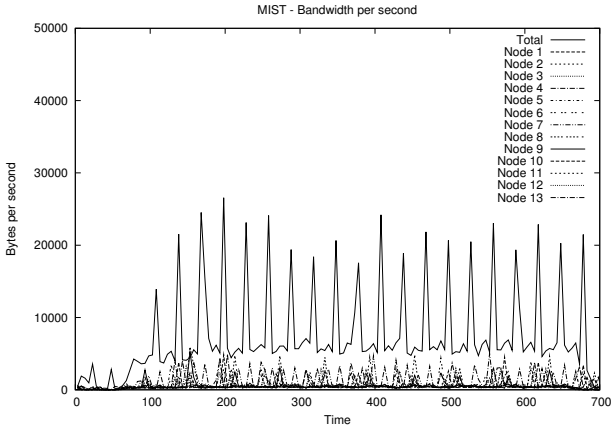

Fig. 10. Total bandwidth use in the UAV experiment. The spikes occur when the UAV reaches one of the groups and starts exchanging messages.

with messages the UAV node is missing, allowing it to carry them to the other group.

In the first experiment, it takes the UAV 30 seconds to travel the 650 meters between the centers of each group. This corresponds to a movement speed of 21.6 m/s, or approximately 78 km/h. There is at no point a continuous network connection between the groups. The UAV node is configured to send periodic Subscription-messages at 1 second intervals, instead of 15 seconds as is used by the other nodes. This is to be able to quickly discover the groups when they come within radio range. The topology is shown in Figure 9.

Figure 10 shows the total bandwidth in the network. As we can see, there are spikes in the traffic each time the UAV arrives with new messages. This is shown more clearly in Figure 11 where only the traffic sent from the UAV is shown.

Messages delivered over time is shown in Figure 12. Each time the UAV reaches one of the groups, there is a sharp increase in messages delivered. As we measure the time it takes to deliver messages to all other nodes, there is a constant delay corresponding to the time it takes for the UAV to travel between the groups. During the last 60 seconds, when no more messages are produced, Mist manages to deliver 100% of the chat messages.

To verify that the protocol works at even higher speeds, we repeat the experiment with a UAV node moving at 43.3 m/s, or 156 km/h. At this speed, the UAV is in contact with the groups for approximately 8 to 9 seconds. This is shorter than the minimum convergence time observed for several common ad-hoc routing protocols [13]. In this experiment, the

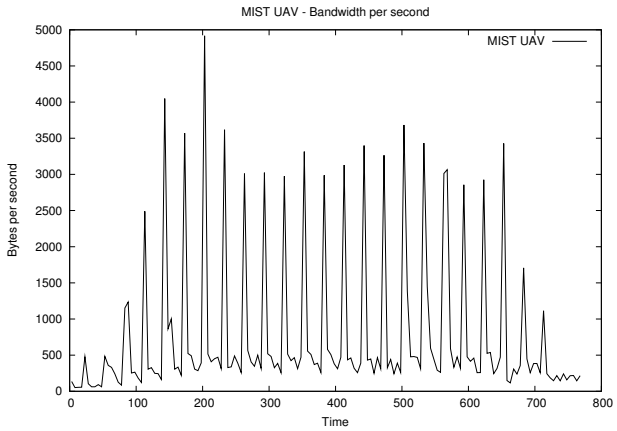

Fig. 11. Bandwidth used by the UAV. As it reaches each group there is a spike in traffic.

TABLE III
MESSAGES DELIVERED IN UAV EXPERIMENT

	21.6 m/s	*43.3 m/s*
Total	100%	99.67%
Group A	100%	100%
Group B	100%	100%

Subscription-messages are sent from the UAV every second to be able to discover the groups on the ground quickly.

Table III shows the average message delivery rates from the UAV experiments. As we can see, all messages are delivered when the UAV moves at 21.6 m/s. When the speed is doubled to 43.3 m/s, some messages are lost, but we still achieve a delivery rate of 99.67%.

V. CONCLUSION

Our chat client based on the improved Mist publish/subscribe protocol performs better in dynamic, mobile networks than XMPP in terms of bandwidth usage and message delivery rates. Although XMPP is faster in static networks, Mist is able to deliver more than 99% of the messages in mobile networks at higher speeds than XMPP. We show that our solution works by evaluating it on a simulated network topology, running full-featured software implementations of the routing protocol, the XMPP server and the Mist middleware. Finally, we demonstrate how a UAV traveling at speeds up to 156 km/h is able to deliver messages between two network partitions.

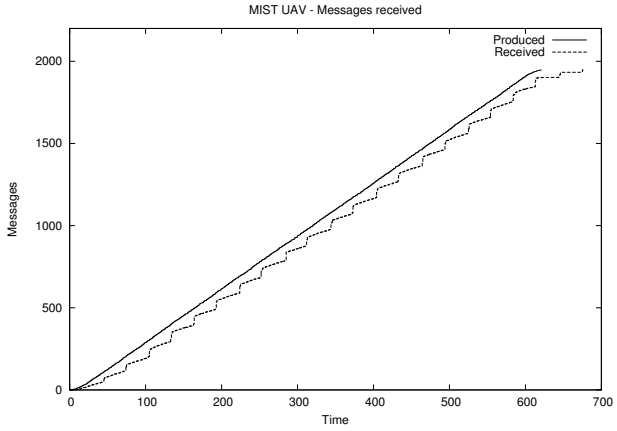

Fig. 12. Messages delivered over time in the UAV experiment. Each time the UAV reaches one of the groups, there is a sharp increase in messages delivered. The delay in message delivery corresponds to the time it takes the UAV to travel the distance between the groups.

REFERENCES

[1] 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.

[2] M. Skjegstad, F. T. Johnsen, T. Hafsøe, and K. Lund, "Robust and Efficient Service Discovery in Highly Mobile Radio Networks using the MIST Protocol," in *MILCOM 2010. The 29th IEEE Military Communications Conference*, San Jose, CA, USA, November 2010.

[3] C. Lindemann and O. Waldhorst, "Epidemic dissemination of presence information in mobile instant messaging systems," in *Kommunikation in Verteilten Systemen (KiVS)*. Springer, 2005, pp. 29–40.

[4] R. Metzger and M. C. Chuah, "Opportunistic information distribution in challenged networks," in *Proceedings of the third ACM workshop on Challenged networks - CHANTS '08*. New York, NY, USA: ACM Press, 2008, p. 97.

[5] 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.

[6] 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.

[7] R. Lass, J. Macker, D. Millar, W. Regli, and I. Taylor, "XO: XMPP overlay service for distributed chat," in *MILCOM 2010. The 29th IEEE Military Communications Conference*, San Jose, CA, USA, November 2010.

[8] R. Lass, J. Macker, D. Millar, and I. Taylor, "GUMP: adapting client/server messaging protocols into peer-to-peer serverless environments," in *Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems*. ACM, 2010, pp. 39–46.

[9] A. D. Urso, "A Serverless Instant Messaging Protocol for Mobile Ad Hoc Networks," *8th International Conference on Creating, Connecting and Collaborating through Computing*, pp. 71–75, 2010.

[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[11] M. Skjegstad and T. Maseng, "Low Complexity Set Reconciliation using Bloom Filters," in *7th ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing (FOMC)*. San Jose, CA, USA: ACM, June 2011.

[12] Isode, "M-Link and XMPP Performance Measurements over HF Radio using STANAG 5066 and IP," http://www.isode.com/whitepapers/xmpp-performance-hf-radio.html, Sept 2010.

[13] M. Abolhasan, B. Hagelstein, and J. C.-P. Wang, "Real-world performance of current proactive multi-hop mesh protocols," in *15th Asia-Pacific Conference on Communications*. Shanghai, China: IEEE, Oct. 2009, pp. 44–47.