ONEChat: Enabling Group Chat and Messaging in Opportunistic Networks

Heming Cui
Columbia University
1214 Amsterdam Avenue
New York, USA 10027
hc2428@columbia.edu

Suman Srinivasan
Columbia University
1214 Amsterdam Avenue
New York, USA 10027
sumans@cs.columbia.edu

Henning Schulzrinne
Columbia University
1214 Amsterdam Avenue
New York, USA 10027
hgs@cs.columbia.edu

ABSTRACT

Opportunistic networks, which are wireless network "islands" formed when transient and highly mobile nodes meet for a short period of time, are becoming commonplace as wireless devices become more and more popular. It is thus imperative to develop communication tools and applications that work well in opportunistic networks. In particular, group chat and instant messaging applications are particularly lacking for such opportunistic networks today.

In this paper, we present ONEChat, a group chat and instant messaging program that works in such opportunistic networks. ONEChat uses message multicasting on top of service discovery protocols in order to support group chat and reduce bandwidth consumption in opportunistic networks. ONEChat does not require any pre-configuration, a fixed network infrastructure or a client-server architecture in order to operate. In addition, it supports features such as group chat, private rooms, line-by-line or character-by-character messaging, file transfer, etc.

We also present our quantitative analysis of ONEChat, which we believe indicates that the ONEChat architecture is an efficient group collaboration platform for opportunistic networks.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*

General Terms

Design,Performance,Reliability

Keywords

opportunistic networks, BonAHA, middleware, library, group chat, instant messaging, p2p

1. INTRODUCTION

Despite the increasing availability of wireless networks, many locations still lack Internet connectivity. At the same time, most mobile devices, from laptops to smart phones, can participate in local (single-hop) wireless networks. Such transient local networks, called opportunistic networks, might be formed by students sitting in a classroom, travelers riding in a subway car, or a audience at a seminar. Opportunistic networks allow participants to communicate to each other.

But text-based chat applications can enhance collaboration and augment oral communication in such networks. Hence, a group chat or instant messaging application for opportunistic networks would be very useful.

A group chat application is a collaborative software (also referred to as groupware) which is designed to help people communicate with each other in real time. IM (Instant Messaging) is another tool which allows real-time text-based communication application among people, either with one another or in a group.

Unfortunately, most existing group-chat and instant messaging applications today work in a client-server manner. Most of the popular solutions also rely on the use of proprietary protocols and servers. Hence they cannot be used in opportunistic networks. Recently, a few IM applications have been proposed for opportunistic networks, but to the best of our knowledge, they can not support group chat efficiently. More details about related work will be introduced in Section II.

We have implemented an efficient group chat application for opportunistic networks called ONEChat (Opportunistic NEtwork Chat). ONEChat was built using the Java [26] programming language. Even though we mostly focus on single-hop local opportunistic networks, ONEChat works in any network that supports multicast communication.

ONEChat does not need to be manually configured, nor is a fixed infrastructure required for it to work properly. ONEChat works very well even in the presence of transient nodes that enter and leave the network quickly. The implementation of ONEChat is greatly simplified by building our program on top of the BonAHA framework [2] and the real-time text protocol [15], both of which are detailed in Section III.

In addition to supporting simple group chat, ONEChat also has several additional and useful features that make it a fully featured application. For instance, it allows for users to create their own private groups, where messages are encrypted and which only users with the knowledge of a shared key can join. ONEChat also supports exchange of small files as well as buddy icon updates from other users.

ONEChat leverages the properties of real-time text to allow transmission of messages in line-by-line mode, with the user indicating the completing a message through a signal such as pressing the Enter key, or in character-by-character mode, where a character is transmitted as soon as it is entered.

The rest of the paper is organized as follows. Section II introduces related work. Section III covers the implementation details of ONEChat. In Section IV, we analyze the performance of ONEChat and other similar applications. Our conclusion is summarized in Section V.

2. RELATED WORK

There are a plethora of IM applications today. ICQ [4], GTalk [5], MSN [6], AOL [7], Skype [8], and MultiChat [9] are among the most popular and widely known ones. However, none of these popular IM programs or their clones can be used in opportunistic networks because they require connection to the Internet, in particular, connection to the servers that are run by the companies that host these IM programs.

Recently, some chat applications for opportunistic networks have appeared. iChat [10], Socialized.NET [11], Opportunistic Chat [12], and DTN (Disruption Tolerant Networks) Jabber Proxy [13] are representative of these class of applications. However, they can not support group chat for opportunistic networks due to the drawbacks listed below.

Both iChat and Socialized.NET work in a P2P manner. However, neither of them support message multicast, so a message has to be sent multiple times in order to reach all users within a group. This would consume a lot of bandwidth.

Opportunistic Chat [12] introduces a Bluetooth-TCP/IP hybrid approach: if two users next to each other want to talk directly, they can set up a Bluetooth link; if they are too far away to be able to use Bluetooth, or if they want to chat within a group, then they should setup a client-server communication link via a TCP/IP network. Therefore, Opportunistic Chat cannot handle group chat for opportunistic networks either.

DTN Jabber Proxy can work in opportunistic networks, but it requires a complicated server configuration and the server proxy needs to be installed and available to the network.

In order to fully support group chat for opportunistic networks, we have implemented a pure-Java, lightweight and configuration-free application called ONEChat. As soon as ONEChat applications start up, they can discover each other without querying any central servers, and they can work without requiring any pre-configuration.

Once a ONEChat application enters or leaves the network, all the other nodes will be notified automatically. For each message, there is only one multicast transmission to all the other group members to save bandwidth. The following section will introduce the implementation details of ONEChat.

3. IMPLEMENTATION OF ONECHAT

In this section, we will explain how ONEChat works with the BonAHA library and the real-time text protocol. We will also explain how the different types of groups and messages are defined in an ONEChat application. We will also detail how some of its essential features of ONEChat, such as creating a group, joining and leaving a group, notification of network entry, and notification of leaving a network, work.

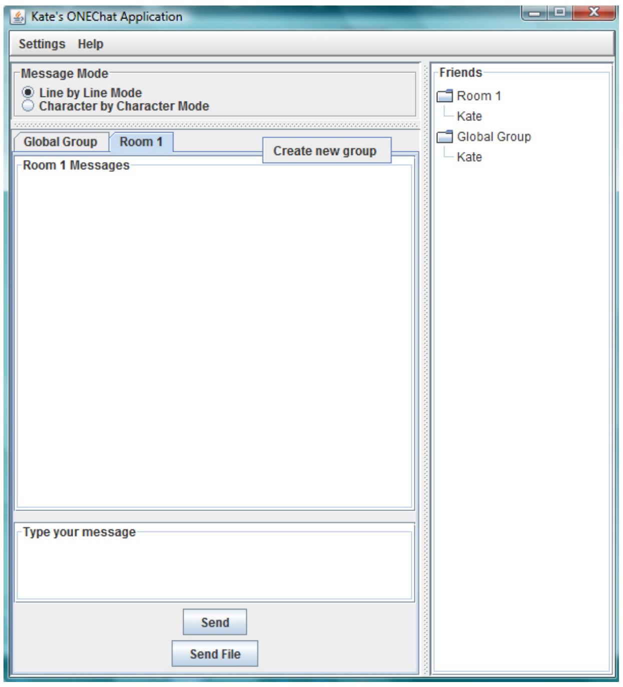


Figure 1: A screenshot of a ONEChat session in operation.

3.1 Introduction to ONEChat

BonAHA [2] is a framework for opportunistic networks based on the multicast DNS and Zero Configuration [3] networking suite of service discovery protocols. The BonAHA library allows for easy development of applications that work in link-local, opportunistic networks. BonAHA can work in local (single-hop) opportunistic networks as well as in regular wired or wireless connections.

For each user, we define his or her *network* as all users within his or her transmission range.

Figure 1 shows a screenshot of ONEChat's user interface. The left part of the user interface contains a list of instant messages between users, while the right side of the UI contains a list of the user's "friends" who are in the same network.

For example, in this screenshot, the user's name is *Kate*, and there are already two groups, *Global Group* and *Room 1*, in Kate's ONEChat application. (The Global Group is always present in the ONEChat applications, while other local rooms can be created as necessary.)

Correspondingly, there are two tabs on the left part of the UI with these two group names, and there are two tree components on the right side which display the groups. The chat messages in a group are shown in the main window under a tab, while a tree component on the right side of the UI lists the users in that group.

3.2 User and message discovery using BonAHA

There are two kinds of messages in ONEChat: *system messages* and *user messages*.

We use BonAHA to send and receive system messages in order to perform the "behind the scenes" work to allow ONEChat applications and users to signal each other, as well as entry and exit in the network. We use the real-time text protocol to send and receive user messages.

In addition to chat, ONEChat also supports file transfer and the buddy icon update features, which are also implemented as user messages. These features will be introduced later in this section.

The message publishing and signalling mechanism for ONEChat is built on top of BonAHA, which uses mDNS (multicast DNS) service discovery protocol [3].

ONEChat is meant to work in opportunistic networks which are highly transient. So it is necessary to keep track of the state of users in the network, such as users entering and leaving the network. ONEChat recognizes these events as system events and uses BonAHA to publish them as system messages.

ONEChat uses two functions from the BonAHA framework to be notified when users enter and leave the network - $serviceUpdated(BNode\ n)$ and $serviceExited(BNode\ n)$.

ONEChat uses the $set(String\ key,\ String\ text)$ function call in the BonAHA framework to publish a system message. This sets the global properties of the user's key, and all the other users within its transmission range receive this message.

Table 1 summarizes how ONEChat uses the BonAHA functions to handle the low-level network events in the opportunistic network.

Table 1: Usage of the four BonAHA functions in ONEChat

Function	Usage
serviceUpdated	Triggered when a new ONEChat user enters the network.
serviceExited	Triggered when a ONEChat user leaves the network.
set	Called when ONEChat publishes a system message.
get	Called when ONEChat retrieves the information published on the network.

3.3 Messaging using RTP and RTT

The user message publishing mechanism in ONEChat is built on top of the real-time text (RTT) protocol, and implemented using the T140 library [24].

The real-time text protocol uses RTP (Real-time Transport Protocol) and defines an RTP payload type [15] for text conversation. In the real-time text protocol, as soon as a character is typed, it is sent and displayed immediately to the recipient. This allows text to be used in the same conversational mode as voice and video.

In ONEChat, we use the real-time text protocol to provide two transmission modes for user messages: line-by-line mode and character-by-character mode. In line-by-line mode, we buffer a group of characters that a user has typed and then transmit them; in character-by-character mode, text is sent and received character by character in a real-time manner.

As shown in Figure 1, users can choose a message mode in the *Message Mode* field. Once they choose the character-by-character mode, the *Send* button is disabled since they do not need it (they just need to type the message and the typed characters will automatically be sent). After the user switches back to the line-by-line mode, this button is enabled.

3.4 Private and secure messaging using ONEChat Groups

We provide a primitive security feature for ONEChat. There are two kinds of groups in ONEChat: the *Global Group* and *protected groups*. All public messages published within one's network are displayed in his or her *Global Group*, which is always present in any ONEChat session and available to all users currently on that opportunistic network.

By default, each ONEChat user always stays in the *Global Group*. All the other groups created by users are protected groups. A protected group must be created with a password, and others who want to join this group must know this password. We assume that this password is distributed using some out-of-band method. ONEChat does not send any password to the network.

All messages published within a protected group will be encrypted with its group password using AES in Counter Mode.

Currently, we do not consider malicious users bent on disrupting service. In multicast scenarios where keys are shared between members, it is easy to authenticate the source and prove that a member of the group has sent a message, but difficult to prevent one member from impersonating another.

This problem is called DOA (Data Origin Authentication). There are already some promising proposals in this area [16] [17] [18] [19].

We have not implemented DOA for this current version of our application, but we will quickly summarize its features. DOA can be done using signatures. There are two kinds of approaches. The first approach involves signing each RTP packet [16] [17]. This approach provides good source authentication but suffers from high computation overhead in signing and verifying the signature for each packet. The second approach involves amortizing a single signature over multiple packets or sessions [18] [19]. This reduces the overhead but it is not satisfactory when transmission is lossy. However, in some scenarios like small group chat through wireless links, the computation overhead is not a bottleneck, and the first approach is more acceptable.

3.5 Messages in ONEChat

As mentioned in Section III.A, there are two kinds of messages in ONEChat, *system messages* and *user messages*.

The first kind is *system message*, which is sent and received by the *set* and *get* functions provided by the BonAHA framework. Table 2 illustrates the usage of all the five kinds of ONEChat system messages.

The other kind is *user message*, which is published by users. In addition, the file transfer and buddy icon update mechanisms also work on top of user message transmissions.

The user message transmission mechanism is based on the real-time text protocol, and this protocol works on top of UDP multicast.

Due to the use of UDP multicast, we have one current limitation in our file transfer implementation. Sending a large file over UDP multicast causes fragmentation because

Table 2: Usage of the five ONEChat system messages

Type	Usage
sys_create	Notify others that a new group was created.
sys_join	Notify group members that I joined this group.
sys_reject	Notify someone that the password he typed was wrong.
sys_enter	Notify others that I entered the network.
sys_exit	Notify others that I left the network.

of the 1,500 byte MTU limit on Ethernet packets. In order to alleviate the fragmentation problem in our current implementation, we only allow files that are of a certain size to be sent over ONEChat, which we enforce by using a file filter. We find that file transfer and buddy icon updates (which depend on the file transfer mechanism) work without problems in our current implementation.

Retransmission of lost packets may result in re-sending a complete file several times. It is necessary to look into RTP-compatible retransmission mechanisms to mitigate packet loss. RFC 2354 [20] proposes several techniques, such as FEC (Forward Error Correction), retransmission, and interleaving, which may be considered to increase packet loss resiliency. In addition, RFC 4588 [21] proposes a comprehensive RTP retransmission payload format for both unicast and small multicast groups. This format is defined in the AVPF profile (RFC 4585 [22]), and is used by receivers to send retransmission requests. There are already some open source multicast file transfer program like UFTP [23], but they are not based on RTP. Thus, the RTP-compatible retransmission mechanism mentioned in RFC 4588 may be more appropriate to mitigate the UDP fragmentation problem.

Both system and user messages have three fields: *message type*, *destination group* and *message content*. The value of the *message type* field can be any one of the values in the first column in Table 2, or *user_lbl*, (a user message under line-by-line mode), or *user_cbc*, (a user message under character-by-character mode).

Since ONEChat is a local chat application, there is only one multicast RTP group for all user messages among different chat groups, so we need a *destination group* field to indicate which group a message belongs to. A message would only be displayed in a group whose name is the same as that of the *destination group*. The *message content* field stores the real content of a message.

3.6 Creating, Joining and Leaving a Group

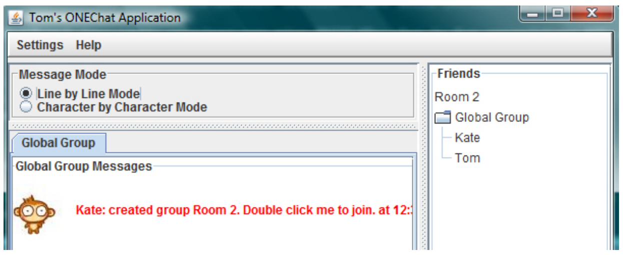
When a user creates a new group, he or she is required to enter a password (the group key), and others who want to join the group need to get this password information from the group creator.

Take an interaction between two users, Kate and Tom, as an example. As shown in Figure 1, suppose Kate creates a group named *Room 2*. This event will be encapsulated into a *sys_create* (Table 2) message and broadcast automatically to all users within Kate's network. Tom receives this message and a message is displayed on his ONEChat UI, as shown in Figure 2.

If Tom is interested in joining *Room 2*, he double-clicks this and attempts to join the group. This results in a popup window which asks Tom to enter a password. After Tom enters a password, the new group *Room 2* will appear in Tom's ONEChat window, and a *sys_join* (Table 2) message will be sent to all users in Group *Room* 2 notifying them that Tom has joined this group.

After Kate receives this message, she uses the group key (her password) to decrypt the message and tries to get the tag. If this succeeds, then Kate knows that Tom has typed the correct password, otherwise a *sys_reject* message will be sent to Tom to expel him from the session. Tom's ONEChat window will then close the *Room* 2 tab.

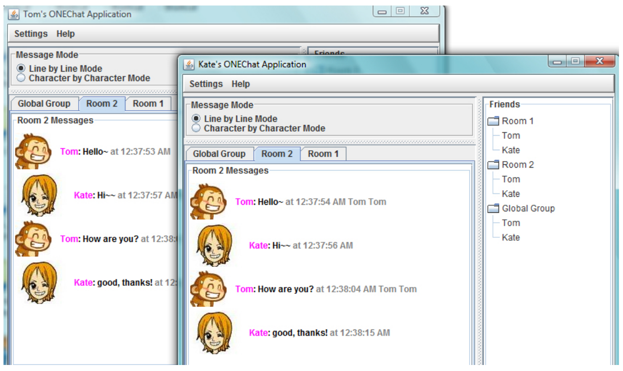
If Tom is able to successfully authenticate and join *Room* 2, he can talk to Kate and others who are members of this group. Figure 3 shows a screenshot of a established connection and group chat.



Kate: created group Room 2. Double click me to join. at 12:

Figure 2: Tom is informed that Kate created group *Room* 2.

In Figure 3, Tom uses an icon (a monkey) and Kate another icon (a girl). The icons in the two users' applications are synchronized with the buddy icon update mechanism: once a user updates his icon, this icon image will be transmitted through multicast to all users within the network, and the other users can view this updated buddy icon. The buddy icon update mechanism is encapsulated as a public user message.



Tom: Hello~ at 12:37:53 AM
Kate: Hi~~ at 12:37:57 AM
Tom: How are you? at 12:38:
Kate: good, thanks! at 12:

Tom: Hello~ at 12:37:54 AM Tom Tom
Kate: Hi~~ at 12:37:56 AM
Tom: How are you? at 12:38:04 AM Tom Tom
Kate: good, thanks! at 12:38:15 AM

Figure 3: Tom and Kate chat within group *Room* 2.

A user can leave a group at any time. Suppose Tom wants to leave *Room* 2, he closes the tab named *Room* 2 in his application. A *sys_leave* (Table 2) message will be automatically multicast to all users within this group.

This message will appear on the message window in Kate's *Room* 2, and Kate knows that Tom has left. Figure 4 depicts the interactive sequence of the creating, joining and leaving group procedures between two users.

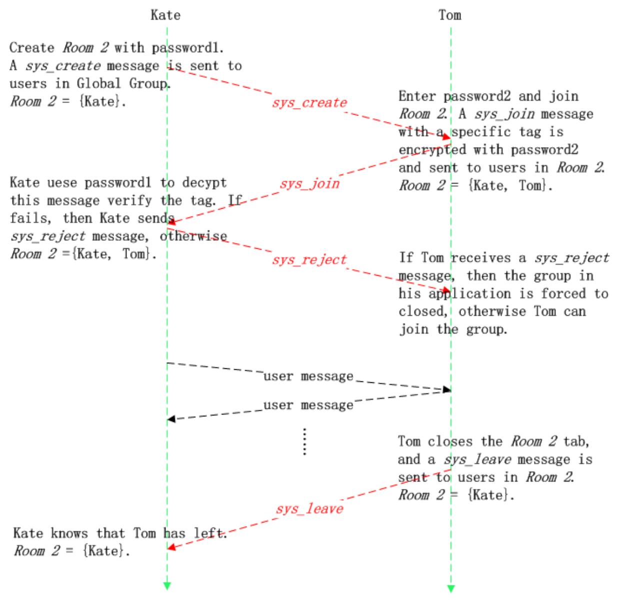

Kate
Tom
Create *Room 2* with password1. A *sys_create* message is sent to users in Global Group. *Room 2* = {Kate}.
sys_create
Enter password2 and join *Room 2*. A *sys_join* message with a specific tag is encrypted with password2 and sent to users in *Room 2*. *Room 2* = {Kate, Tom}.
sys_join
Kate uese password1 to decypt this message verify the tag. If fails, then Kate sends *sys_reject* message, otherwise *Room 2* ={Kate, Tom}.
sys_reject
If Tom receives a *sys_reject* message, then the group in his application is forced to closed, otherwise Tom can join the group.
user message
user message
Tom closes the *Room 2* tab, and a *sys_leave* message is sent to users in *Room 2*. *Room 2* = {Kate}.
sys_leave
Kate knows that Tom has left. *Room 2* = {Kate}.

Figure 4: The sequence of the creating, joining and leaving a group.

3.7 Enter-Network Notification

Once a user enters the network, all the other users in his or her network will be notified by BonAHA's *serviceUpdated* mechanism. Figure 5 shows how a enter-network notification works.

Take Tom and Kate as an example. Once Tom launches a ONEChat application, a *sys_enter* message (Table 2) is generated automatically and published by the *set* (Table 1) function provided by BonAHA. All the other users in Tom's network receive this message immediately. Kate's ONEChat makes sure that this is a *sys_enter* message and displays it on the message list of her Global Group, and Tom is listed in Kate's friend list.

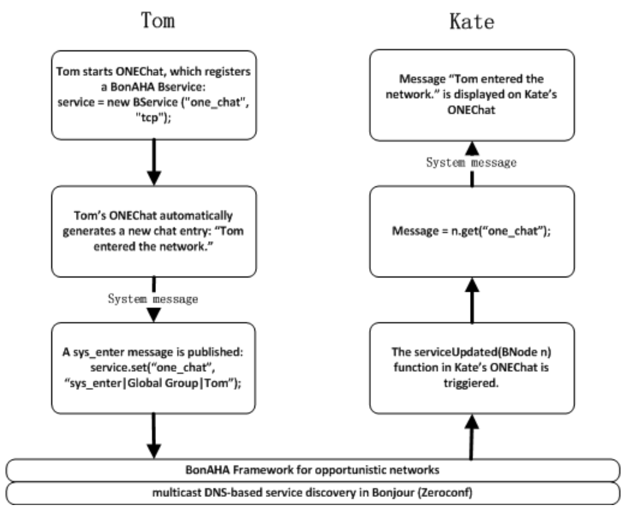

Tom
Kate
Tom starts ONEChat, which registers a BonAHA Bservice: service = new BService ("one_chat", "tcp");
Tom's ONEChat automatically generates a new chat entry: "Tom entered the network."
System message
A sys_enter message is published: service.set("one_chat", "sys_enter|Global Group|Tom");
Message "Tom entered the network." is displayed on Kate's ONEChat
System message
Message = n.get("one_chat");
The serviceUpdated(BNode n) function in Kate's ONEChat is triggered.
BonAHA Framework for opportunistic networks
multicast DNS-based service discovery in Bonjour (Zeroconf)

Figure 5: The enter-network notification procedure.

3.8 Leave-Network Notification

ONEChat can capture a leave-network event as well as an enter-network event. We have developed the procedure of leave-network notification by implementing the *serviceExited* (Table 1) interface function provided by BonAHA. Figure 6 depicts the implementation of this notification procedure.

Take Tom and Kate as an example. Suppose Tom closes his ONEChat application. This action triggers a *sys_exit* event down to his BonAHA framework, and this event is multicast to all the others in Tom's network. At Kate's end, a system message with the message *Tom left the network* is displayed on the message list of her Global Group and the name *Tom* is deleted from her friend list.

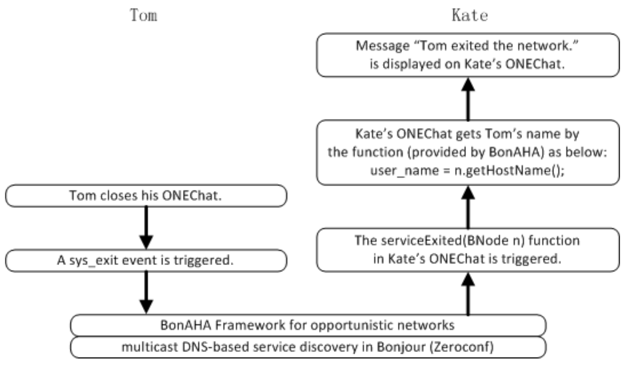

Tom
Kate
Message "Tom exited the network." is displayed on Kate's ONEChat.
Kate's ONEChat gets Tom's name by the function (provided by BonAHA) as below: user_name = n.getHostName();
Tom closes his ONEChat.
A sys_exit event is triggered.
The serviceExited(BNode n) function in Kate's ONEChat is triggered.
BonAHA Framework for opportunistic networks
multicast DNS-based service discovery in Bonjour (Zeroconf)

Figure 6: The leave-network notification procedure.

4. PERFORMANCE EVALUATION

We compare the performance evaluation of ONEChat's messaging system in opportunistic networks by comparing it to peer-to-peer chat and instant messaging clients, since we were not able to find similar IM clients that operate in a manner similar to ONEChat. We also believe that this evaluation validates our design of the ONEChat messaging system as efficient in the opportunistic network scenario.

The performance metric is the total amount of bytes sent to the network (bandwidth consumption) in the below scenario.

Assume that we have a group G_0 with m users: $\{N_1, ..., N_m\}$. Each user N_i is within the single-hop communication range of all other users, and he or she is going to send k_i messages to all the others within this group. Each message has a size of L bytes. Let $B_{ONEChat}$ denote the total amount of bytes sent by ONEChat and B_{P2P} by P2P.

In P2P applications, each message has to be sent to everyone else within the group once (message retransmission is ignored), so the total amount of bytes sent to the network is:

$$B_{P2P} = \sum_{i=1}^{m} L \times k_i \times (m-1)$$

In BonAHA, a multicast message will be sent a few times using the redundant transmission mechanism defined by Zeroconf [3].

Using Wireshark [14], we found that each system message is transmitted at most three times, and each user message is transmitted once. Therefore, the *average transmission times* (in short, *avt*), which is defined as the total number of transmitted messages divided by the total number of distinct transmitted messages. We will consider the worst case scenario for the *avt* value, which is three.

Since ONEChat applications perform message multicasting for group chat, the total amount of bytes sent in the network is:

$B_{ONEChat} = \sum_{i=1}^{m} L \times avt \times k_i$

From the two equations above, we can see that when $(m-1) > avt$, the amount of bytes sent to the network in P2P applications is more than that in ONEChat. Given $avt \leq 3$, if there are more than four users in a group, ONEChat consumes less bandwidth than P2P.

Hence, we can confirm that ONEChat's operation for opportunistic networks is more efficient than regular P2P clients which use unicast for messaging.

5. FUTURE WORK

Our future work for this project includes improving its security against malicious users, and solving the packet fragmentation problem in large file transfers.

6. CONCLUSION

This paper describes a group chat application, ONEChat, for opportunistic networks. There are two main contributions. First, ONEChat eliminates configuration and the necessity of a fixed network infrastructure, making it easily deployable in opportunistic networks. Second, it uses multicast techniques to reduce bandwidth consumption in group chat scenarios. Quantitative performance analysis shows that ONEChat outperforms P2P-based applications like iChat and Socialized.NET in bandwidth consumption as long as there are more than four members in a group.

ONEChat also has several interesting and useful features, such as private groups which are secure and require a shared key to join, as well as line-by-line and character-by-character modes of communication using Real-Time Text (RTT), that make it quite useful and usable as a full-featured application in real opportunistic networks today.

Our implementation of ONEChat (including additional documentation and details of data transfer protocols and RTP) is available for download at [25].

7. ACKNOWLEDGMENT

This work was supported by NSF Grant No. 04-54288 and No. 04-12025.

8. REFERENCES

[1] Suman Srinivasan, Arezu Moghadam, Henning Schulzrinne, *BonAHA: Service Discovery Framework for Mobile Ad-Hoc Applications.* IEEE Consumer Communications & Networking Conference 2009 (CCNC'09), Las Vegas, USA, January 2009.

[2] Suman Srinivasan, Arezu Moghadam, Se Gi Hong, Henning G Schulzrinne, *7DS - Node Cooperation and Information Exchange in Mostly Disconnected Networks.* IEEE International Conference on Communications (ICC), Glasgow, Scotland, Jun 2007.

[3] Zero-Configuration Networking protocols, http://www.zeroconf.org/

[4] ICQ, http://www.icq.com/

[5] Google Talk, http://www.google.com/talk/

[6] MSN Messenger, http://messenger.msn.com/

[7] AOL Instant Messenger, http://www.AIM.com/

[8] Skype, http://www.Skype.com/

[9] J Schull, M Axelrod, L Quinsland, *Multichat: Persistent, Text-As-You-Type Messaging in a Web Browser for Fluid Multi-Person Interaction and Collaboration.* Proceedings of the 39th Annual Hawaii International Conference on System Sciences, Hawaii, USA, 2006.

[10] Apple iChat, http://www.apple.com/macosx/features/ichat.html

[11] Socialized.NET, http://www.socialized.net/

[12] Salvatore Sorce, Francesco Cinquegrani, Salvatore Anzalone, Dario Caccĺña, Antonio Gentile, *A Dynamic System for Personal Communications: the Opportunistic Chat.* International Conference on Intelligent Pervasive Computing, Jeju City, South Korea, 2007.

[13] Ryan Metzger, Mooi Choo Chuah, *Opportunistic information distribution in challenged networks.* Proceedings of the Third ACM Workshop on Challenged Networks, San Francisco, California, USA, 2008.

[14] Wireshark, http://www.wireshark.org/

[15] "RTP Payload for Text Conversation." [Online]. Available: http://tools.ietf.org/html/rfc4103

[16] Perrig, A., Canetti, R., Tygar, D. and D. Song, *Efficient Authentication and Signing of Multicast Streams over Lossy Channels.* in Proc. of IEEE Security and Privacy Symposium S&P 2000, pp. 56-73, 2000.

[17] Istemi Ekin Akkus, Oznur Ozkasap, and M. Reha Civanlar, *Secure Transmission of Video on an End System Multicast Using Public Key Cryptography.* Lecture Notes in Computer Science, pp. 603-610, Volume 4105, 2006.

[18] Namhi Kang, and Christoph Ruland, *MDS: Multiplexed Digital Signature for Real-Time Streaming over Multi-sessions.* Lecture Notes in Computer Science, pp. 824-834, Volume 3391, 2005.

[19] C. K. Wong and S. S. Lam, *Digital signatures for flows and multicasts.* In Proc. IEEE ICNP'98, 1998.

[20] "Options for Repair of Streaming Media." [Online]. Available: http://tools.ietf.org/html/rfc2354

[21] "RTP Retransmission Payload Format." [Online]. Available: http://tools.ietf.org/html/rfc4588

[22] "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)." [Online]. Available: http://tools.ietf.org/html/rfc4585

[23] http://www.tcnj.edu/~bush/uftp.html

[24] T140 library, http://sourceforge.net/projects/rtp-text-t140/

[25] ONEChat home page, http://bonaha.sourceforge.net/web/projects/one_chat/

[26] The Java programming language. http://java.sun.com/