Review and Comparison of Instant Messaging Protocols

Computer Science
Bachelor Thesis

Tim van Lokven
0438006

January 23, 2011

Abstract

In this thesis, I will review and compare the two most popular Instant Messaging protocols in the West. To this end, I will first determine which two protocols are the most popular, and then formulate criteria to assess them. Each protocol will then be examined to uncover the fundamentals that will help with assessing each protocol in accordance with the criteria. I will then use the criteria to grade each protocol, which will finally allow me to compare the protocols with one another to determine which protocol obtains a higher score regarding the criteria, and then point out some possible improvements.

Contents

1 Introduction

In the past decade and a half, Instant Messaging programs have been one of the most commonly used methods of communicating remotely by young people. They have grown up with programs like MSN Messenger, Yahoo Instant Messenger and AOL Instant Messenger, and use them vigorously to stay in contact with people who they don't see often, but also just to chat with people they see every day. It will come as no surprise that a good number of companies have created Instant Messaging clients to try and generate some profit by playing into this very clear trend.

With a large amount of Instant Messaging clients available, that all use different underlying protocols, it raises the question of how these protocols differ. How do they tackle certain problems that arise, and how does the quality of the protocols compare?

In this thesis, I will determine what the two most popular Instant Messaging protocols are, and compare them with the help of a number of criteria that I will describe beforehand. These criteria will all concern the user comfort in some way, since I believe that a good protocol is one that offers everything a user needs, offers a sufficient amount of security and is not too taxing for the user's connection.

Because the Instant Messaging protocols are far too large to review completely, I will focus on the following actions that every protocol should support:

- Authenticating.
- Retrieving the user's contact list.
- Starting a conversation with a friend.
- Sending and receiving messages.

After determining which two protocols I will review, I will give a detailed description of their inner workings, and then assess them in accordance with the criteria. The protocols will then be compared to one another, giving an overview of which protocol functions better in which areas.

2 Popularity of Instant Messaging protocols

In order to find out which instant messaging protocols are the most popular, first it needs to be made clear what a "popular client" is. It could be the highest rated protocol, or the most frequently used one. I've decided to only look at the two most frequently used protocols in the Western world. The reason for this, is that the cultural and lingual differences between a country like China and the West are so large, that it is very hard to say what the popularity of a protocol can be attributed to. Furthermore, it will be much harder to find specifications of non-Western protocols, due to them often not being available in English.

There aren't very reliable statistics available of instant messaging protocol usage, but there is quite a bit known about the usages of instant messaging clients. For that reason, I will look at the official statistics published by the developers of these clients. In the case that these statistics are not made public, I will use the most reliable and recent estimates available by third parties.

There is one issue with this approach. A lot of clients implement several instant messaging protocols. The user counts for these clients won't give you accurate information about which protocols are actually being used.

For instance, the Microsoft Notification Protocol (MSNP) is implemented by several instant messaging clients. Some of the popular ones are Windows Live Messenger, eBuddy and Pidgin. Windows Live Messenger is Microsoft's official instant messaging client, and is the intended client for MSNP. This is the only protocol the client supports. Pidgin and eBuddy support several other instant messaging protocols as well as MSNP. It is therefore impossible to tell if the users of these programs are actually using the Microsoft Notification Protocol, or if they are only using the other protocols.

For that reason, I've decided to disregard instant messaging clients that implement multiple protocols, and only focus on the official clients that the protocols were intended to be used by. An exception to this is XMPP, because it is an open source protocol and doesn't have an official client. The most well known client that uses this protocol is Google Talk.

2.1 Usage Statistics

Table 2.1 contains a number of commonly used Instant Messaging protocols, and gives a approximate number of active users for each. Furthermore, it lists the owners of each protocol, and the Instant Messaging clients that make use of it.

Protocol	**Client**	**Owner**	**Active Users**	**Statistics Date**
MSNP	Windows Live Messenger	Microsoft Corporation	330 million	June 2009
OSCAR	AOL Instant Messenger	America Online	53 million	March 2006
	ICQ		40-50 million	February 2010
YMSG	Yahoo! Messenger	Yahoo!	22 million	September 2006
XMPP	Google Talk (not official)	XMPP Standards Foundation	40-50 million	January 2007
QQ	Tencent QQ	Tencent Holdings	448 million	August 2009

Table 2.1: Active users of widely used Instant Messaging protocols.

2.1.1 Results

It appears that the QQ Protocol has the most active worldwide users. However, the vast majority of these users reside in China. This protocol is hardly ever used in the Western world, however there is very little information on why this is the case. Because I have decided to limit myself to popular protocols in the West, I will not look further into this protocol.

The Microsoft Notification Protocol has 330 million users, and is by far the most frequently used instant messaging protocol in the Western world. Because AOL Instant Messenger and ICQ both use the OSCAR protocol, it comes to about 100 million active users.

According to these results, the following protocols are the most popular in the West:

1) MSNP
2) OSCAR
3) XMPP

Since MSNP and OSCAR have the most active users, I will be reviewing and comparing these Instant Messaging protocols in this thesis.

2.2 MSNP

The Microsoft Notification Protocol, also known as the Mobile Status Notification Protocol, is Microsoft's instant messaging protocol. It's developed to be used by their official clients, the most important being Windows Live Messenger (WLM). The protocol is currently up to version 18 (MSNP18).

The protocol is not open source. However, due to its popularity, a lot of reverse engineering has been done, and a lot of the protocol is public knowledge at this point. The protocol is implemented by numerous third party clients.

Windows Live Messenger was released in 1999 under the name MSN Messenger, and its popularity has since then been ever growing. Especially among the youth, WLM is one of the primary means of communication. It comes pre-installed with Microsoft Windows, which could be one of the reasons of its success.

2.3 OSCAR

OSCAR, or the Open System for CommunicAtion in Real-time, is an instant messaging protocol owned by America Online (AOL). Currently, OSCAR is being used by AOL's two main instant messaging clients: AOL Instant Messaging (AIM) and ICQ.

Despite it being called "open", the protocol is not open source. Just like MSNP, a great part of the protocol has been reverse engineered. As AOL primarily focuses on the American market, AIM is not very popular anywhere else. It has been the most frequently used client in the United States, but in Europe, it has never been able to come close to Windows Live Messenger.

3 Protocol Criteria

To objectively assess a protocol, it is necessary to construct a set of criteria that help determine how well it scores in a number of fields. These will later be used to grade protocols individually, and also to compare them among each other.

For each of the criteria, I will grade a protocol with a number ranging from 1 to 5. The implied meanings of these numbers are the following:

1) Horrible.
2) Bad.
3) Decent.
4) Good.
5) Excellent.

Because I do not wish to value all the criteria equally, they will also be assigned a weight. This weight will range from 1 to 3, and can be interpreted the following way:

1) Lesser importance.
2) Normal importance.
3) Higher importance.

3.1 General

- ❖ The protocol should be open source, and third parties should not be discouraged from creating clients or servers for it.
  Weight: 3.

- ❖ The protocol should be comprehensible, with meaningful command names and a clear structure.
  Weight: 1.

3.2 Security

The following criteria concern the authentication process of signing in to your instant messaging account.

- ❖ The password restrictions must be strict enough to make the password not vulnerable to easy attacks, like the dictionary attack.
  Weight: 3.

- ❖ The password must be transferred to the instant messaging server encrypted.
  Weight: 3.

- ❖ A message sent to another user should be encrypted.
  Weight: 2.

- ❖ The encryption used by the protocol must be sufficiently strong, making it infeasible to crack.
  Weight: 3.

3.3 Efficiency

- ❖ There must not have a great data overhead, messages sent should be reasonably short.
  Weight: 2.

- ❖ There should not be a great number of connections made by the protocol.
  Weight: 1.

- ❖ The protocol should not require a large number of messages for a simple action.
  Weight: 2.

3.4 User friendliness

- ❖ There should be a large number of contacts allowed.
  Weight: 2

- ❖ The users should be able to pick different font styles and colors.
  Weight: 2.

- ❖ Users should be able to choose and change their own screen name.
  Weight: 3.

4 MSN Protocol

In this chapter, I will be examining the MSN Protocol. I will delve into the core aspects of the protocol that are important to review the properties of it that lie within the scope of this thesis.

4.1 Servers

First of all, I will be outlining the different servers that the protocol calls upon, which is important to understand the inner workings of all the processes that are involved when a user authenticates and starts a conversation with a friend.

4.1.1 Dispatch Server

This is often the first server the client contacts. The Dispatch Server redirects the client to one of the Notification Servers, where the client will then authenticate. If the client has previously stored a Notification Server, it can simply connect to that, and skip the Dispatch Server altogether. However, if the client cannot connect to the previous Notification Server, it will need the Dispatch Server to get a referral to a new one. The Dispatch Server that the MSN Protocol uses is *messenger.hotmail.com* on port *1863*.

4.1.2 Notification Server

This is the main server of the MSN protocol. After connecting to it, the client authenticates here. When the client has successfully logged in, it will receive the friends list, including their current statuses (Online, Busy, Away or Offline). This server keeps track of the client's presence, logging it out if it no longer responds. The client can start a conversation with a friend, and the Notification Server will assign a Switchboard Server to them, which they can then use to converse.

4.1.3 Switchboard Server

When two or more clients wish to have a conversation, a Switchboard Server will be assigned to them. It will relay their messages to each other, and carry information like the client's font type and color. Clients will also receive notifications when the other is typing a message, or someone joins or leaves the conversation.

4.1.4 Server overview

Figure 4.1 shows how the main three different types of MSNP servers relate to one another. It depicts that there is one Dispatch Server, which redirects users to several possible Notification Servers, that in turn redirect the user to a number of Switchboard Servers.

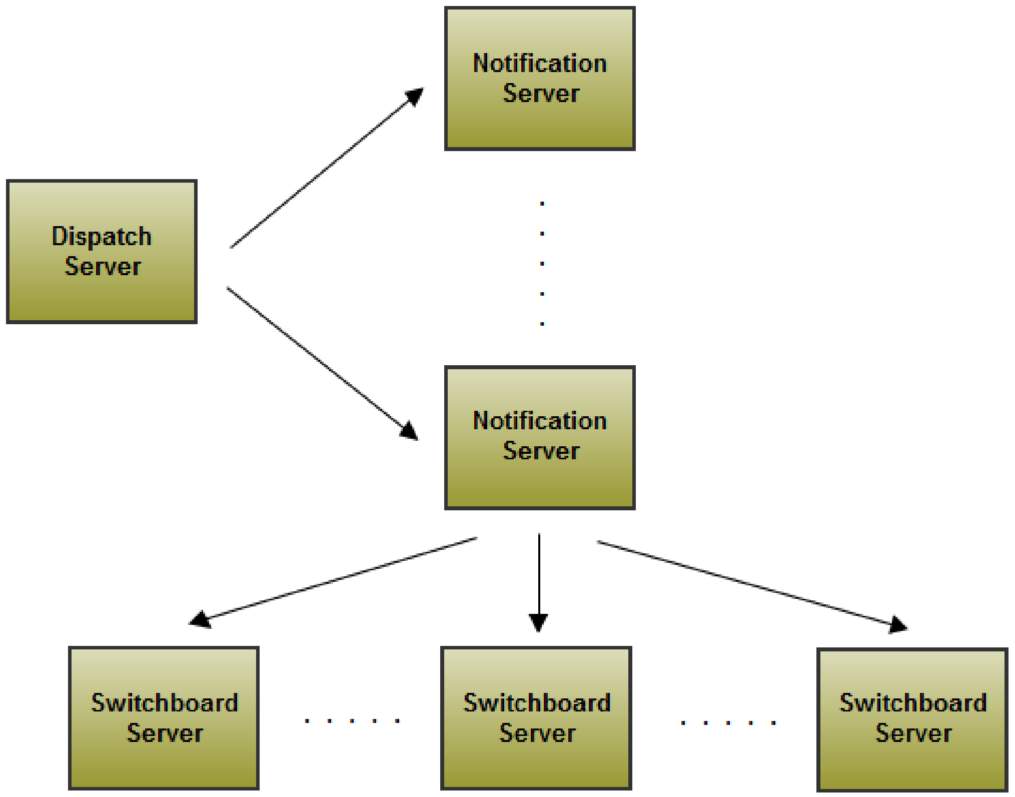


Figure 4.1: Overview of MSNP servers.

4.1.5 Conversation overview

To illustrate how two users connecting to the MSN network end up in the same Switchboard Server, the Figure 4.2 may be referenced. The two users connect to the Dispatch Server, and get referred to different Notification Servers, though this does not have to be the case. When one user starts up a conversation with the other user, a Switchboard Server is assigned to the both of them.
The colored arrows illustrate the referrals each user receives.

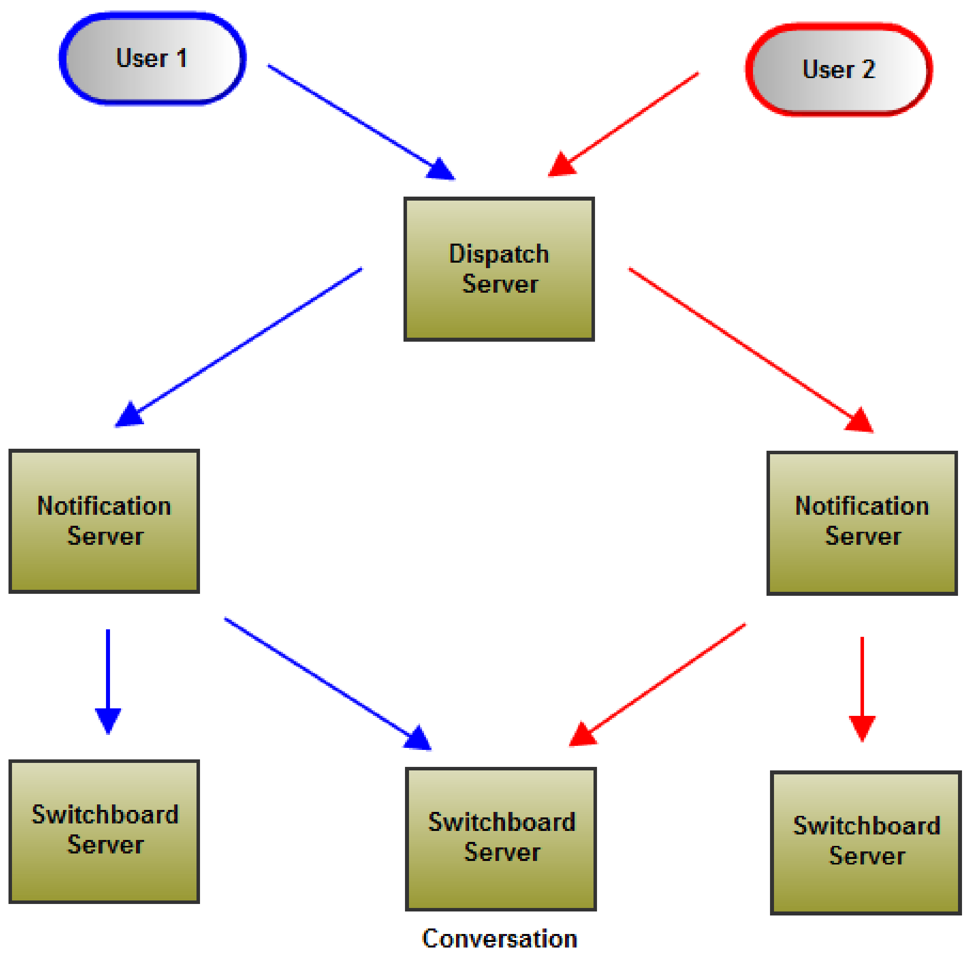


Figure 4.2: Server traffic when starting a conversation.

4.2 Message Layout

Every message from the client to the server is sent in the form of a command. There are four types of commands, but the difference between them is fairly minimal. These types are:

- Normal commands.
- Payload commands.
- Error commands.
- Asynchronous commands.

4.2.1 Command Names

Every command has a command name. This name consists of three capitalized letters in the case of a standard command, and three digits in the case of an error message. This command name is usually an acronym of the intended word, for example *USR* for *user* and *CHL* for *challenge*.

4.2.2 Transaction ID

Any command sent from the client to the server is required to have a Transaction ID. This is a number ranging from 0 to $2^{32} - 1$ which is used to match the client's requests to the server's responses. The message sent from the server in response to a command from the client will have the same Transaction ID.

Commands sent from the server to the client that are not sent in response to a request will either have no transaction ID, or a transaction ID of 0.

4.3 Command Layout

Every type of command has a different layout. Though all similar, they are distinctly different to account for the information they need to be able to contain in certain events.

4.3.1 Normal Commands

Most commands fall under the category of normal commands. They are typically used by the client to make request to the server, and used by the server to respond to these requests. The syntax for these commands are shown in figure 4.3.

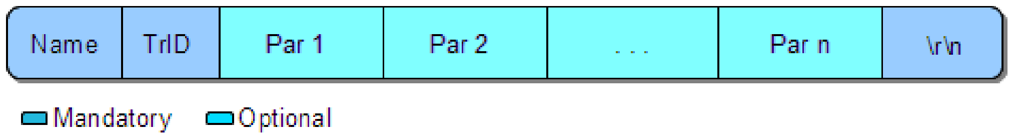


Figure 4.3: Normal command layout.

4.3.2 Payload Commands

A different type of command is used when a large chunk of text needs to be transmitted. These commands are called payload commands, and they contain the number of characters the text to follow will have, so the receiver knows how much data to read. Figure 4.4 shows the layout for these commands.

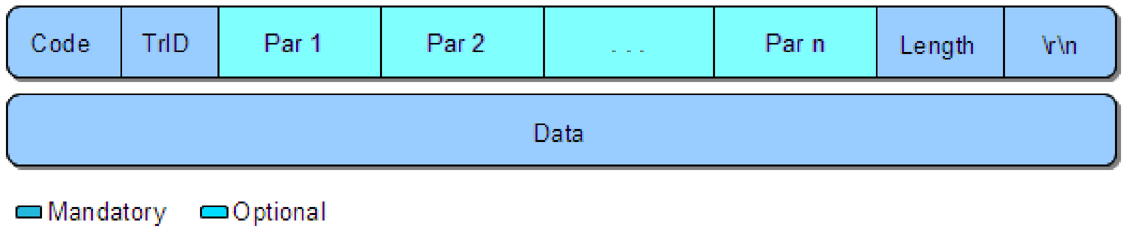


Figure 4.4: Payload command layout.

4.3.3 Error Commands

In the case that something goes wrong, an error command is transmitted. Their command name is a three digit error code. They also have a transaction ID, but no additional parameters. Figure 4.5 shows how these commands are set up.

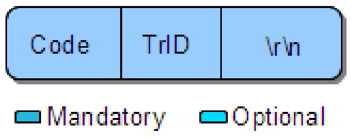


Figure 4.5: Error command layout

4.3.4 Asynchronous Commands

Commands sent by the server that are not in response to a client request are called asynchronous commands. They do not contain an actual transaction ID, so this field is either 0 or empty. The syntax for these commands can be seen in figure 4.5.

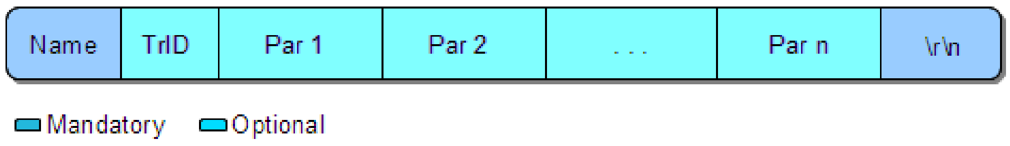


Figure 4.6: Asynchronous command layout.

4.4 Authenticating

4.4.1 Dispatch Server

The first step of logging into the MSN protocol is obtaining the address of a Notification Server. For that purpose, it connects to the Dispatch Server. When connected to it, several steps are taken to get a referral to a Notification Server.

First of all, the client tells the server what versions of the protocol it supports, using the *VER* command. The server then responds with the protocol that will be used henceforth, or will send an error and disconnect if the server doesn't support any of the protocols.

The client then uses the *CVR* to send the server some basic information about itself. This information includes the localization of the user, the operating system the client runs on, the name of the client, and finally the email address the user wishes to login to.
The server responds with the recommended client version to use, and the URLs to download this version of the official client.

Finally the client requests to login with the *USR* command, again sending the server the email address. The Dispatch Server then gives the client the address of a Notification Server to login to.

Example session

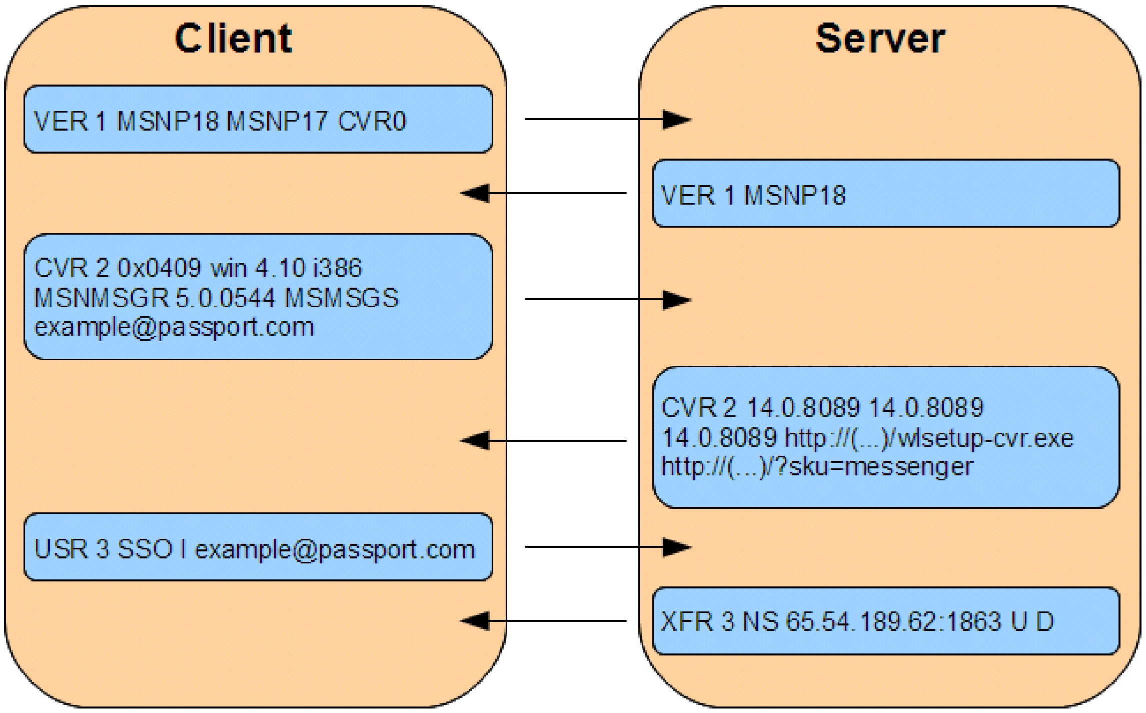

Client
Server
VER 1 MSNP18 MSNP17 CVR0
VER 1 MSNP18
CVR 2 0x0409 win 4.10 i386 MSNMSGR 5.0.0544 MSMSGS example@passport.com
CVR 2 14.0.8089 14.0.8089 14.0.8089 http://(...)/wlsetup-cvr.exe http://(...)/?sku=messenger
USR 3 SSO I example@passport.com
XFR 3 NS 65.54.189.62:1863 U D

Figure 4.7: Example session of getting a referral from the Dispatch Server.

4.4.2 Notification Server

The next step in the process is for the client to authenticate with the Notification Server. This is a more complicated process.

Just like it did with the Dispatch Server, the client first tells the server what version of the protocol it's using, and some basic information about itself, using the *VER* and *CVR* commands. The server responds in a similar to the Dispatch Server.

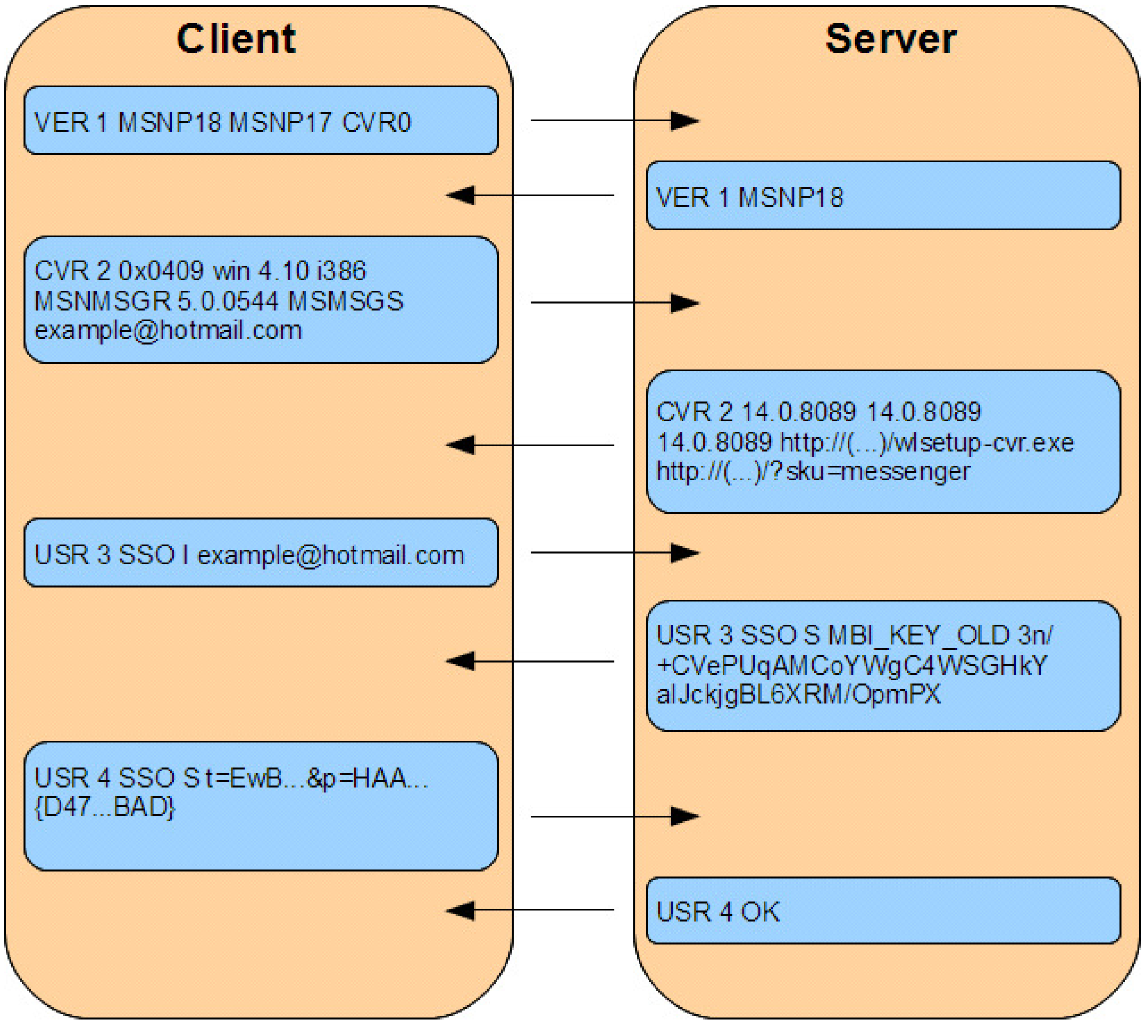
The client then sends a login request again with the *USR* command, but instead of getting a referral like it did from the Dispatch Server, the client will receive a policy and a nonce from the Notification Server.

The MSN client then creates a Simple Object Access Protocol (SOAP) message that contains the policy it received. The message also contains the login information of the user; the email address and the password. The SOAP messages used by the MSN protocol are long and not very readable or interesting, so I won't be looking at them in detail.

The client then connects to another Microsoft Server, specifically the Server Response File at *https://login.live.com/RST.srf*, and sends the newly created SOAP message there. It will receive a SOAP response that contains a security token that must be sent back to the Notification server, and a secret string.
The secret string and the nonce received from the Notification Server are then used to compute a return value. This value is computed by creating a key using TripleDES on the nonce. SHA1-HMAC is then used repeatedly on the key and then variations of it. TripleDES is used one more time to compute the final return value.

Finally, the client sends the security token it got from the SOAP response and the return value it computed to the Notification server, along with the computer's globally unique identifier (GUID). If all went well, the server responds with a confirmation that the client is now logged in.

Example Session


Client
Server
VER 1 MSNP18 MSNP17 CVR0
VER 1 MSNP18
CVR 2 0x0409 win 4.10 i386 MSNMSGR 5.0.0544 MSMSGS example@hotmail.com
CVR 2 14.0.8089 14.0.8089 14.0.8089 http://(...)/wlsetup-cvr.exe http://(...)/?sku=messenger
USR 3 SSO I example@hotmail.com
USR 3 SSO S MBI_KEY_OLD 3n/+CVePUqAMCoYWgC4WSGHkY alJckjgBL6XRM/OpmPX
USR 4 SSO S t=EwB...&p=HAA... {D47...BAD}
USR 4 OK
Figure 4.8: Example session of authenticating with the Notification Server.

Buddy List

Authenticating is not the only thing the MSN protocol uses SOAP requests for, acquiring the buddy list requires one too. The client posts the request to *http://contacts.msn.com/abservice/abservice.asmx*. As to not have to request the entire contact list, which can be quite lengthy, every time the user logs in, the SOAP message contains a time stamp of when the list was requested previously.
It is possible to use the SSL gateway to post this message, but the official client doesn't do this. The SOAP response contains a list of contacts and information about them.

After the client receives the contact list, it has to send this list to the Notification Server with the *ADL* command. This is a payload command that sends an XML message along with it. The layout of the XML message is as follows:

```
<ml l="1">
    <d n="domain1">
        <c n="email@hotmail.com" l="3" t="1"/>
    </d>
    <d n="domain2">
        <c n="test@hotmail.com" l="2" t="1"/>
        <c n="example@hotmail.com" l="2" t="4"/>
    </d>
</ml>
```

Figure 4.9: Example of a buddy list containing two contacts.

The Notification Server will then send a confirmation back to the client to let it know that it has received the contact list, and has accepted it.

4.4.3 Starting a Conversation

Every conversation is held on a Switchboard Server. There are two ways to connect to a server: you can request a new one or you can be invited to an already existing one.

A Switchboard Server is requested by sending an *XFR* command to the Notification Server with the string *SB* as a parameter. The Notification Server will then send back the address of the Switchboard Server assigned to the client, and it will include an authentication string. The client must then connect to the Switchboard Server and authenticate with the *USR* command using the string it received from the Notification Server.

Once connected, the client can send a *CAL* command to the Switchboard Server to invite a contact to partake in the conversation. This command will include the recipients email address.

The contact will then receive the asynchronous command *RNG* from their Notification Server. This command will include some basic information; the email address and the name of the person sending the invitation, the location of the Switchboard Server, an authentication string and a session id.

The invitee connects to the Switchboard Server and authenticates with the *ANS* command, using the authentication string and session id it received in the *RNG* command from the Notification Server. The participants that are already present in the conversation will receive the asynchronous *JOI* message to notify them that someone has joined. The client connecting will receive one or more *IRO* commands that inform it of the participants already present in the conversation.

4.4.4 Instant Messaging

Clients connected to the Switchboard server can now use the *MSG* command to send messages to all the other participants. This is a payload command, with a parameter that determines whether or not the server will respond with *ACK* and *NACK* to inform you about your message's delivery status. The other participants will receive a *MSG* command from the Switchboard Server, containing the message and the sender's email address.
The following is an example to illustrate the layout of a message sent in this manner:

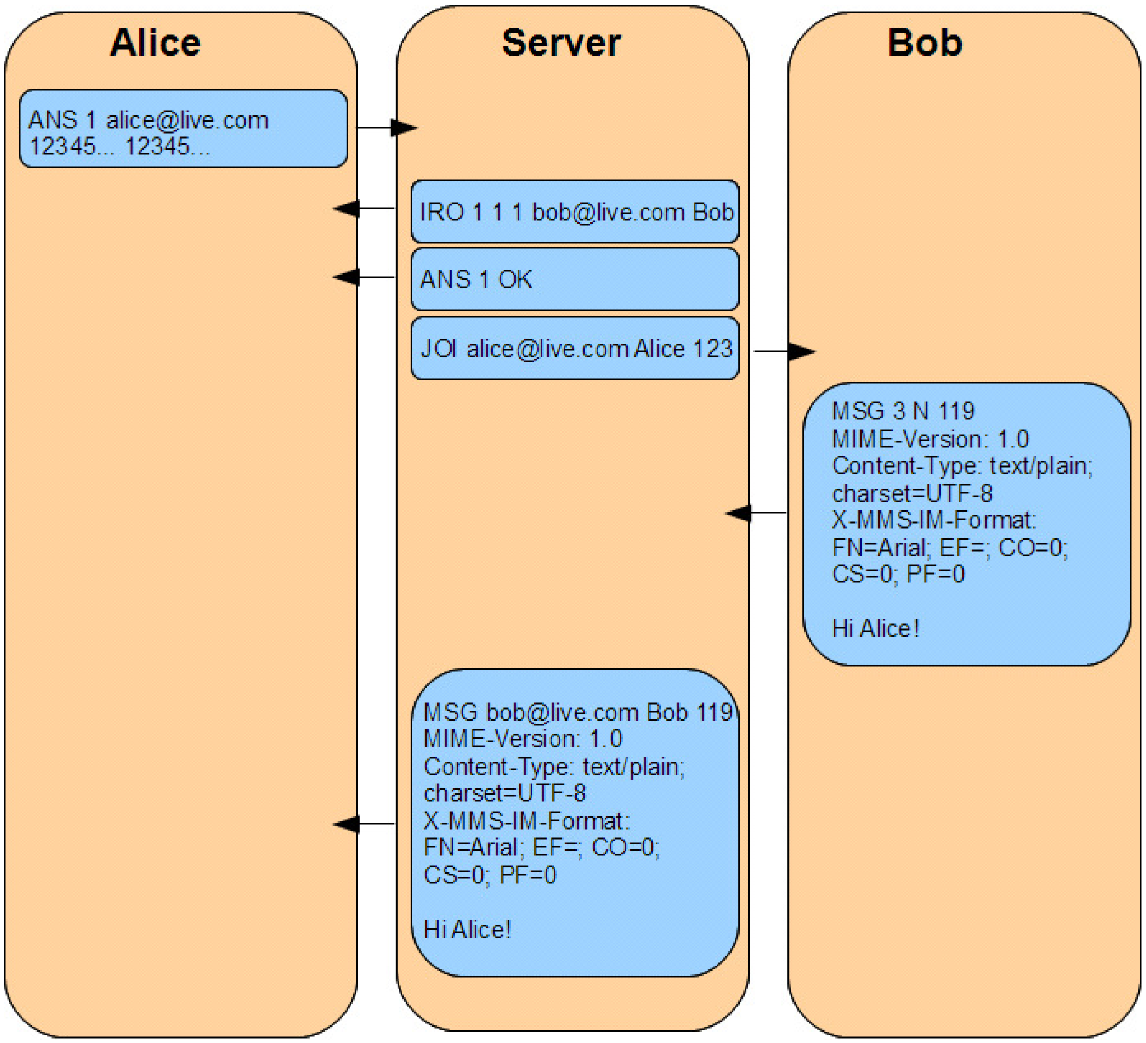
```
MSG 4 N 133\r\n

MIME-Version: 1.0\r\n
Content-Type: text/plain; charset=UTF-8\r\n
X-MMS-IM-Format: FN=Arial; EF=I; CO=0; CS=0; PF=22\r\n
\r\n
Hello! How are you?
```

Figure 4.10: Example of a message sent over the MSN Protocol.

4.4.5 Example Session

In this example, Bob is the only participant in the conversation until Alice joins. After Bob is informed that Alice has joined, he sends her the message "Hi Alice!"


Alice
Server
Bob
ANS 1 alice@live.com 12345... 12345...
IRO 1 1 1 bob@live.com Bob
ANS 1 OK
JOI alice@live.com Alice 123

```
MSG 3 N 119
MIME-Version: 1.0
Content-Type: text/plain;
charset=UTF-8
X-MMS-IM-Format:
FN=Arial; EF=; CO=0;
CS=0; PF=0

Hi Alice!
```

```
MSG bob@live.com Bob 119
MIME-Version: 1.0
Content-Type: text/plain;
charset=UTF-8
X-MMS-IM-Format:
FN=Arial; EF=; CO=0;
CS=0; PF=0

Hi Alice!
```

Figure 4.11: Example session of a conversation through the Switchboard Server.

4.5 Assessment

Now that the core mechanics of the MSN Protocol have been outlined, I will assess the protocol using the criteria that I listed earlier.

4.5.1 General

> *The protocol should be open source, and third parties should not be discouraged from creating clients or servers for it.*

There is no official documentation of the MSN Protocol. Nearly all information available is discovered through reverse engineering. Even now, the meaning of all the data sent over the protocol is still not understood. The protocol uses a challenge system that requires the client to compute a response token in a very complicated way, designed to make it more difficult for third party software to make use of the protocol.

Score: **1/5**

> *The protocol should be comprehensible, with meaningful command names and a clear structure.*

The command names used all consist of three letters, followed by a transaction ID and then a number of arguments. This makes for a very clear structure. The command names are often acronyms, making it easier to remember their meaning. Examples are *SYN* for "Synchronize" and *USR* for "User". However, the function of the parameters are often unclear.

Score: **4/5**

4.5.2 Security

The password restrictions must be strict enough to make the password not vulnerable to easy attacks, like the dictionary attack.

MSN requires a password of at least six characters, and does not enforce using numerical characters or symbols. With this, MSN asks the bare minimum of users when it comes to password creation, resulting in possibly very weak passwords.

Score: **2/5**

The password must be transferred to the instant messaging server encrypted.

The user's password is sent to the server over a Hypertext Transfer Protocol Secure (HTTPS) connection, and it's encrypted through 128 bit SSL. This is the only time the user's password is transferred.

Score: **5/5**

A message sent to another user should be encrypted.

Messages sent from one person to another are not encrypted in any way. This makes the MSN protocol not suited for confidential communication.

Score: **1/5**

The encryption used by the protocol must be sufficiently strong, making it infeasible to crack.

128 bit encryption is very secure. It is infeasible to crack, even with high numbers of processors at your disposal. As thus, it is very widely used in all fields. This level of encryption is easily sufficient for an instant messaging protocol.

Score: **5/5**

4.5.3 Efficiency

There must not have a great data overhead, messages sent should be reasonably short.

Messages sent by the MSN Protocol tend to be surprisingly concise, only containing the bare necessities. There is not a large data overhead when sending messages. However, the server does expect you to send meta data about your messages every time, which seems redundant. This contains information such as font type, size and color and encoding. This could be avoided by establishing this information once initially, and storing it client side. It would then only need to be updated when the user makes a change.

Score: **4/5**

There should not be a great number of connections made by the protocol.

The MSN protocol first connects to a Dispatch Server, which entire functionality is to refer the client to a Notification Server. This Notification Server never seems to change, so the Dispatch Server seems rather unnecessary. The buddy list is retrieved through a HTTP connection, rather than through the Notification Server like you might expect. All in all, this makes for more connections to more different services than required. However, files and display pictures are communicated through the Switchboard Server, so the number of new connections is not as high as it could be.

Score: **3/5**

The protocol should not require a large number of messages for a simple action.

I do not consider authenticating to be a simple action, so I will not take the long authentication process into regard. Starting up a conversation with someone requires several actions. The Notification Server needs to be contacted to get a referral to a Switchboard Server, to which you must then authenticate. The client then has to invite the other participant to join the Switchboard session, making this entire process rather lengthy. However, the majority of a user's actions will be sending messages to another person. This requires only a single command once both participants are connected to the Switchboard Server.

Score: **4/5**

4.5.4 User friendliness

There should be a large number of contacts allowed.

The protocol itself does not restrict the maximum number of contacts that a user is able to have. However, the official client will allow a user to have about a thousand people in their buddy list. Even though this is a large number, the amount of data the server needs to store for additional contacts is fairly minimal, so I do not think a restriction needs to me imposed.

Score: **4/5**

The users should be able to pick different font styles and colors.

While the MSN protocol does not specifically include functionality for fonts and colors, this information is communicated by the official client as preceding meta data before every message that is sent by the user. This meta data is sent in the form of an Internet Media type, better known as Multipurpose Internet Mail Extensions (MIME). I do not find this a very elegant solution, and would prefer a protocol to have a more strict built in format for it. It is now left to the clients to implement a work around to achieve different font styles and colors.

Score: **2/5**

Users should be able to choose and change their own screen name.

Changing the user's display name is a very simple process, requiring only a single command. This makes the process very short and uncomplicated, as it should be. Furthermore, it accepts any characters the user wishes to use, as long as it is URL encoded.

Score: **5/5**

4.5.5 Results

Table 4.12 contains all scores that were given earlier regarding the MSN protocol and includes an average score for each field. Finally, it shows a global average score of how well the protocol performed in all fields.

Criterion	**Field**	**Weight**	**Score**
Open source	General	3	1/5
Comprehensible	General	1	4/5
Field average	General	-	1.75/5
Password restrictions	Security	3	2/5
Password encrypted	Security	3	5/5
Messages encrypted	Security	2	1/5
Strong encryption	Security	3	5/5
Field average	Security	-	3.45/5
Data overhead	Efficiency	2	4/5
Number of connections	Efficiency	1	3/5
Number of messages	Efficiency	2	4/5
Field average	Efficiency	-	3.80/5
Number of contacts	User friendliness	2	4/5
Font styles and colors	User friendliness	2	2/5
Choose display name	User friendliness	3	5/5
Field average	User friendliness	-	3.86/5
Global average	All	-	**3.37/5**

Table 4.12: Grading the MSN Protocol according to the criteria.

5 OSCAR Protocol

In this chapter, I will be examining the OSCAR Protocol. Similar to the MSN Protocol, I will look at its core mechanics to uncover relevant information that will help with assessing the protocol later.

5.1 Servers

Just like the MSN Protocol, OSCAR also makes use of several different servers, that all serve a different purpose. However, no server is directly comparable to an MSNP server, as the functionality they offer is never exactly the same.

5.1.1 Authorization Server

This is always the first server the client contacts, in order to authenticate and get a referral to the Basic OSCAR Service Server (BOSS). This server is only used for the initial authentication, and can be closed after connecting to the BOSS.

5.1.2 Basic OSCAR Service Server

This is the main OSCAR server. The client will have to use this server to log in to the AOL Instant Messenger network. In order to do so, the client will need the cookie it received from the Authorization Server. The server is used for almost all of the main tasks that clients will want to perform, like sending messages and requesting information.

5.1.3 Others

There are several other servers, like the Buddy Icon Server, and the Chat Room Setup Server. However, they don't offer any functionality that lies within the scope of this research paper, so I won't be discussing them any further.

5.2 Frames

A server connection with the OSCAR protocol is split up into several frames, which are sometimes referred to as channels. They are used to allow for parallel streams of communication, without needing to connect to a different server. The frames have numbers, and the protocol allows for up to 256 of them to be used, but only five actually serve a purpose to this date.

5.2.1 #1 - Connection Start Frame

This frame can be seen as the "login channel". It is used to log in to any of the servers. All communication with the Authorization Server occurs on this frame, as well as the initial authentication with the Basic OSCAR Service Server.

5.2.2 #2 - Main Frame

Packets are most commonly sent over this frame. It covers most of the OSCAR Protocol's functionality.

5.2.3 #3 - Error Frame

When something occurs that is undesirable or unexpected, and the server sends an error message to the client over this frame.

5.2.4 #4 - Connection End Frame

When the server signs the client out of the network, it will send it a message over this frame that contains an error code.

5.2.5 #5 - Keep Alive Frame

This is the newest OSCAR frame, and not much is known about it. It appears to be used solely for ping-requests by the server to check that the client is still responding when it has been idle for some time.

5.3 Message Layout

The OSCAR Protocol uses two wrapper protocols to make messages more organized and easier to parse. It also uses a Type-Length-Value tuple to transform raw data into something more recognizable and interpretable.

5.3.1 Frame Layer Protocol

The Frame Layer Protocol (FLAP) is a low level protocol that serves as a wrapper around practically all messages sent over the OSCAR protocol. FLAP messages contain a FLAP id, the frame number the message is sent over, a sequence id for the purpose of synchronization and the length of the message within the FLAP wrapper. Oddly, the FLAP id is always *2A*, so it doesn't seem to serve a useful purpose. The layout of a FLAP message is as follows:

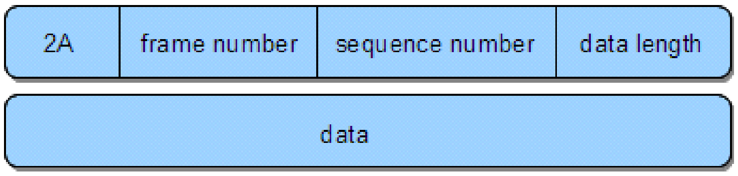


Figure 5.1: Layout a FLAP wrapper.

5.3.2 Simple Network Atomic Communication

All messages sent over the Main Frame are also wrapped in a Simple Network Atomic Communication (SNAC) wrapper. This is another low level protocol that sits on top of the FLAP layer. This means that a message sent over the Main Frame is wrapped twice, first by the SNAC protocol and then by FLAP.

SNAC messages are divided into different categories, called Food Groups. In turn, every Food Group is divided into subtypes of SNAC messages. Each message also contains a request id that is used to match a SNAC request with the server's response. However, this is only necessary when requesting information from the server, not when sending a command. A message also contains flags for some additional information. The layout of a SNAC message is the following:

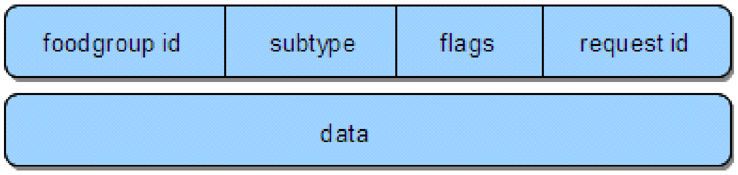


Figure 5.2: Layout of a SNAC wrapper.

5.3.3 Type-Length-Value Tuples

In order to allow raw data to be interpreted more efficiently, it is stored in a Type-Length-Value (TLV) tuple. This wrapper is used for almost every type of data, and makes data fields easily recognizable. A TLV tuple looks like this:

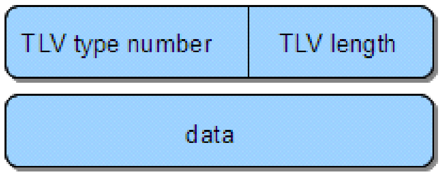


Figure 5.3: Layout of Type-Length-Value tuples.

5.3.4 Server Side Information Items

The server stores information about each account. This includes the account's buddy list and their blocked list. Every item on such a list is stored in a structure. This contains the name of the item, and specifies the item's id, type and group. The id of an item is a unique value that represents the item. There are several different item types, most notably *buddies* and *blocked users*. An item's group is always *0* unless it is a buddy icon. Furthermore, a list item contains some data, consisting of any number of TLV tuples that offer more information about the item. A SSI list item can be represented in the following way:

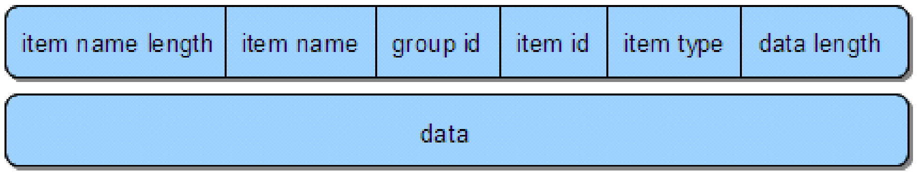


Figure 5.4: Layout of Server Side Information Items.

5.3.5 Instant Messaging Data Structure

When an Instant Message is sent, the data necessary for the message to be received properly is contained in Instant Message Data (IM Data) wrappers. An IM Data unit consists of a type and a subtype, in order to make it easy to identify what kind of data lies within. It also contains the length of the data for easy parsing by the servers and clients. An IM Data wrapper has the following layout:

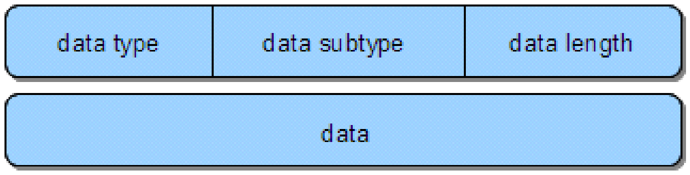


Figure 5.5 Layout of Instant Messaging Data Structures.

5.4 Authenticating

5.4.1 Authorization Server

The first step of logging into the AIM network is obtaining a cookie from the Authorization Server to log into the Basic OSCAR Service Server (BOSS). In order to do this, the client must connect to the Authorization Server at *login.oscar.aol.com*. The server and the client first exchange FLAP version numbers.

The rest of the authorization process consists solely of SNAC messages. The client will start by sending the server a SNAC message with the username of the account that it is logging into. If the account name is recognized, the server will respond with a message containing an authentication key, and the length thereof.

The client then uses this key, along with the password, to construct a md5 hash. It sends this hash back to the server, with some more additional information. This includes the account's screen name, client version numbers and localization information. If all goes well, the server responds with the BOSS address and an authorization cookie, along with some other information.

Example Session

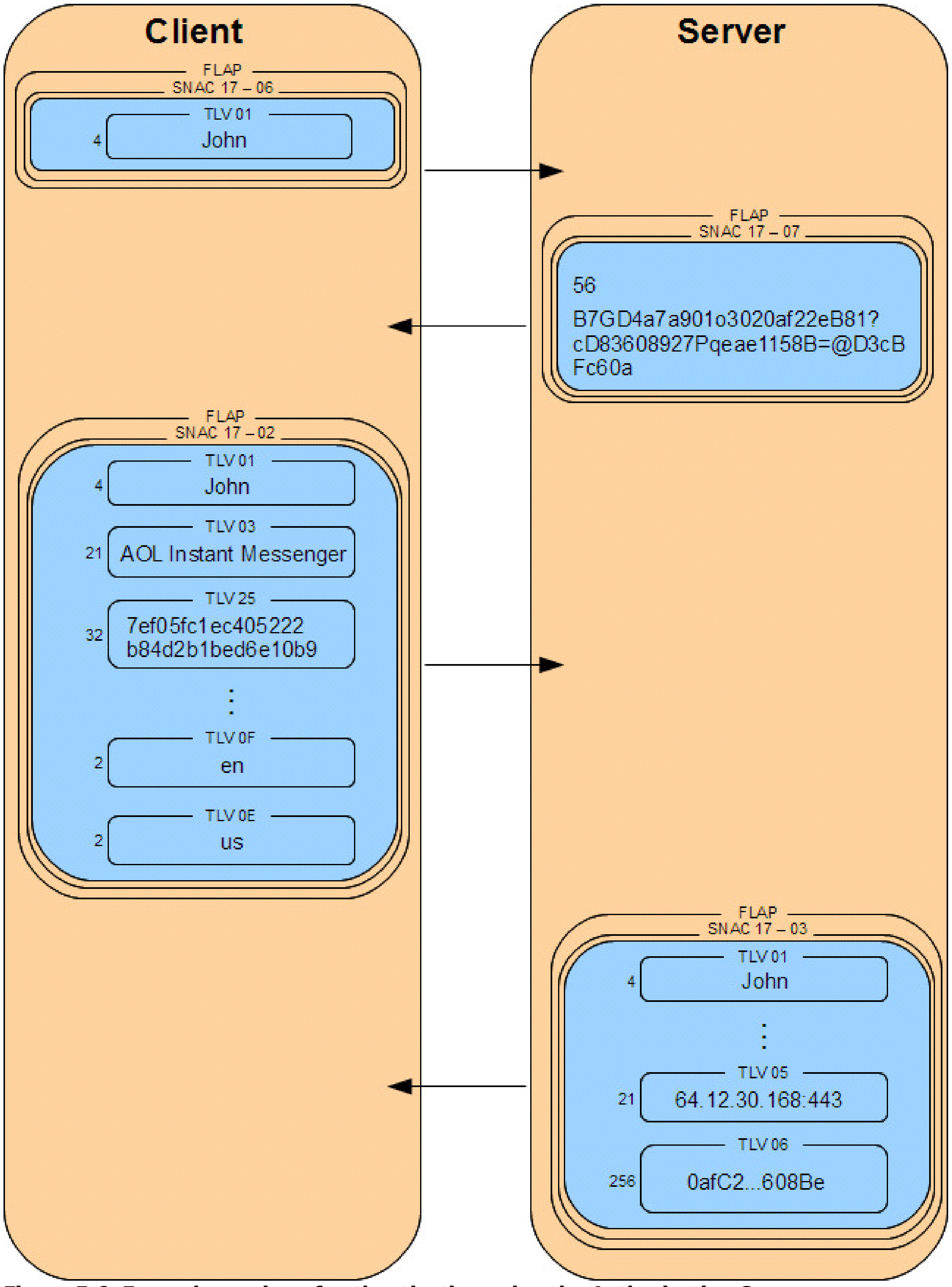


Figure 5.6: Example session of authenticating using the Authorization Server.

5.4.2 Basic OSCAR Service Server

The client then proceeds to actually sign into the AIM network, by again sending its FLAP version number to the server, but this time accompanied by the authorization cookie received from the Authorization Server.

The rest of the communication uses SNAC messages. The server sends a list of supported SNAC Food Groups to the client. The client then responds with the version it supports for these groups. The server then shows the same courtesy, and sends a list back with the versions it supports. The client then requests the list of the so called rate limits from the server. A rate limit is a value that indicates how fast the client is allowed to send messages to the server. For example, if the client exceeds the Alert Level rate limit, it will receive an alert that it is close to being disconnected, which will then happen if the client exceeds the Disconnect Level rate limit. After the client receives this list from the server, it will send a message back acknowledging that it has received the rate limits.

The next step is setting up all the services. The client will send several SNAC requests to the server, and the server will send back responses. The client asks for the limitations and parameters of buddy lists, profiles and instant messaging. It also sends the server its own limitations and profile information.

The client then requests the Server Stored Information (SSI). This information contains your buddy list, your privacy settings and your blocked list. The client can make changes to this information if it wishes to do so. It will then sends a message to the sever activating the Server Stored Information, which makes the changes the client made to the SSI take effect, and is necessary in order for the server to start sending presence notifications.

Finally, the client sends a message to the server indicating that it's ready to be shown as online on the network now.

5.5 Buddy List

The buddy list is stored on the server as Server Side Information. In the authentication process, when the client requests this information from the server, it will receive the buddy list, among other things. An item in the SSI that represents a buddy will contain the buddy's screen name.

The server's respond to a request for the Server Side Information will contain the version of the SSI protocol, the number of items in the list, the actual list itself and the timestamp of when the list was last updated. The version of the SSI Protocol is currently *0*. An example of such a message is the following:

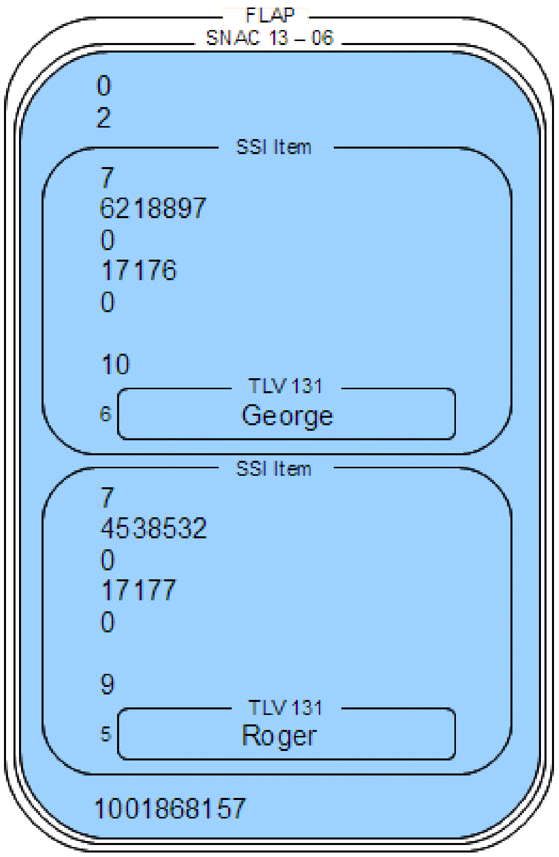


Figure 5.7: Example of a contact list containing two users.

5.6 Instant Messaging

Inter Client Basic Message (ICBM) messages are used to send and receive Instant Messages. Both incoming and outgoing messages are handled in the form of SNAC commands. These commands contain information about the recipient or sender and the message itself. Every message has an ICBM cookie, which helps the server link conversations together. It also specifies the ICBM channel on which the message is sent or received, which is always *1* for plain text messages. Furthermore, it lists the account name of the recipient or sender and the length thereof.

In the case of an incoming message, the SNAC command contains additional information about the sender, including their user status, idle time and account creation time.

The SNAC command also contains two IM Data structures. The first structure defines the capabilities the recipient needs to support in order to correctly receive the message, and the second one contains the actual message, and some information about it. This information includes the used encoding and language. The following diagram shows a representation of an ICBM message for an outgoing instant message:

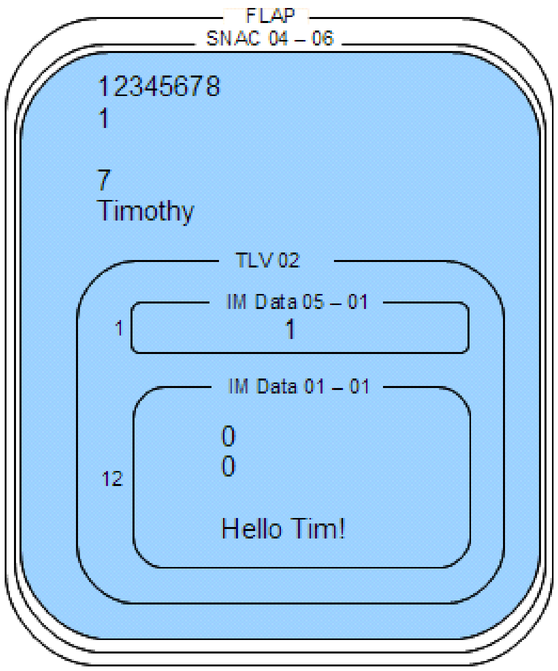


Figure 5.8: Example of a message sent over the OSCAR Protocol.

5.7 Assessment

After outlining the core mechanics of the OSCAR protocol, I will now assess it in the same fashion and using the same criteria as were used earlier to assess the MSN protocol.

5.7.1 General

> *The protocol should be open source, and third parties should not be discouraged from creating clients or servers for it.*

Despite what the name *Open* System for Communication in Real-time implies, the specifications for the protocol are proprietary. AOL has gone to great lengths to keep competitors from implementing clients that are compatible with their protocol. However, OSCAR does not make use of a challenge system that requires the computation of a response.

Score: **3/5**

> *The protocol should be comprehensible, with meaningful command names and a clear structure.*

The FLAP and SNAC wrappers that the OSCAR protocol makes use of results in low legibility, because the data in these wrappers does not consist of ASCII characters, making them very hard to read. Furthermore, the command names are not meaningful, they are merely numbers that need to be looked up in a list in order to be able to figure out their function.

Score: **1/5**

5.7.2 Security

The password restrictions must be strict enough to make the password not vulnerable to easy attacks, like the dictionary attack.

The OSCAR protocol itself does not enforce any password restrictions. However, the AIM signup page requires a password of six to eight characters, that doesn't contain any symbols. This means they actually prevent a user from choosing a strong password, by not allowing symbols or passwords of over eight characters. In doing so, they cause their passwords to be unnecessarily simple and easy to crack.

Score: **1/5**

The password must be transferred to the instant messaging server encrypted.

The password of an AIM account is not directly sent to the Authorization Server. An md5 hash of the password is first computed, which is then transmitted over the protocol. This is the only time the password is sent by the client.

Score: **5/5**

A message sent to another user should be encrypted.

Messages over the OSCAR protocol are not encrypted in any way, making it very easy for someone with a packet sniffer to read every word that is communicated between users.

Score: **1/5**

The encryption used by the protocol must be sufficiently strong, making it infeasible to crack.

MD5 hashes are no longer considered sufficiently safe. Anyone can use an online website that makes use of a Rainbow Table to decrypt an md5 hash in many cases. Considering many people do not choose a strong password, and this is also not enforced by the protocol, this method is very unsafe. However, if a sufficiently strong password is used, it is still very difficult to crack an md5 hash.

Score: **2/5**

5.7.3 Efficiency

> *There must not have a great data overhead, messages sent should be reasonably short.*

No message sent using the OSCAR protocol is actually short. The FLAP and SNAC wrappers add a fair amount of data to every message. The usage of TLV tuples and IM Data structures also adds to this. As a result, practically every message sent has a large data overhead.

Score: **2/5**

> *There should not be a great number of connections made by the protocol.*

The protocol mainly communicates through the BOSS, but also uses separate severs for authentication and buddy icons. It does not use additional servers for conversations between users, so the total number of connections is relatively low.

Score: **4/5**

> *The protocol should not require a large number of messages for a simple action.*

Starting up a conversation with the OSCAR protocol requires significantly fewer actions than the MSN protocol demands. Participants in a conversation do not converse over a separate server, instead the communication goes through the BOS Server. Though this may be more chaotic, since the conversations have to be separated with the help of ICBM cookies. All other actions typically require only one message to be sent to the server, resulting in an overall low number of sent messages.

Score: **5/5**

5.7.4 User friendliness

There should be a large number of contacts allowed.

When the OSCAR protocol sends the user's contact list to the client, it includes a 16-bit word parameter that tells the client how many contacts there are. This means that the maximum number of contacts the protocol supports is 2^{16} (65536) people. However, the official client only supports a maximum of a thousand friends in a user's contact list.

Score: **4/5**

The users should be able to pick different font styles and colors.

The protocol itself does not have any support for different colors or font styles. The only way to change the layout of messages, is to include this information in the actual content. The official client makes use of this by using HTML to include markup information in the message, so that it may be displayed as desired.

Score: **2/5**

Users should be able to choose and change their own screen name.

The OSCAR Protocol only supports the editing of the user's screen name's format. This means that it is possible to change capitalizations and punctuations, but it is not possible to make greater changes to the screen name. This means that the user will have to register for a new account when a new screen name is desired.

Score: **2/5**

5.7.5 Results

The following table contains all scores that were given earlier regarding the OSCAR protocol and includes an average score for each field. Finally, it shows a global average score of how well the protocol performed in all fields.

Criterion	**Field**	**Weight**	**Score**
Open source	General	3	3/5
Comprehensible	General	1	1/5
Field average	General	-	2.50/5
Password restrictions	Security	3	1/5
Password encrypted	Security	3	5/5
Messages encrypted	Security	2	1/5
Strong encryption	Security	3	2/5
Field average	Security	-	2.36/5
Data overhead	Efficiency	2	2/5
Number of connections	Efficiency	1	4/5
Number of messages	Efficiency	2	5/5
Field average	Efficiency	-	3.60/5
Number of contacts	User friendliness	2	4/5
Font styles and colors	User friendliness	2	2/5
Choose display name	User friendliness	3	2/5
Field average	User friendliness	-	2.57/5
Global average	All	-	**2.67/5**

Table 5.8: Grading the OSCAR Protocol according to the criteria.

6 Protocol Comparison

6.1 Servers

Both the OSCAR Protocol and the MSN Protocol make use of a number of different servers, each with its own distinct functionality. It is noteworthy that they have made different choices in this regard.

While MSNP uses a Dispatch Server to refer to the protocol's main server, OSCAR makes opted not to do this. They choose to have the client connect directly to the server where authorization takes place. As opposed to the MSN Protocol, they have opted to handle this very important action on a separate server, where the MSN Protocol has the main server handle it. MSNP also requires the client to send a SOAP file to the Login Server over SSL as a part of the authorization process.

Though OSCAR's main server doesn't take care of authorization, it still takes care of a great deal of actions. All conversations are communicated by this server, and it keeps track of the different ones by cookies. MSNP likes to split up the conversations and have separate servers resolve them. OSCAR keeps things structured by using Frames, which helps easily differentiate between different kinds of messages. However, since the vast majority of data is transmitted over the second Frame, its usefulness is limited.

6.2 Commands

The biggest difference between the two protocols when it comes to sending commands to the servers, is that OSCAR makes use of wrappers and the MSN Protocol does not.

OSCAR includes the sequence number in the FLAP wrapper, and MSNP includes this number as the second parameter of every command. The first parameter is the command name, which OSCAR includes in the SNAC wrapper. This allows OSCAR to have a certain hierarchy for its commands, which is something the MSN Protocol does not have. While this makes OSCAR commands more structured, it decreases the readability of the command names.

OSCAR further structures its data with Type-Length-Value tuples, Server Side Information Items and Instant Messaging Data Structures. MSNP chooses to simply send data as plaintext information, which increases legibility for persons, but also makes the data more chaotic.

6.3 Authorizing

Authenticating using the MSN Protocol is a much longer process than it is with the OSCAR Protocol. The former makes use of an SSL connection where a separate Login Server will compute a security token and a secret string that is returned to the client upon receiving its user information. The secret string is then further manipulated by the client, using a nonce received from the Notification Server, to computer a return value. Combined with the security token, it is then used to finally log in to the system.

OSCAR uses a more direct approach. The client sends its username to the Authorization Server, which sends back a response string. This is combined with the user's password, the md5 hash of which is sent to the server, resulting in an authorization cookie that will allow the user to log in to the system.

6.4 Buddy Lists

The OSCAR takes a more intuitive approach to the retrieval of contact lists. It can simply be requested over the BOSS, and will be received in the form of SSI items. This information can be easily cached, because there is also a command available to request the time the buddy list was last changed on the server, so that the client often will not have to request the list.

MSN requires the contact list to be retrieved through a SOAP message sent to a separate server. This information can also easily be cached, as the SOAP request includes the time the client has last updated the contact list. An XML format is used to store the buddy list information. The client then has to send this information to the Notification Server, which seems to be a very roundabout approach.

6.5 Messaging

MSNP uses an entire process to start a conversation with another user, which involves a lot of messages being sent by the clients and servers before the conversation even having started. A separate Switchboard Server is assigned to the users, and they are notified of each other's presence. Messages sent back and forth are then handled solely by the Switchboard Server, without the involvement of the main server. The messages are marked up by the usage of Internet Media Types.

The OSCAR protocol takes an entirely different approach. The main server handles conversations, and does so in a much simpler fashion. There is no real process to start a conversation, clients simply send each other messages. The clients keep track of ICBM cookies to link conversations together, so that they may be displayed in separate windows for the user's convenience. Although both protocols do not have support built in to handle the mark up of messages, AOL Instant Messenger uses HTML to achieve this effect.

6.6 Criteria

6.6.1 General

The following table shows the side-by-side scores of the MSN Protocol and OSCAR in the general area.

Criterion	**Weight**	**Score MSNP**	**Score OSCAR**	**Winner**
Open Source	3	1/5	3/5	OSCAR
Comprehensible	1	4/5	1/5	MSNP
Weighted Average	-	1.75/5	2.50/5	OSCAR

Figure 6.1: Comparison of both protocols regarding the General criteria.

Both protocols score higher than the other on one criterion, and in the other one. However, since I marked the Open Source criterion to be more important, OSCAR has a higher weighted field average. However, both protocols have low scores here, considering that a score of 3/5 should be the bare minimum of what is passable.

6.6.2 Security

Next up is the security area. Again, I have provided a comparison of the protocol's scores in this field.

Criterion	**Weight**	**Score MSNP**	**Score OSCAR**	**Winner**
Password restrictions	3	2/5	1/5	MSNP
Password encrypted	3	5/5	5/5	Tie
Messages encrypted	2	1/5	1/5	Tie
Strong encryption	3	5/5	2/5	MSNP
Weighted Average	-	3.45/5	2.36/5	MSNP

Figure 6.2: Comparison of both protocols regarding the Security criteria.

OSCAR doesn't manage to score higher than MSNP a single time in this area. MSN allows stronger passwords, both protocols encrypt their passwords, and neither one encrypts the messages sent by users. The MSN Protocol also uses a far stronger password encryption than OSCAR does. It should be no surprise that MSNP has a far greater average score than OSCAR does in the field of security.

6.6.3 Efficiency

The following table depicts the scores of the MSN Protocol and the OSCAR protocol side by side.

Criterion	**Weight**	**Score MSNP**	**Score OSCAR**	**Winner**
Data overhead	2	4/5	2/5	MSNP
Number of connections	1	3/5	4/5	OSCAR
Number of messages	2	4/5	5/5	OSCAR
Weighted Average	-	3.80/5	3.60/5	MSNP

Figure 6.3: Comparison of both protocols regarding the Efficiency criteria.

Both protocols score fairly well in this category. OSCAR falls behind when it comes to data overhead, due to its numerous use of wrappers with every message that is sent. It does however score higher at the other two criteria, making less connections and requiring less messages to be sent. Despite this, the average score is still slightly less than MSNP's, due to the difference in weights between the different criteria. Both protocols have a respectable score in this area.

6.6.4 User Friendliness

For the final set of criteria, the following is the side-by-side comparison of the two protocols in regards to user friendliness.

Criterion	**Weight**	**Score MSNP**	**Score OSCAR**	**Winner**
Number of contacts	2	4/5	4/5	Tied
Font styles and colors	2	2/5	2/5	Tied
Choose display name	3	5/5	2/5	MSNP
Weighted Average	-	3.86/5	2.57/5	MSNP

Figure 6.4: Comparison of both protocols regarding the User Friendliness criteria.

OSCAR's biggest letdown here is not offering the option to change your display name. Having to make a separate account for such a simple action is not acceptable. The protocols score equally well at the other two criteria, since they allow a similar number of contacts, and neither protocol directly supports font styles and colors. This results in MSNP having a far higher average score, since OSCAR received a very low score for the most important criterion.

7 Conclusion

Now that the protocols have been compared for every field of criteria individually, all that remains is reviewing their scores across the board. The following table gives a concise overview of the grades I have assigned to each protocol.

Field	**Score MSNP**	**Score OSCAR**	**Winner**
General	2.00/5	2.50/5	OSCAR
Security	3.45/5	2.36/5	MSNP
Efficiency	3.80/5	3.60/5	MSNP
User friendliness	3.86/5	2.57/5	MSNP
Overall	3.41/5	2.67/5	MSNP

Figure 7.1: Comparison of both protocols regarding all the criteria.

The MSN Protocol has a respectable score in every field, except for General. This results in an overall score of 3.41 out of 5, which is easily a passing grade. OSCAR has a far less favorable grade list, only doing well in the efficiency field. This results in a poor list for this protocol. Microsoft's MSN Protocol achieves a very clear victory over AOL's OSCAR Protocol.

An obvious point of improvement for both protocols is releasing the official protocol specifications, no longer battling against the creation of third party clients. Reverse engineering will reveal enough information about protocols for third party clients to exist, which is illustrated by the many clients available. If the creators of the protocol want their clients to be used, they will just have to make them superior to any third party clients, rather than trying to discourage those from being created.
Another advantage of making the protocols open source, is that add-ons may be created by the community to improve the clients, and add functionality. This can be clearly seen in the case of internet browsers like Mozilla Firefox, that has a very large number of add-ons developed for it, increasing its functionality drastically.

Another step that should be taken is encrypting messages sent to the servers. Users are often not aware that their messages can be read extremely easily by anyone who listens in on the network. Microsoft does not offer any way to send encrypted messages over the network, but there is third party software available to achieve this. AOL does offer a way to send encrypted messages, but charges money for this feature. Even though most users will not care that their messages are not being encrypted, since instant messaging is mainly used for informal communication, it can still be viable to offer this functionality to reach a broader audience.

Instant Messaging clients are facing a similar kind of competition that internet browsers have been facing. MSN Messenger has always been installed on every Windows computer, much like Internet Explorer, which is one reason why this client is so popular. Traditionally, a lot of users always simply used what was available to them. In this day and age though, people have become more aware that there are alternatives, and Internet Explorer usage has dropped drastically. In order to maintain Windows Live Messenger's dominant position, Microsoft will have to keep working on making the client, and the underlying protocol, the best it can be.

All in all, both protocols still have room for improvement. If they want to stay ahead of competitor's protocols, like the open source protocol XMPP, they will have to keep evolving and offering more and better functionality.

8 References

[1] M. Mannan and P. van Oorschot. A protocol for secure public Instant Messaging. (2006) *Financial Cryptography and Data Security.*

[2] R. Jennings, E. Nahum, D. Olshefski, D. Saha, Z. Shae and C Waters. A study of Internet instant messaging and chat protocols. (2006) *IEEE Network.*

[3] D. Kormann and A. Rubin. Risks of the Passport single sign-on protocol. (2000) *Computer Networks.*

[4] Nielsen//NetRatings. (2002) *http://www.nielsen-online.com/pr/pr_020617.pdf*

[5] How secure is the encryption used by SSL? *http://www.inet2000.com/public/encryption.htm*

[6] OSCAR Protocol Specification. *http://web.archive.org/web/20080308233204/http://dev.aol.com/aim/oscar/*

[7] jOSCAR Specification. *http://code.google.com/p/joscar/wiki/FrontPage*

[8] M. Mintz and A. Sayers. (2003) *MSN Messenger Protocol. http://www.hypothetic.org/docs/msn/*

[9] MSNC - MSNPiki. (2008) *http://msnpiki.msnfanatic.com/index.php/MSNC*

[10] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. *http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf*

[11] Rainbow Tables. *http://kestas.kuliukas.com/RainbowTables/*

[12] Wikipedia. Comparison of Instant Messaging protocols. *http://en.wikipedia.org/wiki/Comparison_of_instant_messaging_protocols*

[13] Wikipedia. Active Instant Messaging client users. *http://en.wikipedia.org/wiki/Instant_messaging*

[14] Windows Live Blog. 330 million MSN users. (2006) *http://messengersays.spaces.live.com/Blog/cns!5B410F7FD930829E!73591.entry?sa=240631081*