Anonymous Broadcast Messages

Dragan Lazic
School of Computer Science
University of Guelph
Guelph, ON, CANADA

Charlie Obimbo
School of Computer Science
University of Guelph
Guelph, ON, CANADA

***Abstract*— The Dining Cryptographer network (or DC-net) is a privacy preserving communication protocol devised by David Chaum for anonymous message publication. A very attractive feature of DC-nets is the strength of its security, which is inherent in the protocol and is not dependent on other schemes, like encryption. Unfortunately the DC-net protocol has a level of complexity that causes it to suffer from exceptional communication overhead and implementation difficulty that precludes its use in many real-world use-cases. We have designed and created a DC-net implementation that uses a pure client-server model, which successfully avoids much of the complexity inherent in the DC-net protocol. We describe the theory of DC-nets and our pure client-server implementation, as well as the compromises that were made to reduce the protocol's level of complexity. Discussion centers around the details of our implementation of DC-net.**

Keywords—Dining Cryptographer network; Privacy; sender-untraceability

I. INTRODUCTION

The issue of privacy and anonymity on the Internet has become a challenging one, especially with the growing influence that the Internet has on our day-to-day social lives. With the increased use of social networking sites and mobile Internet based communication, being anonymous and maintaining privacy has becoming something that the average computer has great difficulty in achieving. Preserving the anonymity of the communicating parties is crucial in situations where knowledge of the identity of the source of communicated messages could create a conflict of interest, jeopardize the integrity of a process or endanger the participants. The concept of maintaining anonymity is simple to conceive. However, they can be rather difficult to implement. The trails left by the protocols and technologies involved in digital communication can be difficult to erase or hide to a point where the identity of the communicating parties is not exposed. The trails created in these communications are often required for the communication itself often to maintain a quality of service or integrity of the message being communicated. This paper describes a digital, computer based form of communication that preserves the anonymity of all communicating parties. The program takes heavy influence from David Chaum's Dining Cryptographers problem and the DC Net concept.

A. Background

According to Nissenbaum [1], online anonymity is "unreachability", i.e. the inability of communication, or action of an individual to be traced to a specific person at a specific address in the real world.

A precise, mathematical definition of anonymity has been elusive [2][3]. Uncommon attributes of an individual may still be used to "fish-out" a person's identity, even though one may not know their name, phone number, or address explicitly from a database [4][5][6][7].

Online anonymity has its pros and cons in the society. It can provide a means for free speech and criticism of established power without fear of reprisal [8]. An example of this is when it was used in the media of communication in the North Africa, Middle-East uprising [9][10][11]. The essence of anonymity – and the need to assure deterrence from repercussions created the need for the setting up of a system that protected message conveyors from being identified in the quasi-changed political systems. However, online anonymity can also have detrimental effects in the society. Examples of this include anonymous hacking [12], and communication by terrorists [13].

The anonymous communication method described in this paper is based upon the Dining Cryptographers problem. The Dining Cryptographers problem was first proposed by David Chaum [14][15] in 1988. David describes a thought experiment and proposes a solution, which he develops into a theoretical Dining Cryptographers Protocol (AKA. DC-net) that can be used for broadcasting of unconditional anonymous messages (Chaum 1988). Prior to the development of the DC-net protocol, Chaum developed the concept of multiparty-secure sender-untraceability protocol' which he called the mixed-net protocol. The mixed-net protocol idea was then used and actually implemented in the onion routing protocol of TOR.

To illustrate the theory behind the DC-net the story of the dining cryptographers is often used. The original Dining Cryptographers problem begins with three cryptographers having dinner together at a restaurant. Their waiter informs them that arrangements had been made for the bill to be paid anonymously. One of the three cryptographers might be paying for the bill, or it could be the U.S. National Security Agency (NSA). The three cryptographers want to respect each other's right to privacy but they would also like to know if the NSA covered the bill. They devise their plan, while hiding behind their menus they each flip a coin so that only the person sitting on their right can see. Then they say aloud if the face of the coin they flipped coincides with the face of the coin flipped by the person to their left.

An odd number of differences uttered by the cryptographers would indicate that one of them paid (assuming that the bill was paid once). Yet the payer remains anonymous to the rest of the Diners in that case.

For simplicity, the truth table below indicates the results of the sums of the differences uttered as a result of the comparison of the coin-toss, had none of them paid. It is easy to verify that the above statement is correct.

TABLE I. RESULTS OF THE SUMS OF THE DIFFERENCES UTTERED AS A RESULT OF THE COMPARISON OF THE COIN-TOSS

D1	D2	D3	Differences of Utterances
T	T	T	O
T	T	H	2
T	H	T	2
T	H	H	2
H	T	T	2
H	T	H	2
H	H	T	2
H	H	H	0

1) Peer-to-Peer

With peer-to-peer the clients all make connections to each other in a ring design. For each bit that is transmitted, the results from the first stage of the DC-net protocol are combined by sending them around the ring network. Each user sends their result to the "next" user in a previous determined direction on the ring network. Now each user has 2 stage-one result bits and XORs them. Then they pass the XOR result onto the next user. This process repeats until the results have Figure 1: Dinning Cryptographers

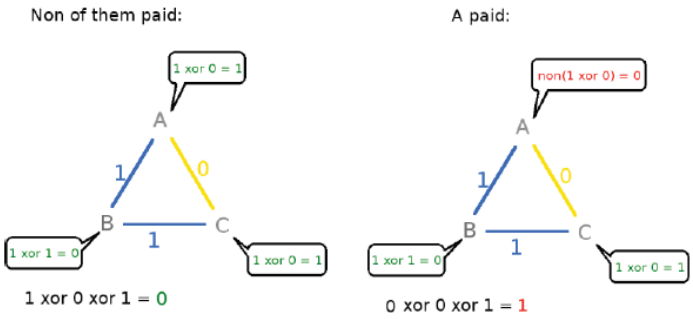

Fig. 1. Dinning Cryptographers

made an entire circuit of the ring (equal in length to the number of participants). Now every member of the ring should have the stage-two result and the entire process repeats for the next bit. The obvious problem with this implementation the overhead of waiting for each client to process then send and also the restructuring that would have to happen every time a new client joins the ring. Figure 2 shows a basic layout of Peer-to-Peer connection.

2) Hybrid Client-server

In this implementation the users send their results to a server, which XORs them and sends back the results. However the pair-wise shared secrets between clients is still communicated in a peer-to-peer manner with direct connections between peers.

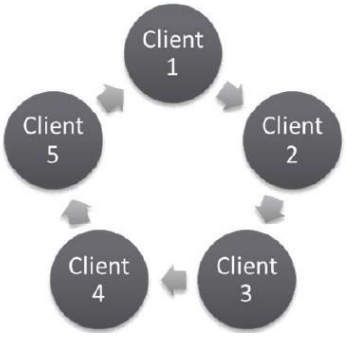

Fig. 2. Peer-to-Peer Model

The hybrid client-server implementation has the advantage of communication efficiency compared to the peer-to-peer implementation. The communication overhead for each client is drastically reduced since a circuit of the ring network doesn't need to be made in order to broadcast messages. The primary disadvantage of this implementation is the necessity of a server in addition to the existing peer-to-peer connections, which increases complexity. Figure 3 shows a basic layout of the Hybrid Client-server connection.

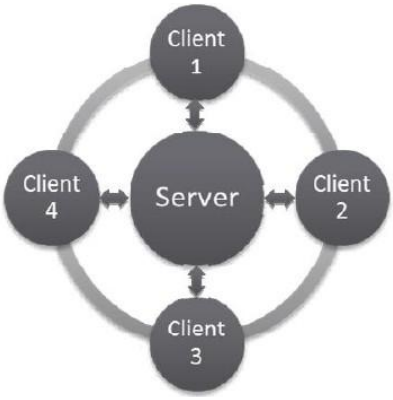

Fig. 3. Hybrid Client Server Model

II. LIMITATIONS

The DC-net protocol is straightforward and elegant in theory. It's also highly secure. However, it has several limitations, which may make it undesirable in certain use-cases. The limitations of the DC-net protocol can be generally grouped into three areas: collisions, disruption and complexity.

A. Collisions

Due to the nature of the DC-net protocol only one byte can be processed at a time. Otherwise if multiple clients send a message the XORed result on the server at the end would be unreadable text. To mitigate this problem, a system of start and end messages will be used. Before a clients message is sent to a server, a specially selected start character is sent before the message. This start character means that the server will then know to expect a message and that the end of the message will be followed with an end-message character. During that time it send a message to the clients notifying them that a message is being received and to not send.

Another similar collision issue is a race condition for sending the start message. To address this a check was put in place, if the XORed result on the server side came out to be anything other then a 0 or a start message the server would then not print the message but instead wait until it was XORed zeroes again and then sent a broadcast notifying of a collision and asking the clients to send again.

B. Complexity

The level of complexity and overhead that exists and can be added to this protocol is rather larger. The existing complexity adds to the communication overhead, which only gets worse with more connections. The clients are constantly sending data to the server, which then has to be processed by the server, even when no actual message is being broadcast. Also if you were to choose to encrypt the connection that would add additional complexity and overhead for both the client and server and decrease performance further. All these overhead results decrease the performance of the server, and as more clients connect performance continues to drop. However this is a necessary limitation and without it, their would be no anonymity.

C. Integrity

The DC-net protocol has no way of checking the integrity of the clients or the server. Essentially this allows anyone to do as they please within certain confines. For example a rouge server can be hosted that works at identifying the clients that connect and broadcast, or a client might be capable of jamming someone or the server from broadcasting. Adding additional checks in place to prevent this kind of action however could result in the loss or the risk of loosing anonymity.

III. Implementation

The implementation deed 24asone in this project is different from the ones outlined earlier [16]. The Client-Server scenario detailed in the previous section required too much communication overhead [17], and the peer-to-peer scenario was even worse. In the archetypical DC-net protocol model discussed previously, both stages of the DC-net protocol are performed for each single bit of data transmitted. This was determined to be extremely wasteful. To lower communication overhead of the DC-net protocol, random number generators were introduced to replace the function of "coin flips". This allows DC-net protocol rounds to be conducted 8-bits at a time.

The server is implemented as a dedicated application that acts as a broadcast hub for the clients as well as the calculator for stage two of the DC-net protocol. The server cannot participate in the chat in the same manner as a client would, however it does send informational broadcasts when required. However, just like in the original Client-Server DC-net scenario, the server is used to keep sessions of the chat. The clients have the ability to send and receive messages. The clients handle stage one of the DC-net protocol and send the results to the server. The clients have two operating rooms, one for regular chat and another for anonymous chat. When all clients have entered into the anonymous room and indicated their readiness the anonymous mode will begin and a count down timer for their session will also start.

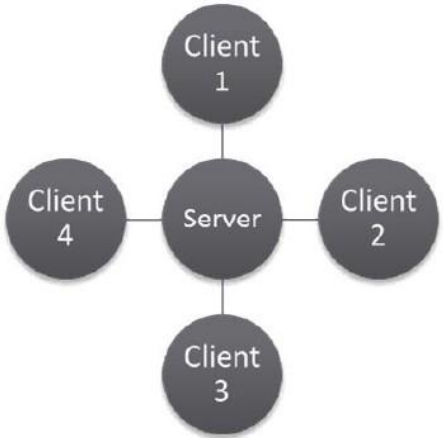

Fig. 4. The client-server connection model

A. DC-Net Implementation

Given the two-sided client-server design and the two stages involved in the DC-net, implementation was done in two stages. The first stage was to ensure that there was an operating general chat program to work of off. The bases for the chat program was found online and modification were made to insure it better fit the needs of the project [18]. The general chat program offers a simple chat interface that requires a unique nickname and the IP address of the server. Once the chat client was done adding the DC-net protocol was the next stage and the bulk of the implementation.

Once the client connects and enters the anonymous room and everyone is ready, the server generates a random seed for each client in the ring at the point of connection. This random seed is then sent to each client as well as the client to their "left". Therefore each client will have two seeds. Once anonymous mode has been activated in the room, the client uses the two seeds to generate the results of the 'coin toss'. The DC-net protocol cycle operates thusly. During stage-one, each client uses their seeded random number generators to produce two sets of bits, 1 byte each in size. These two sets are XORed to produce a single stage-one-result byte. The clients then immediately send the stage-one-result byte to the server using the primary communication channel (implemented by a TCP byte stream).

The server logs the stage-one results as they are received but performs no further action until all stage-one result bytes have been received from every client. Once all the results have been logged from every participating client, the server XORs all the results in the log, producing the stage-two result. Finally the server broadcasts the stage-two result to all clients using the secondary communication channel (implemented by TCP text stream). When the clients receive the stage-two result, a new round of the DC-net protocol is triggered.

B. Client Classes

The client is a stand-alone program consisting of several classes.

1) ClientRunner.java and ChatClient.java

ClientRunner is a very simple and small class. It essentially is used to start the client. ChatClient, is the class containing all

of the code for the GUI. This class also handles creating and executing the socket connection for the basic chat client. Besides the methods used for the creation and handling of the GUI, the class also has two other methods, one that creates a byte stream over the socket connection to receive data and another that does the same except to send data. Figure 5 shows the GUI design for the normal chat room while Figure 6 shows the design for the anonymous chat room.

2) ServerReader.java

The class starts an infinite loop and, using the receiver method created in the ChatClient class, it listens on the read byte stream for any data sent in by the server. Once the class finds data on the stream, it will act in accordance to that data.

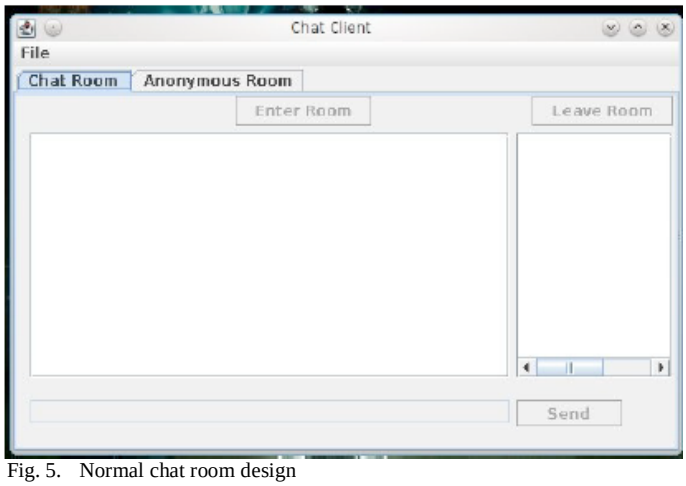

Fig. 5. Normal chat room design

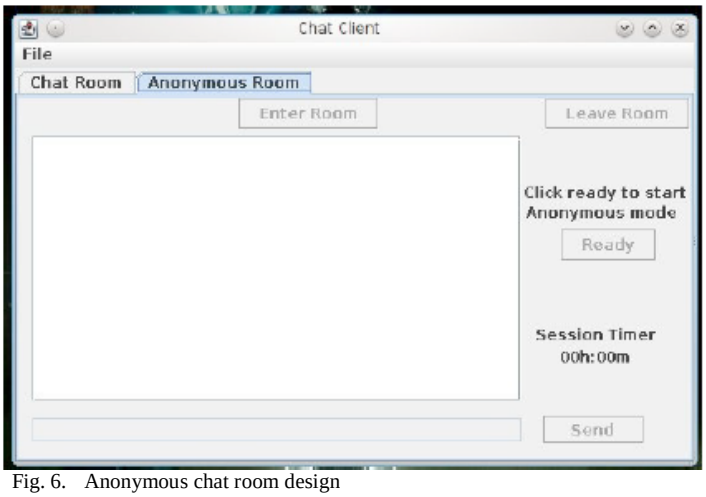

Fig. 6. Anonymous chat room design

This class essentially acts as a communication hub for the client to the server. Should the server need to send anything relevant to the client it will be received and handled by this class. If the server needs to send notice of a collision or that a message is being received then this class will receive the message and print it in the client chat window.

3) ByteSender.java

When anonymous mode is activated this class is triggered and starts an infinite loop. It uses the seed given to the client by the server and also by the other client, to generate the pseudo-random 'coin tosses'. These coin tosses will be represented by a string of numbers. Each number's binary value will equal 8 coin tosses. Once the coin tosses have been generated they are XORed. The class then makes a check against a Boolean variable to see if the user has made an inputted a message in the chat client. If the user has inputted a message it will start reading the bytes of the message and XORing them against the results of the XORed generated coin tosses. If the user doesn't input a message then only the XORed results of the coin tossed will be sent to the server. This procedure continues until the client receives a signal to stop

C. Server Clases

1) Server.java

The Server class creates many of the foundation objects for the DC-net protocol and for the chat environment in general. The class creates the vectors collection that contains the socket connection for each client, the socket objects for both the client and the server and the generator for the pseudorandom seed to be used by the client.

The class starts an on-going loop as it waits for an incoming connection from the clients. Once a connection is established it creates a user land thread for the client that made the connection. The thread is then added to the appropriate vector object in the collection. The class then creates an instance of another class, CThread, and assigns it to the recently added client thread. The Server class continues this process for every client connection it makes.

2) CThread.java

The CThread class creates the required objects and posses the needed methods to cover all the actions executed by the client. It begins by calling the input and output streams created as a result of the clients connection to the server. It then starts an infinite loop, checking the IO streams for traffic and reacting appropriately to any traffic. The class also acts as the main point of communication between the client and the server on the server side.

All messages sent by the client, whether they are meant for the server or for broadcast will come down to and be handled by CThread. The class will also handle any operations that the server must make due to the client. For example, if the client moves rooms from the general chat to the anonymous room or if they client gets put on a wait list. The class will check if the clients nickname is unique, and if it meets the requirements to enter anonymous mode. If the client cannot enter anonymous mode CThread will notify the client why. Overall the CThread class handles a lot of the smaller tasks related to the Client-Server interaction.

3) ByteReader.java/SeedSender.java/SessionTimer.java

SeedSender, will parse through the vectors collection and send each active client object in the vector their appropriate seeds. SeedSender will also notify the clients that the server is ready once all the seeds have been distributed. Afterwards SeedSender will call the ByteReader class and terminate itself. Once called, ByteReader will immediately call the SessionTimer class to start tracking the session time. An infinite loop is then created, and the class calls a method in CThread of each client to fetch a byte of data coming in from

the client. This will essentially be the XORed results that the clients are continually sending to the server. The fetched byte will be stored in a byte array, using one element for each client. It then takes the bytes and XORs them together as it should for a stage two process. It will also convert the XORed result to a string and check the character result. If the result is a start message then the server will know a message is about to be sent. Otherwise it should result in a zero if there was no collision.

When SessionTimer is called it will set the timer for every active client to 15min. This will be the default duration of the anonymous session. As it counts down, at every minute interval it will update the timer for each client. Once the timer has hit the 0 mark, it will check the waiting list for the anonymous room and move over any clients on the waiting list into the room and reset itself.

IV. RESULTS

The implementation of the DC-net protocol created for this project is a functional implementation. It operates in as described by the DC-net protocol and performs both stages of the DC-net protocol for each bit of data sent. However it is not a complete or direct implementation of the protocol as it eliminates all the peer-to-peer communications. The implementation is also relatively practical. It offers both standard user-attributable messaging as well as anonymous messaging. By eliminating all peer-to-peer communication and centralizing the entire DC-net protocol to operate through a server, I have also significantly reduced the communication overhead associated with the DC-net protocol. The pure client-server implementation developed for this project also largely solves the problem of complexity in the DC-net protocol. With all communications centralized at the server, there is reduced overhead and a reasonable number of network connections.

A. Known Issues

The program is not perfect and therefore has some unresolved bugs.

- Leaving the room while waiting for anonymous mode to initiate will not be recognized by the server
- Server notice of incoming message will be attached to a client message if connection speeds are very fast. Example: running on a Local host or a fast LAN.

This is just a list of what was found in test.

V. CONCLUSION

There are cases where privacy preservation is important even in instances where communication is taking place. Examples of this include online-surveys. The Dining Cryptographer network (DC net) was devised by David Chaum for anonymous message publication. This is an elegant and straightforward protocol in theory.

However, it has three main drawbacks, Collisions, Complexity and Integrity. Collisions: according to the protocol, only one byte can be processed at a time, multiple clients sending a message would result in a futile attempt at XORing.

A. *The DC net is not easily scalable and its performance quickly deteroriates when large numbers of clients are added, this is refered to as the Complexity problem in this paper. Lastly DC net protocol has no way of checking the integrity of the clients or the server, thus it has problems with its integrity.*

This paper presents shows how these issues can be resolved, and the research provides an implementation of the DC-net protocol that is practical to deploy and. The application represents a proof of concept for a pure client-server implementation of the DC-net protocol, which avoids the complexity problem found in the implementation scenarios.

VI. FUTURE WORK

Numerous improvements to the implementation can be made with regard to security. To increase the security of the protocol and reduce the chance that a malicious third-party can intercept information with which to compromise the security of the protocol, the primary communication channel used specifically for the DC-net protocol can be encrypted. Currently the server has no ability to police the clients on the server.

Functionality for administration of the server should be added to a production quality implementation to allow the administrators to remove and ban users, among other possible functions. Further improvements to add production quality to the server would be the development of a GUI for the server. A malicious disruption detection method should be implemented to detect users who use customized clients designed for disrupting the DC-net communication on the server. Once identified, these users can be forcibly ejected from the server and their IP address may be banned.

REFERENCES

[1] Nissenbaum, H. 1999. The Meaning of Anonymity in an Information Age. The Information Society , 15:141-144

[2] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. 2007. l-diversity: Privacy beyond K-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) , 1 (1), 3.

[3] Li, N., T. Li, and Venkatasubramanian, S. 2007. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. International Conference on Data Engineering (ICDE) , pp.106-115.

[4] R. Collier, C. Fobel, L. Richards, and G. Grewal A Formal and Empirical Analysis of Recombination for Genetic Algorithm Based Approaches to the FPGA Placement Problem IEEE Canadian Conference on Electrical and Computer Engineering, Montreal, Canada, 2012.

[5] 5. Ohm, P. 2010. Broken Promises of Privacy: Responding to the Surprising Failure of Anonymization, UCLA LAW REVIEW 57 p. 1701 - 1777. Available from: http://papers.ssrn.com/sol3/papers.cfm ?abstract_id=1450006. Retrieved October 1, 2013.

[6] Sweeney, L. 2002. 'K-anonymity: A Model for Protecting Privacy.' International Journal of Uncertainty, Fuzziness, and Knowledge-based Systems 10(5): 557-570.

[7] Arvind, Narayanan and Vitaly Shmatikov. 2008. Robust De-Anonymization of Large Sparse Datasets, in Proc. of the 2008 IEEE Symp. On Security and Privacy 111 -121.

[8] Akdeniz, Y. 2002. Anonymity, Democracy, and Cyberspace. Social Researc h, 69(11), 223-237.

[9] Sharp J. M., Specialist in Middle Eastern Affairs. Egypt: Background and U.S. Relations. June, 2013. http://www.fas.org/sgp/crs/mideast/RL33003.pdf. Retrieved: October 1, 2013.

[10] Ryan, Yasmine (26 January 2011). "How Tunisia's revolution began – Features". Al Jazeera. Retrieved 13 February 2011. http://www.aljazeera.com/indepth/features/2011/01/2011126121815985483.html. Retrieved: October 1, 2013.

[11] Heydemann S. Syria's Uprising: sectarianism, regionalization, and state the Levant. Fride and Hivos, 2013. http://www.fride.org/download /WP_119_ Syria_Uprising.pdf. Retrieved: October 1, 2013.

[12] Kelly, Brian (2012). "Investing in a Centralized Cybersecurity Infrastructure: Why 'Hacktivism' can and should influence cybersecurity reform". Boston University Law Review 92 (5): 1663–1710. http://www.bu.edu/law/central/jd/organizations/journals/bulr/volume92n4/documents/KELLY.pdf. Retrieved October 1, 2013.

[13] Hancock, Jeffrey T. and Beaver, David I. and Chung, Cindy K. and Frazee, Joey and Pennebaker, James W. and Graesser, Art and Cai, Zhiqiang. Behavioral Sciences of Terrorism and Political Aggression, 2010 vol 2:2 pp 108 - 132. Retrieved October 1, 2013.

[14] Chaum, David. "The Dining Cryptographers Problem: Unconditional Sender and RecipientUntraceability." Journal of Cryptology, 1988: 65-75.

[15] Chaum, David. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. "Communications of the ACM, 1981: 84-88.

[16] S0cketka, (2011, May 30). Dining cryptographers problem [online]. Available: http://en.wikipedia.org/wiki/Dining_cryptographers_problem

[17] Oracle's Java Tutorials, Lesson: All About Sockets [online]. Available: http://download.oracle.com/javase/tutorial/networking/sockets/index.html.

[18] Simple Java Chat Client-Server ,(2008, January 14). Client-server chat download[online].Available: http://breakdesign.blogspot.com/2008/01/simple-java-chat-client-server.html