

5-18-2007

Enhancing Crowds to Support Truly Anonymous FTP Transactions

Paul Flowers
University of New Orleans

Follow this and additional works at: <https://scholarworks.uno.edu/td>

Recommended Citation

Flowers, Paul, "Enhancing Crowds to Support Truly Anonymous FTP Transactions" (2007). *University of New Orleans Theses and Dissertations*. 530.
<https://scholarworks.uno.edu/td/530>

This Thesis is protected by copyright and/or related rights. It has been brought to you by ScholarWorks@UNO with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in University of New Orleans Theses and Dissertations by an authorized administrator of ScholarWorks@UNO. For more information, please contact scholarworks@uno.edu.

Enhancing Crowds to support truly anonymous FTP transactions

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
In
The Department of Computer Science

by

Paul Flowers

B.S. University of Memphis, 1980

May 2007

Dedication

In dedication to my father and in memory of my mother

To my loving wife, Nonia, always and forever

Acknowledgements

First of all, I thank Him who is above all.

I am grateful to Dr. Golden G. Richard III for his patience, understanding and guidance in this endeavor.

I would also like to acknowledge the other members of my thesis committee Dr. Shengru Tu and Dr. Vassil Roussev.

Table of Contents

List of Figures	vi
List of Tables	viii
Abstract	ix
Chapter 1: Introduction	1
1.1 Approaches to anonymous web transactions.....	2
1.1.1 Proxies	2
1.1.1.1 Anonymizer	2
1.1.1.2 Lucent personalized Web Assistant	3
1.1.2 Onion Routing	4
1.1.3 Crowds	6
1.2 Capabilities of the different systems	7
1.3 Objectives	8
Chapter 2: Crowds Background	9
2.1 Elements of the Crowds Implementation	9
2.1.1 Crowds Communication Layer	9
2.1.2 Crowds primary protocol	11
2.1.3 Encryption Schemes	14
2.1.4 The centralized server – blender	15
2.1.5 The proxy – jondo	16
2.2 The flow of Crowds	17
2.3 Goals and security considerations	22
2.3.1 Types of anonymity	22
2.3.2 Types of attackers	22
2.3.3 The degrees of anonymity	23
2.3.4 Anonymity properties provided by Crowds	24
2.4 Empirical data measuring performance by the implementers	26
Chapter 3: Jondos Configuration Menus	29
Chapter 4: Anonymous FTP and Crowds	32
4.1 File Transfer Protocol	32
4.2 File Transfer Example	34
4.3 File Transfer Protocol Client Design	36
4.3.1 User Interface	37
4.3.1.1 Local Interface	37
4.3.1.2 Remote Interface	38
4.3.1.3 ASCII/Binary	38
4.3.2 Client Protocol Interpreter	38
4.3.2.1 Connect	38
4.3.2.2 Handle Incoming Message	39
4.3.2.3 Command Interpreter	39
4.3.3 Client Data Transfer Unit	39
4.3.3.1 Incoming Data Message	39
4.3.3.2 Client Data Transfer	40
4.3.3.3 Saving File	40
4.3.4 Commands Included in the FTP Client Design	40
4.4 FTP Server's Responses	42

4.5 Modifications to the Crowds Implementation	42
4.5.1 HTTP Modifications	42
4.5.1.1 Filtering unwanted HTTP headers	42
4.5.2 FTP Modifications	43
4.5.2.1 Data and Control Connections	43
4.5.2.2 Connection to HTTP server or FTP server	44
4.5.2.3 Visual Display Modifications	44
4.6 Caveats relative to the Crowds implementation	44
4.7 Empirical Data measuring performance for the FTP protocol	45
Chapter 5: Visual display of the <i>jondos</i> path	47
Chapter 6: Conclusion	49
Chapter 7: Future Work	50
Bibliography	51
Vita	52

List of Figures

Figure 1: A typical proxy system	3
Figure 2: Lucent Personalized Web Assistant	4
Figure 3: Onion Routing	5
Figure 4: Paths in a Crowd	7
Figure 5: Communication Layer	9
Figure 6: Communication Layer	10
Figure 7: The Select Mode in the Event Loop	11
Figure 8: HTTP transactions	13
Figure 9: The Block Cipher	14
Figure 10: The Stream Cipher	15
Figure 11: New joiner connection to blender	17
Figure 12: Blender sends messages to current members	18
Figure 13: Commit signals sent to all <i>jondos</i>	19
Figure 14: Display the new joiner message	19
Figure 15: Display joining message	20
Figure 16: Degrees of Anonymity	23
Figure 17: Performance Graph without embedded images	27
Figure 18: Performance Graph without embedded images	28
Figure 19: Performance Graph with embedded images	28
Figure 20: Configuration Menus – Main Window	30
Figure 21: Configuration Menu – List Window	30
Figure 22: Configuration Menu – Edit Window	30
Figure 23: Configuration Menu – Help Menu	31
Figure 24: FTP components	33
Figure 25: FTP Transfer Example	35

List of Figures

Figure 26: FTP Client	36
Figure 27: FTP Client User Interface	37
Figure 28: FTP Connection Dialog Box	38
Figure 29: Performance FTP Graph	46

List of Tables

Table 1: Anonymity Properties of Crowds	24
Table 2: Performance Data without embedded images	27
Table 3: Performance Data with embedded images	28
Table 4: Performance FTP Data	46

Abstract

Ensuring privacy on the Internet is one of the most daunting challenges that we presently face. Crowds is the implementation of an approach to provide privacy to web transactions. The system's strategy is to seek concealment through numbers. A crowd consists of a collection of users that intend to participate in web exchanges; each user being represented by a process on their machine called a jondo. The jondo either submits the request to the server or forwards it to another jondo. The randomly achieved sequence of *jondos* that traverse the distance from the initiator to the server provides degrees of anonymity. The present version of Crowds employs HTTP as its sole protocol to secure anonymity with the exception of embedded protocols. It is the intention of this thesis to extend this system's capability by adding the FTP protocol to its cache of viable protocols traversing the Crowd's implementation.

Chapter 1: Introduction

In the March 2001 issue of PC World the following account was given: In 1994, a loan officer for a bank who also served on his county's health board discovered that he had free access to the patient records of people who lived in his county. He cross-referenced the names from his customer databases with the names of people who had been diagnosed with terminal illnesses. The banker called due the loans of dozens of people who had been diagnosed with cancer.

In 1998 an employee of Motorola received startling news at work: She had been automatically enrolled in an anti depression counseling program by the company's human resources department. Eventually, the company admitted that they had received a list of drugs her mail-order prescription company was sending her on a regular basis. Included on the list was a prescription antidepressant.

Data collection is ubiquitous in today's society and it seems that every transaction we participate in is recorded. The profiles and dossiers compiled using the influx of analytical tools available is certainly evident relative to the Internet. Currently, Internet communications do not conform to established practices, since traditionally the invasion of privacy relative to mail and telephone communications are prohibited without judicial intervention. There is certain information the corporate and consumer worlds deem confidential and do not want shared. Certain financial information, private medical and health concerns, participation in support groups pertaining to chemical dependency problems, and corporate trade secrets are just some areas in which concealment is desired. The preservation of privacy involves the concealment of message content and inhibiting the detection of the sender and associated receiver. Arguments concerning Internet privacy will be entertained for many years to come, but most people would agree with one point, however; total lack of privacy is not something desired. In light of commercial concerns, consumers' privacy, the nation's security, and many other variables, a balance between an individual's privacy and the preservation of our society should be sought.

The following scenario should illustrate that encryption of a message will not suffice for every circumstance in which privacy is sought. SSH, PGP and other available software technologies may not achieve the degree of privacy desired. An individual discovers unethical or even criminal activity at his place of employment. Unable to ignore what he has seen for his job sake and embrace some form of rationalization to justify this ignorance, he decides to act. He sends the appropriate authorities an e-mail thinking his identity would be safe since the message was encrypted. The local administrator, colluding with the perpetrators, sees that he has sent an email to unethical.com. Even though the message was encrypted, sender or receiver anonymity is preferred. The employee would be safe if the administrator did not know which employee sent the message (sender anonymity) or if the administrator did not know who was the recipient of the message (receiver anonymity).

1.1 Approaches to anonymous web transactions

1.1.1 Proxies

1.1.1.1 Anonymizer

There are currently two basic approaches for achieving anonymous web transactions. One widely known method is to insert a proxy between the initiator and the receiver of the message, thus; concealing the identity of the sender. Web pages are retrieved by the proxy, acting as an intermediary, and forwarded to the sender. The IP address of the proxy is revealed to the Web site and the address of the sender is secure. In some cases the content can also be encrypted between the host and the Web site. An area of concern relative to the anonymizing agent is that anonymizer has to be a trusted third party, since the sender's address is clearly seen by the anonymizer and the user's actions can be logged by the agent's own ISPs.

Remember all connections are logged, the Web site, the proxy, and your own ISP, therefore; traceable if the proxy reveals its data. Any passive attacker who compromises the agent has the ability to monitor and record all communication between the sender and receiver.

The proxy also is a single point of failure, all private communication ceases if the proxy discontinues functioning properly. The well known proxy 'Anonymizer' [1] (ref. Figure 1) applies this technique.

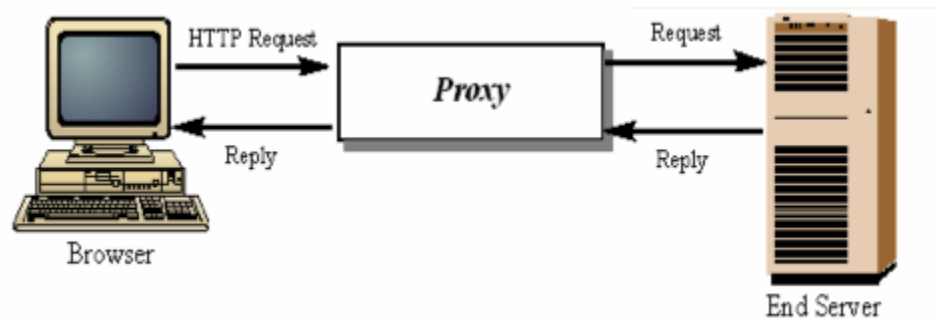


Figure 1: A typical proxy system

1.1.1.2 Lucent Personalized Web Assistant

The Lucent Personalized Web Assistant [2] is also a HTTP proxy (ref. Figure 2) with a desirable twist; it provides an effective method for generating aliases and other features for Web services while securing your personal information. Many Web sites offer newsletters, discounts, confirmation codes, and other personalized information in exchange for usernames, secret passwords, email addresses, and other personal information. It can be easily seen that extensive profiles of individuals can be developed who frequently peruse their sites. The Lucent Personalized Web Assistant software system consists of three components: the Persona Generator, the Browser Proxy, and the Email Forwarder. The Persona Generator requires a valid email address and arbitrary password from the user and generates an alias username, password, and email address for the Web site of interest. Aliases provided by LPWA, remain constant on all subsequent visits to that particular site. The Browser Proxy provides filtering for certain HTTP headers and serves as a proxy. The Email Forwarder routes email addressed to the alias email address to the genuine email address of the user. Once again the pseudonym agent (LPWA) has to be a trusted third party and is a single point of failure.

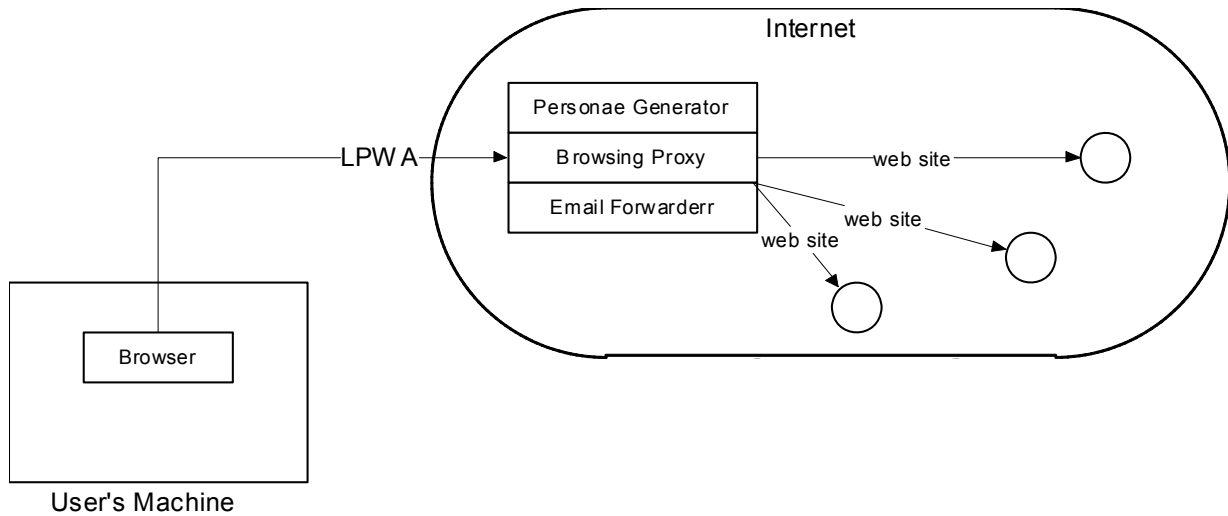


Figure 2: Lucent Personalized Web Assistant

1.1.2 Onion Routing

The second approach for realizing anonymous web transactions is to employ mixes. A mix is a store and forward device that accepts a number of fixed-length messages from numerous sources, performs cryptographic transformations on the messages, and then forwards the messages to the next destination in a random order [5]. The mix can be considered an enhanced proxy. It conceals the sender from the receiver and also provides unlinkability against a global eavesdropper. Unlinkability between sender and receiver signifies that one can not ascertain whether the sender and receiver are communicating with one another even though it can be verified they are participants in transmission of messages. Since a mix accepts a number of fixed-length messages from various sources in a fixed time interval, it would be an arduous task to track messages based on bit-pattern or size. The subsequent random submission of these messages to there next destination makes it difficult to track the messages by ordering. Routing the messages through a multitude of mixes would enhance the unlinkability between sender and receiver.

Onion Routing [3] applies this technique(mixes) to camouflage communication over the Internet. The infrastructure provided by onion routing protects the system against traffic analysis attacks, the ability to detect the sender and corresponding receiver. This system concentrates on connection anonymity, the concealment of the identity of the user by disguising the path between the sender and receiver.

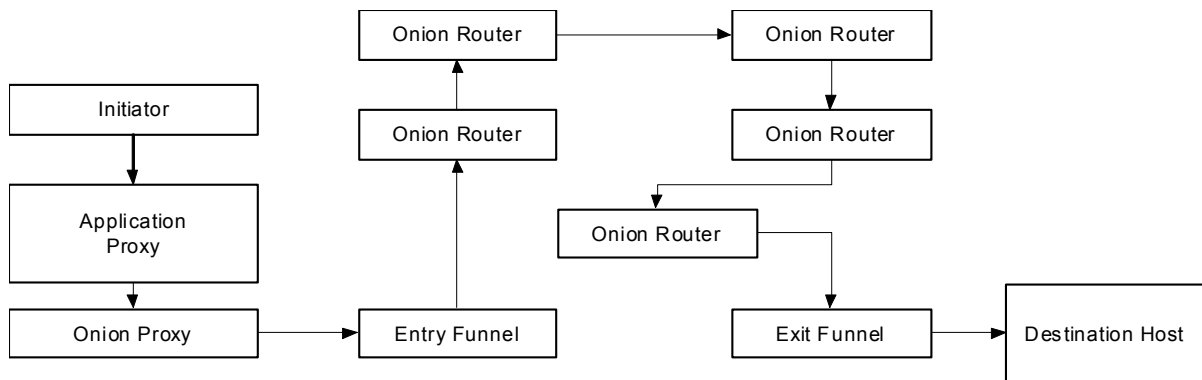


Figure 3: Onion Routing

The system consists of a number of dynamically constructed connections within a network of mixes. The network of onion routers contain routers that act as entry funnels for client connections to the network. Other routers function as exit funnels that provide the generation of TCP connections to the designated Internet services. The client application initiates an anonymous connection by establishing a link with the application proxy which accepts a protocol-specific proxy and converts the data stream into an application-independent format accepted by the onion routing network. The packets are then forwarded to an onion proxy that constructs and manages the anonymous connections. The onion proxy constructs an onion, a multiply encrypted layered structure containing routing information, and then proceeds to the entry funnel.

The funnel receives the onion, decrypts it, and determines from the information provided the next hop in the route. The decrypted layer is removed and the process is repeated until the exit funnel is encountered. At this point the packet is identical to the original message and delivered to the destination host.

1.1.3 Crowds

Crowds (ref. Figure 4) is the implementation of another approach to increase privacy during web transactions. This method employs the concept of concealment through numbers. A crowd consists of a collection of users that intend to participate in web transactions. Each user runs a process on their machine called a jondo. The jondo represents a faceless participant in the gathering that seeks anonymity in their web exchanges. The web client configures the faceless ally as its proxy in web communications. All requests submitted from the browser are sent directly to the jondo. The jondo either submits the request directly to the web server or forwards the message to another jondo in the crowd. The randomly achieved sequence of *jondos* that traverse the distance from the initiator to the server is known as the path (ref. Fig. 4). In this protocol, it can not be determined if a jondo's predecessor is the initiator of the request or just forwarding the request. Communication between any two *jondos* is encrypted using a shared key, known only to those *jondos*. Eventually the request is submitted to the server in plaintext form. All subsequent messages generated by that jondo traverse the same path until a new jondo is accepted into the crowd.

Therefore, Crowds offers another approach to maximize privacy on the web. Crowds prevents the receiver, the web server, from acquiring identifying information, even the IP address and domain. Request headers, inherent in the HTTP protocol, that reveal certain data and minimize privacy can be extracted from the messages. Crowds conceal the sites visited by the user. Crowds also takes measures to conceal this information from other collaborating *jondos*.

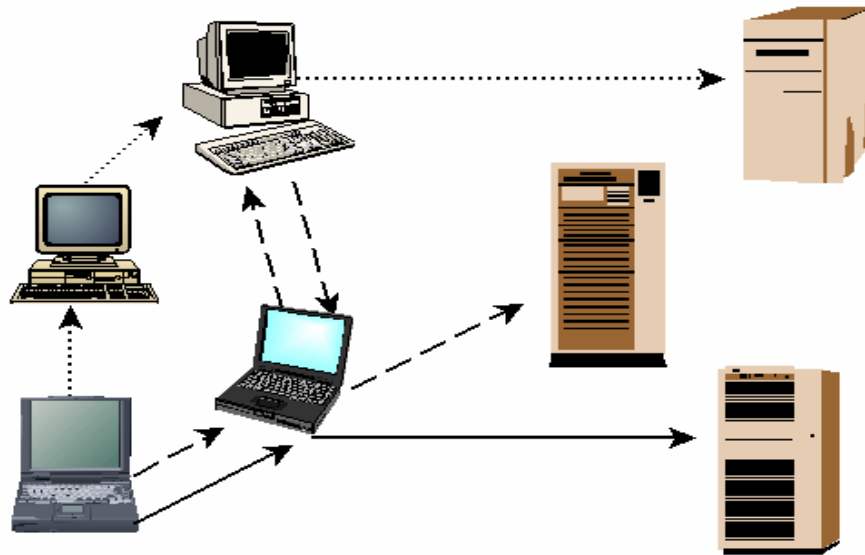


Figure 4: Paths in a crowd

1.2 Capabilities of the different systems

Each system can be categorized according to connection anonymity, data anonymity or personalization [7]. Connection anonymity conceals the user's identity by masking the path from the sender to the receiver. Data anonymity extracts information that could identify the user. Personalization provides methods for web site registration without revealing the initiator.

LPWA connection anonymity is considered low since the path has just one intermediary, the proxy. Data anonymity and personalization is high because it uses persistent aliases for user names, passwords and e-mail addresses. The Anonymizer data anonymity is considered high since it removes identifying information in the HTTP headers, extracts Java and Javascript for browsing and rewrites HTTP pages to enable it to request any embedded links that exist in the page. Its personalization is absent because of its inability to preserve data anonymity. Its connection anonymity is limited. Onion routing and Crowds possess a high degree of connection anonymity since their paths to the server are well disguised. Data anonymity and personalization were not designed into their functionality.

1.3 Objectives

My main objective is to ensure that other protocols that differ from the 'HTTP' protocol can be employed in the 'Crowds' implementation. Cryptographical operations, i.e. the operations involving the encryption schemes, must be sustained to ensure that the security analytical calculations, established by the original 'Crowds' implementation, won't be compromised. The analytical calculations of the original 'Crowds' system are based on certain encryption schemes, which if modified, would alter the security calculations, a condition to avoid minimizing complications. The 'FTP' protocol was chosen based on two striking differences between the protocols. The FTP protocol employs two parallel TCP connections to transfer a file, whereas; the HTTP protocol utilizes just one. The FTP protocol must also maintain state, whereas the HTTP protocol is stateless. The original 'Crowds' software system retrieves FTP requests only if they are embedded in a web page.

It is the intention of this thesis to compose a FTP client that can connect to a FTP server, examine the file system of the remote server, and retrieve text and binary files while traversing the *Crowds* implementation. The client will only use anonymous authentication to preserve user's anonymity, and the client will not address FTP commands that modify the server's file system. Modifications of the proxy's code will be made since they are essential to provide access to the FTP server and ensure the reception of server responses. The client will be written in Perl with the graphical components employing the Perl 'Tk' module.

Additionally, I intend to develop software that will display graphically the paths of selected *jondos* from origin to destination.

I will also generate a graphical user interface which will provide the user with the effortless ability to load and edit configuration values relevant to the 'Crowds' implementation. A brief help menu will also be available to aid in understanding these attributes and associated values. The configuration menus created to accomplish this task are: the start, list, edit, and help menus.

Chapter 2: *Crowds* Background

2.1 Elements of the *Crowds* Implementation

To appreciate the complexity and intricacies of the *Crowds* implementation one should consider its individual components: the communication layer utilized by *Crowds* for message passing, the primary communication protocol, the encryption schemes employed, the blender, and the jondo.

2.1.1 *Crowds* Communicating Layer

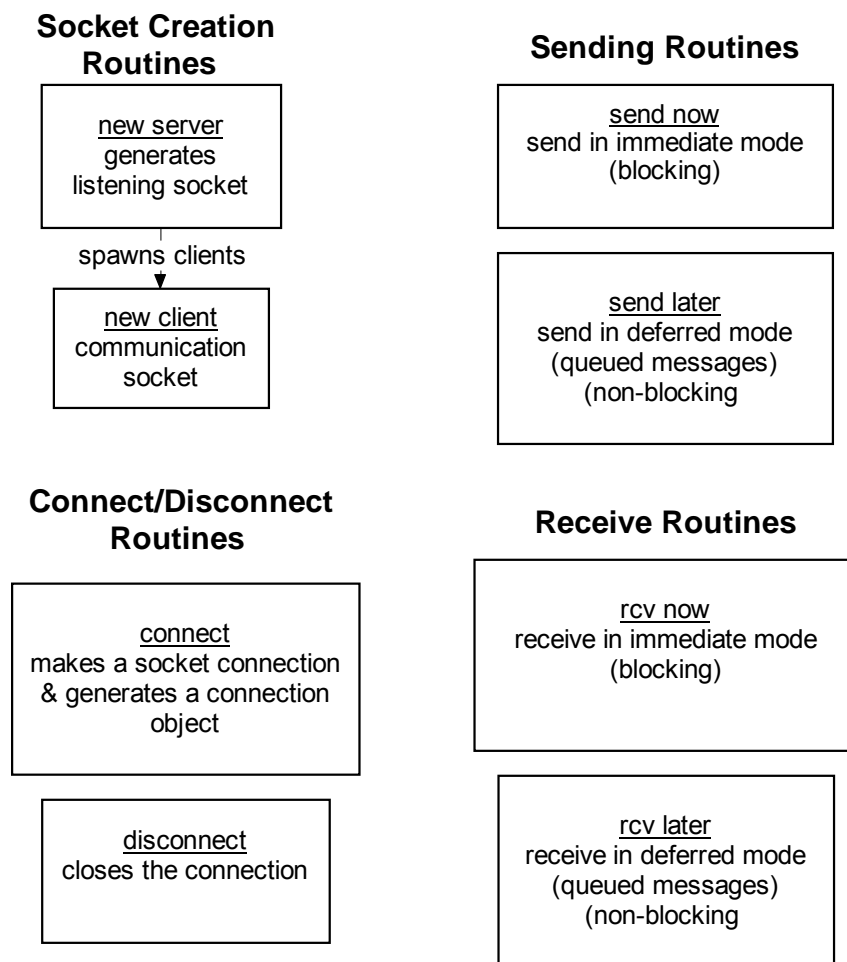


Figure 5: Communication Layer

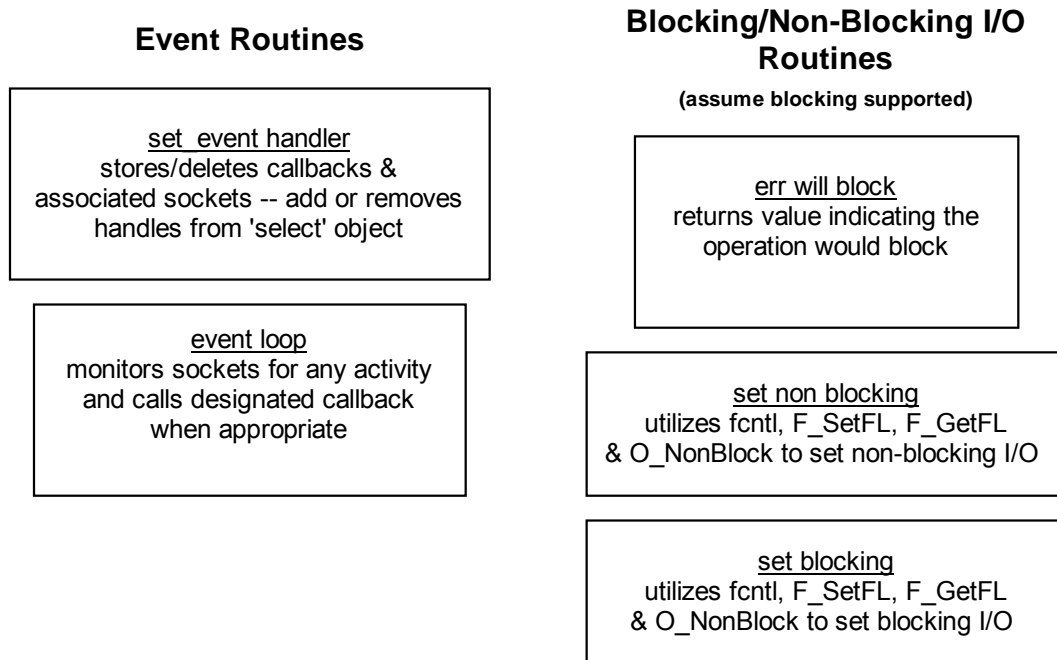


Figure 6: Communication Layer

The communication layer handles multiple clients by employing the 'Select' module and non-blocking I/O. The 'select' method in the event loop of the communication layer monitors the sockets to determine if they are ready for reading or writing, and if any activity is observed the appropriate callback (routine) is called. The 'fcntl' function is used to place the sockets into non-blocking mode. These two methods are used to avoid blocking and thus minimizing the wait time incurred by clients for access to the server.

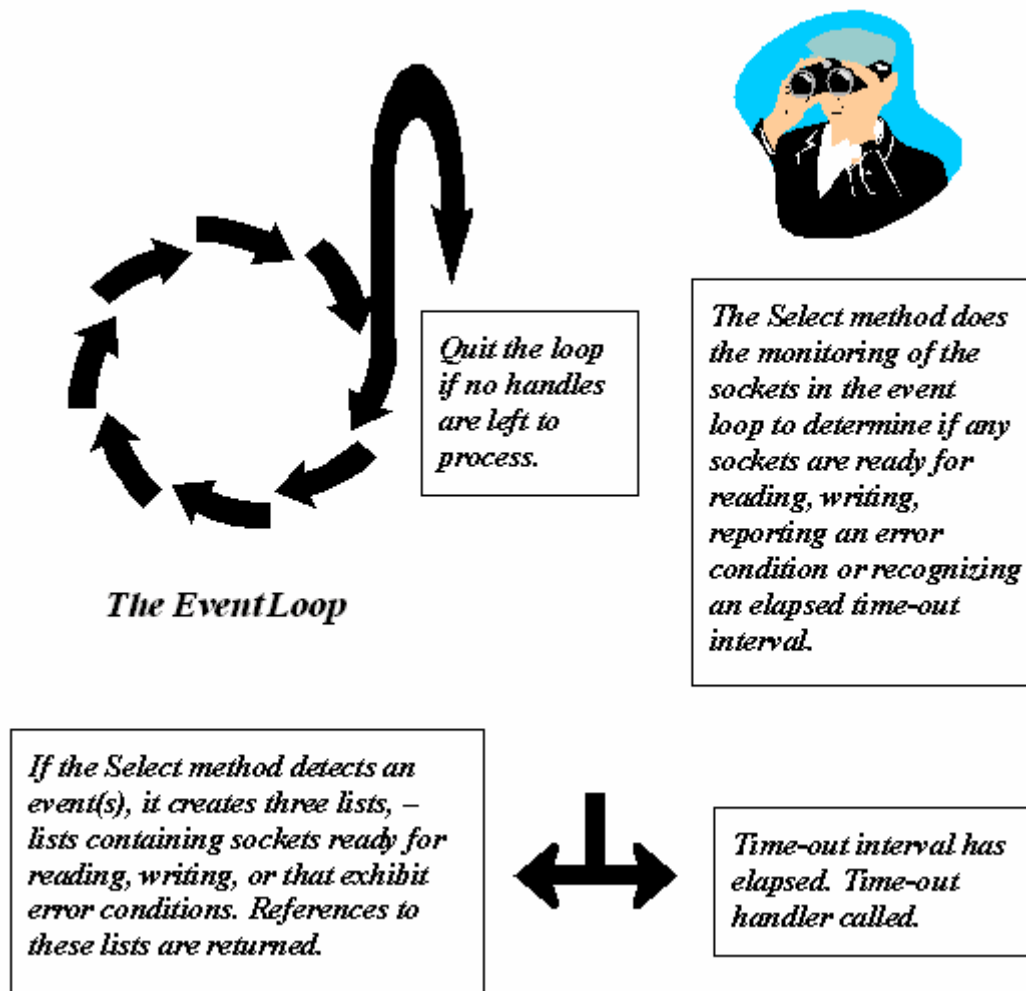


Figure 7: The Select Method in the Event Loop

2.1.2 Crowds Primary Protocol – Hypertext Transfer Protocol

HTTP is a stateless protocol in which the web server does not have to maintain any information about the client. The client initiates a request, the server responds, and the transaction is completed. The client's request message consists of a request line, headers, and sometimes a body.

Request line	Headers	Blank line	Optional body
--------------	---------	------------	---------------

The request line is subdivided into a request method, a Universal Resource Locator, and the HTTP version.

Request method	Space	URL	Space	HTTP version
----------------	-------	-----	-------	--------------

The URL contains four elements: the retrieval protocol, the server's address, server's port, and the path to the document requested. The protocol used to retrieve the data in this version of 'Crowds' is HTTP.

Retrieval protocol	:://	Server's address	:	Server's port	/	Path
--------------------	------	------------------	---	---------------	---	------

The client (browser) begins by parsing the Universal Resource Locator, and extracting the components above: the protocol to be used, the hostname of the computer to be contacted over the network, the port number, and the document path encountered to gain access to the information requested. The browser connects to the computer with the given hostname using the appropriate protocol. A default port number of 80 is applied if a port number is not specified.

A message is then sent by the browser to the server using the following format: the client's request method, the Universal Resource Indicator, the HTTP version, the headers, and if required, the body. The client's request method is a command elicited by the client to the server indicating the objective the client wishes to achieve. The method 'GET' is the primary method employed in this version of 'Crowds'. Other methods available in HTTP version 1.0 are HEAD, POST, PUT, DELETE, TRACE, etc. The 'GET' method's objective is to retrieve a document from a specified location on the server.

The following is an example of the 'GET' method: GET /example.html HTTP/1.0. The 'GET' method's function in this case is to retrieve the document example.html at path '/' using version 1.0 of the HTTP protocol. After this information has been sent, the headers can be sent to the server next. This optional information presented in the message pertains to the client's configuration and types of documents that the client prefers to handle.

The headers are subdivided into four categories: general headers, request headers, response headers, or entity headers. The general headers can exist in client requests or server responses. The request headers convey to the server the client's configuration and the formats of the preferred documents.

The response headers disclose information about the server and how it deals with requests. The entity headers employed in both client requests and server responses provides information about the entity body in the HTTP message. The entity body, which comprises the data portion of the message, is sent after the headers. This division is not always applicable, but is used to supply data when needed, such as in response to CGI programs.

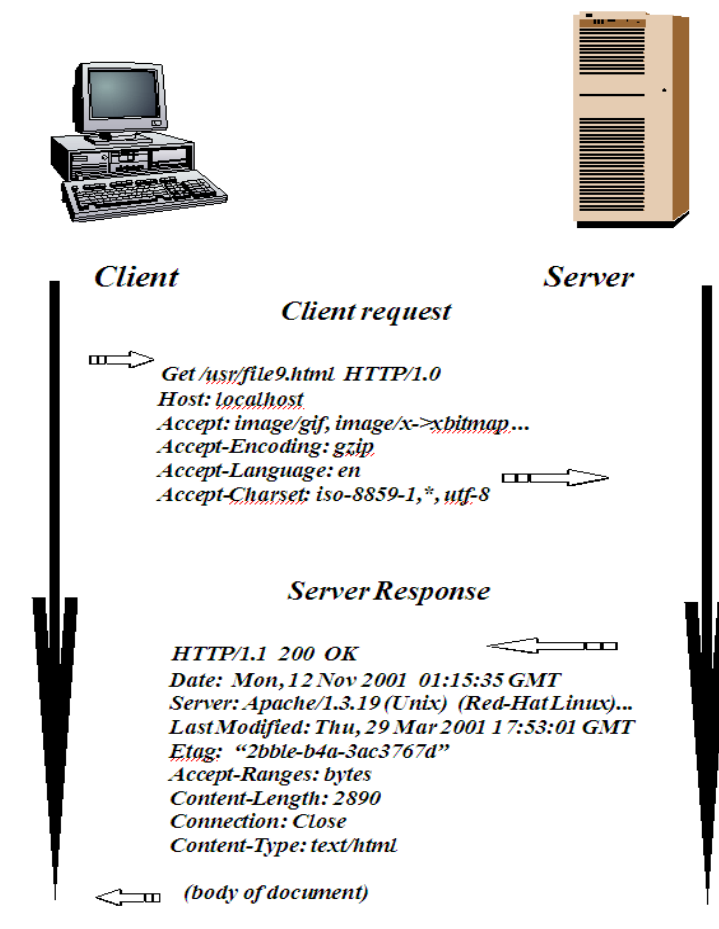


Figure 8: HTTP transactions

The server responds utilizing the following format: the HTTP version, status code, status description, headers, and entity body. The following is an example of the first line of the server's response: HTTP/1.0 200 OK. This line indicates the version of the HTTP protocol the server is using. It also conveys to the client that the request made was successful and the data requested will be sent after the header information. Next the server sends the necessary header information and lastly the data requested.

2.1.3 Encryption Schemes

The *Crowds* implementation employs symmetric key-based algorithms: both block and stream algorithms. Block cipher routines are used in *Crowds* to generate a block cipher object and the associated encrypting and decrypting routines in the following instances:

- 1) Generation and verification of the blender's administration cipher using the blender's administration passphrase as the key providing access to the blender to manage administrative tasks.
- 2) The block cipher is also used for communication between the blender and the jondo or for encryption and decryption of requests and response keys from jondo to jondo.

The stream cipher package encompasses routines that encrypt and decrypt requests and replies pertaining to web transactions. The key to the request and response ciphers is a 128 byte string which is randomly generated. Encryption and decryption is accomplished through XORing of the plaintext.

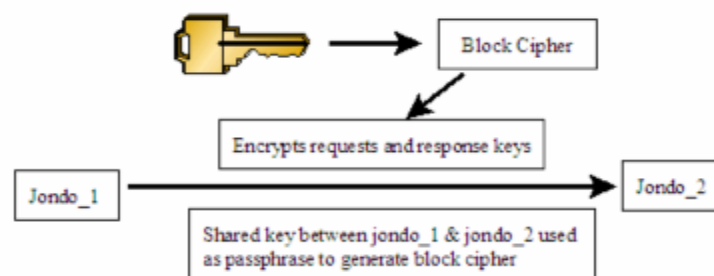


Figure 9: The Block Cipher

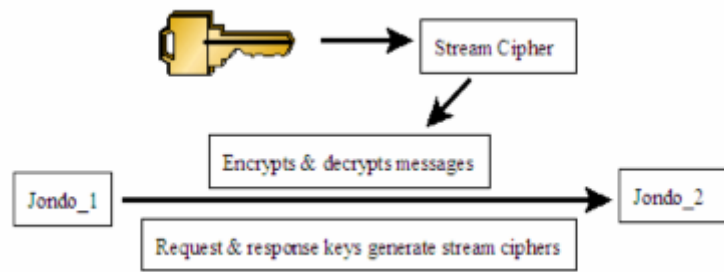


Figure 10: The Stream Cipher

2.1.4 The centralized server – Blender

A centralized server known as the blender manages crowd membership. The blender governs admittance to the crowd, the time in which access to the crowd is permitted, and the conveyance of the membership list to the other crowd members. Imposing restrictions on crowd membership is essential due to the following consideration: the propagation of a multitude of collaborators by a single attacker which reduces the ability of the system to maintain protection. Due to this potential difficulty the blender controls access by authentication based upon a pre-established account containing the account name and corresponding password of each potential user.

The design of the software system requires that all of the jondo paths to the server are static. The impetus behind this decision is to preclude the successful attack of collaborating *jondos* on the system. The colluding *jondos* could increase the probability of identifying the initiating jondo by linking multiple paths to the same jondo.

Static paths serve as the motivating factor for the blender to impose a time for a jondo to access the crowd, since the correlation of any newly admitted jondo path and the new joiner can be ascertained. Hence, the blender sends out a join commit signal to notify all current members that the re-establishment of their paths from scratch is essential. The commit time is generated by the blender within fifteen minute intervals when newly formed paths are required.

The preservation of the integrity of the crowd membership list is upheld by each member. The list is updated when notification of changes are received from the blender. The jondo can also revise its list to the most recent changes obtained by detecting *jondos* who have failed.

The primary drawback to this approach to group membership maintenance is that the blender is a trusted third party relative to key distribution and membership reporting.

2.1.5 Jondo

As mentioned earlier, a crowd consists of a collection of users that intend to participate in web transactions, in which each user runs a process on their machine called a jondo. The jondo represents a faceless participant in the gathering that seeks anonymity in their web exchanges. The browser must configure the jondo as its proxy prior to web communications. Subsequently, all requests submitted from the browser are sent directly to the jondo. Other values associated with the jondo must also be loaded prior to the commencement of web transactions. These values are contained in a configuration file on the user's system. The configuration file contains the following attributes:

name = the name registered with the blender

password = the password registered with the blender

cookies = 0

blender = the IP address of the blender

bport = the blender's port associated with the 'Crowds' implementation

jport = the jondo's port

commit = 1

verbose = displays informative text to STDERR

firewall = 0

protocol = determines what protocol is used HTTP or FTP

display = visual display of the jondos paths

2.2 Flow of Crowds

Initially, the blender is started and listens for incoming requests. The jondo commences operations by loading the values of the attributes stored in its configuration file, initiating a connection to the blender, and starts its own internal and external servers. The jondo's internal server processes requests from the blender and other *jondos*, while the jondo's external server processes messages from the browser. After connection to the blender is established, the jondo sends the following message to the blender:

encrypted time	:	jondo's name	:	jondo's port	:	firewall flag
----------------	---	--------------	---	--------------	---	---------------

Upon successful validation of the jondo by the blender, it adds the jondo to the 'members' database file, generates shared keys to give this jondo the ability to communicate with all of the other members in the crowd, and transmits to the new member the following message:

encrypted time	=	commit time	=	membership list
----------------	---	-------------	---	-----------------

The membership list includes all the previously joined *jondos* including the new jondo and consists of the following:

member_name	,	jondo's IP-address	:	jondo's port	,	key	;
-------------	---	--------------------	---	--------------	---	-----	---

This order is repeated until all members' data has been sent.

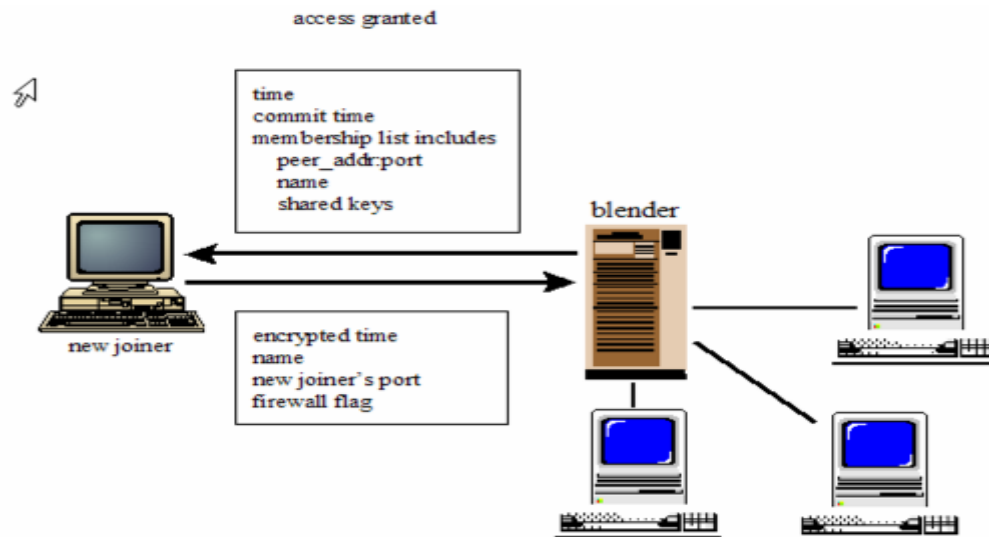


Figure 11: New joiner connecting to blender and blender's response

The acceptance and processing of this information allows the jondo to become an effective participant in the crowd. The jondo is now a new_joiner and the connection to the blender is terminated.

The blender sends an update to all *jondos* with the exception of the new joiner. The message contains information relevant to the new member and permits the other members to communicate with the new member. This information includes the IP address and port of the new member and a unique key is sent to every jondo in the list for communicating with this particular jondo.

The blender prepends a blender header to the message so the jondos being updated can differentiate between a blender and a proxy message, connects to the internal server of each jondo, and transmits the following encrypted message:

encrypted time	,	new jondo's IP address	,	new jondo's port	,	shared key	,	new jondo's firewall flag
----------------	---	------------------------	---	------------------	---	------------	---	---------------------------

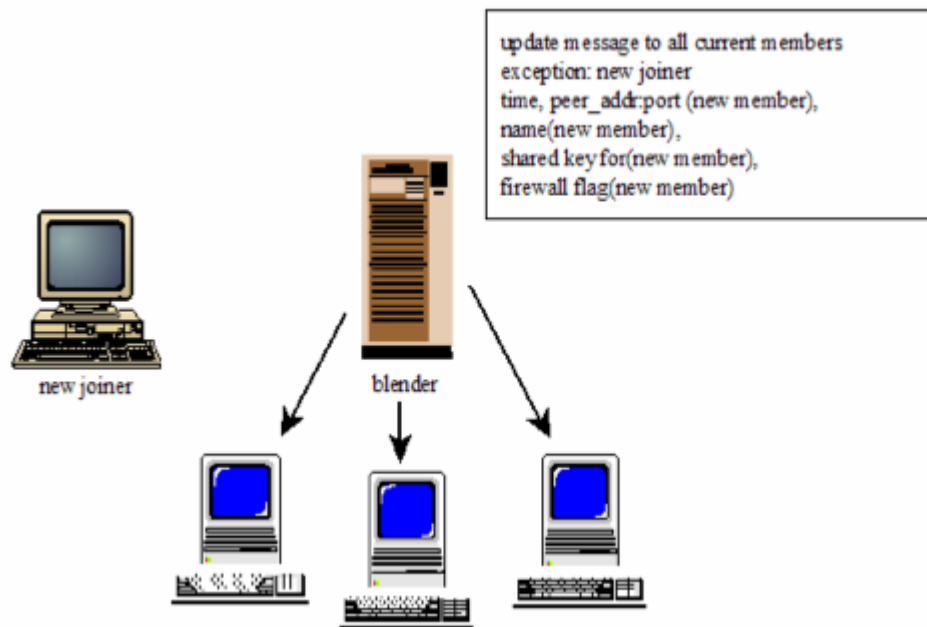


Figure 12: Blender sends update message to current members

The blender's server listens for connection requests in the event loop. A timeout occurs if an event does not take place in the allotted time. The blender's timeout handler determines if the commit time has been reached and sends the commit message to every jondo when the commit time is encountered. The blender prepends a blender header to the message and sends the following commit message which consists of the following entries:

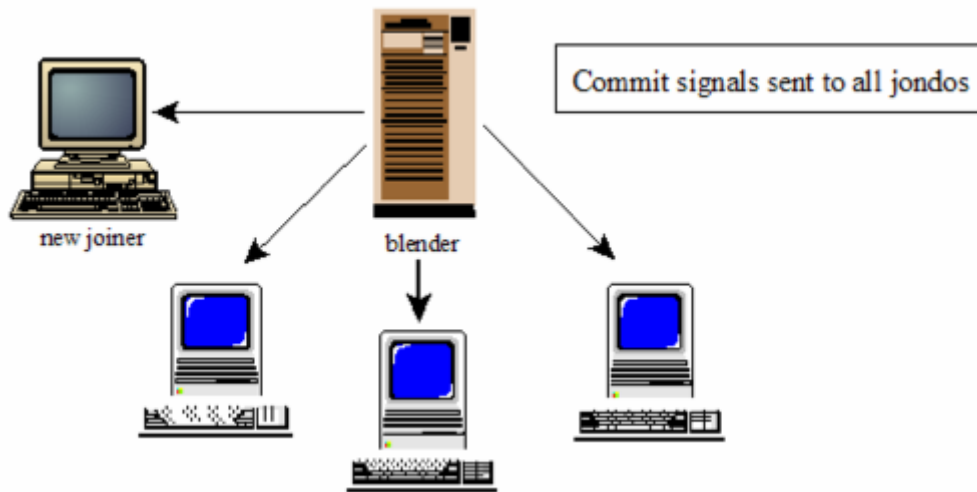


Figure 13: Commit signals sent to all jondos

If the browser tries to establish a connection with its proxy (jondo's external server), prior to the new jondo receiving the commit message from the blender, its jondo transmits the 'display_new_joiner message' to the browser which is displayed by the browser. Any subsequent messages sent from the browser display the same message until the commit message has been received.

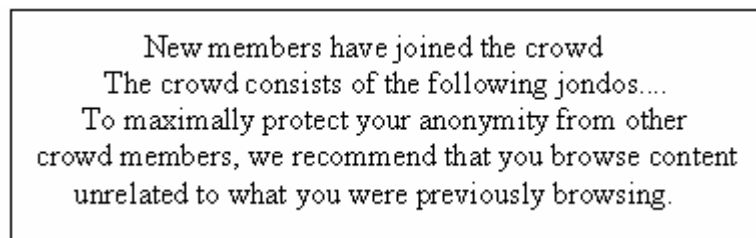
```
Welcome new crowd member

Welcome to the crowd. You should be able to
browse or execute an anonymous FTP
transaction using the crowd just after the
$commit_time.
Until then, to browse (non-anonymously) just
change the HTTP proxy setting in your browser.
```

Figure 14: Display the new joiner message

After the reception of the commit message, the new joiner's commit status changes from new_joiner to normal and further processing of the message occurs. The shift in commit status causes the program to check if the protocol contained in the jondo's initial message is valid, i.e. the HTTP or FTP protocol. It checks whether the message is a crowd query, a request used by the user to obtain a list of all active jondos in the crowd. And it probes to see if previously requested images are contained in the image cache. If these conditions do not exist, the message is considered new, the image cache is cleared, and the message undergoes further processing.

Prior to the reception of the commit message from the blender, the jondos that have been previously browsing continue to browse. After the commit message has been transmitted and received then the browsing jondos commit_status changes from normal to committed. The shift in the commit_status triggers the jondo to display the 'display_joining_message' to its browser.



New members have joined the crowd
The crowd consists of the following jondos....
To maximally protect your anonymity from other
crowd members, we recommend that you browse content
unrelated to what you were previously browsing.

Figure 15: Display joining message

If new material is browsed the path will be reconstructed. The motivation behind the reconstruction of the paths is to protect the new joiners anonymity. If the pre-existing paths remained static then any new path constructed would be attributed to the new joiner. The program also checks for protocol validity, a crowd query message, or an image request. An absence of these conditions indicates a new message any pre-existing paths are destroyed, and the message undergoes further processing.

If there is only one member in the crowd a direct connection to the web server is initiated and all forwarding routines are bypassed.

Assuming the crowd is truly a crowd, more than one member, the first line of the initiating message consists of the following elements to maintain consistency with the HTTP protocol:

request method	url	protocol version
----------------	-----	------------------

The server's address and port are extracted from the URL for the subsequent connection to the server. The subsequent messages from the browser via the jondo are HTTP headers followed by the HTTP entity body. Any unwanted headers that may impact anonymity are filtered out. The headers that remain are stored for image retrieval later in the process. If a jondo is randomly selected it is added to the path of the initiator and a connection is established with that jondo. The jondo can either select itself (pseudo-connection) or another jondo.

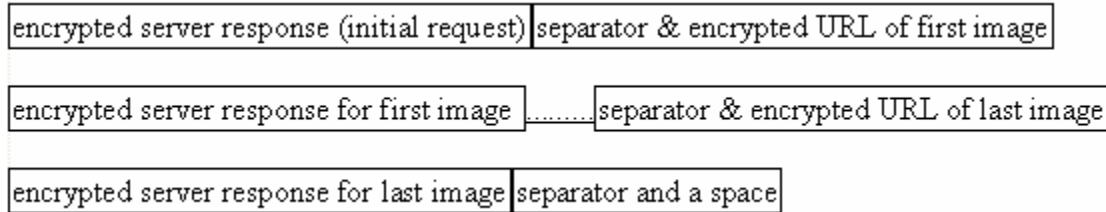
The initial message sent to the randomly selected jondo includes the 'proxy' header (indicates message source), the sending jondo's port, the encrypted request and response keys and a mapped id of the sender's path id.

crowds/1.1:proxy	:	jport	:	key1	:	key2	:	path_id
------------------	---	-------	---	------	---	------	---	---------

Forwarding routines continue to employ randomly selected jondos to generate a path to the server. During this process, the system also generates stream ciphers to encrypt and decrypt requests and replies between jondos. This forwarding process continues until a submitting jondo is randomly selected by the system. The request presented to the server by the submitting jondo transmits the requests in plaintext.

The server delivers the appropriate message to the submitting jondo and the submitting jondo parses out any images contained in the web page. The URLs of the images are stored for later submittal and the server's response is concatenated and stored. After the server closes the connection, the stored message is encrypted and sent back to the submitting jondo's predecessor. The connection is terminated and the client (submitting jondo) is disconnected. If there were any URLs of images stored they are now removed one by one and connection is made to the server using these URLs. A request is made to the server using the URL and the same request method and stored headers.

This message is sent to the server and the server responds by sending the requested image to the submitting jondo. This jondo in turns sends this image to the previous jondo. The message and images are sent using the following format:



The message is sent along the path from one jondo to its client (previous jondo) until the originating jondo is reached. The originating jondo removes the separators and submits the original server's response to the browser.

If the web page contains embedded URLs (e.g. images) the browser sends these requests to its jondo. The browser's jondo does not forward the requests on the path but waits for the contents of the request to return and then transmits the contents to the browser to prevent timing attacks.

2.3 Goals and Security Considerations

2.3.1 Types of Anonymity

As discussed in [7] there are three types of anonymity: sender anonymity, receiver anonymity, and unlinkability of sender and receiver. Sender anonymity means that the identity of the sender is hidden. Receiver anonymity signifies that the identity of the receiver is hidden. Unlinkability of sender and receiver means that a communication link between the sender and receiver cannot be determined even though it is known that both are participants in some communication.

2.3.2 Types of Attackers

A local eavesdropper can minimize privacy since this attacker observes all communication originating from the user's computer.

Collaboration among the jondos can be a detriment to anonymous communication by combining their data and departing from the prescribed rules of communication. Combination of these attackers can also exist to negate privacy.

2.3.3 The Degree of Anonymity(Figure 16)

The degree of anonymity is viewed as an informal continuum with six defined areas (Continuum references sender anonymity):

Absolute Privacy ----- exists when the presence of communication by the sender cannot be detected.

Beyond Suspicion ---- defined when communication is observed, but the initiator can not be distinguished from any other potential communicator.

Probable Innocence -- the realm in which the communicator might be somewhat suspect, the attacker is unable to have greater than 50% confidence that any node initiated the message.

Possible Innocence -- the region in the continuum where there is a significant probability that Someone else is responsible.

Exposed ----- indicates the identity of the sender has been compromised.

Provably Exposed --- communicator can be identified and proven to others.

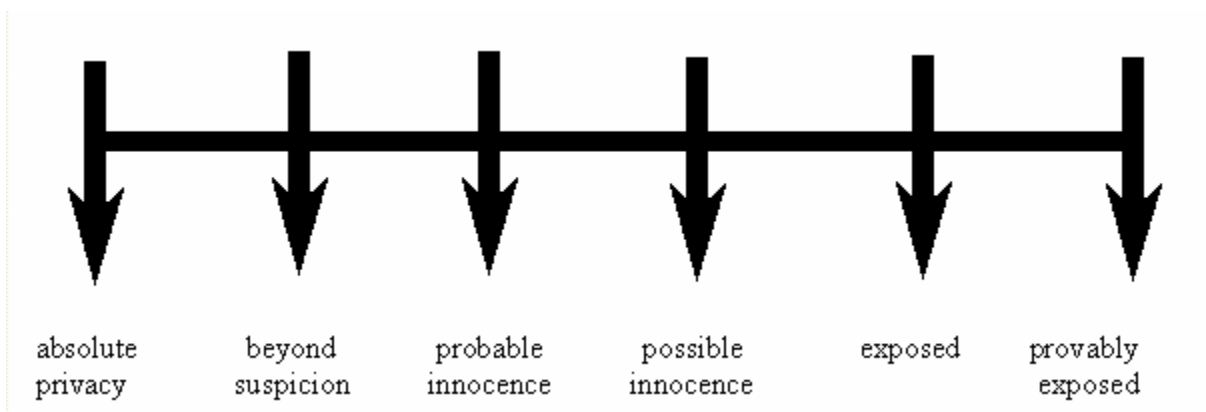


Figure 16: Degrees of Anonymity

2.3.4 Anonymity Properties provided by *Crowds*

Attacker	Sender Anonymity	Receiver Anonymity
local eavesdropper	exposed	$P(\text{beyond suspicion}) \rightarrow 1$ $n \rightarrow \infty$
c collaborating members $n > \frac{pr}{pr-1/2} (c+1)$	probable innocence $P(\text{absolute privacy}) \rightarrow 1$ $n \rightarrow \infty$	$P(\text{absolute privacy}) \rightarrow 1$ $n \rightarrow \infty$
end server	beyond suspicion	N/A

Table 1: Anonymity Properties of *Crowds*

As discussed in [6] the system does not offer sender anonymity in reference to a local eavesdropper since the local attacker observes all communication originating from the user's computer and, hence; its degree of anonymity is exposed.

There are two cases to consider when analyzing receiver anonymity in reference to the local eavesdropper as the attacker. In the first scenario the initiating jondo is the submitting jondo and the message passed to the server is in plaintext, thereby; revealing the server's identity to the local eavesdropper, since the attacker can read all messages from the initiator. The probability that the jondo is the submitting jondo is 1 divided by the number of members in the crowd, thus; the probability that the initiating jondo is the submitting jondo is inversely proportional to the crowd size. The second case occurs when the initiator is not the submitting jondo and all requests emanating from the jondo are encrypted and the anonymity of the receiver is not compromised, since the local eavesdropper cannot ascertain the receiver's identity. It can be clearly seen that the first scenario is the weakest relative to receiver anonymity but still suggests that the probability of '*beyond suspicion*' approaches 1 as the crowd size approaches infinity.

In reference to aggressive behavior from the end server, receiver anonymity is obviously non-existent. The end server can not predict which member is the path initiator since all of the members have an equal possibility of being the initiator, therefore; sender anonymity is beyond suspicion.

In regards to an alliance of corrupt jondos, receiver anonymity would be compromised if one or more of the collaborators were participants in the path, since the plaintext of messages are routed through path members. The paper derived a theorem based on probability which stated that the path initiator had '*probable innocence*' against '*c*' collaborators if $n \geq (p_f / p_f - 1/2) (c + 1)$

where ' p_f ' is the probability of forwarding, '*c*' denotes the number of collaborators, and '*n*' indicates the number of members in the crowd. This equation indicates that if the probability of forwarding is high, the number of collaborators that can be tolerated approaches half the crowd and still ensures '*probable innocence*' for the path initiator. If the probability of forwarding is close to one-half then collaborator tolerance is decreased if '*probable innocence*' for the initiator is to be maintained. The equation also suggests that as $n \rightarrow \infty$ the probability that a collaborator exists on the path approaches zero, if *c* and p_f are held constant. The supposition that collaborators cannot monitor a path they do not occupy leads us to the conclusion that the probability of '*absolute privacy*' approaches '1' as '*n*' approaches infinity for sender and receiver anonymity.

Timing considerations were also discussed and have a direct bearing on the code. Inherently, HTML is a language whose web pages can contain embedded urls of images.

When the user's browser receives a web page that contain these urls it automatically issues subsequent request(s) to acquire these embedded images. The prompt request with a brief duration may indicate to a collaborator that its predecessor is the initiating jondo. To preclude this timing attack the implementation causes the submitting jondo to parse out the embedded urls for the images and subsequently request them from the server and send them along the same path back to the initiating client.

2.4 Empirical data measuring performance by the implementers

The equipment and software involved in measuring performance in the AT&T research lab [6] were 150 MHZ Sparc 20 machines containing the SunOS operating systems. The browser employed for this activity was Netscape 3.01 configured to allow four simultaneous network connections. The Apache web server was utilized on a 130 MHZ SGI workstation running Iris 5.3. Figures 17 & 18 show the results that the lab achieved while retrieving web pages without the additional overhead of embedded images. Figure 21 displays the documented results of recovering web pages with embedded images. The variables in figure 14 are page size, in kilobytes, the path length (number of jondos in the path) , and the mean latency in milliseconds. Jondos appearing repeatedly on the path are counted for each time they appear in the path length.

An empirical result that is manifested when the path length increases from one to two results from the fact that encryption starts at page length two. When only one jondo is in the crowd that particular jondo makes a direct connection to the server and hence encryption does not occur. To impede the overhead of encryption a path key is employed. The hindrance to an increase in latency and thus in performance results from the path key being able to allow the initiating jondo to encrypt the requests and the submitting jondo to decrypt the message. The reversal of this procedure occurs with the server responses, the submitting jondo encrypts the replies and the initiating jondo decrypts the message. All of the intermediate jondos are not involved in these cryptographic processes and thus latency is minimized.

Encryption overhead is more of a detriment when embedded images are retrieved (Fig. 19). AT&T labs used 1-kilobyte images on the same server for their test. The serial transmission of the images was also mentioned as a factor that increased latency as opposed to other browsers that retrieve images concurrently.

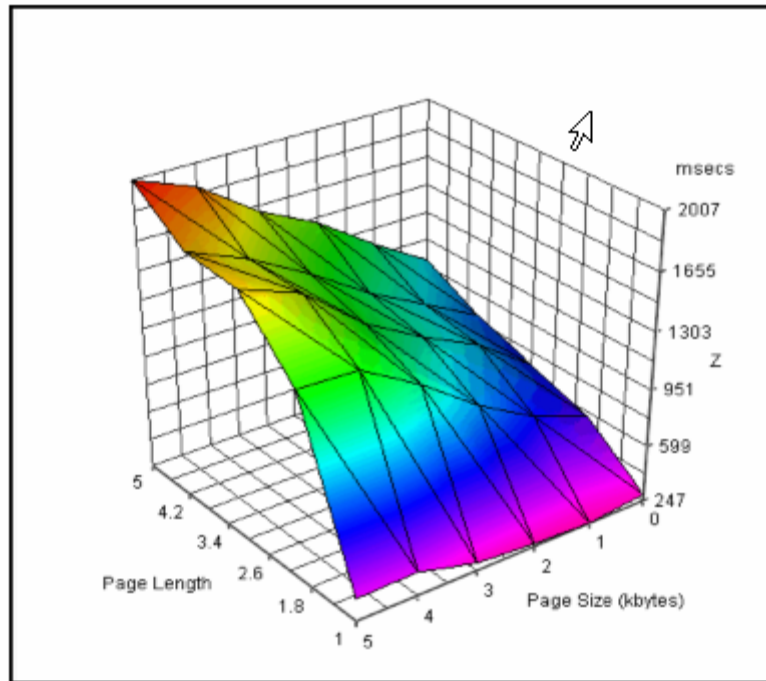


Figure 17: Performance Graph without embedded images

Path Length	Page Size (kbytes)					
	0	1	2	3	4	5
1	288	247	264	294	393	386
2	573	700	900	1157	1369	1384
3	692	945	1113	1316	1612	1748
4	814	1004	1191	1421	1623	1774

Table 2: Performance Data without embedded images

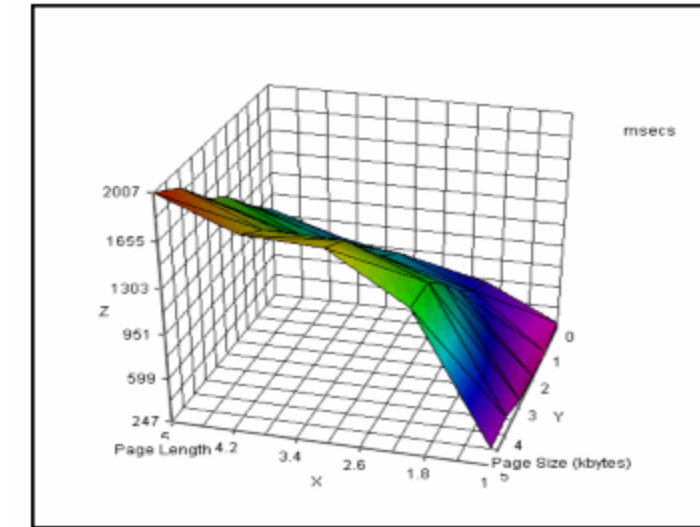


Figure 18: Performance Graph without embedded images

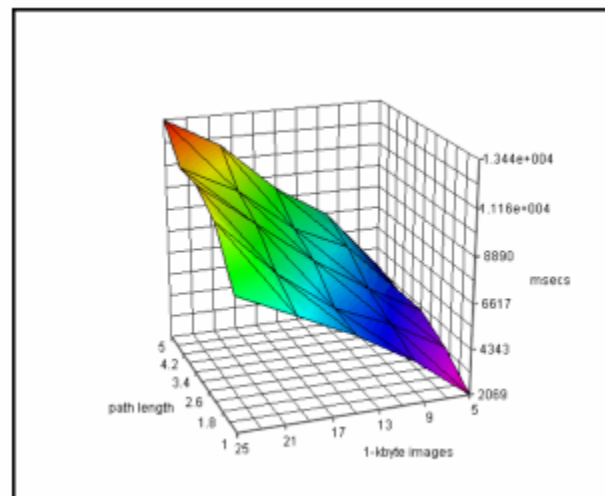


Figure 19: Performance Graph with embedded images

Path Length	Number of 1-kbyte images				
	5	10	15	20	25
1	2069	4200	5866	7219	8557
2	3313	4915	6101	8195	10994
3	4127	5654	7464	9611	11809
4	4122	6840	8156	10380	11823
5	4508	7644	9388	11889	13438

Table 3: Performance Data with embedded images

Chapter 3: Jondos Configuration menus

To provide the user with the ability to load and edit the jondos configuration values efficiently, I have generated a graphical user interface. The configuration menus created to accomplish this task are: the Start, List, Edit, and Help menus. A brief Help menu will also be available to aid in understanding these attributes and associated values.

When the program is invoked, the List, Edit, Help, and Exit buttons (ref. Figure 20) are generated and associated with their corresponding routines. Each aforementioned buttons in the initial window invokes a routine which produces a new window and also disables the button producing the effect. Disabling the button inhibits the production of multiple windows while the procedure is being executed.

The List procedure (ref. Figure 21) loads the attributes and corresponding values from the configuration file. The attributes and values are displayed in a scrolled listbox and a Dismiss button is provided to withdraw the window from view.

The Edit routine (ref. Figure 22) exists in a scrolled text area that provides the capability to edit the values of the associated configuration attributes. The attributes are displayed on left justified labels in the text area and the values of the attributes are exhibited in right justified entry widgets. The entry widgets allow the user to revise the values. The Save button stores the information with the attributes and values passed by reference and deposited into an array which is saved in the configuration file of that jondo. A Dismiss button is also included for window withdrawal.

The Help procedure (ref. Figure 23) includes a scrolled text area that contains a menu bar which encompasses the following menu buttons: blender, jondo, headers and flags. Items are added to each of these menus and the appropriate information is displayed when the item is activated.

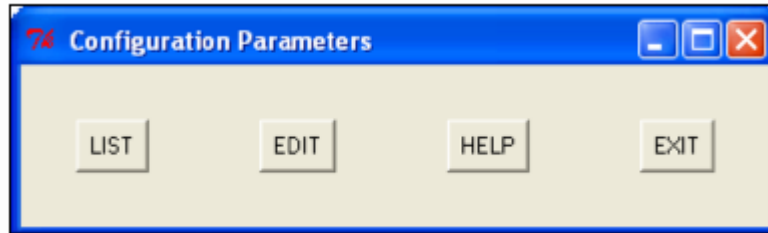


Figure 20: Configuration Menus – Main Window

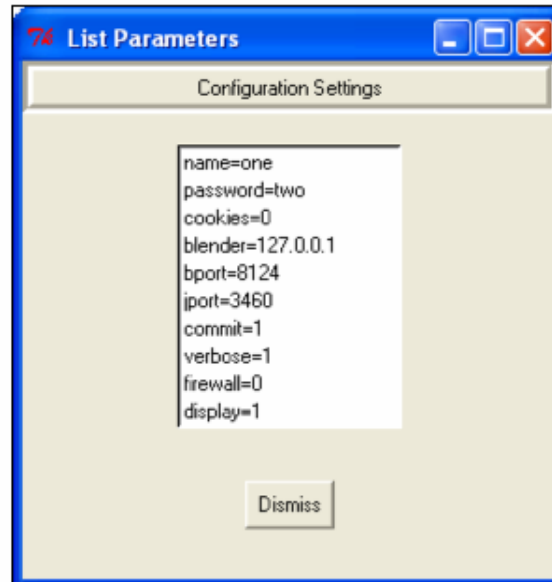


Figure 21: Configuration Menus – List Window

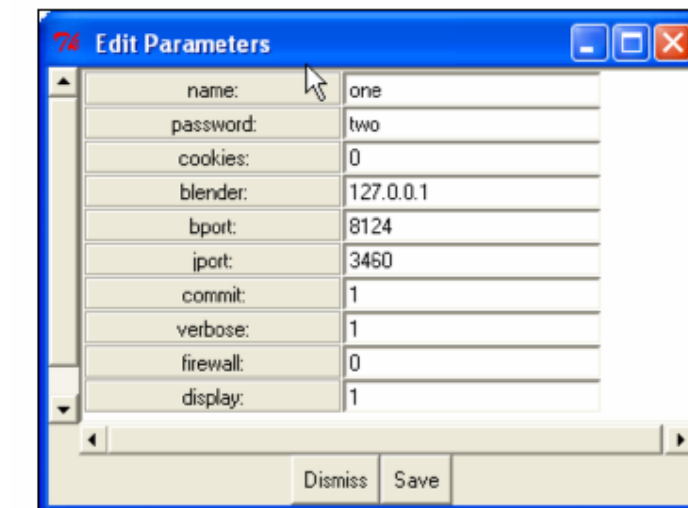


Figure 22: Configuration Menu – Edit Window

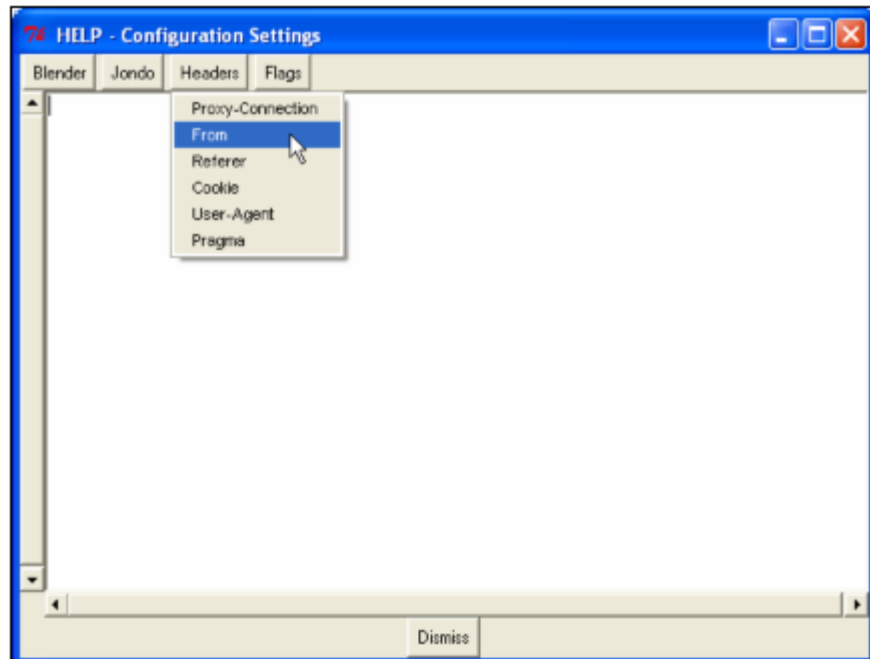


Figure 23: Configuration Menu – Help Window

Chapter 4: Anonymous FTP and *Crowds*

As mentioned previously, this software system retrieves only embedded FTP requests. I am developing a graphical FTP client that can connect to a FTP server, examine the file system and retrieve text and binary files while traversing the *Crowds* implementation. First, I will discuss the FTP protocol that will help to clarify the similarities and differences between the two protocols. I will continue the discussion with an example of an FTP session. Next, I will explicate the functionality of each component in the client's design. The subsequent section will list the commands that will be incorporated in the client followed by a description of the server's responses. Finally, I will describe essential modifications in the original software to support the protocol.

4.1 File Transfer Protocol (Ref. Figure 24)

FTP is a protocol used to transfer files between computers. The basic model of FTP, as described by the RFC 959 paradigm consists of three components for the client and two for the server. The client components are the user interface, the client protocol interpreter, and the client data transfer unit. The server components are the server protocol interpreter and the server data transfer unit. The FTP protocol employs two TCP connections, one for the control connection and the other for the data connection. The control connection is made between the client and server protocol interpreters and similarly the data connection is established between the data transfer units of the client and server.

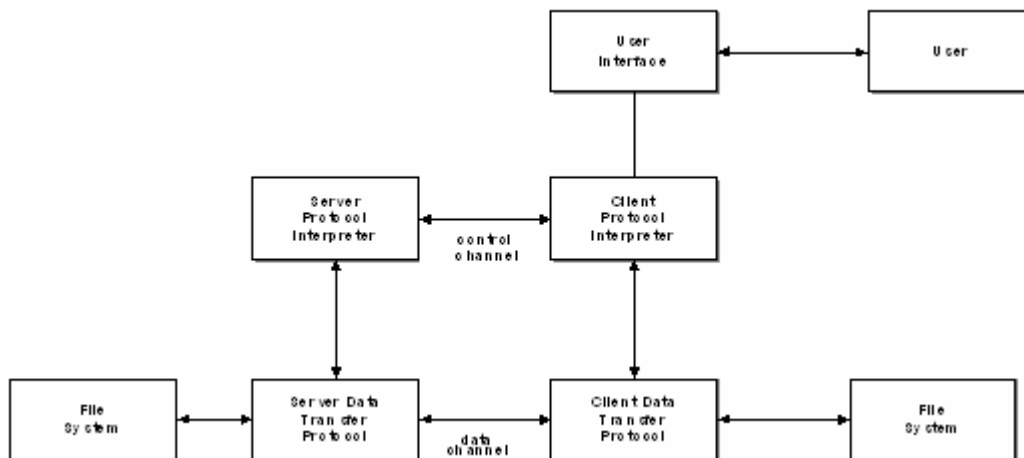


Figure 24: FTP components

The function of the control connection is to send commands to manage directories, transferring files, authentication, etc. The data connection is the channel in which the file is actually received or sent. The FTP control channel is persistent, it remains open until the interactive session is closed whereas a data channel is non-persistent, a new data channel is generated for each file transferred within a session and then closed. The FTP client first establishes a TCP control connection with the FTP server on the default port of 21. When a data connection is established the default port of 20 is used at the server side. Communication over the control connection utilizes client commands and server responses. Each command or response is only a short line using the NVT ASCII character set.

When communicating over the data connection one must consider that the computers exchanging files may have different character sets, file structures, and different file formats. To resolve this compatibility issue, the client defines the file type, the data structure, and the transmission mode.

The acceptable file types in the FTP protocol are ASCII, EBCDIC, and Image. FTP transfers files across the data connection using one of the following structures: file structure, record structure, or page structure. The transmission modes utilized by FTP are the stream mode, block mode, or the compressed mode.

4.2 File Transfer Example (Ref. Figure 25)

When using FTP in file transferring one of three events will occur:

A file will be retrieved from the server to the client applying the 'RETR' command.

A file will be copied to the server from the client utilizing the 'STOR' command.

A list of directory contents are sent from the server to the client employing the 'LIST' command.

This example will illustrate the retrieval of a directory's contents from the server's file system by executing the 'LIST' command.

A control connection is established between the server and client using the default port 21.

The server sends the following message to identify itself: '220 linux FTP server (ver...)'.

The anonymous user starts its login sequence with the message: 'USER anonymous'.

The server responds with the message: 331 'Guest login OK' and prompts the user for its password.

The user submits its 'guest' (or a fictitious (anonymous) email address) for its password, 'PASS guest'.

The server responds with the message: '230 guest login OK'.

The client then informs the server of its IP address and the port it will use for data connections on the client system: PORT IP address port #.

The server prepares itself to open the data connection between port 20 (default server side) and the Ephemeral port received from the client.

The server sends the following message indicating this preparation: '150 data connection will open soon'.

The client sends the 'LIST' command to display the directory contents.

The server sends the message: '125 Data Connection OK'.

The server subsequently sends the directory contents over the data connection and then the message '226 Directory Send OK, Closing the Data Connection'.

The client can continue sending FTP commands and receive server responses or terminate the session by sending the message 'QUIT' to the server.

The server responds to the 'QUIT' command with the message: '221 Server Closing GOODBYE'.

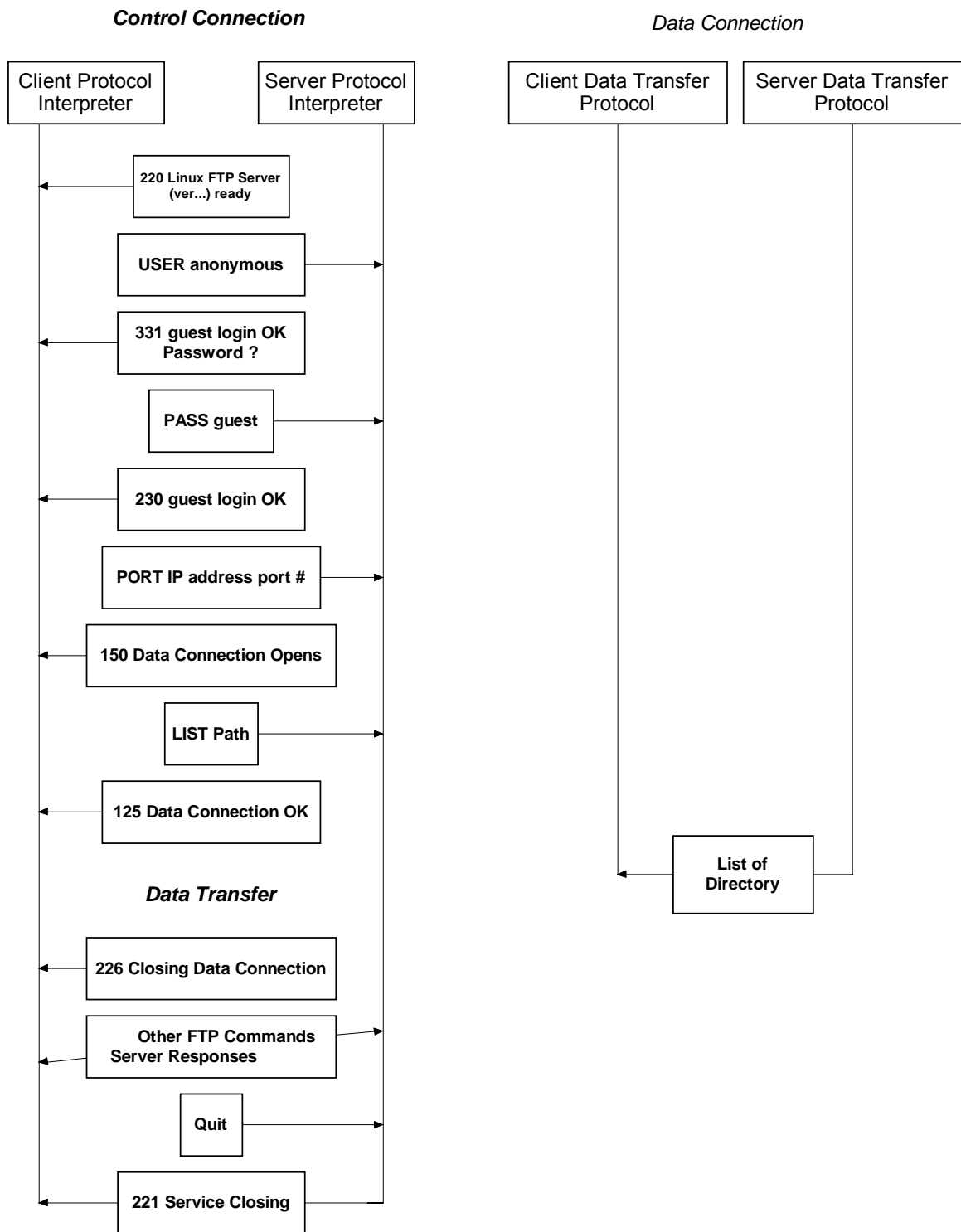


Figure 25: FTP Transfer Example

4.3 File Transfer Protocol Client Design

The FTP client used in conjunction with the *Crowds* implementation is a graphical user interface that contains the code that applies the FTP and *Crowds* protocols to establish a connection and transfer files from the FTP server. The client's design will follow the paradigm defined in RFC 959. It will consist of the User Interface, the Client Protocol Interpreter and the Client Data Transfer Protocol. As represented in Figure 26 the User Interface in this design will consist of three software elements: Local Interface, Remote Interface, and Ascii/Binary. The Client Protocol Interpreter will be implemented with the Connect, Handle Incoming Messages and the Command Interpreter sections. The functionality of the Client Data Transfer Protocol will be achieved using the components: Incoming Data Message, Client Data Transfer, and Saving File.

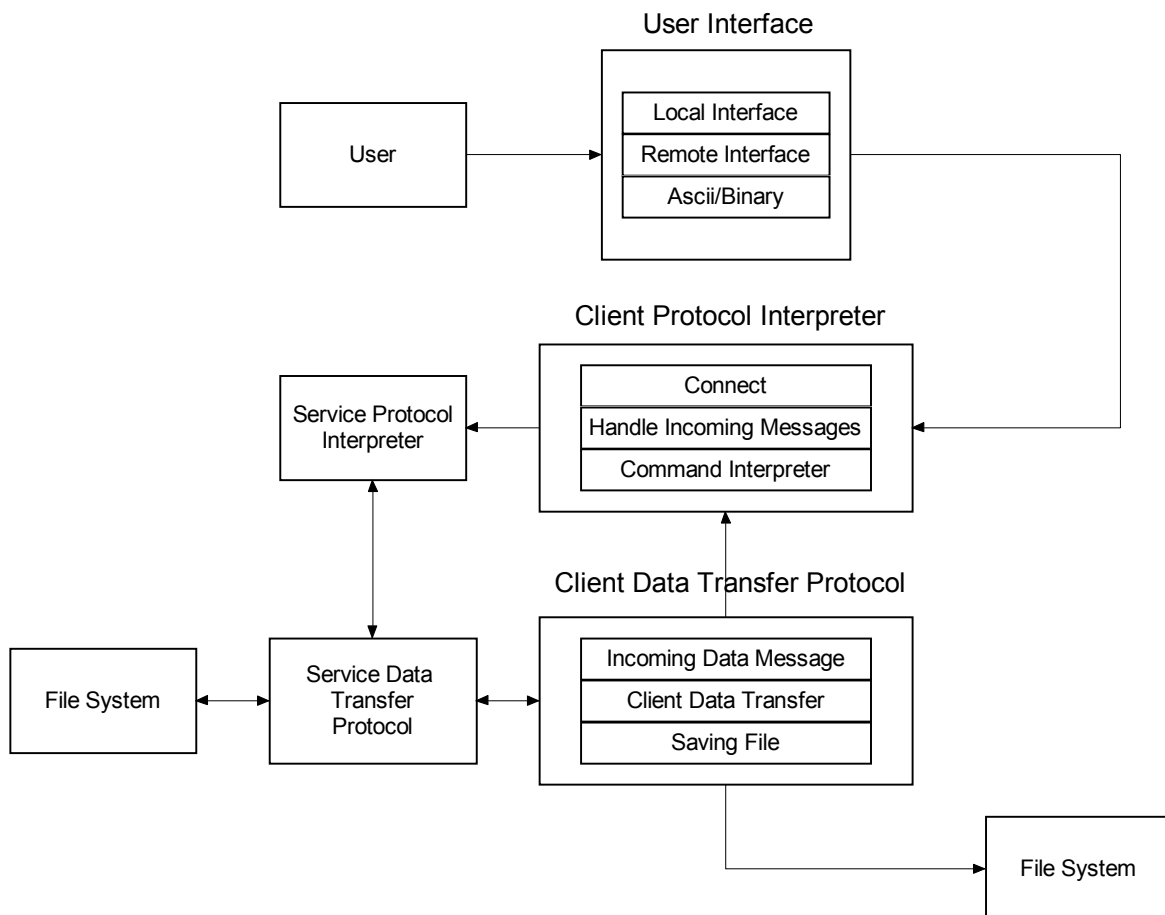


Figure 26: FTP client

4.3.1 User Interface

4.3.1.1 Local Interface

Initially, when the FTP client is executed, the hierarchy of the local file system is displayed beginning with the root directory. Any directory in the local file system can be expanded or collapsed, revealing or concealing its associated contents. The management of the local file system can be accomplished either by graphical means or by using the appropriate buttons on the interface. The absolute path to the file or directory is required when using change directory in the local file system. The interface will also allow the user to generate a new directory in the local file system. If the entry selected is a file or link to a file then the file's data is subsequently disclosed in a pre-existing window. The local interface will generate a display area to accept user input and display server responses (ref. Fig. 27). Informational or error messages from the jondo are also displayed in this area. The interface will also send the user requests or server responses to the Client Protocol Interpreter for further processing and ultimately to the Server Protocol Interpreter.

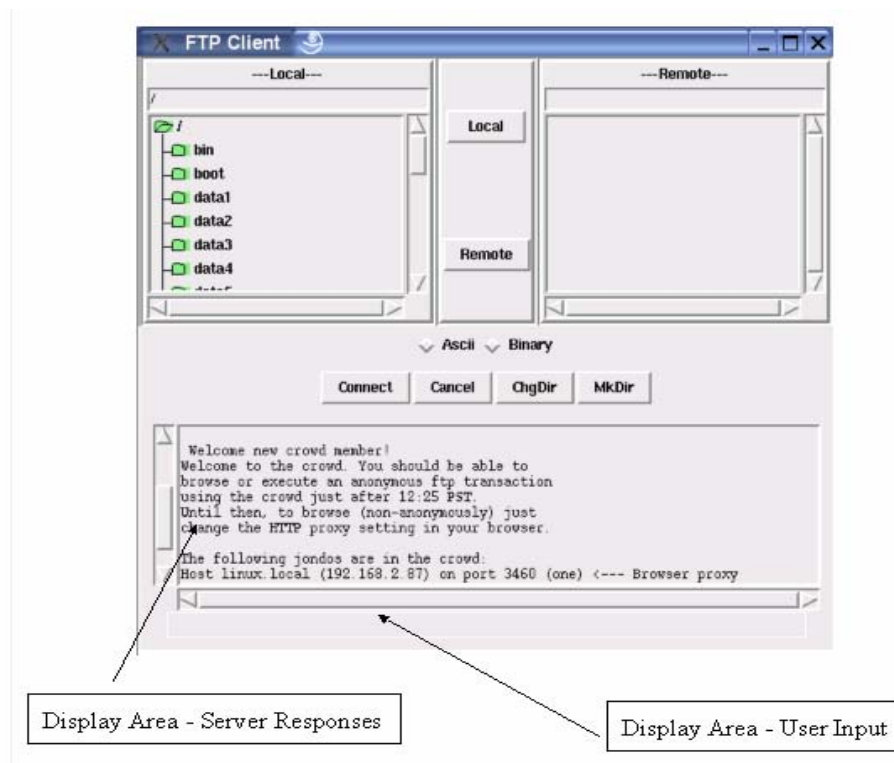


Figure 27: FTP Client User Interface

4.3.1.2 Remote Interface

After connection is established with the FTP server, the client will display the remote file system in the user interface remote area. Selecting and activating a directory entry displays or conceals the directory's contents. If a file or a link to a file is selected in the remote file system, the file is stored automatically in the current directory of the local file system.

4.3.1.3 ASCII/Binary

This FTP client will be able to transfer ASCII files and binary or image files across its data connection. In ASCII each character is encoded using ASCII. The image or binary files are sent as continuous streams of bits without any interpretation or encoding. The user selects the type of file by activating one of the buttons on the user interface. The client will use the default values for file structure and transmission mode.

4.3.2 Client Protocol Interpreter

4.3.2.1 Connect

This component of the Client Protocol Interpreter illustrated in Figure 28 functions as the connection mechanism to the user's jondo and ultimately to the FTP server by traversing the dynamically generated path of randomly selected jondos. The dialog box provides the user with the ability to insert the addresses and ports of the FTP server and its jondo, The Dialog Box is activated by the 'Connect' button on the user interface. The initiating message to the server is also transmitted in this routine to initiate the FTP session.

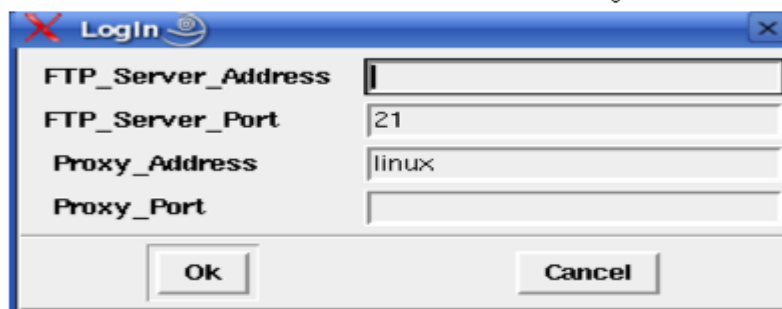


Figure 28: FTP Connection Dialog Box

4.3.2.2 Handle Incoming Message

This software element distinguishes between informational or error messages from the jondo and responses from the FTP server. Jondo messages are in HTML format and undergo further processing to convert the files to plain text in the display area of the user interface. Server responses are sent to the Command Interpreter for additional processing.

4.3.2.3 Command Interpreter

The function of this Command Interpreter is to process user requests from the user interface. This component also issues commands to the FTP server (Server's Protocol Interpreter) on the control connection, receive numerical reply codes and the associated descriptions from the FTP server, decipher these codes and process them accordingly. This element also drives the 'Client Data Transfer Unit'.

4.3.3 Client Data Transfer Unit

The Client Data Transfer Unit receives, processes and stores data from the Server's Data Unit. Using the file type, file structure and transmission mode parameters set up by the client over the control connection, the Server Data Unit either sends a list of the contents of a directory or transfers a file from the Server's file system over the data connection to the Client's Data Unit. In this implementation the Unit is composed of the 'Incoming Data Message', the 'Client Data Transfer' and the 'Saving File' components.

4.3.3.1 Incoming Data Message

The routine retrieves and stores messages from the server's 'Data Transfer Protocol' that are directories, files, or links.

4.3.3.2 Client Data Transfer

This component of the transfer unit establishes the data connection and processes the messages received from the server's data unit.

4.3.3.3 Saving File

This is the unit that saves files to the current directory of the local file system.

4.3.4 Commands Included in the FTP Client Design

FTP protocol commands sent from the FTP client consists of four uppercase ASCII characters followed by optional arguments and can be subdivided into six groups: access commands, file management commands, data formatting commands, port defining commands, file transferring commands, and other various commands. These commands will be incorporated in this design.

Access commands – The user accesses or terminates its connection with the remote system.

<i>Command</i>	<i>Description</i>
USER	Sends the username for the login
PASS	Sends the password for the login
QUIT	Logout command to end a session

File management commands -- Allows the client to access the file system on the remote system.

<i>Command</i>	<i>Description</i>
CWD	Change the current directory on the server
CDUP	Change to the parent directory
LIST	List the contents of the directory
PWD	Print the current directory of the serve

Data formatting commands – Permits the user to select the file type.

<i>Command</i>	<i>Description</i>
TYPE	Data representation (file) type, ASCII, binary,etc

Port defining commands – Defines the port number for the client's data connection

<i>Command</i>	<i>Description</i>
PASV	Server chooses a specified port for the data connection
PORT	Server chooses a specified port for the data connection

File transfer commands – Allows the client to transfer files.

<i>Command</i>	<i>Description</i>
RETR	Retrieves file - file transferred from server to client
STAT	Elicits the current status of the server

Other commands – Provides miscellaneous information to the client

<i>Command</i>	<i>Description</i>
SYST	Retrieves the OS information on the server
HELP	Elicit helpful information from the server
NOOP	No Operation - check if server is alive

4.4 FTP Server's Responses

The server sends at least one response as a result to the client's command. The response consists of two parts: a three digit number and a text part. The first two digits of the numeric part define the status of the command. The third digit provides additional information. The text part defines additional required parameters or provides descriptive text.

Response examples:

200 Command OK

331 User name OK; password is needed.

4.5 Modifications to the *Crowds* implementation

4.5.1 HTTP modifications

4.5.1.1 Filtering unwanted HTTP headers

The program has the ability to filter out the 'Cookie' header request to maintain privacy. Additions to the client's configuration file and a trivial modification to the code would allow the user the ability to sanitize other request headers the client may deem necessary to ensure anonymity. The capability to remove the following headers was added to the program: the 'From' header, the 'Referer' header, and the 'User Agent' header. The 'From' header contains the client's email address but most browsers do not send 'From' headers for obvious reasons. The 'Referer' header provides the URL of the previously visited web page providing a means to build dossiers of the client. The 'User Agent' header reveals the name and version of the browser and may include the operating system being used. This header would enable an attacker to use known weaknesses of the browser or operating system to fulfill its goal.

4.5.2 FTP modifications

The jondo's(proxy) code must be modified to address the following conditions. The modifications are essential to satisfy the FTP protocol and provide access to the FTP server and ensure the reception of server responses.

4.5.2.1 Data and Control Connections

In this version of Crowds, a connection is opened using the HTTP protocol and terminated after each request and the corresponding response. In order to support the FTP protocol the proxy program must be modified to ensure the control connection remains open during the entire interactive FTP session. To accomplish this task the program has been modified to sustain this connection until the 'Quit' command from the user has been given. This modification shouldn't compromise privacy since only user commands and server responses travel along this channel.

In the FTP protocol another connection (data connection) must be generated. The data connection opens in conjunction with a file transfer or the 'LIST' command and then closes. This particular action, the opening and closing of the data connection must occur after each transaction. In the HTTP protocol only one connection is required. In the HTTP protocol version 1.0 there is just one transaction prior to the connection being terminated. The HTTP protocol connection parallels the data connection of the FTP protocol in the sense that after a single transaction occurs the channel is closed. The difference lies in the data that is transmitted over the channel, in the HTTP protocol the client's requests consists of the HTTP headers and the body; whereas, in the FTP protocol the user's requests consists of only commands. The response in the HTTP connection involves a web page and in the FTP protocol server responses are returned along the channel. The software must be adjusted to recognize these differences and accommodate both protocols.

To preclude a potential attacker from being able to associate multiple paths to a particular jondo, static paths were used in the HTTP protocol [6].

To apply this design concept to the FTP protocol, control and data channels will use the same static path to the FTP server to sustain the anonymity properties. The FTP client program and modifications to the jondo program provide for this feature.

4.5.2.2 Connection to HTTP server or FTP server

The code must be modified to distinguish between a connection to a HTTP server versus a FTP server. This version of '*Crowds*' code is based on the HTTP protocol only. Error messages default port numbers, etc. pertain to the HTTP protocol only and must be changed to include the FTP protocol.

4.5.2.3 Visual Display Modifications

The server program, i.e. the blender, must be modified to include another server that allows each jondo to transmit information to the blender so its path can be constructed. Other procedures are added to establish the connection object and respond to subsequent messages sent from the jondos. Other additions to the code will enable the blender to manipulate databases with data from the jondos to display the path for each jondo. The jondo program will provide additional code enabling it to connect to the blender and transmit pertinent path information to the blender to store in appropriate databases.

4.6 Caveats relative to the *Crowds* implementation

Adherence to the following cautions must be heeded to ensure anonymity while using the '*Crowds*' system:

Obviously the client will only use anonymous authentication to preserve user's anonymity. Any content of the client that reveals the identity of the client would render the system useless.

The FTP client will not address FTP commands that modify the server's file system since most servers restrict modification to its file system by an anonymous client.

ActiveX , JavaScript, and any other executable code must be turned off because executable content can establish a connection to the server bypassing the '*Crowds*' implementation.

4.7 Empirical data measuring performance for the FTP protocol

The equipment and software involved in measuring performance in the NSSAL lab were machines with Xeon 3.8GHz Xeon processors running the Centos Linux operating systems. The FTP client was employed for this activity and was configured to allow three simultaneous network connections. The vsftp server ran on an lightly loaded identical machine running Centos Linux. Figures 29 show the results that the lab achieved while retrieving text files. The variables in Table 4 are page size(axis y), in kilobytes, the path length (number of jondos in the path – axis x) , and the mean latency in milliseconds(axis z). Jondos appearing repeatedly on the path are counted for each time they appear in the path length.

Since the path lengths are randomly established at run time, the length of the path to establish a performance chart can be obtained with the following formula $(p_f / 1 - p_f) + 2$ [6]. The probability of forwarding (p_f) can be adjusted in the jondo's program in the routine 'handle_proxy_request'. One of the jondos or all of them can be changed to obtain the path length desired. The Perl modules Time::HiRes and Time::Elapsed were used to measure the times in milliseconds.

An empirical result that is manifested when the path length increases from one to two results from the fact that encryption starts at page length two. When only one jondo is in the crowd that particular jondo makes a direct connection to the server and hence encryption does not occur. To impede the overhead of encryption a path key is employed. The hindrance to an increase in latency and thus in performance results from the path key being able to allow the initiating jondo to encrypt the requests and the submitting jondo to decrypt the message. The reversal of this procedure occurs with the server responds, the submitting jondo encrypts the replies and the initiating jondo decrypts the message. All of the intermediate jondos are not involved in these cryptographic processes and thus latency is minimized.

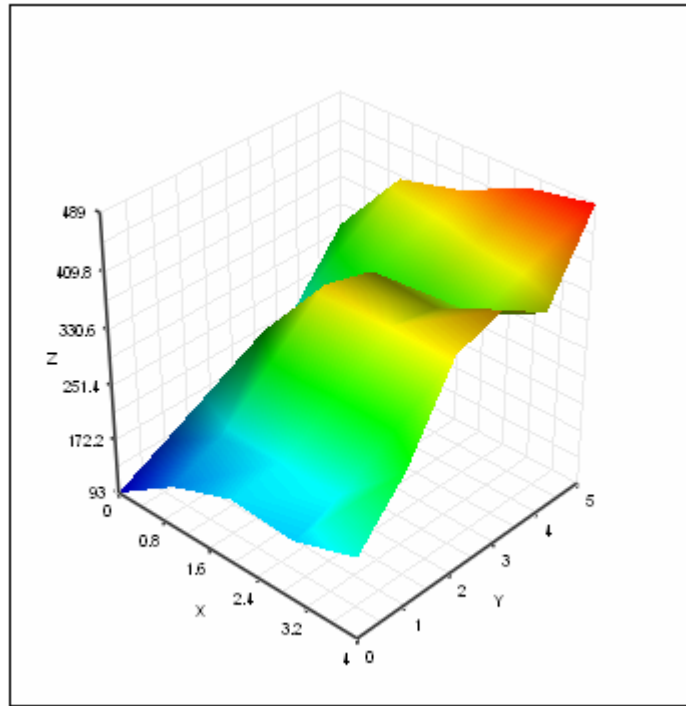


Figure 29: Performance FTP Graph

Path Length	Page size (kbytes)					
	0	1	2	3	4	5
1	93	128	124	147	220	300
2	153	182	299	283	284	405
3	186	196	405	387	306	428
4	179	200	392	380	332	471
5	209	284	399	421	379	489

Table 4: Performance FTP Data

Chapter 5: Visual display of the jondos path

Initially, each jondo is assigned a unique path identification number upon start-up. Each jondo in the path needs to be identified by a unique number to preclude infinite loops. This distinctive property is employed to develop code that graphically displays the paths of the jondos. The elements in the jondo's code used to support this visual system are the original path identifiers, and the *Translate* and *Next* associative arrays. First, unique identification numbers are assigned to each jondo. Next, before a jondo connects to its successor, it is assigned a different identification number randomly selected.

The new id number for the jondo is stored as the value in the *Translate* hash with the old path id as the key. This process is repeated with each jondo on the path until the server is reached. The successor of a jondo is randomly selected from a set of active jondos and is stored in the *Next* hash with the path id number again as the key.

The visual display system is activated by setting the display attribute in the jondo's configuration file. Initially, the jondo connects to the blender and sends the blender its name and unique id number. Subsequently, the jondo connects and transmits relevant information to the blender whenever it connects to itself, to another jondo or to the FTP server. These messages are delimited by the number '#' sign.

The visual display section of the blender utilizes these messages in conjunction with three databases to store pertinent information aiding in the generation of the graphical system. The databases include the following information: 1) the initiators' names and identifiers 2) the jondos' identifiers and the *Translate* hash and 3) the initiators' identifiers and a corresponding arrays that stores the jondos in its path.

The processing of the message from the jondo commences by determining the number of delimiters contained in the message. One delimiter denotes that this is the initial message.

Two delimiters indicate that the message is sent by the submitting jondo. More than two delimiters indicates that this is not the first message or the last message.

The display system is dynamically constructed by determining if the identification number is one of the originating id numbers and if it is, to add the next jondo to its path. If the identification number is not one from one of the initiators then the code reverses the direction of the path by using the *Translate* hash until the originator is found, then once again the next jondo is added to this originator's path.

The hash of the database file that consists of the initiating jondos and the arrays that contain their paths is processed with the URL of the server. The routine uses this data to display the initiators and their paths graphically. Initially, the originating jondo is drawn in its own window and a unique color is assigned to that jondo. This process is repeated for each jondo in the hash. Next, for each jondo that was selected, its window is retrieved, the position and color for the next jondo in its path is determined and it is drawn. The name of the jondo is also written in the display. After all the jondos in the path are displayed the server is also displayed. This process is repeated until all the data in the hash has been displayed.

Chapter 6: Conclusion

FTP provides the user with access to an abundance of information: software, drivers, graphic images, music, various documents etc. Anonymous FTP allows the user access to this wealth of information without being an official user of the system. The costs of this anonymity are the access restrictions to the server's file system and the inability to store files to the server. It is assumed that receiver anonymity is achieved in anonymous FTP since authentication is absent; but what if the server is malicious. Could the server record the IP addresses of the user, the times and frequencies of accesses, or other activities unknown to the client. It is clearly seen that sender anonymity and unlinkability are not available in this protocol. Truly anonymous FTP could be sought by masking the path using the approach provided by *Crowds* thus providing degrees of sender and receiver anonymity.

The FTP protocol can be implemented on the *Crowds*' system with the security concerns remaining intact. Other application programs utilizing TCP should also be able to utilize the system providing a source of anonymity while traversing networks

If a user seeks sender anonymity, receiver anonymity and unlinkability he could use a protocol that provides the ability to establish different personae along with the *Crowds* implementation that masks the path to the server. A combination of anonymity agents can be employed to allow a user to choose the type and degree of anonymity desired.

Chapter 7: Future Work

I would complete the command set of the FTP client to enhance the capability of the client. These commands would not include those requests that would alter the file system of the remote server, since this would be prohibited in an anonymous connection.

In reference to the data connection attributes, i.e. the file type, data structure, and transmission mode, I would allow the user to automatically select the file type, expand the data structure to include the 'record' and 'page' structure options, and include the 'block' and 'compressed' mode as transmission mode alternatives.

Additionally, would add more features to the client to be selected by icons, menus, and other cosmetically pleasing images to enhance the graphical user interface, thus; providing the user with the ability to accomplish various tasks with ease.

Bibliography

- [1] The Anonymizer Service, www.anonymizer.com.
- [2] Eran Gabber, Phillip B. Gibbons, David M. Kristol, Yossi Matias, and Alain Mayer, "Consistent, Yet Anonymous, Web Access with LPWA", Communications of the ACM, Volume 42, Number 2, February 1999, pages 42-47.
- [3] David Goldschlag, Michael Reed and Payl Syverson, "Onion Routing for Anonymous and Private Internet Connections", Communications of the ACM, Volume 42, Number 2, February 1999, pages 39-41.
- [4] Michael K. Reiter and Aviel D. Rubin, "Anonymous Web Transactions With *Crowds*", Communications of the ACM, Volume 42, Number 2, February 1999, pages 32-38.
- [5] Reed, M., Syverson, P., and Goldschlag, D. *Onion Routing for Anonymous and Private Internet Connections*. (January 28, 1999).
- [6] Michael K. Reiter and Aviel D. Rubin, "*Crowds* : Anonymity for Web Transactions", AT&T Labs Research.
- [7] Pfizmann, A, and Waidner, M. 1987. Networks without user observability. Computers & Security 2, 6, 158-166.
- [8] Srinivasan, Sriram. *Advanced Perl Programming*. Sebastopol: O'Reilly & Associates, 1997.
- [9] Wall, Larry., Tom Christiansen, and Jon Orwant. *Programming Perl*. 3rd ed. Sebastopol: O'Reilly & Associates, 2000.
- [10] Lidie, Steve., and Nancy Walsh. *Mastering Perl/Tk* Sebastopol: O'Reilly & Associates, 2002.
- [11] Request For Comments 959
- [12] Gourley, David., and Brian Totty. *HTTP The Definitive Guide*. Sebastopol: O'Reilly & Associates, 2002.
- [13] Stein, Lincoln D., *Network Programming with Perl*. Addison-Wesley, 2001
- [14] Syverson, P., Reed, M., and Goldschlag D. Private Web browsing. Journal of Computer Security. Sec. 5,3 (1997) 237-248
- [15] Tor: An anonymous Internet <http://tor.eff.org/>

VITA

Paul Flowers received a Bachelor of Science degree in Mechanical Engineering from the University of Memphis in 1980.