Universitatea Alexandru Ioan Cuza Iaşi
Facultatea de Informaticã

Lucrare de Licenţã

Gazebo F2F : A Friend-to-Friend Application

propusă de

Ştefan Silvestru

Sesiunea : iunie , 2009

Coordonator ştiinţific
Prof. Dr. Gabriel Ciobanu

Universitatea Alexandru Ioan Cuza Iaşi
Facultatea de Informaticã

Gazebo F2F : A Friend-to-Friend Application

propusă de

Ştefan Silvestru

Sesiunea : iunie , 2009

Coordonator ştiinţific
Prof. Dr. Gabriel Ciobanu

Declarație privind originalitatea şi respectarea drepturilor de autor

Prin prezenta declar cã Lucrarea de licență cu titlul *"Gazebo F2F : A Friend-to-Friend Application"* este scrisã de mine şi nu a mai fost prezentatã niciodata la o altã facultate sau instituție de învãțãmânt superior din țarã sau strãinatate. De asemenea, declar cã toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate in lucrare, cu respectarea regulilor de evitare a plagiatului :

- toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altã limbã, sunt scrise între ghilimele şi dețin referința precisã a sursei;
- reformularea în cuvinte proprii a textelor scrise de cãtre alți autori deține referința precisã;
- codul sursã, imagini, etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi dețin referințe precise;
- rezumarea ideilor altor autori precizeazã referința precisã la textul original.

Iaşi, 16 iunie 2009

Absolvent *Ştefan Silvestru*

____

(semnãtura în original)

Declarație de consimțământ

Prin prezenta declar ca sunt de acord ca Lucrarea de licență cu titlul *"Gazebo F2F : A Friend-to-Friend Application"* , codul sursã al programelor şi celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare sa fie utilizate în cadrul Facultãții de Informaticã. De asemenea, sunt de acord ca Facultatea de Informaticã de la Universitatea Alexandru Ioan Cuza Iaşi sã utilizeze, modifice, reproducã, şi sã distribuie în scopuri necomerciale programele-calculator, format executabil şi sursã, realizate de mine în cadrul prezentei lucrãri de licență.

Iaşi, 16 iunie 2009

Absolvent *Ştefan Silvestru*

____

(semnãtura în original)

Acord privind proprietatea dreptului de autor

Facultatea de Informaticã este de acord ca drepturile de autor asupra programelor-calculator, format executabil şi sursã, sã aparținã autorului prezentei lucrãri, *Ştefan Silvestru.*

Încheierea acestui acord este necesarã din urmatoarele motive:

Iaşi, 16 iunie 2009

Decan *Gheorghe Grigoraş*	Absolvent *Ştefan Silvestru*
___________________________	___________________________
(semnãtura în original)	(semnãtura în original)

Contents

Introduction

"The concept of P2P is increasingly evolving to an expanded usage as the relational dynamic active in distributed networks, i.e. not just computer to computer, but human to human." [22]

Looking in the past, we discover that the most predominant computing mode was centralized processing, where a central computer also called a mainframe, handled all processing. Nowadays we use distributed processing, where a number of computers handle all processing, and although distributed physically, they are connected through a communication network.

P2P technologies give a lot of freedom to their users (e.g. filesharing with people you don't know, that might be on the other part of the world or next door to you) but not without a price : poor security. That is because there are no means of authenticating all peers in a scalable manner and without a central trusted authority. Therefore, the possibility of connecting to a potentially malicious node (although it seems very peaceful and having many sharable resources) is very high. So, are there any viable solutions?

In 2001, Dan Bricklin coined the term **Friend-to-Friend networking** (F2F), refering to computing in local P2P networks, where people know each other and how much they can trust one another. Through some simple techniques, and based on Stanley Milgram's "small world phenomenon" [34] or the "six degrees of separation", such local P2P networks can connect one to another and form a global P2P network, with enhanced security and authetication of users. Simply put, a social network overlay is placed on top of the actual P2P network, thus forming a web of trust.

This thesis's topic is Friend-to-Friend, because even if it is coined since 2001, it still represents a novelty in the field of P2P networks, as it has been overlooked for quite a long time, and has a great potential by solving some P2P security issues. The future of computing will definitely rely if not on pure F2F technologies, but certainly on social connections between users.

Based on the above ideas we structured the thesis as follows : Chapter 1 talks about networks in general, covering up topics like protocol stacks and network architectures (from server based to Peer-to-Peer). Chapter 2 presents the Friend-to-Friend model thoroughly, with goods and bads, and some potential fields of interest. Chapter 3 summarizes the ideas behind WASTE and Turtle, two applications that comply, more or less, to the F2F model. Chapter 4 gives an overview of some cryptographic primitives which are used in our application. Chapter 5 introduces Gazebo F2F – our Friend-to-Friend application.We describe its design and implementation from a high level, and also offer a real case scenario involving two users that interact in order to authenticate one to the other and chat over the insecure network. Chapter 6 covers the tools used for building Gazebo F2F. The thesis ends with some conclusions and directions for further improvements.

Contributions

Current trends in distributed networks (e.g. Peer-to-Peer) show the increasing needs for a more secure computing process if we refer to scientific computing, and for more secure communication protocols if we refer to regular applications that are specially designed to work in distributed networks. Thus, we have focused on writing a new communication protocol that would work in distributed networks, not in the "opened" and regular manner of P2P, but on a new model called *Friend-to-Friend (F2F),* which takes advantage of the social relations that exist between users. This bold objective could not have been made possible though, without a thorough understanding of prior protocols (that converge or are related to the F2F model) like Freenet [A2] [A3], Turtle [A8] [A9], Tarzan [A4], OneSwarm [A5] or WASTE [32].

Our work has materialized in the creation of Gazebo F2F, a protocol that shows both the advantages and shortcomings of the F2F model. As it is shown later in the paper, we choosed to develop this protocol, based on the top-down model : a rigorous modular design, followed by the implementation of each component with regard to the blueprints of the application.

The lack of scientific papers and implementations on the chosen model, was regarded from the beginning as both a disadvantage – because of the undetailed F2F principles, and as an advantage - because of the freedom for innovation based on the given guidelines.

From Gazebo F2F's beginning to this last version (the one that comes with this paper) we concentrated on developing a 'pure' F2F application, putting a lot of time and effort in the creation of the actual mechanism that lets people connect one another in a secure manner, thus future work will focus more on services Gazebo should offer. The most notable novelties it cames with, are : the users' accounts system with local and remote authetication involving "*mobility archives"* and the initial network setup protocol and model (which is later re-used in the connection establishment process between authenticated users).

Last but not least, we present in a detailed manner and propose some new fields of interest and application regarding the F2F model. Of these, we can enumerate computer gaming – based on a possible approach that is described in [A1] (the Hydra project), data storage and streaming, and even the well known filesharing.

Chapter 1

Networks and Network Architectures

1.1 Network Models

1.1.1 ISO/OSI Model [9]

The OSI (Open Systems Interconnection) Reference Model is the standard model for networking protocols and distributed applications. It is divided into 7 layers(Application, Presentation, Session, Transport, Network, Data-Link and Physical) each having a certain functionality (Figure 1). Each layer is a collection of similar logical functions providing services to the layer above it and receiving services that are below it.

The user resides at the highest level (Application) and communicates with others at the same level.The top 4 layers are called "Host layers" being responsible for things that are related to the application and the lower 3 - "Media Layers" – which are responsible for data conversion into physical signals and for their transport to end-users through physical media.

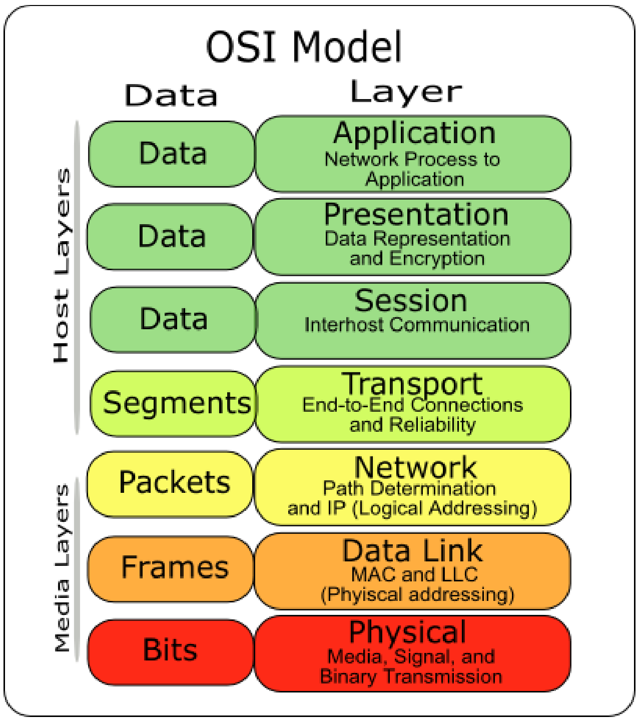

OSI Model
Data
Layer
Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)
Frames
Data Link
MAC and LLC (Phyiscal addressing)
Bits
Physical
Media, Signal, and Binary Transmission

Figure 1 : ISO/OSI Layers' functionality [10]

1.1.2 TCP/IP Model

The TCP/IP model (***T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol)* or Internet Protocol Suite is the collection of communications protocols used to connect hosts on the Internet and similar network. Similar to the OSI model it can be viewed as a set of layers (Figure 2).

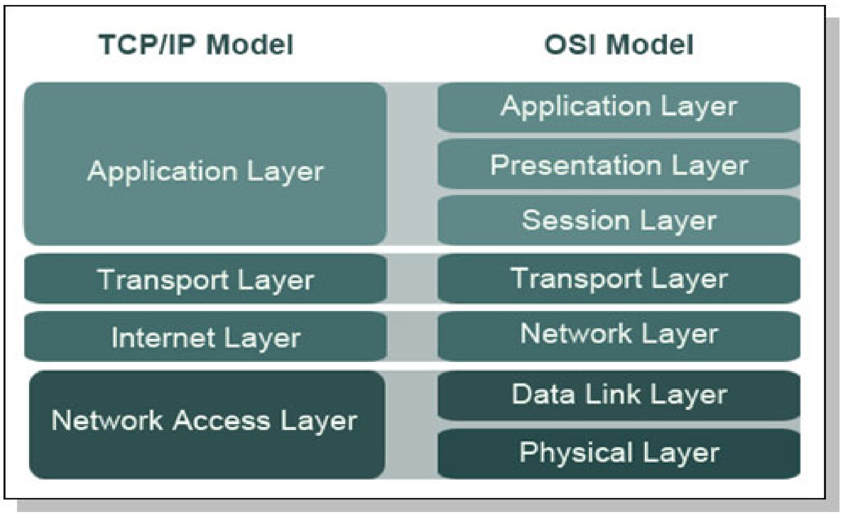


Figure 2: TCP/IP and OSI models comparison [11]

There are 4 of them (Application , Transport, Internet and Network Access), this model being intentionally designed simpler. Each layer is responsible for solving certain problems ; upper layers are closer to the user, managing abstract data , and rely on lower layers to transform this data into physical signals and send it through the network to other users. "The TCP/IP design generally favors decisions based on simplicity, efficiency and ease of implementation." [9]

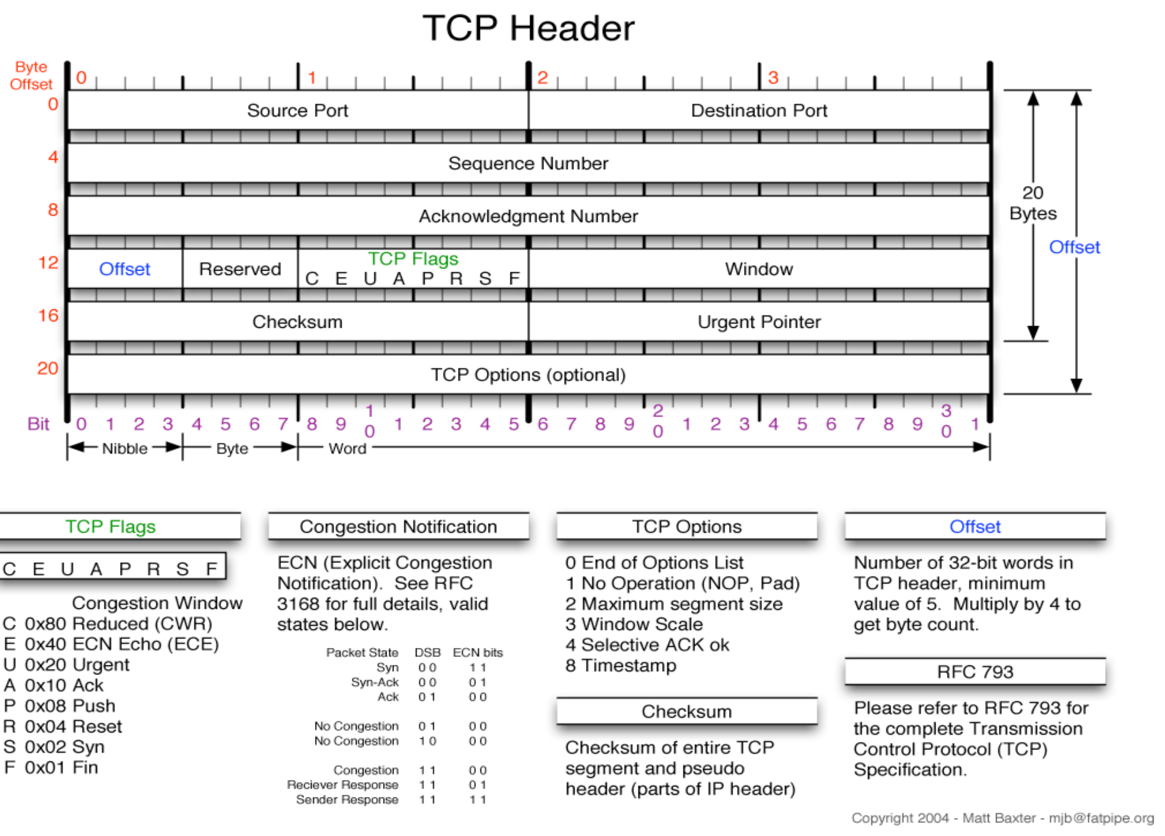
1.2 Network Protocols

TCP and IP are the core protocols of the Internet Protocol Suite.

1.2.1 TCP [12]

TCP operates at a higher level (Transport Layer), providing reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. It creates "user datagrams" using data received from Application Layer to which it appends a particular header (Figure 3). TCP communicates program to program (and not mchine to machine) through 16 bits identifiers called **ports**. TCP creates a virtual circuit between the 2 ports thus establishing the connection, it transfers user data throughout the session and in the end it terminates the connection. The combination between an IP address (computer location) and a port (mapped to the application program process) is called a **socket address**.

Among its other management tasks, TCP controls message size, the rate at which messages are exchanged, and network traffic congestion.



TCP Flags

C E U A P R S F

Congestion Window

C	0x80	Reduced (CWR)
E	0x40	ECN Echo (ECE)
U	0x20	Urgent
A	0x10	Ack
P	0x08	Push
R	0x04	Reset
S	0x02	Syn
F	0x01	Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Reciever Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0	End of Options List
1	No Operation (NOP, Pad)
2	Maximum segment size
3	Window Scale
4	Selective ACK ok
8	Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

Figure 3 : TCP Header [13]

1.2.2 IP [15]

IP (***I****nternet* ***P****rotocol*) works at the Internet Layer; it contains addressing and some control information that allow it to route packets from source host to destination host (Figure 4).
IP provides connectionless, best-effort delivery of datagrams through an internetwork and fragmentation and reassembly of those datagrams.It does not add safety , flow control or error recovery with regard to lower levels.Datagrams sent by IP could get lost, corrupted or duplicated . Solving these issues are tasks for higher layers like Transport and Application.

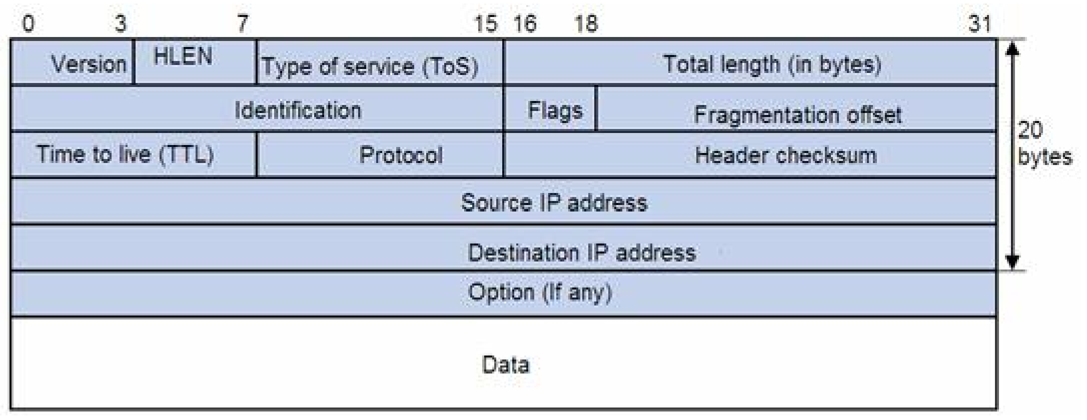


Figure 4 : IP packet format [14]

"Perhaps the most complex aspects of IP are IP addressing and routing. Addressing refers to how end hosts become assigned IP addresses and how subnetworks of IP host addresses are divided and grouped together. IP routing is performed by all hosts, but most importantly by internetwork routers, which typically use either interior gateway protocols (IGPs) or external gateway protocols (EGPs) to help make IP datagram forwarding decisions across IP connected networks."[15]

1.3 Network Architectures

"A network is either a peer-to-peer network (also called a workgroup) or a server-based network (also called a client/server network)." [16]

1.3.1 Server Based Networks

1.3.1.1 Overview

These computer networks can be described as having a central super-computer (called **server** or host-computer) that receives request and provides various services to **clients** (remote processors) (Figure 5). Servers are powerful computers or processes , dedicated to providing certain services like: managing databases (database servers) , disk drives (file servers), network traffic (network servers), printers (print servers), e-mails (mail servers), applications (application servers) and so on. Clients are PCs or workstations that provide an interface to allow a computer user to request services of the server and to display the results the server returns. Usually a client software process initiates a communication session while the server waits for requests.In a server-based network, users may have accounts and passwords to log on to the server and to access shared resources. Server operating systems are designed to handle the load when multiple clients access server's resources.

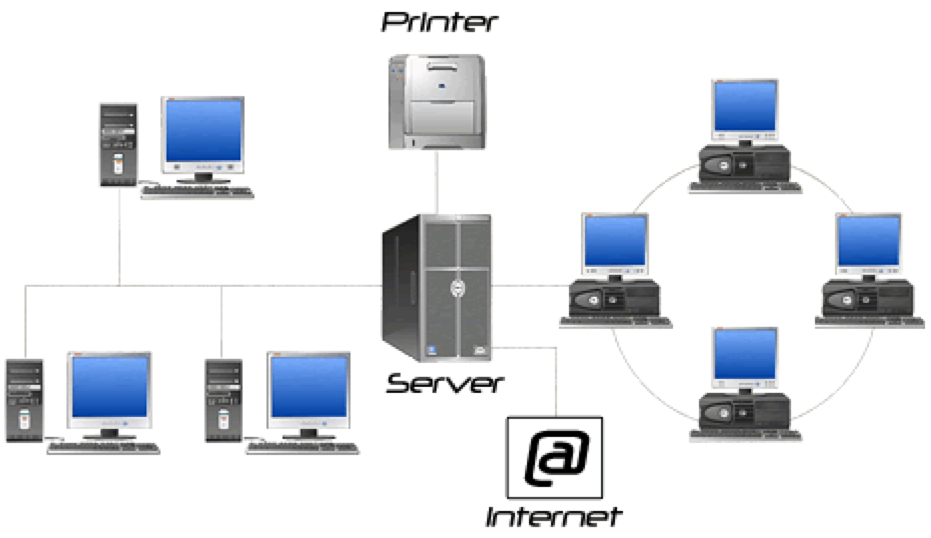


Figure 5 : Server based network [17]

1.3.1.2 Server Based Architectures [18]

2-Tier Architecture

The most basic type of client-server architecture employs only clients and servers.It describes a system where the client requests resources and the server provides them responding directly to the request.It means that the client acts as one tier and the server with its own applications acts as another tier.(Figure 6)

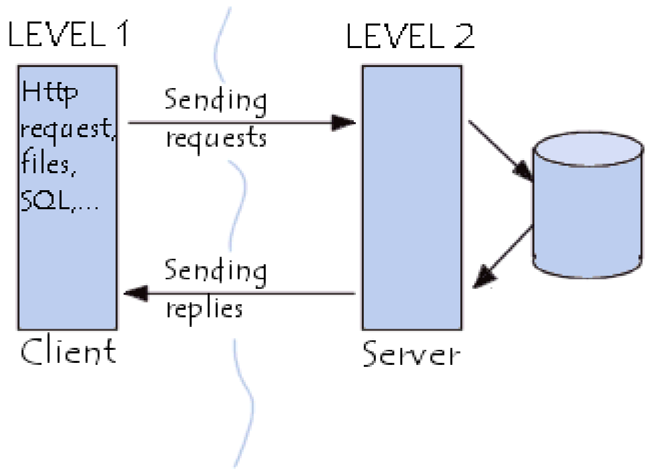


Figure 6 : 2-Tier Architecture [19]

3-Tier Architecture

Compared to a 2-Tier architecture the 3-tier has an intermediary level called a middleware.
The client, which requests resources, has a user interface for presentation . It communicates with the middleware (application server) which provides the requested resources by calling a data server on which reside the wanted data.(Figure 7)
This architecture has a greater degree of flexibility, increased security and increased performances than the previous one.

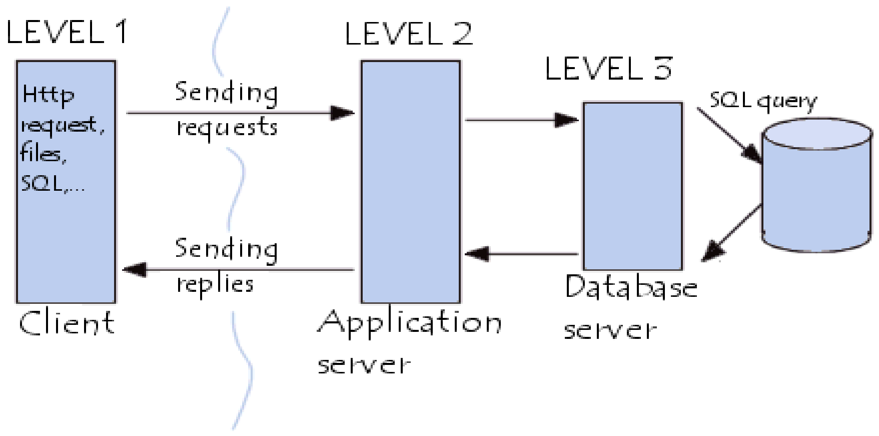


Figure 7 : 3-Tier Architecture [20]

Multi-Tiered Architecture

"In 3-tier architecture, each server (tier 2 and 3) performs a specialised task (a service). A server can therefore use services from other servers in order to provide its own service. As a result, 3-tier architecture is potentially an n-tiered architecture."[18] (Figure 8)

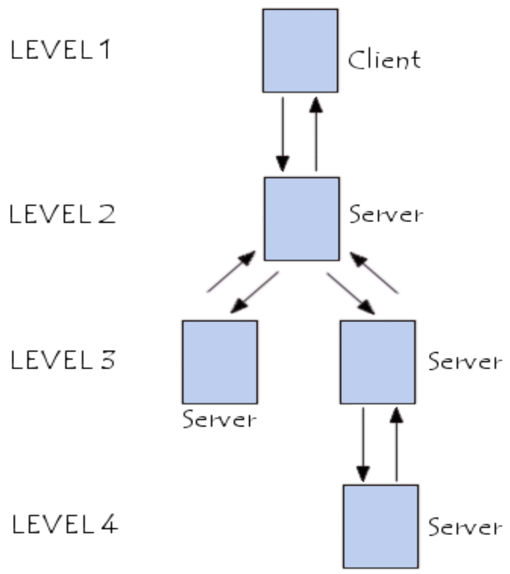


Figure 8 : Multi-Tiered Architecture [21]

1.3.1.3 Advantages and Disadvantages

Advantages

- Easier data management because files are in one location
- Ease of maintenance - it is possible to repair , upgrade or replace a server with the client being unaware and unaffected by changes
- Greater security as data is stored in a single place – servers can control better the access to resources
- The update of a piece of information/data is much easier as it is changed in a single place
- The server hardware is designed to resolve requests from clients quickly
- All data are processed on the server so this reduces network traffic and also users' need for high-performance workstations
- Increases productivity

Disadvantages

- Traffic congestion - if the number of simultaneous client requests increases
- Lack of robustness – if a critical server fails then users' requests could not be fulfilled until the server is put back on track
- Server farms are quite expensive and as they grow they are harder to maintain

1.3.2 Peer-to-Peer Based Networks [22]

1.3.2.1 Overview

"A ***P**eer-to-**P**eer* (or P2P) computer network uses diverse connectivity between participants in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide the core value to a service or application. P2P networks are typically used for connecting nodes via largely ad hoc connections. Such networks are useful for many purposes. Sharing content files (see file sharing) containing audio, video, data or anything in digital format is very common, and real time data, such as telephony traffic, is also passed using P2P technology." [22] (Figure 9)

"The term peer-to-peer is used to indicate a form of computing where two or more than two users can share files or CPU power. They can even transmit real time data such as telephony traffic on their highly ad hoc networks. Interestingly, the peer-to-peer network does not work on the traditional client-server model but on equal peer nodes that work both as "clients" and "servers" to other nodes on the network." [24]

Peer :
"A pure P2P network does not have the notion of clients or servers but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network." [22]

Node :
"In networks, a processing location. A node can be a computer or some other device, such as a printer. Every node has a unique network address, sometimes called a Data Link Control (DLC) address or Media Access Control (MAC) address." [25]

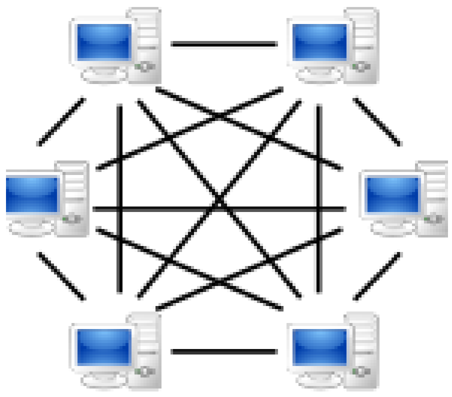


Figure 9 : Peer-to-Peer based network [23]

1.3.2.2 Classifications

Peer-to-peer networks can be classified in many different ways, from what they can be used for (file sharing, media streaming, online chat , software publication and distribution, etc) to their degree of centralization. From the above definitions we observe two approaches for this architecture : **pure** and **hybrid**.

In 'pure' P2P networks peers are equal one to another, acting as both client and server - therefore no specialised server is needed to control the connections between peers. Nodes have to organize themselves based on what information they have about the network and on interaction with locally reachable nodes (called neighbours). Data is distributed across multiple peers (Figure 9).

There are many types of 'hybrid' P2P systems , most of them having a central server that keeps information about peers and respond to requets for that information. Although data is distributed across peers, the server has information about it and makes it easy for peers to search for specific sharable resources (Figure 10).

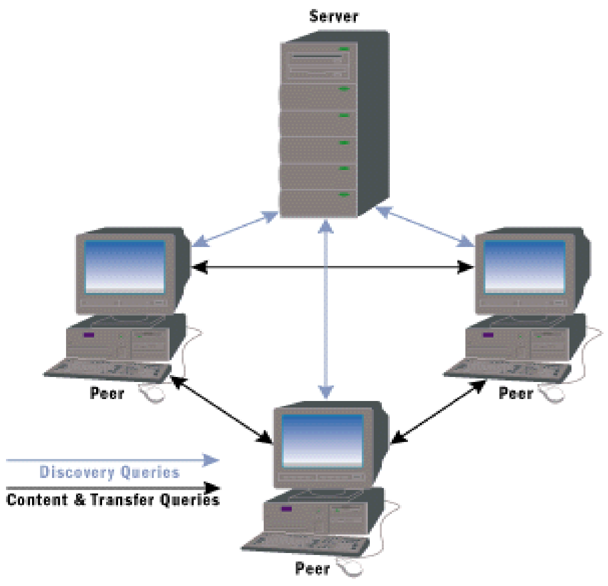


Figure 10 : Hybrid P2P network [26]

"An **overlay network** is a computer network which is built on top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. For example, many peer-to-peer networks are overlay networks because they run on top of the Internet. Dial-up Internet is an overlay upon the telephone network." [27] (Figure 11)

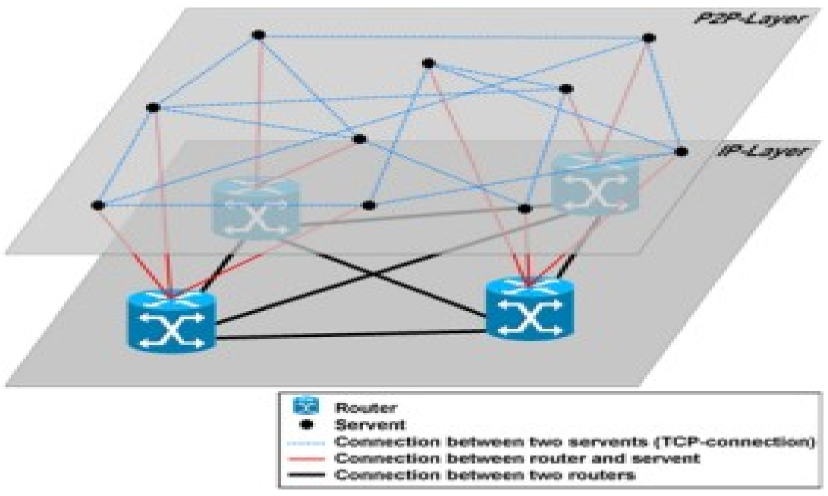

P2P-Layer
IP-Layer
Router
Servent
Connection between two servents (TCP-connection)
Connection between router and servent
Connection between two routers

Figure 11 : Overlay network concept [30]

"The P2P overlay network consists of all the participating peers as network nodes." [22]
Between every two nodes that know each other exists a link, so if a peer "discovers" another peer in the network then peers that have knowledge of this first peer will also have knowledge of the last "found" peer. Depending of how nodes are linked one to another in the overlay network we can classify P2P networks as being **unstructured** or **structured**.

Unstructured P2P networks :

In unstructured systems there is no control over the network topology or resource placement. When a peer joins the network it connects freely to any other peer(arbitrarily selects its neighbours). Over time its knowledge about the network may grow and connect to other peers in its peer list (Figure 12).

If such peers want at some point to look up for a certain data then they must flood that query to as many peers as possible for a higher finding success ratio. The problem is that the query may not get resolved (because peer's knowledge of the network is too little) especially if the peer looks for rare data (compared to popular/highly-rated data which could be found at several peers). Also flooding may cause a lot of traffic over the network so this search method is quite inefficient although many popular P2P networks are unstructured. Due to absence of topological constraints these kind of networks perform well and need little maintenance in dynamic environments where peers join and leave the network frequently.

As shown before the main drawback with these systems is the underdeveloped **routing system**. The two fundamental routing operations are **flooding** and **random walks**. With flooding a query is sent to every neighbour which send the query to their neighbours and so on until the maximum hop counter is reached. In random walks the query is forwarded to a randomly picked neighbour. A random walk is costly effective in terms of packets sent over the network but the search results may be very poor whereas flooding is the opposite. In flooding many messages are useless because of redundant retransmission as neighbours of peer's neighbours may be neighbours of the peer. [A11]

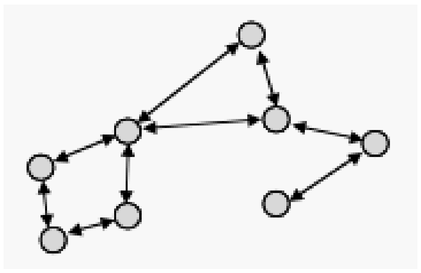


Figure 12 : Unstructured P2P network [29]

Structured P2P networks :

"Structured P2P network employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired file, even if the file is extremely rare. Such a guarantee necessitates a more structured pattern of overlay links. By far the most common type of structured P2P network is the distributed hash table (DHT), in which a variant of consistent hashing is used to assign ownership of each file to a particular peer, in a way analogous to a traditional hash table's assignment of each key to a particular array slot." [22]

1.3.2.3 Fields of Interest and Application

- Bioinformatics: As it deals with large datasets P2P networks are an attractive solution for example for carrying out tests for drug candidates .
- Education : Because of the fast distribution and large storage space P2P networks have been implemented in academic environments for facilitating file sharing among educational institutions globally.
- Military : "The U.S. Department of Defense has already started research on P2P networks as part of its modern network warfare strategy." [22]
- Business : In the past years a lot of investments have been made in P2P networks. "Besides File Sharing, companies are also interested in Distributing Computing, Content Distribution, e-marketplace, Distributed Search engines, Groupware and Office Automation via P2P networks. There are several reasons why companies prefer P2P sometimes, such as: Server space and bandwidth saving; Real-time collaboration—a server cannot scale well with increasing volume of content; a process which requires strong computing power; a process which needs high-speed communications, etc. At the same time, P2P is not fully used as it still faces a lot of security issues." [22]
- TV: "Quite a few applications available to delivery TV content over a P2P network (P2PTV)" [22]
- Telecommunication: Peoples' demands for clearer real-time voice and video streaming have increased globally.

1.3.2.4 Security Risks and Threats [31]

Because P2P applications are installed on local client machines that are directly linked to the Internet, those nodes are widely exposed to malicious attacks that are used for many reasons - from blocking the access to the network and implicitly to resources (DOS attacks) to gaining personal advantages (Sybil attack). Often P2P application protocols are stealthy, encrypting themselves or tunneling undetected through open ports, thus, giving the opportunity to abuses. Therefore we should outline the elements that must be considered in order to protect ourselves : connection control, access control, operation control, anti-virus, and protection of the data stored on our machines.

"Over and above the potential for productivity loss and bandwidth and storage resource abuse through employee usage of unauthorized software, P2P networks can:

- Open up back doors into the network, allowing hackers direct access to corporate assets and putting the organization in breach of privacy legislation
- Enable the exchange of copyrighted material, rendering the corporation vulnerable to breach of copyright lawsuits
- Overload network bandwidth with unauthorized file sharing activities
- Allow bundled adware applications to be installed on the network without the user's knowledge" [33]

We outline some threats that P2P applications are vulnerable to:

External Threats :

- Theft – companies can lose a great amount of money due to stealing of source code, sensitive information (research projects, new pharmaceutical drugs) or new movies and music
- Bandwidth Clogging – P2P file-sharing applications can create congestion in network traffic due to large file transfers, resulting in a higher response time for internal users which usually leads to income loss
- Bugs - if the P2P software that is installed on client systems has bugs it may expose the network to great security risks which vary from external attacks to system crashes
- Encryption cracking – by taking a lot of desktop computers and linking them togheter results in a large amount of coumputing power and distributed processing could be used to crack strong encryption (e.g.: DES – in 1999)
- Trojans, viruses and backdoors – A user may download and install a broken P2P application that looks valid but instead install a backdoor or trojan on its computer. Also with file-sharing a user may download infected files that could harm or even crash its system
- Confidentiality - some P2P applications give clients direct access to files that are stored on user's hard drive , so gaining knowledge of the operating system and connecting to folders that are hidden shares a malicious peer can get unauthorized access to sensitive information
- Authentication – when using P2P you have to be able to determin who is accessing your resources and if he is authorized to do it

Internal Threats:

- Interoperability – infrastructure diversity (different platforms, different systems, different applications working together in a given infrastructure) poses a series of security issues associated with interoperability
- Private businesses on public networks – many companies conduct private businesses on public networks which may lead to various security risks that can put the businesses in jeopardy
- Adding and removing users – if there does not exist a feasible policy for user addition and deletion, the system may get vulnerable especially to attacks from former users that know how the system works
- General security – P2P systems share many security problems like possible data tampering, unreliable transport , latency problems and so on
- The human factor – should always be taken into consideration when security is an issue. There will always be malicious users that will try to gain as many advantages as possible from P2P systems and there will always be users that will make wrong decisions regarding the accesibility of their files (which may lead a (novice) user to unwillingly share sensitive documents or even the whole hard disk)

Next , we will look at some specific P2P attacks: [A10]

Rational attacks :

For P2P to be effective, peers should cooperate, but usually they tend to represnt their own interest, trying to consume system's resources and minimize the consumpsion of own resources. Such nodes will not attempt to disrupt routing, censor data or corrupt the system unless the access of the node for shared resources increases. If many nodes get self-interested and refuse to contribute with resources the system may destabilize.Therefore P2P systems must be designed robust against these kind of attacks if they want to be successful.

File poisoning :
This attack's goal is to replace a valid file with a fake and useless one. Sometimes the music industry releases a lot of fake audio files in order to protect their copyrighted materials. Messages passing through a malicious node could get poisoned , thus giving a fake file a higher score than the real data , thus getting downloaded more time and discouraging in this way, users that are looking for copyrighted data. Nowadays, this attack is successful because clients are not willing to share their data (rational attack) , corrupted files are not removed fast from machines and usually users stop downloading if it the download looks stalled.

Sybil attack :
In this attack a single malicious identity forges multiple identities and presents them to the system as valid, thus taking control on some part of the network. As those fake identities gain the trust of the peers , they may gain more rights and responsabilities over certain files and may choose to corrupt them.Without a central authority that controls the identities this attack is hard to stop.

Eclipse attack :
Once an attacker has gained the control over some strategic routing paths (usually by Sybil attacks) he can launch an eclipse attack. This means that he can separate the network in subnetworks. Thus, sooner or later messages will pass from one subnetwork to the other,the malicious user having many ways of attacking the system in a more efficient manner : he can disturb routing and make it inefficient, drop all messages and completely separate the two subnetworks or replace good files with corrupted ones.

1.3.2.5 Advantages and Disadvantages

Advantages :

- All peers provide resources (bandwidth, storage space , processing power, etc)
- In a pure P2P network there is no single point of failure – if a peer is down the rest are still able to communicate
- In a pure P2P network there is no need for maintenance of the system as it is self-maintained or in a hybrid system less maintenance is needed than in a server based network
- P2P prevents network congestion and traffic overload by distributing data and balancing requests ocross the network, things that could not be possible in a server based network
- P2P networks scale well as new nodes join the network, thus, due to a lack of centralized control and because of peers interacting with each other

Disadvantages :

- Many application need high security which is not provided by current P2P solutions
- A server based network will run as long as the provider of those servers' services wants to, whereas in a P2P network if the users start abandoning the system services won't be available to anyone
- Searching data in a pure P2P system is quite expensive (think at flooding) - hybrid architectures circumvent this

Chapter 2

Friend-to-Friend

2.1 Overview

"A friend-to-friend (or F2F) computer network is a type of peer-to-peer network in which users only make direct connections with people they know." [43]

In [44] Dan Bricklin coined the term **Friend-to-Friend networking** (F2F) describing it as follows : "In very small areas, though, the technique of knowing everybody and how much you can trust them, and being able to revoke access or change rules, may be a reasonable simplification. Like the use of centralized servers to make file sharing more manageable (and easier to implement, upgrade, etc.), explicit lists of network nodes you control in some way and will trust (and how much) can lead to useful systems without central control. "

Nodes in F2F overlays can forward files or file requests (for example) in an anonymous way i.e without telling the receiving node from whom they got the file or to whom it is delivered. Users in such an anonymous network can't find out who else is participating beyond their circle of friends.

"F2F software is not an F2F network by itself; such software can be used to participate in an existing F2F network, or to launch a new F2F network." [43] Applications that use public servers to communicate to their friends or use private servers to access resources are not F2F networks.

2.2 Fields of Interest and Application

Online reputations could be constructed and verified in anonymous F2F overlays by rating for example files, by multiplying their current ratings with the provider's reputation. If a file is broken or is of poor quality its rating can be manually decreased or the file could even be removed.

Third party storage like file or web servers that are public to everyone in the network could be used for faster downloading and also preventing a user's ISP from logging the addresses of his friends and blocking them through a firewall due to high network traffic. [A7] talks about storing data reliably in distributed storage infrastructures in P2P systems, where nodes choose their neighbours based on existing social relationships.

Because users make direct connections only to people they know , filesharing in F2F networks is a bit tricky because a node's circle of friends may not be very "rich" in resources and that node may not find the files it looks for. Therefore a new technique is needed to share files with unknown people but without the identities revealed. Cryptic6 [46] solves this problem [47] by refering to Stanley Milgram's "small world phenomenon" [34] or the "six degrees of separation" where it is shown that most people in the world are connected to each other by a chain of six friends. Turtle F2F[48] (see 3.2) implements a similar idea; a node broadcasts a query for a file to its friends and those friends to each of their friends and so on until the file is found (or not) . If the file was found at more than one hop (not at that node's friends) and the node wants to download it, then, the file will be transfered on the query's route that found the file, backwards. The node won't have knowledge of the route, thinking it is downloading from a friend of his, thus, keeping identities hidden.

New functionalities can be implemented in F2F applications to take advantage of the fact that friends in a F2F network are often spatially close one to the other. Metropolitan network connections are very fast so audio and video streaming between friends can me made possible. Also, tedious tasks or group projects can be completed in a collaborative way between friends of some degree (i.e not necessarily friends in the true sense of the word but buddies or schoolmates) (collaborative software are meant to produce "output" and don't necessarily rely on social links or prior knowledge between peers). Social meatings could take place in F2F networks, building stronger relations between participants. Such close friends could be easily convinced to leave their F2F application opened for longer periods of time; someone could take advantage of the nowadays enormous computing power given by a single computer and start a distributed task in his network of friends, task that could run in the processors' idle time.

Computer gaming could also benefit from the F2F model. Usually when a company releases a (massively) multiplayer game on the market, it has to be prepared with the underlying infrastructure (especially game servers) to support all players that want to play that game. The network model used is client-server , all game traffic passing through that company's servers for routing and actions validation(so that players don't cheat). If the game gains more and more supporters, then a lot of money would have to be invested in more infrastructure(which may be problematic for small firms) and network/server maintenance would become very difficult.A solution for that would be :
Change the network model to hybrid P2P such that public servers only manage accounts and new games creation. Users would access the public server only for authentication and for choosing a game in which to play. They would thus make directly connections to authenticated nodes . Session keys could also be assigned by servers before a new game starts. Actions validation could be done by the application on each users' computer and re-tested when actions are received by peers. Nodes are often closer one to another than to the game servers so games' latency times would decrease astonishingly due to this spatial locality.
An approach to this model has been made by the Hydra project in [A1].

2.3 Security Issues

F2F networks are fully protected against Sybil attacks (see 1.3.2.4) because connections are made based on social links. A potential malicious friend can't pretend having more than one account and even if he claims he has, he does not know how the network looks like from his friend's point of view.

Also file poisoning and rational attacks (see 1.3.2.4) are less likely to happen in F2F networks than in regular P2P networks, mainly because of the social links and interaction that exists between nodes and the reputation that is wanted to remain intact in front of friends.

By exchanging secret keys face-to-face, man-in-the-middle attacks could be totally avoided and secure communication over insecure channels could be achieved.

2.4 Advantages and Disadvantages

Advantages :

- Improved security - cryptographic keys can be exchanged face to face thus avoiding man-in-the-middle attacks
- Once a node knows all IP addresses of his friends he can block(through his firewall) all other IPs that try to access that application's port
- Malware can be avoided using strong reputation networks
- Fewer leechers – a user is more inclined to act responsibly when using a friend's resource like bandwidth

Disadvantages :

- Difficult setup - each node has to connect manually to others in the network
- Poor services – not enough friends are motivated to run the application for a long time so the shared resources (files, computer cycles, etc) can be hard to find

Chapter 3

Related Work

A node in an F2F network requires a lot of effort to set up and maintain, because all peers must be connected manually , and it may be problematic if a user wants to try several such applications. Mainly because of this, there are not many pure F2F applications, but we will try to present two, that comply to the idea of F2F.

3.1 WASTE

"WASTE is a P2P and F2F protocol and software application developed by Nullsoft in 2003 that features instant messaging, chat rooms and filesharing capabilities. After its release, WASTE was removed from distribution by AOL, Nullsoft's parent company. Several developers have modified and upgraded the WASTE client and protocol. The SourceForge edition is considered by many to be the "official" development branch, but there are several forks. " [32]

WASTE enables the creation of decentralised and secure private networks.(Figure 13) Users share public keys one to another and then connect to them manually by providing an IP address and a port number. In a subnetwork at least one node has to be "public"(be able to accept incoming connections), because if two firewalled computers can't establish a connection then a public node is used to route the traffic between the two.

Usually a password/network name, is assigned to each subnetwork to avoid collisions. There also exist networks without a passphrase called "nullnets". Because of that, nullnets can merge, making it a great place for filesharing and open discussions.

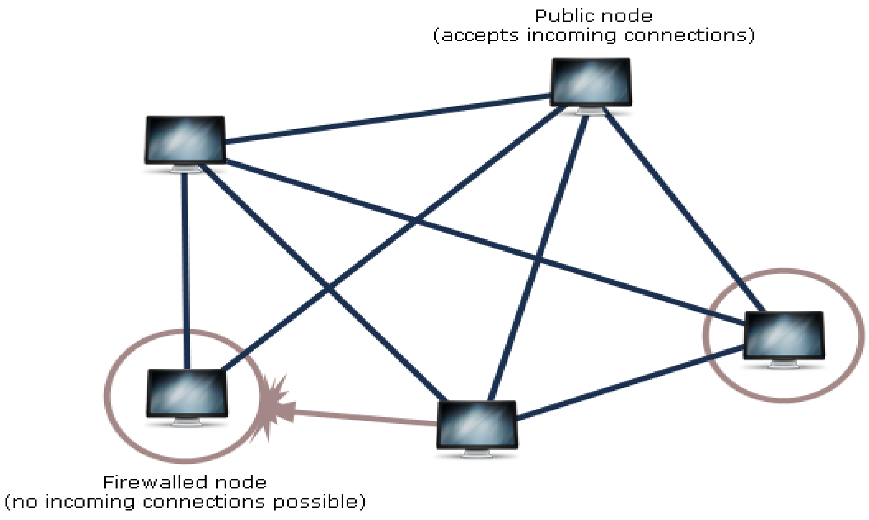

Public node
(accepts incoming connections)
Firewalled node
(no incoming connections possible)

Figure 13 : Decentralized mesh [28]

Among other functionalities , WASTE can create private and public chat rooms, browse and trade files between users. Its file searching feature is very fast because it connects small groups(10-50 users) that resolve a query faster than in larger, public groups.
Along with the trade of RSA public keys (which makes the communication secure) , WASTE has some interesting features that make it anonymous , like obfuscation of the protocol (making it difficult to detect WASTE is being used) or "saturation" which adds random traffic, making traffic analysis difficult.

WASTE does not have only strengths, but shortcomings too. The initialisation process may be too difficult for a regular user because public keys have to be traded and in some cases port forwarding on the firewall may have to be manually enabled. If a user has gained access to a network then it is hard to kick him out of it; all other users have to "migrate" and recreate the initial network without one or more "inconvenient" user(s). As shown before , at least one node has to be "public". If , after some time the network "loses" that public node (or if functioning for some time, nodes in that network change their IP addeesses), all other nodes that connected through it must contact each other (by mail , instant messaging, face-to-face, etc) in order to reconnect by some means. Therefore a minimal coordination has to take place in order for a network to "survive" for a long time.

3.2 Turtle

"Turtle is a free anonymous peer-to-peer network project being developed at the Vrije Universiteit in Amsterdam, involving professor Andrew Tanenbaum." [49]
Technically it is a F2F network, designed for censorship resistance. Users connect one another based on real life social links. Turtle comes with a novelty when considering the exchange of keys. The key agreement protocol bases itself on exchanging personal questions - the answers expected are assumed to be known by the other ends and not by any eavesdroppers.

Turtle F2F is mainly intended for the safe sharing of sensitive information, but it supports file searching over the network of friends, friends of friends and so on. Searches are flooded through the network, the results being forwarded back along the reverse path. Communication between nodes uses virtual circuits tunneled through TCP connections. Data of arbitrary length can thus be carried over the network in an anonymous manner. Queries and query hits are also sent over virtual circuits, the virtual circuit architecture being capable of supporting other applications, including real-time communication.

Information exchange is done over secure channels, so linking a query initiator to a responder can only be made by analyzing the traffic, but with little changes in the protocol Turtle could protect itself against traffic analysis. As shown in section 2.3, F2F networks are immune to "Sybil" attacks and Turtle F2F is no different. Security break in one node only affects that node's friends in their network interactions – this fail-mode property is called "confined damage". Also, because of this "web of trust" between users, DoS attacks and various malicious behaviour of the network are less likely to happen than in regular P2P networks. [A8] [A9]

Chapter 4

Cryptography

"*Cryptography* is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Cryptography is not the only means of providing information security, but rather one set of techniques." [B1]

Next, we outline public-key and symmetric-key cryptography, key agreement schemes and present one protocol from each class.

4.1 Symmetric-key Cryptography

4.1.1 Overview

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key. Informally, the raw data (called **plaintext)** is encrypted with a key, the resulting (scrambled) data (called **ciphertext**) is then sent out-of-band or on the media to the destination, where it must be decrypted with the same key it was encrypted to produce the initial message (Figure 14) .

A great advantage of symmetric-key algorithms is their speed because most of them use simple mathematical operations like addition or operations of bits. The main drawback is related to key management, because the same key that encrypts the message must also decrypt it , thus , the key has to be shared between the sender and the receiver(s) of the message.

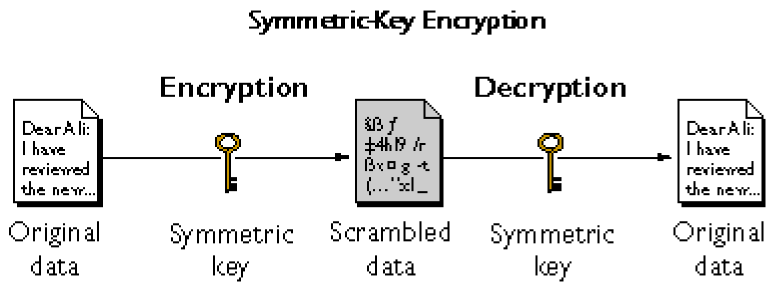


Figure 14 : Symmetric-key encryption [35]

Symmetric-key ciphers can be classified as **block** ciphers or **stream** ciphers.

A block cipher takes as input a block of raw data and a key and outputs an encrypted block of the same size. Usually the message is longer than the block size, so it must be split in many chunks of the size of a block."The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs which have been designated cryptography standards by the US government. Despite its deprecation as an official standard, DES (especially its still-approved and much more secure triple-DES variant) remains quite popular; it is used across a wide range of applications, from ATM encryption to e-mail privacy and secure remote access." [36]

Stream ciphers create a random key material long enough to combine each bit/character from the plaintext with each bit/character from the key material. "The output stream is created based on a hidden internal state which changes as the cipher operates . RC4 is a widely used stream cipher." [36]

4.1.2 AES

"Advanced Encryption Standard (AES) is an encryption standard adopted by the U.S. government. The standard comprises three block ciphers, AES-128, AES-192 and AES-256, adopted from a larger collection originally published as Rijndael. Each AES cipher has a 128-bit block size, with key sizes of 128, 192 and 256 bits, respectively. The AES ciphers have been analyzed extensively and are now used worldwide, as was the case with its predecessor,the Data Encryption Standard (DES)." [37]

This cryptosystem is based on doing calculations in the Galois Field (GF(2^8)) with the irreducible polynom $f(x)=x^8+x^4+x^3+x+1\in\mathbb{Z}_2[x]$. The fix block size is of 16 bytes, AES operating on a 4×4 array of bytes, called **state**.

AES algorithm has 4 main procedures on which it is based :

- `AddRoundKey` - each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule
- `ShiftRows` - cyclically permutes the rows of the state a number of positions
- `SubBytes` - a non-linear substitution which operates independently on each field of the state which replaces a byte with another one according to a lookup table
- `MixColumns` – considers each column of the state as a polynom over GF(2^8) ; it combines the four bytes in each column

A high-level description of the algorithm could be the following:

- Expand the key using Rijndael's key schedule
- AddRoundKey
- For a number of rounds(10 for 128-bit keys, 12 for 192-bit keys or 14 for 256-bit keys) do :
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey
- SubBytes
- ShiftRows
- AddRoundKey

Only the first step works with the key , the other ones and especially the 4 procedures apply to the input matrix (state).

"As of 2006, the only successful attacks against AES implementations have been side-channel attacks.The National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for US Government non-classified data. In June 2003, the US Government announced that AES may be used to protect classified information." [37]

4.2 Asymmetric-key Cryptography

4.2.1 Overview

Public-key cryptography is a cryptographic approach that uses asymmetric key algorithms instead of or in addition to symmetric key algorithms. In asymmetric key algorithms the key used for encryption is not the same key used for decryption. Each user has a pair of cryptographic keys : a **public key** and a **private key**. The private key is kept secret, while the public key may be widely distributed.

Public key cryptography can be used for **public key encryption**(a message is encrypted with a user's public key and is decrypted by that user using its private key, key that should be known just by him, as this key-pair is just like an identity)(Figure 15) or **digital signatures**(a message is signed with a sender's private key and can be verified by anyone having access to that message and to sender's public key).

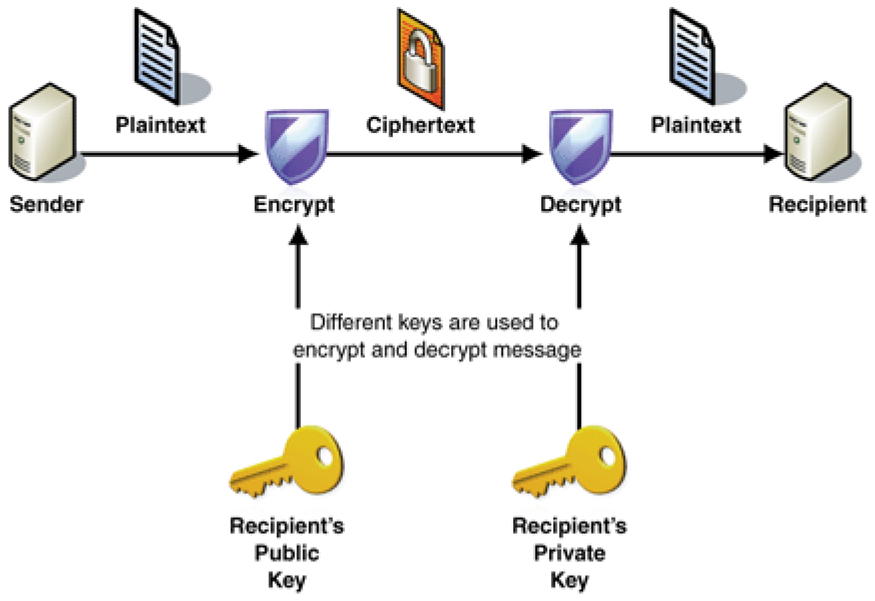


Figure 15 : **Public key data encryption and decryption** [38]

"In contrast to one-time pad (is an encryption algorithm in which the plaintext is combined with a secret random key or pad as long as the plaintext and used only once. A modular addition is typically used to combine plaintext elements with pad elements [39]), no public-key encryption scheme has been shown to be secure against eavesdroppers with unlimited computational power. Proofs of security for asymmetric key cryptography therefore hold only with respect to computationally-limited adversaries, and can give guarantees (relative to particular mathematical assumptions) of the form "the scheme cannot be broken using a desktop computer in 1000 years", or "this algorithm is secure if no improved method of (for instance, integer factoring) is found". " [36]

4.2.2 RSA

RSA is an algorithm for public-key cryptography suited for both signing and encryption.It is widely used in e-commerce protocols and it is believed to be secure because of its long keys and good implementations. Its security is based on the computational complexity of a "hard" problem called "integer factorization of very big numbers".

The cryptosystem looks like this :

- choose p and q two distinct primes and compute n = pq
- compute φ(n) = (p-1) (q-1)
- choose e such that : 1 < e < φ(n) and (e,φ(n)) = 1 (they are coprime)
- determine d which satisfies the congruence relation $de \equiv 1 \, (mod \, \varphi(n))$

For each key K = (n,p,q,e,d) , (n,e) is the public key and (p,q,d) is the private key.

Given a plaintext x we can encrypt it using RSA by calculationg $e_K(x) = x^e \, mod \, n = y$. Thus, y is the ciphertext that has to be sent to destination. There it can be decrypted by calculating $d_K(y) = y^d \, mod \, n$, and if everything went alright the result should be the initial message x.

Nex we present a working example, with small numbers :

Let p = 101 and q = 113 . The n = 101 * 113 = 11413 and φ(n) = (101-1) (113-1) = 11200 . Because $\varphi(n) = 2^6 5^2 7$ e should be picked such that it is strictly less than n and it can not be divided by 2, 5, or 7. So we choose e = 3533 .
Using Euclid's algorithm we compute $d = e^{-1} mod \, 11200 = 6597$.

Next we choose a "plaintext" x = 9726 and compute the ciphertext $y = 9726^{3533} \, mod \, 11413 = 5761$. To decrypt y we compute $x' = 5761^{6597} \, mod \, 11413 = 9726$ which equals the initial message x.

In real world, if p and q are chosen too small then n can be factorized very fast and with p and q known, the cryptosystem is completely broken.

Public-key cryptosystems are much slower than symmetric-key ones because they use expensive computational operations like power raising big numbers, modular multiplication and modular multiplicative inverse calculation, and so on.

4.2.3 Diffie-Hellman key exchange

Diffie-Hellman key exchange is a cryptographic protocol that allows two parties to establish a shared secret key over an insecure communication channel. The key can then be used to securely transport data between the two parties over the insecure channel using a symmetric-key scheme.

We present the protocol along with an example : [40]

We consider 2 parties called Alice and Bob. These two parties agree to use a prime number p=23 and (a primitive root mod p called) base g=5 .

Alice chooses a secret integer *a*=6, then sends Bob $A = g^a \, mod \; p$

- $A = 5^6 \, mod \, 23 = 8$

Bob chooses a secret integer *b*=15, then sends Alice $B = g^b \, mod \; p$

- $B = 5^{15} \, mod \, 23 = 19$

Alice computes the secret $s = B^a \, mod \; p$

- $s = 19^6 \, mod \, 23 = 2$

Bob computes the secret $s = A^b \, mod \; p$

- $s = 8^{15} \, mod \, 23 = 2$

Thus , both Alice and Bob have the same secret key s which they can use to encrypt and decrypt data in a symmetric fashion (Figure 16).

Only a, b and $g^{ab} = g^{ba} \, mod \; p$ are secret. All other values and parameters are sent in clear over the insecure channel.Of course, in real world bigger numbers(a , b and p) have to be picked to consider the cryptosystem secure. The Diffie-Hellman protocol bases its security on the discrete logarithm "hard" problem ("if *g* and *h* are elements of a finite cyclic group G then a solution x of the equation $g^x = h$ is called a discrete logarithm to the base *g* of *h* in the group *G* " [42])

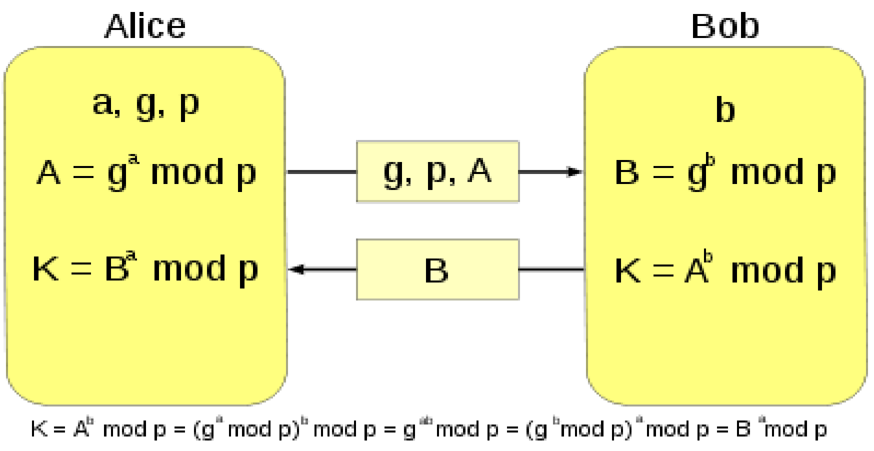

Alice
Bob
a, g, p
A = g^a mod p
K = B^a mod p
g, p, A
B
b
B = g^b mod p
K = A^b mod p
K = A^b mod p = (g^a mod p)^b mod p = g^ab mod p = (g^b mod p)^a mod p = B^a mod p

Figure 16 : Diffie-Hellman key exchange [41]

Chapter 5

Application architecture and protocols specification

Next, we present our Friend-to-Friend application - Gazebo F2F. This chapter will cover its architecture, a developer's guide and an usage scenario. Also, in the end, we will talk about what can be done in future in order to improve the application. Because the source code is publicly available, we will talk less about the actual implementation in the chosen programming language and try to focus more on how the application was built, seen from a higher level.

5.1 Design

Gazebo builds an overlay network (see 1.3.2.2) on top of the actual physical network, where users make direct connection only to people they know and trust. Although the physical network is P2P, adding the overlay as described above, makes Gazebo a Friend-to-Friend application, which creates a new F2F network. Thus, the network topology, as seen by each node, is star ("list of friends"), like in Figure17, and not a full mesh ("chat room"), like in Figure 9.

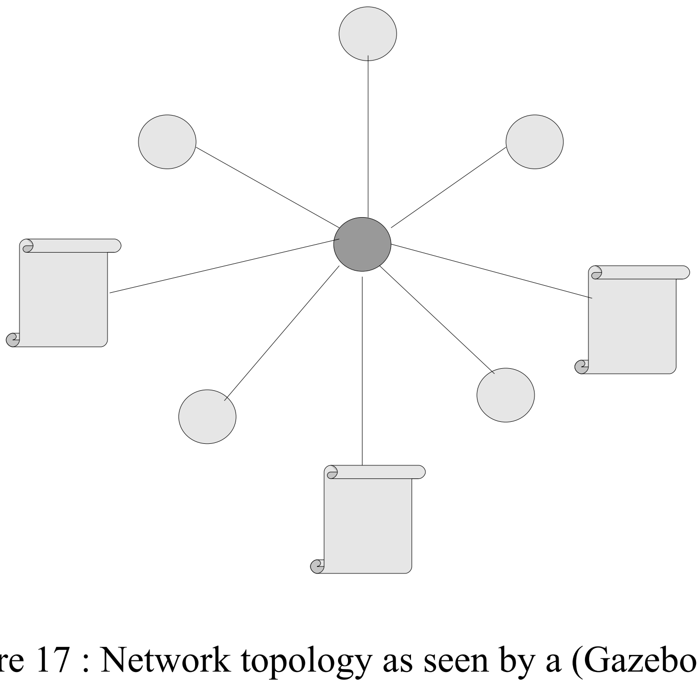


Figure 17 : Network topology as seen by a (Gazebo F2F) node

After a new user has created an account, it holds an asymmetric key pair (see 4.2). After it finds out its friends' public keys, it creates a new certificate for every user he wants to connect to. Then, these certificates are exchanged **out-of-band**, in a face-to-face manner. For two nodes to establish a connection, each must have the other's certificate and the original one (which it created and "shared" with the other node). Such a node that has both certificates is called a **pender** and is depicted in Figure 17 as a certificate. If at some point both penders are online, then, a protocol takes place between the two, in order to authenticate one to the other. If the protocol finishes successfully then each node is called and viewed by the other as a **friend**; in Figure 17 a friend is depicted as a circle.

After the initial setup, each time two friends connect one to the other, a (session) key exchange protocol takes place. This is done because encrypting and decrypting the data in a session in an asymmetric manner is very unfeasible due to the long time spent with these operations. Therefore a session key is agreed and all data is encrypted/decrypted symmetrically (see 4.1) with it.
After the authentication and session key exchange, the two friends can communicate in a secure way across the insecure network.

Gazebo protocol does not share any other sensitive information between nodes, so even if they are friends (in the Gazebo meaning), they are also individualities, and keep their secrets hidden (besides the session key and each other's public keys they do no have information of anything else). Moreover, if friends of some node connect to others, that node does not have information of the network overlay topology, beyond its own circle of friends (Figure 18).

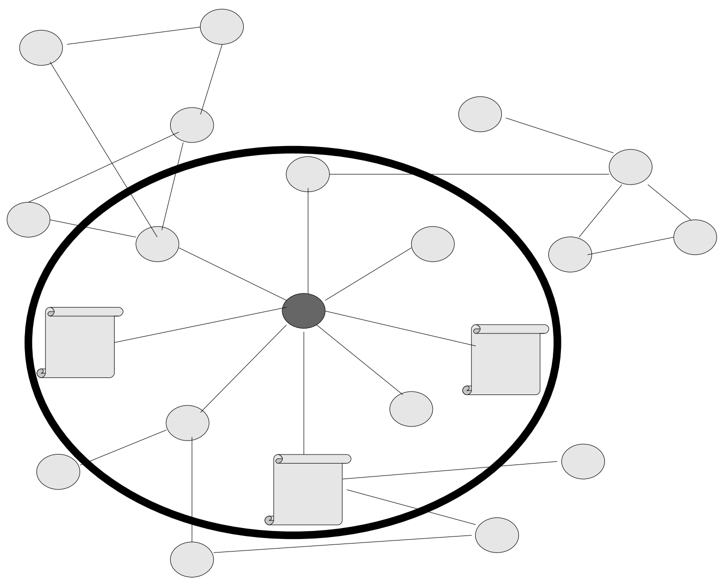


Figure 18 : Circle of friends and network knowledge

Even though the setup may be a bit difficult, the presented model has some interesting security features. As with all "true" F2F networks, this one is also immune to "Sybil" attacks (see 1.3.2.4). If one node in the network is broken, it is a single point of failure and does not affect others. At most, it might affect some friends of the broken node by trying to send malicious files, but it is very unlikely to have a devastating effect over the good functioning of the network. Moreover, the nodes do not have any secret information of the others, so again, the network will not be affected. More than that, because of the web of trust that exists between users, friends will try to act more responsible, with respect to resources and to the network itself, so DoS attacks or any other malicious behaviour over the network are highly improbable.

5.2 Developer's guide

In the previous section we saw from a high level, the F2F model that Gazebo implements. Next, we will try to take a more in depth look at the actual implementation. The "story" will follow, more or less, the actual steps that were made during the implementation. From now on, we will mark mentions of any kind with (!!!).

(!!!) Gazebo's internal structure is shown in Figure 19, with the mention that this is how it was initially designed. The software version that comes with this paper was built entirely based on this architecture, but it does not implement (yet) services like collaborative work, file service or video streaming, nor the forwarding component.

5.2.1 Account management

If a user wants to use Gazebo, it will definitely need an account. The novelty that Gazebo comes with, when refering to accounts, is that they reside on clients' computers because there is no central server to keep these information and nodes have to authenticate one another.

A registration wizard is runned, so that the user could provide information about itself (in an interactive manner), which is needed to create a new account. Information that are requested are split in three : *personal information* – name, location, e-mail, other comments; *network related information* – network connection type, IP address, workspace and download directory; *security related information* – username, password, private and session key size.

(!!!) In the original design Gazebo has file searching and sharing capabilities over the network so, the real network connection type is needed because, if a node has a poor connection, we want to avoid making it forward files and messages, as it would consume its short bandwidth (not implemented yet).

The username is less important and could even be the same as one of your friends because the public key is the primary identifier and has to be unique and not the username. A password of at least 8 characters is required because the account resides on the local computer and the protection level has to be high. The public-private key pair is recommended to be of at least 2048 bits; the session key size should be chosen based on how much security we need during a session (in a local network you may want a smaller key size – fast encryption/decryption , whereas when communicating over the Internet you may want to pick a bigger key size – slower but more secure).

In the end, an "User Profile" object (see Figure 19) is updated with all these information and is serialised. Physically, an archive with the extension ".gzb" is created in the "Workspace directory", which was picked at registration time.

The most important file in the archive is "Settings.gbo", which contains the serialised "User Profile", which among other personal information that were required at registration, contains the private key! This file is then encrypted – using AES (see 4.1.2) – with the password the user provided. A copy of the public key is also found, in "key.pub.gbo". In "HMAC.gbo" a keyed-hash message authentication code of the "Settings.gbo" file (using the same user provided password) is present – it is used for data integrity and authentication (good to know if the settings were tampered in any way). A "challenge message" that was prior hashed (using SHA-512) and encrypted (with the password provided by the user) can be found in "Challenge.gbo". (Figure 20)

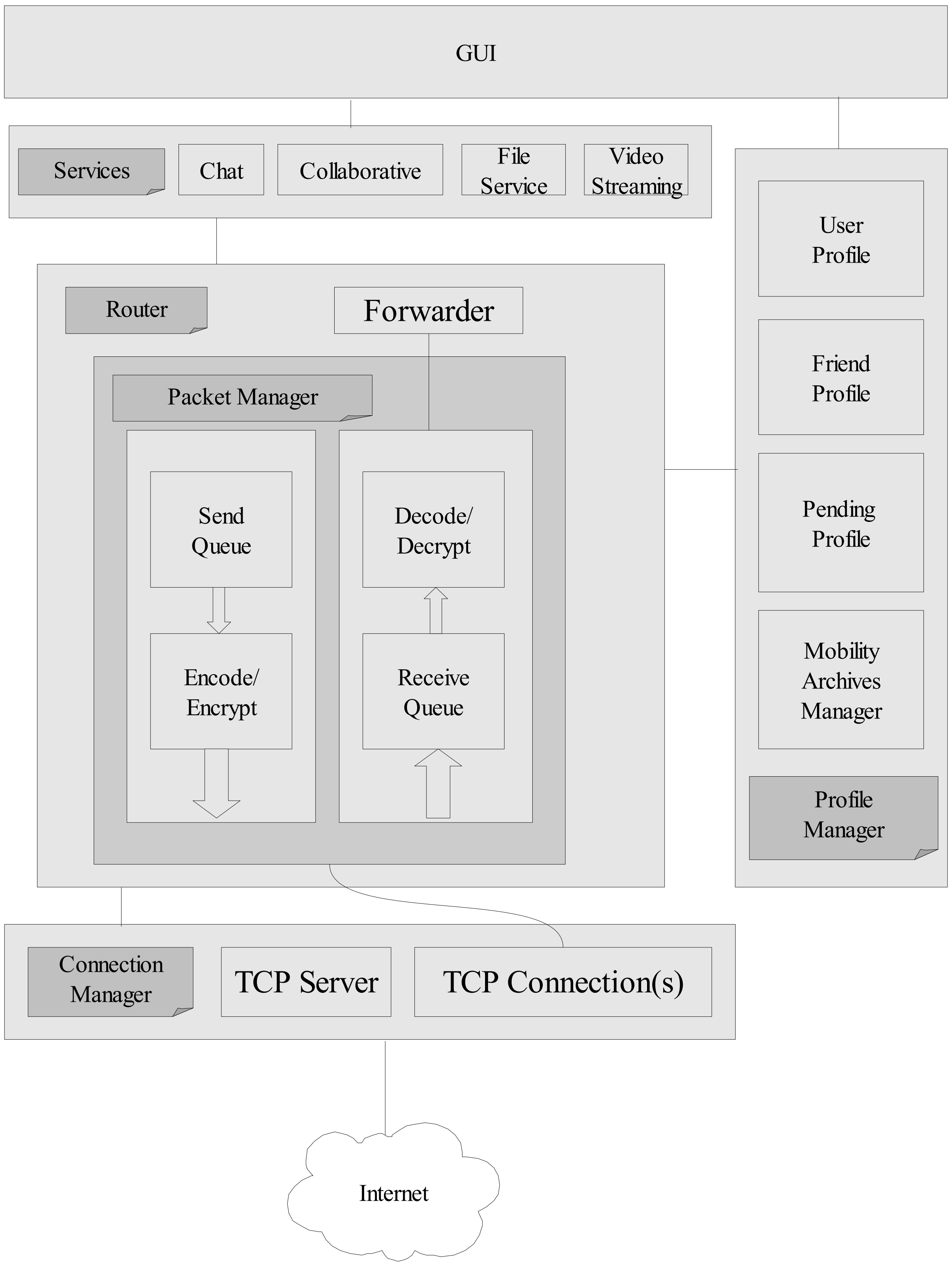
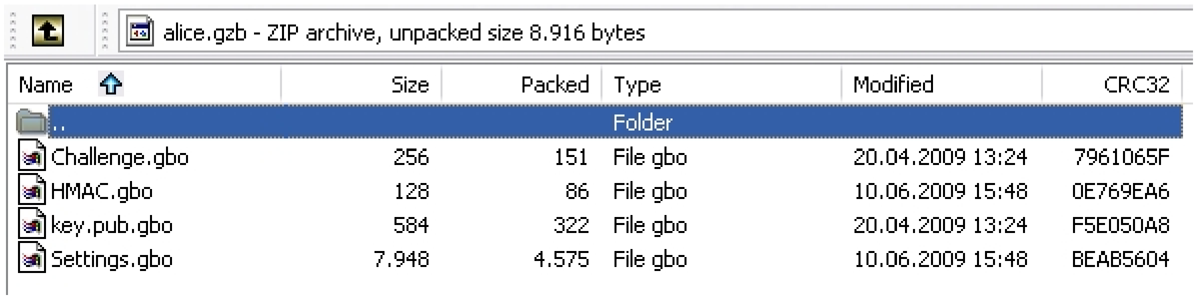


Figure 19 : Gazebo F2F's architecture



alice.gzb - ZIP archive, unpacked size 8.916 bytes

Name	Size	Packed	Type	Modified	CRC32
..			Folder		
Challenge.gbo	256	151	File gbo	20.04.2009 13:24	7961065F
HMAC.gbo	128	86	File gbo	10.06.2009 15:48	0E769EA6
key.pub.gbo	584	322	File gbo	20.04.2009 13:24	F5E050A8
Settings.gbo	7.948	4.575	File gbo	10.06.2009 15:48	BEAB5604

Figure 20 : Archive contents

Finally, a line containing user's name and location of the archive on disk, is appended to the "user.archive.map" file. This is needed because, when logging in, a user should only provide its username and password and would not need to worry about its archive location on disk. If the account location is not found in that mapping file or the user wants to log in his account from a different computer (by copying the .gzb archive on that computer – that's why it is called a *mobility archive*), Gazebo allows profiles to be loaded, by specifying the archive path on disk and a password to open it.

When someone wants to log in, Gazebo hashes a challenge message, encrypts it with the password provided by the user and if it matches the content of Challenge.gbo file then the user is let in. Also a keyed hash message authentication code is generated for the Settings.gbo using the key provided by the user. If it does not match what's in HMAC.gbo the the user is warned that the archive was tampered and is asked if it wants to continue. In either case, nothing can be done to "un-tamper" the data, because we don't know in which way it was altered.

(!!!) A Gazebo account is compromised only if an attacker succeeds to break the password that protects the account – the password that the user registered with and uses to log in. That's why the password has to be strong enough (long password) to at least withstand some attacks (brute force, etc) if the archive is "stolen". If the password is broken, then, the attacker can find out the private key and with it, decrypt the network traffic in the initialisation process, when users connect one to the other, then find out the session keys and with them decrypt all the "secure" traffic between the "broken user" and its friends. Account alteration could lead to forbidden access to someone's own account, and thus, the need for a new account and a new (difficult) setup.

If the attacker uses key loggers then, even if the account is on a server(in client server applications), it is compromised. Therefore, we worry only for the case when someone steals the archive and wants to break it. A logical device could be created on disk for each archive to "stay" in it, so when an operation has to be made with the archive a password to be requested.
Because the protocol does not share any sensitive information regarding friends (like computer information, passwords, etc), even if an attacker breaks into one node, it can't damage only by that other friends, or affect the good functioning of the F2F network.

5.2.2 Certificates

(!!!) From now on, we will only consider the interaction between 2 nodes (named Alice and Bob), because it can be applied next, to any number of nodes. Thus, we will show what Alice has to do in order to connect to Bob and start using the tools provided by Gazebo.

(!!!) We will consider some notations, in order to present the protocols in an easier to read and understand manner:

- "A" and "B" will refer to Alice and Bob, two "correct" users and friend from real life (we put *index* if refering to either of them)
- "E" will refer to Eve, an eavesdropper
- *K(index)* is Alice or Bob's randomly generated symmetric keys at some point
- *SK* is the (symmetric) session key Alice and Bob use after they become friends – for every session is a different key, even if we note it in the same way
- *PK(index)* is the public key
- *Info(index)* refers to personal information like name, username, e-mail address, etc
- $\{data\}_{K(index)}$ - data is encrypted with the key K using AES (see 4.1.2)
- $\{data\}_{PK(index)}$ - data is encrypted with the public key PK using RSA (see 4.2.2)
- || means (string) concatenation with delimiter – e.g. "abc" || "def" means "abc#def"

After Alice creates an account and logs in, she wants to connect to Bob – a friend of hers. To do this, she needs to generate a certificate (with personal information), which has to be shared with Bob. The certificate C_A thus has the form :

$$C_A=\{K(A)\}_{PK(B)}||\{ID(A), Info(A)\}_{K(A)}$$

Alice randomly generates a symmetric key K(A) with which she encrypts a randomly generated identifier and some personal info – her name and IP address, and a question for Bob to answer (the question is for security reasons). Because in this stage Alice does not know from Bob (and vice versa) anything else than his public key, she encrypts K(A) with Bob's public key. The two encrypted strings are then concatenated and the resulting string is dumped on disk.

Bob also generates a certificate C_B and exchanges it with Alice's. For security reasons, the exchange is required to be made out-of-band (face-to-face), and not over the network. A Pending Profile(see Figure 19) object has knowledge of both certificates – the one generated and the one received and loaded in the application. After both users loaded the exchanged certificates into their accounts, each of the two nodes is called a **pender**.

5.2.3 Pender to Friend Transformation

If both penders are online at the same time then, Gazebo proceeds with a three steps protocol, to autheticate users one to the other and "transform" them from penders to friends :

(!!!) We consider Alice to be the initiator of the protocol (she connects to Bob first). She sends him the following message M1 :

$$M1=\{'PENDER'||k(a)\}_{PK(B)}||\{K(A), K(B), Info(A)\}_{k(a)}$$

K(A) and K(B) are the two keys generated by Alice and Bob for their certificates – see C_A and C_B above. Again, because Alice and Bob don't share any secret, only their public keys, we use the same method that we applied for the certificates: generate a symmetric key with which we encrypt a large amount of data and encrypt that key with the public key of the pender. Because we presume that we always know the correct public key of a friend (i.e. the public key is taken directly from our friend even if it is posted on websites, sent through e-mail, etc thus the possibility of being replaced with an attacker's public key is zero) and also consider that it is hard/impossible to break, it means that the only person that will be able to decrypt the message that contains the symmetric key is the holder of the private key and that is our friend. k(a) is the generated symmetric key for this message. Info(A) contains information about Alice like: IP address, public key, name, e-mail, etc.

Bob receives M1, splits it in two by the delimiter, decrypts the two messages and tests if : the first 6 characters form the string "PENDER", the IP address received corresponds to the one from Alice's certificate, K(A) from the certificate corresponds to K'(A) received, and K'(B) received corresponds to K(B) from the certificate. If all of these are true, Bob is asked through the interface if he is happy with the answer provided by Alice to his initial certificate question. If he is, then he keeps Info(A) and initiates a message M2 (similar in meaning to M1) which he sends to Alice :

$$M2=\{'ACCEPT'\|k(b)\}_{PK(A)}\|\{K(B),K(A),Info(B)\}_{k(b)}$$

Alice receives M2, decrypts the messages and conducts the same tests Bob has, seen from her point of view (if the first 6 characters form the string "ACCEPT" and if the IP address and keys from the certificates match). If everything is ok, Alice is asked if she is fine with the answer Bob provided to her question. If she is, then she sends a message M3 to Bob :

$$M3=\{'ACCEPT'\}_{PK(B)}$$

Bob receives M3 and checks to see if the decrypted message is "ACCEPT".

(!!!) At any point, if any condition fails, then the protocol fails and the connection is terminated.

(!!!) The question and answer are just another security measure. If hypothetically Eve intercepts the network traffic between Alice and Bob when they conduct the above protocol and wants to forge one of them's identity, she will need to forge M1 or M2, but she won't know how to answer to the question correctly because it is a personal information that is known only by Alice and Bob.

After the protocol finishes, Alice has Info(B), Bob has Info(A), they are authenticated and are **friends** (in the Gazebo F2F meaning) one to the other. The Pending Profile objects that kept the certificates are removed from both accounts and replaced with Friend Profiles (see Figure 19) objects that are updated with Info(A) respectively Info(B); keys from certificates and other security information are discarded because they are obsolete.

5.2.4 Friend to Friend Connection Establishment

Next, we will consider that Alice and Bob became "friends" after conducting the above protocol. Now, if both of them are online, they will try to connect one to the other and conduct another protocol in order to exchange a session key in a secure manner, without prior knowledge of any secret i.e. both know only the public key of its friend. For this, Gazebo F2F implements the Diffie-Hellman key exchange protocol (see 4.2.3) and a custom key exchange protocol that is actually used (because it follows Gazebo's guidlines for protocols and is much simpler than DH) and presented next :

Again we consider Alice to be the initiator of the protocol. She sends Bob the following message M1 where IP(A) is Alice's IP address and SK(A) is her's piece of session key :

$$M1=\{'FRIEND'\|IP(A)\|SK(A)\}_{PK(B)}$$

Bob receives M1, decrypts it, and if the first 6 characters form the word "FRIEND" and IP'(A) is Alice's IP from the Friend Profile object, then Bob generates his piece of session key SK(B) and concatenates SK(A) with SK(B) (without the delimiter) to form the session key SK. With it, he encrypts the message "FRIEND", concatenates to it his IP address and his piece of session key, encrypts everything with Alice's public key and sends her this message M2 :

$$M2=\{IP(B)\|SK(B)\|\{'FRIEND'\}_{SK}\}_{PK(A)}$$

with

$$SK=SK(A)+SK(B)$$

where the "+"operation refers to the classical string concatenation.

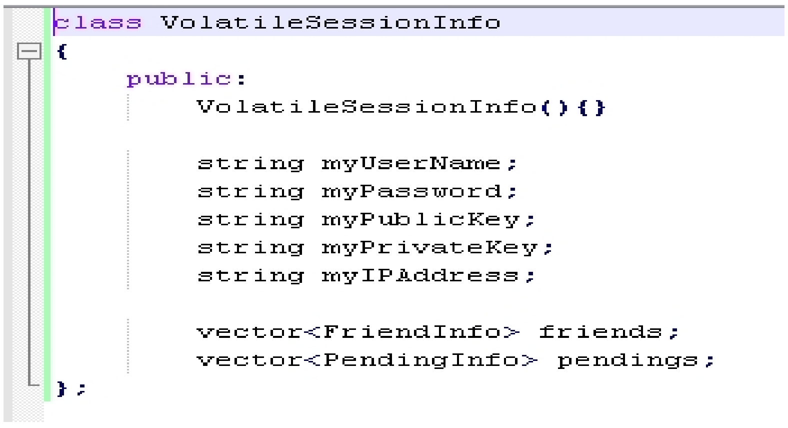
Alice receives M2 and tests if the IP'(B) is the same to the one in the Friend Profile object, concatenates SK(B) to SK(A) and decrypts the inner message. If it is "FRIEND" it means that Alice and Bob have the same session key. Alice has to send one more message to Bob so he can be certain that Alice has the same key as him, and that there is not Eve on the line that tries to forge Alice's identity. Thus, she encrypts the message "2FRIEND" with the session key and sends this M3 to Bob :

$$M3=\{'\text{2FRIEND}'\}_{SK}$$

Bob receives M3 and decrypts it with his SK. If the message matches "2FRIEND" it means that Alice has the same session key Bob has and that they now can rely on this key to securely communicate over the network.

5.2.5 The Global VolatileSessionInfo Object

After a user logs into its account, one of the things that happens in the background is the initialisation of a *VolatileSessionInfo* global object. Because Gazebo's internal architecture is quite complex, and as its name suggests, this object keeps the useful information needed in a session (i.e. from when the user logs in to when the user logs out), information that is scattered in numerous places. It is called global, not because it resides in a global shared place, but because it is visible to everyone by passing a pointer of the object to other objects that need it (Figure 21)



```
class VolatileSessionInfo
{
    public:
        VolatileSessionInfo(){}

        string myUserName;
        string myPassword;
        string myPublicKey;
        string myPrivateKey;
        string myIPAddress;

        vector<FriendInfo> friends;
        vector<PendingInfo> pendings;
};
```

Figure 21 : VolatileSessionInfo class

For each friend we keep a FriendInfo stucture that keeps some personal information about our friend, the session key, some flags regarding connection and a message queue. For each pender we keep a PendingInfo stucture that keeps some personal information about the pender, some flags regarding connection and a message queue.

5.2.6 Threads Structure

Because of Gazebo's highly complex internal structure and because wxWidgets (see 6.2) threads can't have a shared workspace location and communicate one to the other through it (by updating variables, flags, etc), we use message queues to send messages from one component to the other. Moreover, it is recommended that GUI operations should be made only in the main thread and not in any other thread. Thus, the need for a way to communicate between components is paramount.

The current implementation for queues can actually send entire objects and not just string messages. The operations supported are Post (send), Receive and ReceiveTimeout (wait a specified time for a message to arrive and then return).

(!!!) SinkData class is derived from wxObject and has two fields : a public key and the actual (string) data. It is used with message queues, to transport data between components. Because a public key uniquely identifies a friend, it is used to identify the network interface on which the message should be sent.

After the main frame initialises all GUI components, a number of threads are started (Figure 22) :

Sink Send – Receives a SinkData object, encrypts the data with the session key found in the VolatileSessionInfo object (that corresponds to the public key received), and throws it to the corresponding connection thread that will send the message on the network to our friend.

Sink Receive - Receives a SinkData object, decrypts the data with the session key found in the VolatileSessionInfo object (that corresponds to the public key received), and throws it to main frame to establish what service should process that message.

Server – Waits for clients to connect, and handles them by starting a new connection thread for each of them.

Client – From time to time (25 seconds in the current implementation) this thread tries to connect to those friends and penders that are not already connected. It starts a new connection thread to handle each of them.

(!!!) The main frame initialises queues for all 4 threads described above to whom passes them as pointers. When the application is exited, the main frame posts an "EXIT" message to all these queues and the threads are exited. Also, all threads that handle connections are exited. When a subsidiary thread wants to communicate with the main thread (which handles all GUI actions) it does not use a queue because, managing a queue in the main thread is very time consuming, so, the user interface would respond really slow or even freeze. Therefore we use *events*. Events can incorporate from string messages to entire objects. The main frame has a table of handles for every event that might appear, so when an event is fired having the destination "main frame", the event type is looked up in the table and is resolved by a specific function. Thus, the main thread is "bothered" only when it is absolutely necessary and does not have to manage a queue, so it is more responsive to users' GUI actions. (see Figure 22)

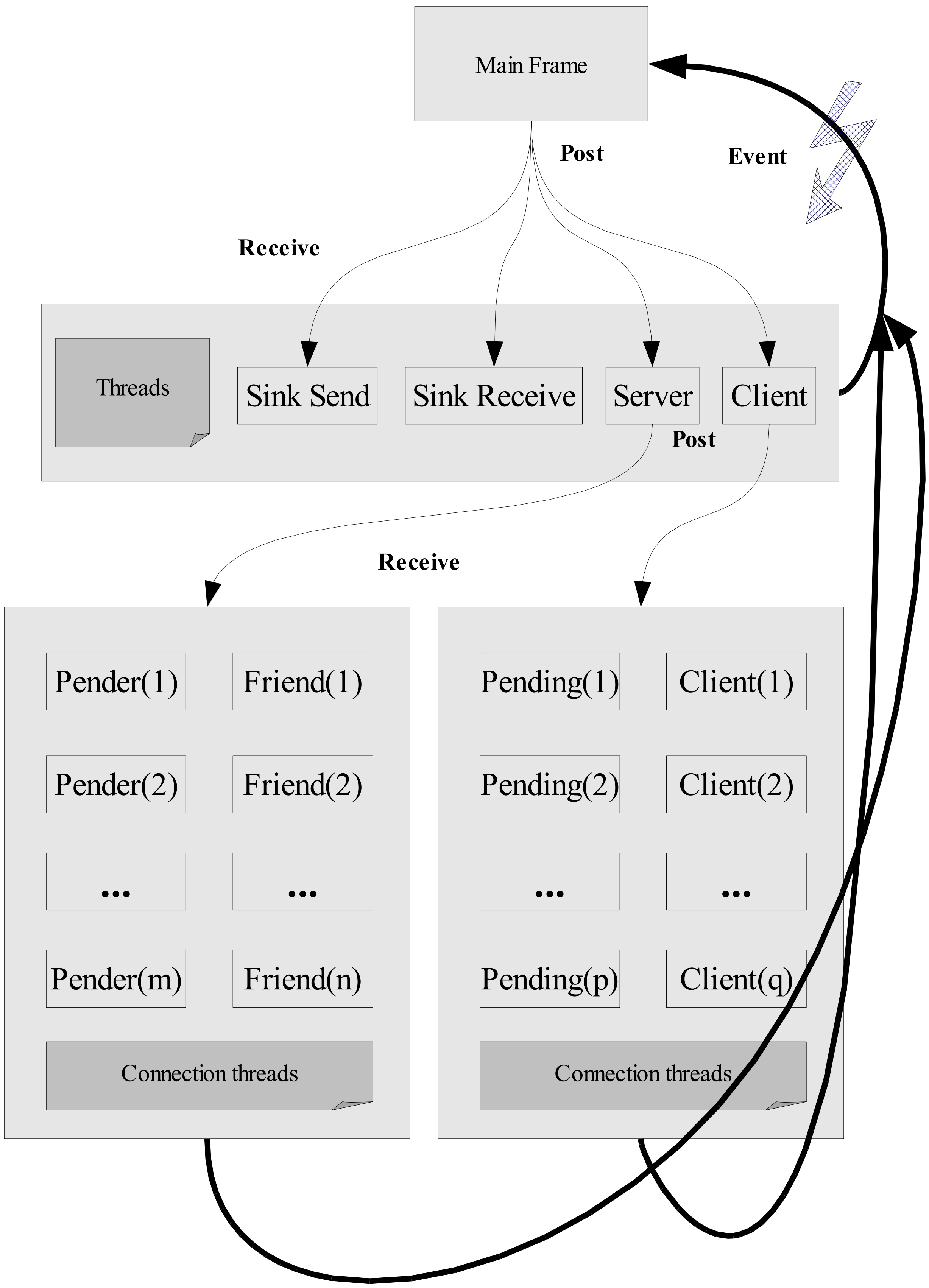


Figure 22 : Gazebo F2F's threads structure

5.2.7 Chat Service

After all these security protocols and setup Alice surely wants to chat with Bob. Figure 23 shows you what happens in the background when a message is sent or received by the chat service.

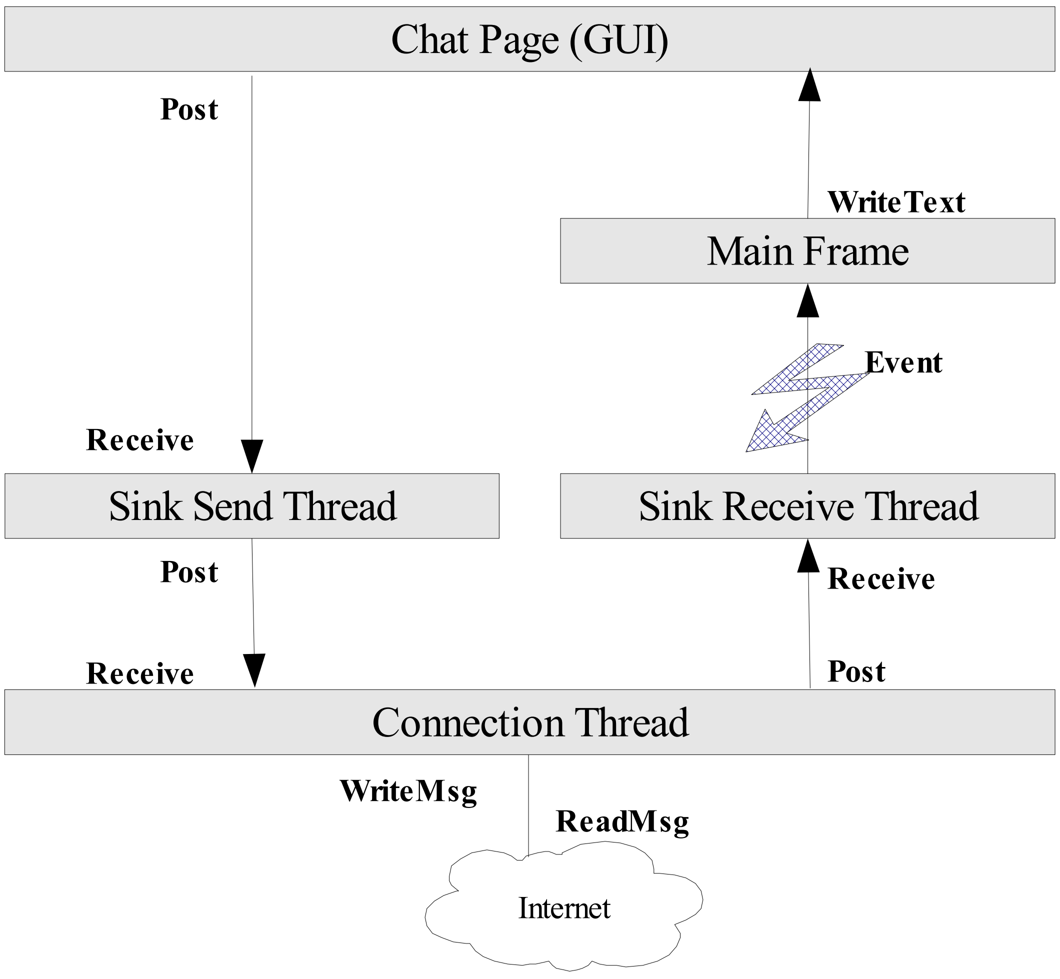


Figure 23 : Gazebo F2F's chat service

Basically, Alice types a message in the chat page she opened (to talk to Bob) and presses the Send button. Each chat page that is opened is also identified by that person's public key (Alice wants to talk to Bob, so that page will be identified by Bob's public key, which is the unique identifier). The message is shown on Alice's screen and then a SinkData object (containing Bob's public key and Alice's message) is posted to the Sink Send thread. There, the message is encrypted with the session key Alice and Bob agreed to use, the ciphertext is posted to the connection thread that handles the connection to Bob, and there it is sent on the network.

Bob receives Alice's message in the connection thread which handles the connection to Alice, which sends it to the Sink Receive thread. There, based on the public key received from the connection thread, looking in the VolatileSessionInfo object, the session key Bob and Alice agreed to use is found and the message is decrypted. An event with a SinkData object containing the decrypted message and Alice's public key is fired, with the destination Main Frame. Based on the public key received, Main Frame looks up for an opened chat page that is identified by it. If none is found, Main Frame opens one. The plaintext is then displayed on Bob's screen.

5.3 Future Development

Here, we present some components that should be implemented in future versions, that we think might improve Gazebo F2F :

Forwarder

Like Figure 19 presents it, this should be a control component. It should receive messages from Sink Receive thread and decide what to do with them. If their destination is a Gazebo service then they should be sent to Main Frame or directly to those services, to process them. There may be particular situations (filesharing – see next) where messages should just be forwarded to another friend, so they are just thrown to the Sink Send thread with special flags activated, encrypted and thrown again on a specific network interface.

Collaborative work

Gazebo F2F should implement a collaborative framework because it suites best the idea of friend-to-friend computing (because usually people that work on a project have prior social interactions and are connected by social links of some sort). Source code, charts, plots, art work, etc. could be created and maintained in an interactive manner, users having a variety of tools to ease their work on school or work projects.

Video streaming

Because of the spacial locality between friends, a resource consumer service like video streaming could be implemented. Basically, the idea is to browse a video file on a friend's computer and play it on your computer without first downloading the whole of it and then playing it, but, use your friend's Gazebo as a streaming server and your Gazebo as a streaming client (and vice versa when is the case). Similar idea for audio files.

File browsing and sharing

A logical shared directory could be created for each user that uses Gazebo. Files that users want to share with their friends should be put there. Then, these friends could browse files from this directory or search for a particular file and download it.

In section 2.2 we presented a possible approach for implementing filesharing in F2F applications. A user searches for a file on the network; his request is broadcasted to all of his friends and rebroadcasted on and on to friends of friends, for a number of hops. Whenever there is a match, a message should be sent back on the initial route it came, but in a hop by hop manner (from friend to friend and not direct connection). Figure 24 presents a scenario where Alice has three friends (Bob, Charlie and Dean) and looks for a file called "patch.exe". She broadcasts the request, and after a while there are a few matches. Gazebo should implement a detection mechanism that should not allow the same request move back and forth between the same users : e.g. Alice-Charlie-Hans-Bob creates a "redundant walk". There should be three matches, corresponding to the three friends Alice has. Lets say Alice chooses to download the file from Charlie. As seen from George's point of view, Charlie requests "patch.exe" for himself. Actually, Charlie is a bridge between Alice and George. In the end, Alice does not know who George is (and vice versa), and thinks that Charlie had that file. The same thing would've happened if Alice had chosen to download "from" Bob or Dean.Also, the Connection Type field should be taken into consideration : if a user has a poor connection (e.g. Modem) the forwarding mechanism should be disabled and he should not be used as a bridge like Charlie is (for Alice, Hans and George) in the above scenario.

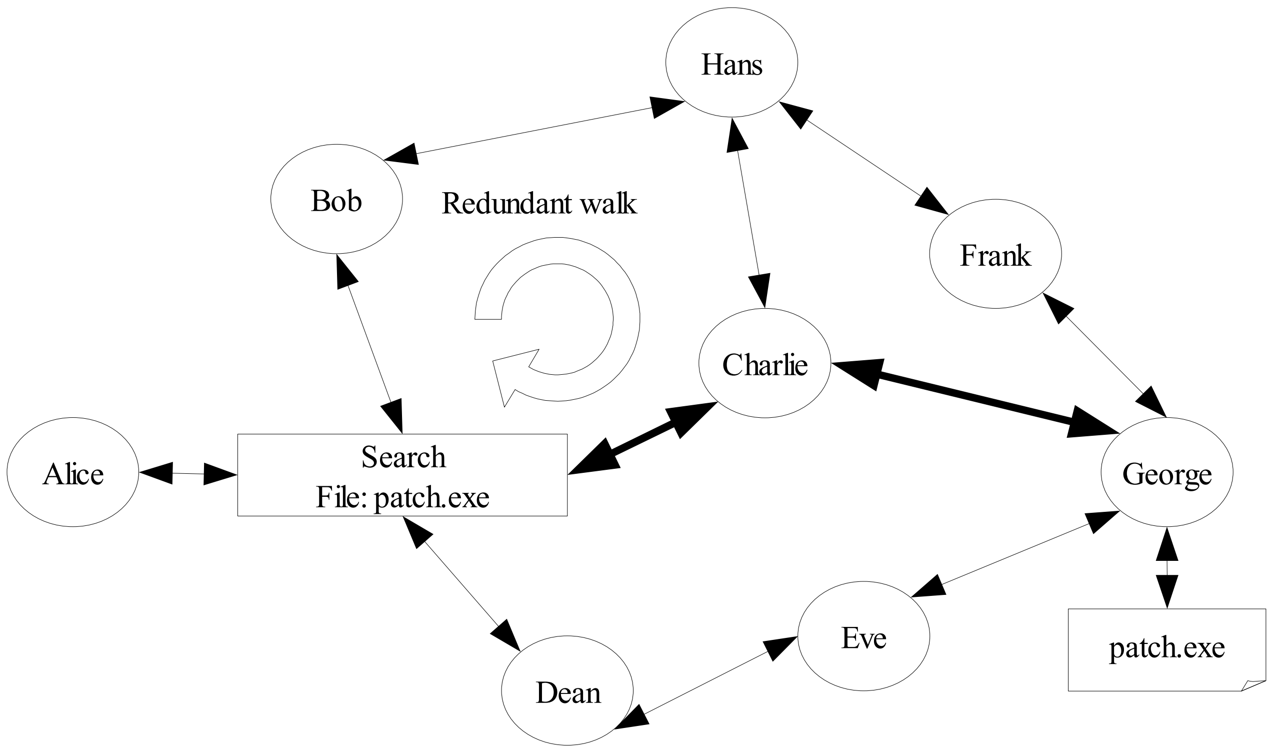


Figure 24 : Network file search

Internal firewall
The difficult setup has one more advantage : because we know every IP address of the users we want to become friends with (through the certificates), an access control table could be built inside Gazebo F2F, table that should keep all allowed IP addresses and grant them access to connect to us, and block all others.

5.4 Usage Scenario

This section provides a real case scenario involving two users and real life friends, called generically Alice and Bob, who follow the Gazebo F2F protocol steps presented earlier (in a more general way), in order to reach the point where both are authenticated one to the other and can chat in a secure manner over the insecure network.

(!!!) Again, we will see these steps through Alice's eyes, Bob just doing similar things (when he is not mentioned). In figures where 2 pictures are placed side by side, the left one is Alice's and the right one is Bob's.

When Alice starts Gazebo F2F, she should see a frame similar to the one presented in Figure 25. Before she logs in, she has to create an account, which she does, by clicking on the left most icon from the group of 3, that are at the bottom of the frame. A wizard should pop up, wizard that she has to follow in order to create a new account.

The information required can be split in three : *personal information* – name, location, e-mail, other comments; *network related information* – network connection type, IP address, workspace and download directory; *security related information* – username, password, private and session key size. Because the account resides on the local disk, a strong password should be chosen. The session key size should be picked, based on the level of insecurity of the network : if we use the protocol only in a secure local network then a smaller key size can be picked to speed up the transfer process, whereas if we use it over the Internet, even if packets can get delayed a bit, we can choose a larger key size. In the end, Alice's account will be an archive called "alice.gzb" (is she pickes her username to be alice) that resides in *path-to-Gazebo/profiles/alice* .

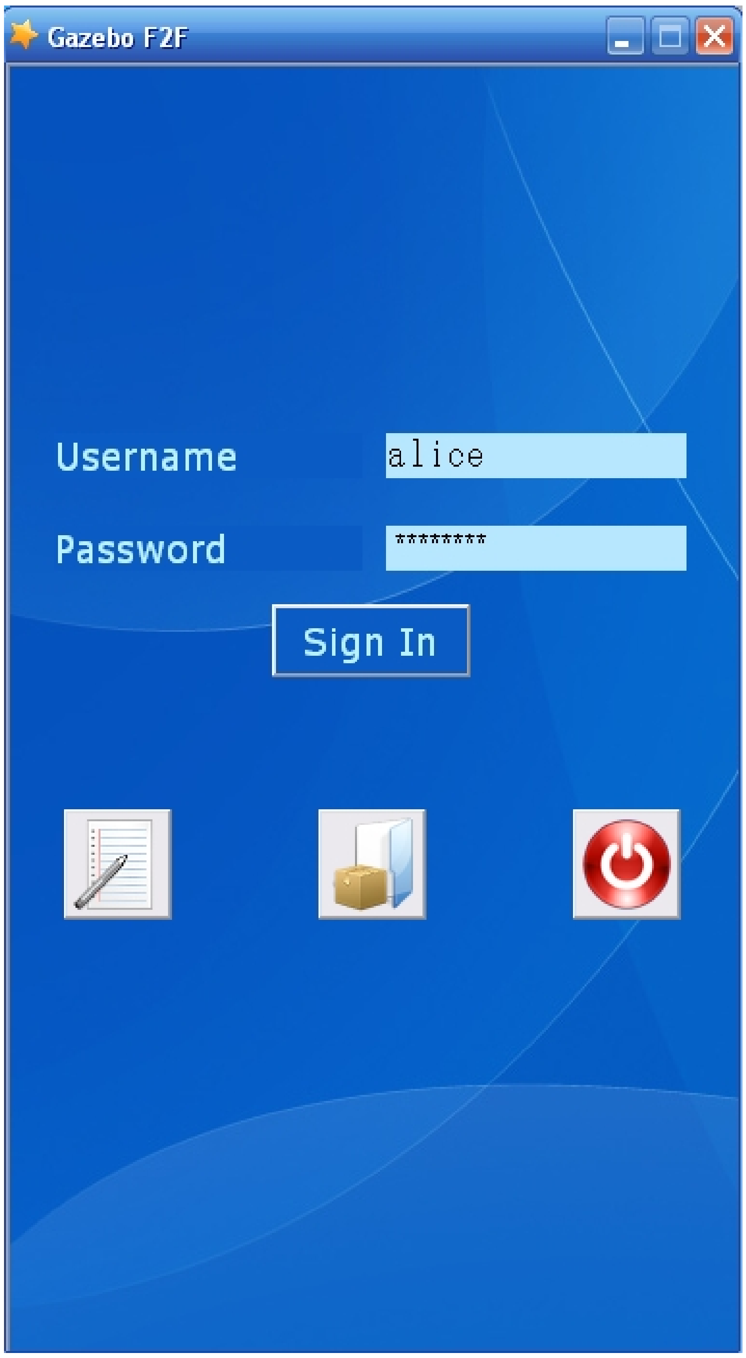


Figure 25 : Gazebo F2F's login frame

Later on, if Alice wants to access her account on a different machine than the one she made her account on, she has to have the ".gzb" file with her. On a different computer, she has to click on the middle icon from the group of 3 on the bottom of the frame. A dialog box, like the one in Figure 26 should appear. There, she has to introduce the path to her archive and the password of her account. On the local machine she can just fill up the username and password fields that are on the login frame, and click on the "Sign In" button. The right most icon (from the group of 3), exits the application.

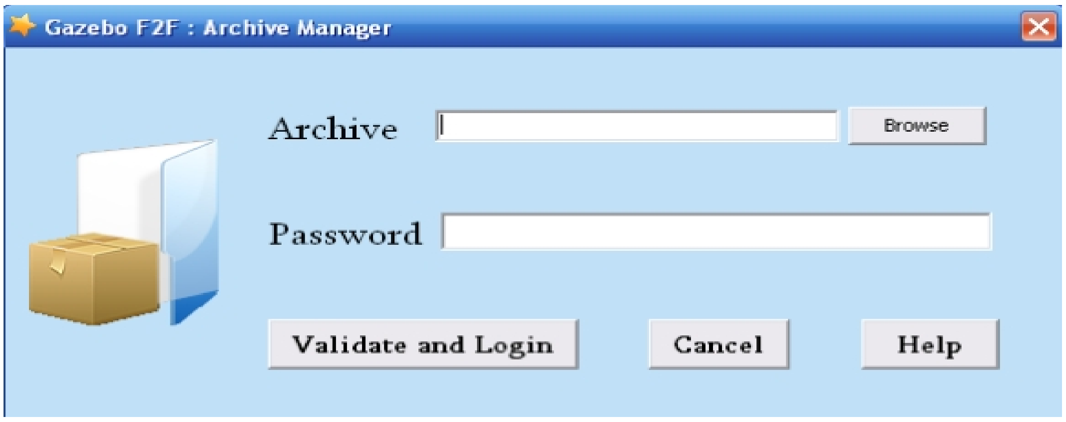
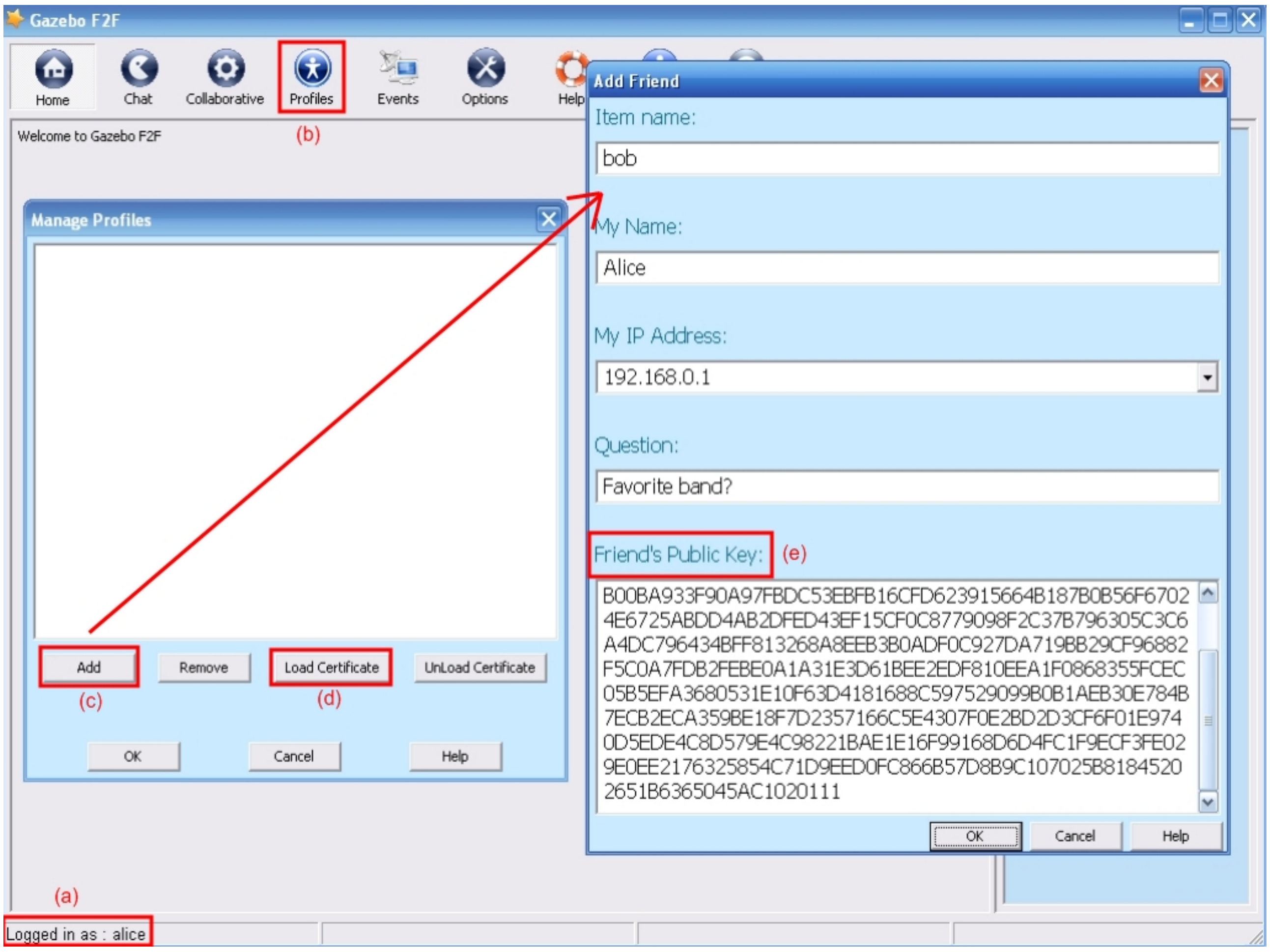


Figure 26 : Gazebo F2F's mobility archives manager

After she logged in, she should see Gazebo F2F's main frame, and in the bottom left corner, a message stating that the user alice has logged in (Figure 27 - (a)).



Welcome to Gazebo F2F

Logged in as : alice

Item name: bob

My Name: Alice

My IP Address: 192.168.0.1

Question: Favorite band?

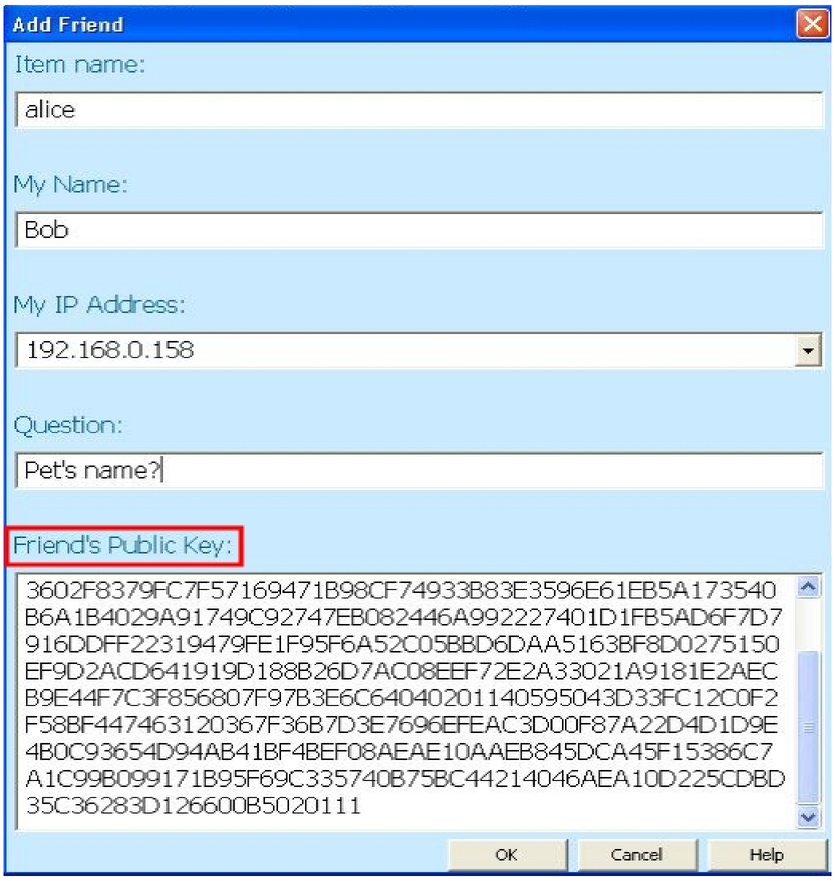
Friend's Public Key:

```
B00BA933F90A97FBDC53EBFB16CFD623915664B187B0B56F6702
4E6725ABDD4AB2DFED43EF15CF0C8779098F2C37B796305C3C6
A4DC796434BFF813268A8EEB3B0ADF0C927DA719BB29CF96882
F5C0A7FDB2FEBE0A1A31E3D61BEE2EDF810EEA1F0868355FCEC
05B5EFA3680531E10F63D4181688C597529099B0B1AEB30E784B
7ECB2ECA359BE18F7D2357166C5E4307F0E2BD2D3CF6F01E974
0D5EDE4C8D579E4C98221BAE1E16F99168D6D4FC1F9ECF3FE02
9E0EE2176325854C71D9EED0FC866B57D8B9C107025B8184520
2651B6365045AC1020111
```

Figure 27 : Profiles manager

Alice would like now to connect to Bob. After Bob created an account of his own, the two users should exchange their public keys (that are found in *path-to-Gazebo/profiles/username*), which are unique identifiers.

In the main frame, Alice clicks on the Profiles tab (Figure 27 – (b)) which pops up the Profile Manager. She then clicks on the Add button (Figure 27 – (c)) and a form, like the one the red arrow points to, pops up. She fills in the required fields, the last of them being Bob's identifier (the public key we took from him) (Figure 27 – (e)). Bob does the same, and fills in a form like the one shown in Figure 28, providing Alice's public key as the unique identifier.


Add Friend

Item name:
alice

My Name:
Bob

My IP Address:
192.168.0.158

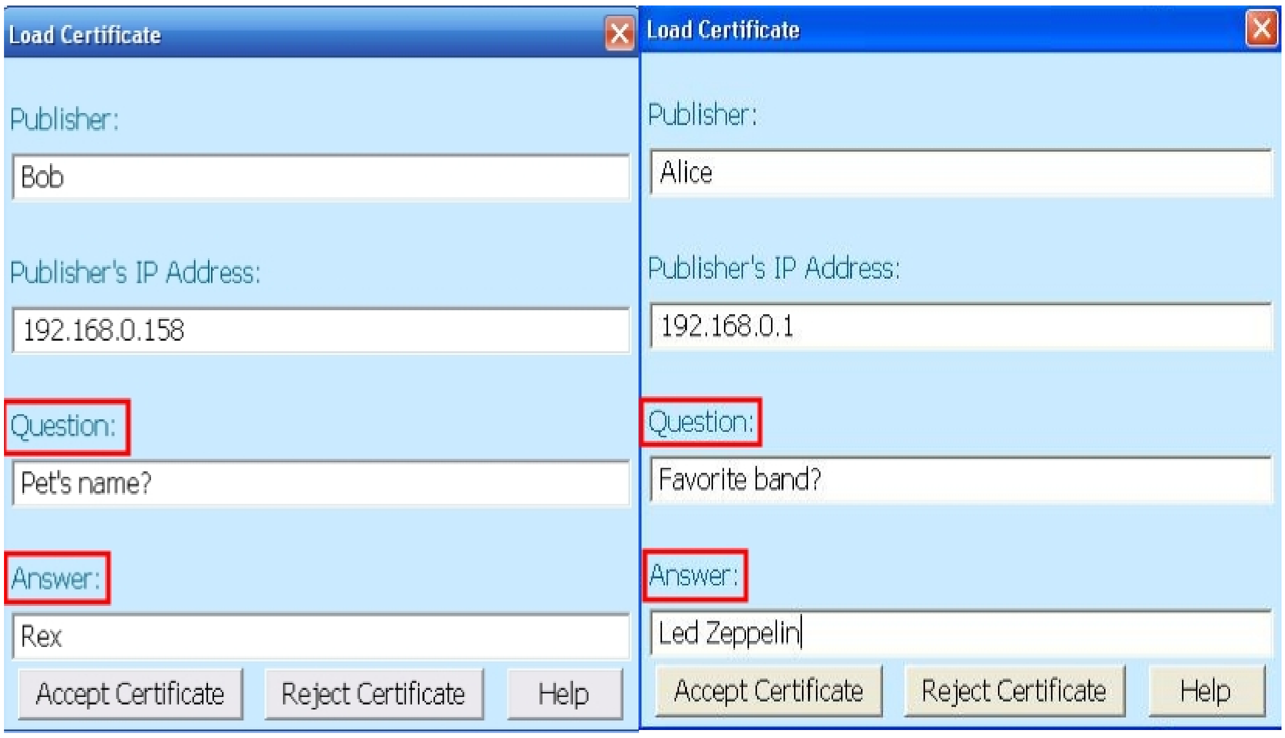
Question:
Pet's name?

Friend's Public Key:
```
3602F8379FC7F57169471B98CF74933B83E3596E61EB5A173540
B6A1B4029A91749C92747EB082446A992227401D1FB5AD6F7D7
916DDFF22319479FE1F95F6A52C05BBD6DAA5163BF8D0275150
EF9D2ACD641919D188B26D7AC08EEF72E2A33021A9181E2AEC
B9E44F7C3F856807F97B3E6C6404020114059504 3D33FC12C0F2
F58BF447463120367F36B7D3E7696EFEAC3D00F87A22D4D1D9E
4B0C93654D94AB41BF4BEF08AEAE10AAEB845DCA45F15386C7
A1C99B099171B95F69C335740B75BC44214046AEA10D225CDBD
35C36283D126600B5020111
```

Figure 28 : Adding a new friend

After this step, each user will have a certificate (that resulted from filling up the form) that must be exchanged with the other one. The exchange is mandatory to be made in an out-of-band manner (like face-to-face) and not over e-mail or any other network communication means. Anyway, other secrets have to be shared also (how to answer the questions) and although the certificates are encrypted, they could be intercepted and possibly get broken, if sending them over the network.

After the physical exchange, those certificates have to be loaded in both users' accounts. This is done by selecting from Profiles, the item on top of which we want to load the certificate, and press the Load Certificate button (Figure 27 – (d)). The left part of Figure 29 shows the dialog box that appears when Alice tries to load the certificate she received from Bob (the right part is when Bob loads the certificate received from Alice). She also has to answer a question (this is for security reasons), the answer being provided by Bob in their face-to-face meating.



Load Certificate	Load Certificate
Publisher: Bob	Publisher: Alice
Publisher's IP Address: 192.168.0.158	Publisher's IP Address: 192.168.0.1
Question: Pet's name?	Question: Favorite band?
Answer: Rex	Answer: Led Zeppelin

Figure 29 : Loading certificates

After the certificates have been loaded, Alice and Bob are called **penders,** and a new Pendings entry is added in the main frame's tree (Figure 30).

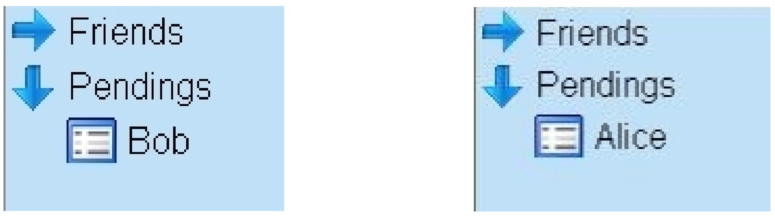
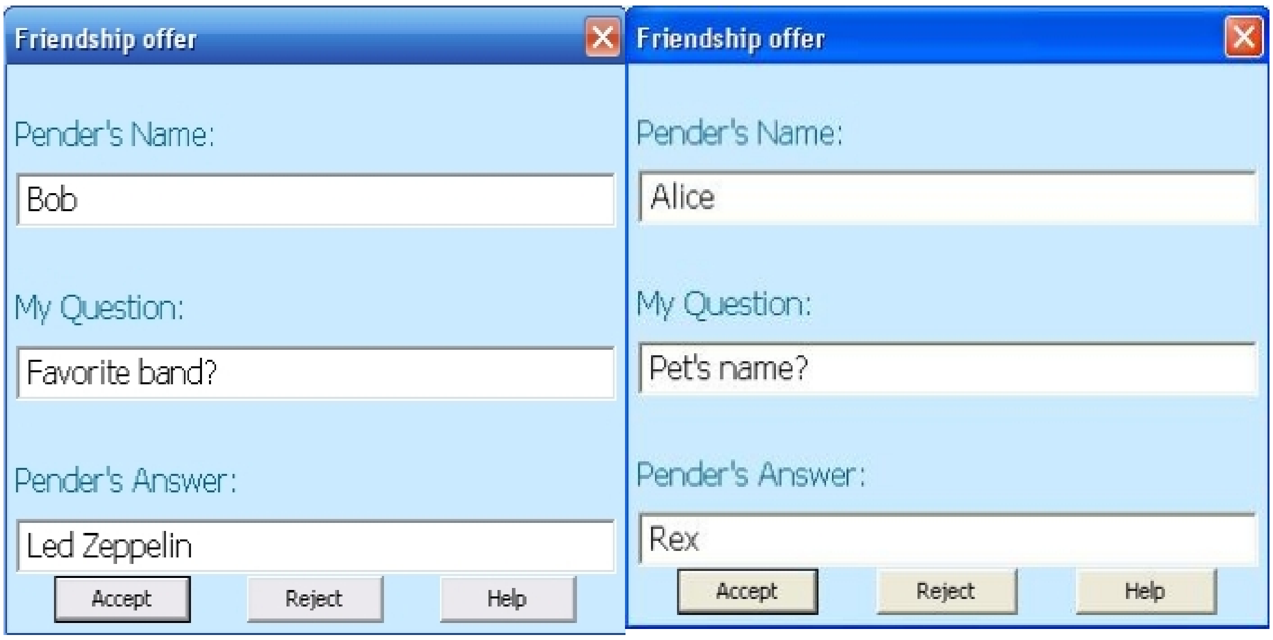


Figure 30 : Pendings entries

At some point, when both users are online, the protocol that "transforms" penders to friends, takes place. A dialog box on each side pops up, and each user is asked if accepts the friendship offer. There is also presented each pender's question and answer (Figure 31). If a malicious user would intercept the protocol communication and would want to forge someone's identity, he would not know how to answer these questions, because they are personal secrets and are shared only between friends.


Friendship offer

Pender's Name:

Bob

My Question:

Favorite band?

Pender's Answer:

Led Zeppelin

Friendship offer

Pender's Name:

Alice

My Question:

Pet's name?

Pender's Answer:

Rex

Figure 31 : Friendship offer

The Pendings entry of each is removed, and a Friends entry is added to each main frame's tree. Now, both users are authenticated to each other, and each of them is called a **friend**. If they are still online or when they both get online, they automatically connect one to the other (Figure 32).

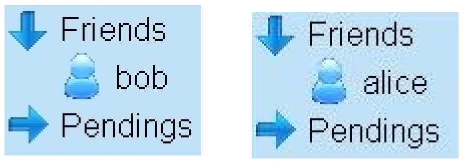
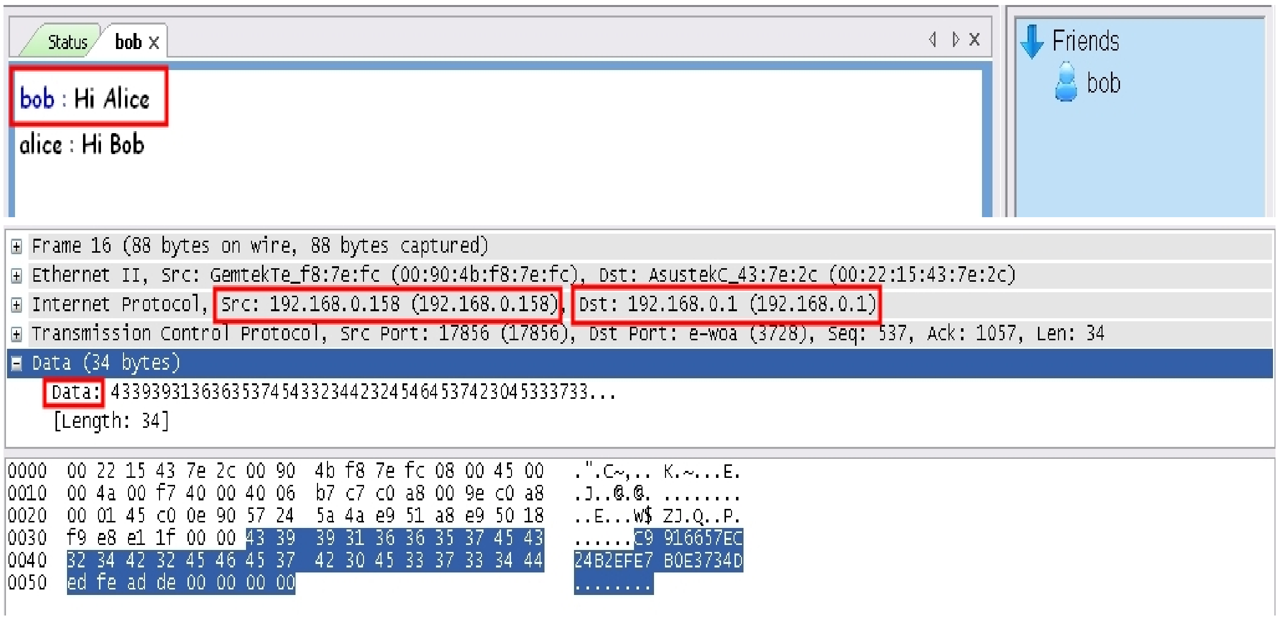


Figure 32 : Users online

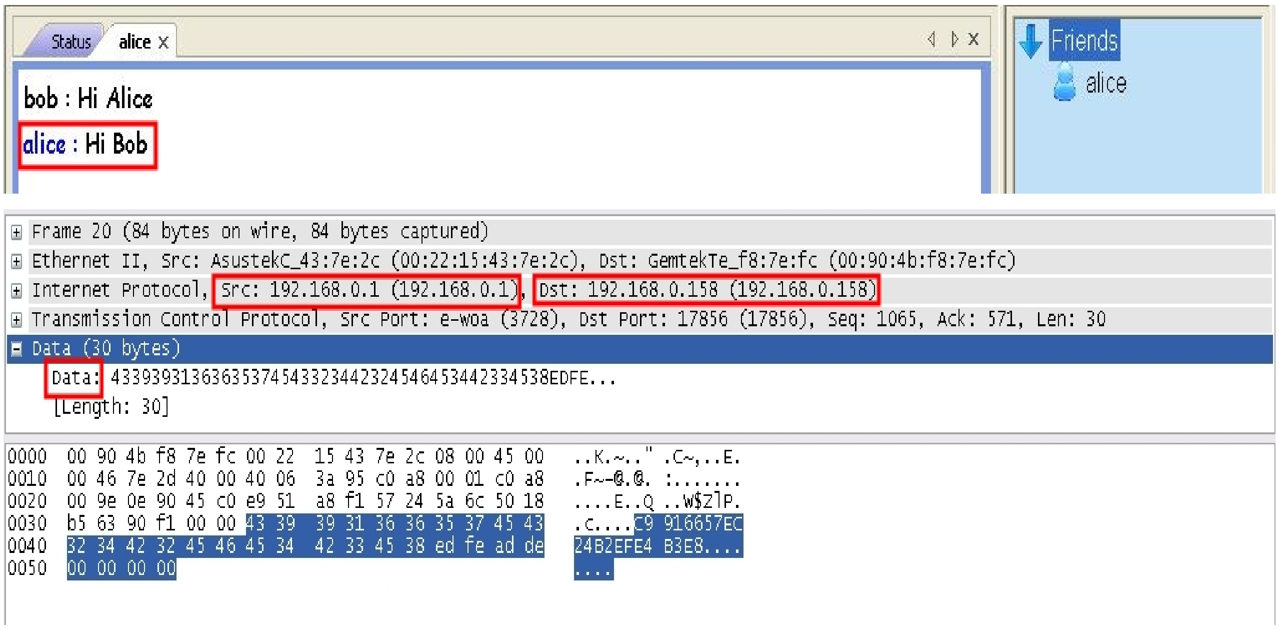
We also made some experiments with Wireshark [45], a network protocol analyzer, to provide a proof that the communication with Gazebo F2F is secure (data is encrypted). We choosed to run our tests on Gazebo F2F's chat service. Figure 33 presents a chat session between Alice and Bob which is intercepted. Bob (IP : 192.168.0.158) sends the message "Hi Alice", and Alice (IP : 192.168.0.1) receives it and displays it. The data is clearly not in plaintext, but in hexa, due to the current implementation which first encrypts the data and then transforms all ASCII characters from the ciphertext in hexa. Alice sends Bob the message "Hi Bob", which is again encrypted, as it can be seen in the figure.

Alice's IP address : **192.168.0.1**
Bob's IP address : **192.168.0.158**



```
bob : Hi Alice
alice : Hi Bob
```

```
⊞ Frame 16 (88 bytes on wire, 88 bytes captured)
⊞ Ethernet II, Src: GemtekTe_f8:7e:fc (00:90:4b:f8:7e:fc), Dst: AsustekC_43:7e:2c (00:22:15:43:7e:2c)
⊞ Internet Protocol, Src: 192.168.0.158 (192.168.0.158), Dst: 192.168.0.1 (192.168.0.1)
⊞ Transmission Control Protocol, Src Port: 17856 (17856), Dst Port: e-woa (3728), Seq: 537, Ack: 1057, Len: 34
⊟ Data (34 bytes)
    Data: 433939313636353745433234423245464537423045333733...
    [Length: 34]

0000  00 22 15 43 7e 2c 00 90  4b f8 7e fc 08 00 45 00   .".C~,.. K.~...E.
0010  00 4a 00 f7 40 00 40 06  b7 c7 c0 a8 00 9e c0 a8   .J..@.@. ........
0020  00 01 45 c0 0e 90 57 24  5a 4a e9 51 a8 e9 50 18   ..E...W$ ZJ.Q..P.
0030  f9 e8 e1 1f 00 00 43 39  39 31 36 36 35 37 45 43   ......C9 916657EC
0040  32 34 42 32 45 46 45 37  42 30 45 33 37 33 34 44   24B2EFE7 B0E3734D
0050  ed fe ad de 00 00 00 00                            ........
```

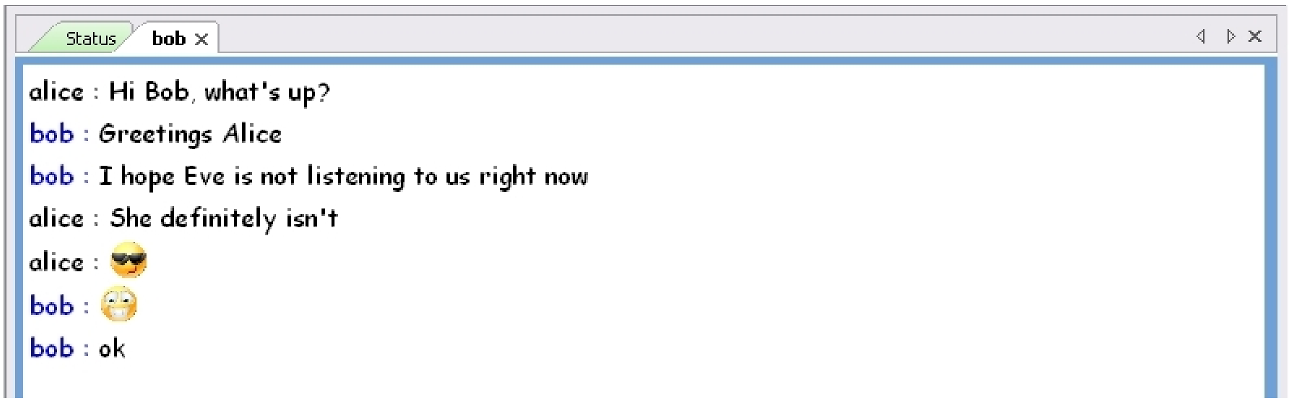


```
bob : Hi Alice
alice : Hi Bob
```

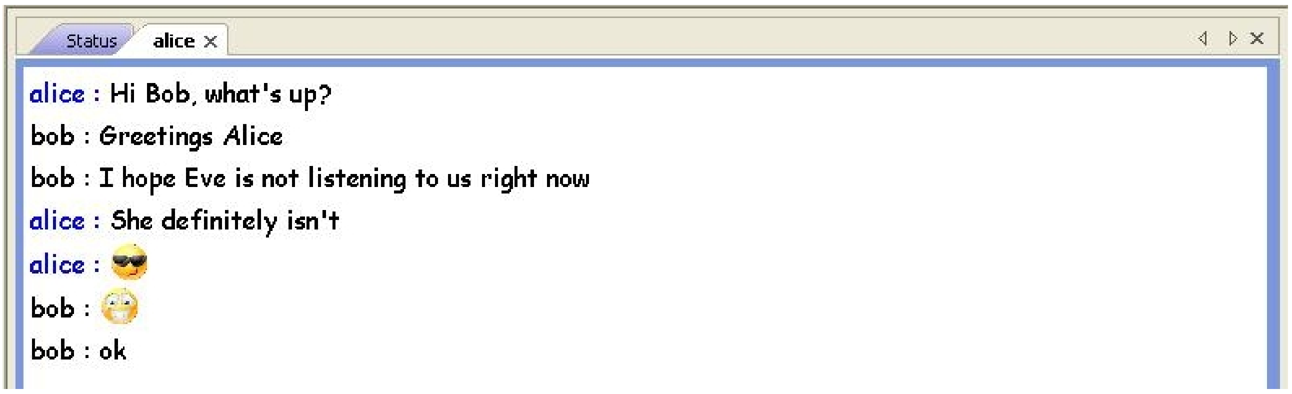
```
⊞ Frame 20 (84 bytes on wire, 84 bytes captured)
⊞ Ethernet II, Src: AsustekC_43:7e:2c (00:22:15:43:7e:2c), Dst: GemtekTe_f8:7e:fc (00:90:4b:f8:7e:fc)
⊞ Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.158 (192.168.0.158)
⊞ Transmission Control Protocol, Src Port: e-woa (3728), Dst Port: 17856 (17856), Seq: 1065, Ack: 571, Len: 30
⊟ Data (30 bytes)
    Data: 4339393136363537454332344232454645344233453 8EDFE...
    [Length: 30]

0000  00 90 4b f8 7e fc 00 22  15 43 7e 2c 08 00 45 00   ..K.~.." .C~,..E.
0010  00 46 7e 2d 40 00 40 06  3a 95 c0 a8 00 01 c0 a8   .F~-@.@. :.......
0020  00 9e 0e 90 45 c0 e9 51  a8 f1 57 24 5a 6c 50 18   ....E..Q ..W$ZlP.
0030  b5 63 90 f1 00 00 43 39  39 31 36 36 35 37 45 43   .c....C9 916657EC
0040  32 34 42 32 45 46 45 34  42 33 45 38 ed fe ad de   24B2EFE4 B3E8....
0050  00 00 00 00                                        ....
```

Figure 33 : Secure chat - proof of concept

Thus, Gazebo F2F users can chat in a secure manner (Figure 34)



alice : Hi Bob, what's up?
bob : Greetings Alice
bob : I hope Eve is not listening to us right now
alice : She definitely isn't
alice :
bob :
bob : ok



alice : Hi Bob, what's up?
bob : Greetings Alice
bob : I hope Eve is not listening to us right now
alice : She definitely isn't
alice :
bob :
bob : ok

Figure 34 : A chat service session

Chapter 6

Tools

6.1 Microsoft Visual Studio

"Visual Studio is a complete suite of tools for building both desktop and team-based enterprise Web applications. In addition to building high-performing desktop applications, you can use Visual Studio's powerful component-based development tools and other technologies to simplify team-based design, development, and deployment of enterprise solutions. "[1]

6.2 wxWidgets [2]

"wxWidgets (formerly wxWindows) is a widget toolkit for creating graphical user interfaces (GUIs) for cross-platform applications. wxWidgets enables a program's GUI code to compile and run on several computer platforms with minimal or no code changes. It covers systems such as Microsoft Windows, Mac OS X, Linux/Unix (X11, Motif, and GTK+), OpenVMS, OS/2 and AmigaOS. "[3]

It is one of the most complete GUI toolkits, having many utility classes and a lot of documentation. "Whenever possible, wxWidgets uses the native platform SDK. This means that a program compiled on Windows will have the look and feel of a Windows program, and when compiled on a Linux machine, it will get the look and feel of a Linux program.".Also it is free for personal and commercial use.[4]

The library is implemented in C++ but it has bindings for other programming languages as : C# (wx.Net), Java (wx4j), Perl(wxPerl), Python (wxPython),Haskell (wxHaskell), Lua (wxLua), Ruby(wxRuby) and many more.

Known applications like aMule(P2P filesharing application), Audacity (sound editor), BitTorrent(P2P filesharing application), Code::Blocks (C/C++ IDE), RapidSVN(Subversion client) or TortoiseCVS(CVS client) have been built using wxWidgets.[3]

6.3 Crypto++ [5]

It is a free C++ library for cryptography;it includes ciphers, message authentication codes, one-way hash functions, public-key cryptosystems, and key agreement schemes .

"The library is an powerful and elegant tool for performing complex cryptography. It uses advanced C++ features such as templates, multiple inheritance, and exceptions to achieve that power and elegance.

For people who are familiar with C++, the library will appear intuitive and easy to use. Others may need to view it as a learning opportunity. If you are a C++ beginner and you are under a very tight schedule, or if you are "afraid" of the more advanced features of C++, this library may not be for you. Having said that, you are invited to see for yourself how easy or hard it is to use by looking at some of the other answers in this category. "[6]

It is a cross-platform library ; C++ with Crypto++ code can be compiled on different platforms and operating systems (Windows, Unix, Linux, Solaris, MacOS).

"Some versions of Crypto++ have been validated by NIST(National Institute of Standards and Technology) and CSE(The Communications Security Establishment) for FIPS 140-2 level 1 conformance."[5]

There are many products that use this library ; some of them are : (commercial products like:) WinSSHD(SSH2 server for Windows), Tunnelier (SSH2 client for Windows), Microsoft Office Groove (peer-to-peer groupware platform, with collaborative editing, instant messaging, file sharing, voice chat, and other tools), Steam (digital distribution, digital rights management, multiplayer and communications platform) (and noncommercial products like:) MacCVS Pro (free CVS client for Mac OS), Keep In Touch(secure instant messenger), Grapevine (a distributed decentralized peer-to-peer static data storage network, similar to Freenet and Gnutella), yaSSL (Yet Another SSL, SSL/TLS Implementation with OpenSSL compatibility), PKIF(full-featured, standards compliant PKI enablement library, written in C++ with bindings for C# (and COM/.Net) and Java).[7]

6.4 Boost

"The Boost C++ Libraries are a collection of peer-reviewed, open source libraries that extend the functionality of C++. Most of the libraries are licensed under the Boost Software License, designed to allow Boost to be used with both open and closed source projects. Many of Boost's founders are on the C++ standard committee and several Boost libraries have been accepted for incorporation into both the Technical Report 1 and C++0x.

The libraries are aimed at a wide range of C++ users and application domains. They range from general-purpose libraries like the smart_ptr library, to operating system (OS) abstractions like FileSystem, to libraries primarily aimed at other library developers and advanced C++ users, like the metaprogramming template (MPL).

In order to ensure efficiency and flexibility, Boost makes extensive use of templates. Boost has been a source of extensive work and research into generic programming and metaprogramming in C++." [9]

Conclusions and Future Work

This thesis presented a Friend-to-Friend (F2F) protocol called Gazebo F2F, which enables the setup of a new F2F network, where users establish connections only to their "friends" i.e. persons that are socially related to them. Based on this model, Gazebo F2F poseses an enhanced and customizable security which has some interesting properties : it is immune to most P2P attacks (like DoS, Sybil or Eclipse) and each node is a single point of failure i.e. a broken node can't jeopardize the good functioning of the network and it usually discovers and recovers quickly from its weird and wrong behaviour. Because the network topology, as seen by a node, is only related to its own circle of "friends", Gazebo F2F introduces implicitly another security feature : anonymity. Users' mobility is not restricted by the lack of an authorised central authority, because Gazebo F2F introduces a new feature called "mobility archives". As shown, all these enhancements could not have been made possible without a thorough, top-down, systematic and modular approach.

The shortage of scientific papers on the Friend-to-Friend topic made our work difficult, because we only had some guidelines of the model that we could follow, but without some proper examples and implementations. There are of course applications that comply to or share features with the F2F model, having various fields of application : Freenet [A2] [A3] - a decentralized, censorship-resistant distributed data store, Turtle [A8] [A9] – an anonymous P2P network intended for the safe sharing of sensitive information, but it also supports file searching and retrival; Tarzan [A4] – a P2P anonymous IP network overlay; OneSwarm [A5] – a privacy-preserving P2P client and data sharing protocol; [A6] presents a lightweight desktop Grid framework which enables to set up a computational friend-to-friend Grid environment with the help of using instant messaging systems; whereas Gazebo F2F's objective has been to implement a "pure" F2F setup mechanism and not just services on a framework that would make compromises when it comes to security. Because of this, we proposed a number of services that should come with future versions, like : filesharing – with regard to the F2F model, data (video/audio) streaming and an integrated platform that would support collaborative work between "friends".

Although the F2F model has been overlooked and underrated, we hope in a further research (on the presented topic) that might uncover substantial advantages, which in turn, could offer P2P and other distributed networks, solutions to some of their security problems.

References

Books

[B1] A.J. Menezes, P.C. van Oorschot and S.A.Vanstone. *Handbook of Applied Cryptography.* CRC Press. August 2001

Articles

[A1] L. Chan, J. Yong, J. Bai, B. Leong. Hydra : A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*. Melbourne, Australia, 2007

[A2] I.Clarke. A Distributed Decentralized Information Storage and Retrieval System. *Ph.D. Thesis*. Division of Informatics, University of Edinburgh, 1999

[A3] I.Clarke and S.G.Miller. Protecting Freedom of Information Online with Freenet. An *IEEE Internet Computing article*. January-February 2002

[A4] M.J.Freedman and R.Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *ACM CCS'02,* November 18–22, 2002, Washington, DC, USA

[A5] T. Isdal, M. Piatek, A. Krishnamurthy, T. Anderson. Friend-to-friend data sharing with OneSwarm. *Technical report, UW-CSE*. February, 2009

[A6] K.Kraaner. Friend-to-Friend Computing. *Master Thesis*. University of Tartu, Faculty of Mathematics and Computer Science, Institute of Computer Science. May 2008

[A7] J. Li and F. Dabek. F2F: Reliable Storage in Open Networks. In *5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, Santa Barbara, CA, USA, February 2006.

[A8] P. Matejka. Security in Peer-to-Peer networks. *Master Thesis*. Charles University, Prague, 2004

[A9] B.C. Popescu, B. Crispo, and A.S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System. In *12th International Workshop on Security Protocols*, Cambridge, UK, April 2004

[A10] B. Pretre. Attacks on Peer-to-Peer Networks. *Semester Thesis*. Dept. of Computer Science, Swiss Federal Institute of Technology (ETH) Zurich. Autumn 2005

[A11] S. Schmid and R.Wattenhofer. Structuring Unstructured Peer-to-Peer Networks. *S. Aluru et al. (Eds.): HiPC 2007, LNCS 4873, pp. 432–442*. Computer Engineering and Networks Laboratory, ETH Zurich. 2007

Web resources

[1] http://msdn.microsoft.com/en-us/library/52f3sw5c.aspx

[2] http://www.wxwidgets.org/

[3] http://en.wikipedia.org/wiki/WxWidgets

[4] http://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits
[5] http://www.cryptopp.com/
[6] http://www.cryptopp.com/fom-serve/cache/44.html
[7] http://www.cryptopp.com/wiki/Related_Links
[8] http://en.wikipedia.org/wiki/Boost_C%2B%2B_Libraries
[9] http://en.wikipedia.org/wiki/OSI_model
[10] http://upload.wikimedia.org/wikibooks/en/2/21/Osi-model-7-layers.png
[11] http://www.learn-networking.com/wp-content/oldimages/osi-model.jpg
[12] http://en.wikipedia.org/wiki/Transmission_Control_Protocol
[13] http://www.software-engineer-training.com/wp-content/uploads/2007/12/tcp_header.png
[14] http://www.h3c.com/portal/res/200705/31/20070531_107804_image015_195599_57_0.jpg
[15] http://en.wikipedia.org/wiki/Internet_Protocol
[16] http://technet.microsoft.com/en-us/library/cc527483(WS.10).aspx
[17] http://www.libantext.com/images/net.gif
[18] http://en.kioskea.net/contents/cs/cs3tier.php3
[19] http://static.commentcamarche.net/en.kioskea.net/pictures/cs-images-2-tier.gif
[20] http://static.commentcamarche.net/en.kioskea.net/pictures/cs-images-3-tier.gif
[21] http://static.commentcamarche.net/en.kioskea.net/pictures/cs-images-n-tier.gif
[22] http://en.wikipedia.org/wiki/Peer-to-peer
[23] http://en.onsoftware.com/wp-content/uploads/2008/01/p2p-network.png
[24] http://www.otcbeyond.com/VoIP-Definitions.html
[25] http://www.webopedia.com/TERM/N/node.html
[26] http://dev.rdxx.com/Files/Pic/Img/Framework/0682712005574928.gif
[27] http://en.wikipedia.org/wiki/Overlay_network
[28] http://wasteagain.sourceforge.net/img/mesh.png
[29] http://www.pepers.org/ulanc/pepers/images/architectures/unstructured_indirect.gif
[30] http://www.lkn.ei.tum.de/forschung/gruppen/p2pg/p2p.jpg
[31] http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p10.html
[32] http://en.wikipedia.org/wiki/WASTE
[33] http://www.facetime.com/solutions/p2pcontrol.aspx
[34] http://en.wikipedia.org/wiki/Small_world_phenomenon
[35] http://publib.boulder.ibm.com/infocenter/tpfhelp/current/topic/com.ibm.ztpf-ztpfdf.doc_put.cur/gtps5/ssldig01.gif
[36] http://en.wikipedia.org/wiki/Cryptography
[37] http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[38] http://i.msdn.microsoft.com/Aa480610.ch7_x509techsupp_f01(en-us,MSDN.10).gif
[39] http://en.wikipedia.org/wiki/One-time_pad
[40] http://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange
[41] http://en.wikipedia.org/wiki/File:Diffie-Hellman-Schl%C3%BCsselaustausch.svg
[42] http://en.wikipedia.org/wiki/Discrete_logarithm_problem
[43] http://en.wikipedia.org/wiki/Friend-to-friend
[44] http://www.bricklin.com/f2f.htm
[45] http://www.wireshark.org/
[46] http://cryptic6.sourceforge.net/
[47] http://cryptic6.sourceforge.net/6deg.ppt
[48] http://turtle-p2p.sourceforge.net/index.html
[49] http://en.wikipedia.org/wiki/Turtle_F2F