Università
Ca'Foscari
Venezia

Master Cycle Degree in

Computer Science

Final Degree

Ca' Foscari
Dorsoduro 3246
30123 Venezia

Protect-to-Prevent: Security of Routing Mechanisms in Peer-to-Peer Networks

Supervisor

Flaminia LUCCIO

Majoring

Alberto STEVANATO

Matricola 823028

Academic Year

2012 / 2013

"*I know you are listening to me, I advice your presence. I know you are scared by us, scare by the change. I do not know your future, I'm not here to tell you how it is going to end. I came to tell you how it will start. Now I'll hung up the phone and I'll show all these people what you do not want they to see. I'll show them a world without you, a world without rules and controls, without frontiers and boundaries. A world in which all is possible. What will happen then, depends on you and on them.*"

(from the film *Matrix*, Watchowsky Brothers)

To all those people who believe in a better future.

Abstract

In the last twenty years P2P technology has begun to spread, mainly because of the increasing need of sharing a large quantity of files. Critical operations such as the exchanging of messages or the search of a file are governed by specific routing mechanisms, that depend on the different P2P structures which are involved.

In this thesis we will first present different types of routing protocols, and we will highlight the structure and the mechanisms by which they carry out the update of the contacts and the transfer of files over a network. We will then present some of the weaknesses of which these protocols are suffering, showing how their routing mechanisms can be used in a wrong way to produce unexpected behaviors by the hosts. We will then show some of the countermeasures adopted in order to mitigate the effects of these attacks. In the last part of the thesis we will first use the Peersim simulator to simulate some of the known routing attacks to the Pastry protocol, we will then simulate some new attacks, and we will finally discuss some possible countermeasures.

SUMMARY OF CONTENTS

INTRODUCTION

In the last two decades the need of new technologies to store and retrieve information has highly increased. Initially, databases were the main structures to store various kind of information, and today are widely used combined with Web sites and Web applications. From the nineties, new technologies to store and retrieve information began to spread and among all, Peer-to-Peer (P2P) oriented technologies. They evolved starting from the simple sharing of music files and other files to the recent developments in P2P technologies: television streaming, live Web streaming and cloud systems. These new technologies began to spread due to their simplicity in the implementation phase and the decentralized behavior they posses that is, each user can independently download or upload contents inside the network. In the last years, file sharing systems were the most known P2P technologies, since the illegal or legal diffusion of multimedia contents over the Web was controlled directly or indirectly by a P2P network.

The first attempts to provide a P2P network for file sharing, i.e. Napster, was a partial ruin due to its centralized behavior. This system was based on a central server in which each reference to the shared contents was stored. For this reason, many people issued the Napster developer for copyright infringements, being able to localize each shared file at any time. Since then, multiple attempts were made trying to provide a more solid application by which to exchange multimedia contents (but not only) through a P2P network. Presently, P2P networks offer anonymity to their users, so the download of copyrighted (illegal) material has became common among users. But anonymity also offers the possibility to evil users to mount several network attacks, given that they can potentially be able not to be recognized. These security problems are surely important, since the entire network can be blocked by a proper attack. Many studies were made on this topic, but solutions against P2P attacks are not simply or always feasible. On the other hand, the security field is full of countermeasures against P2P attacks with many of them basing their strength points over a centralized security control. Even if recently the attention has been moved over other usages of P2P technologies, the security problem remains central. Routing in P2P networks is an important task to achieve in a secure way, since without advices on the security of the routing mechanisms the network will be exposed to consistent damages. Moreover, routing mechanisms behind each P2P protocol present security weaknesses, since many possible misuses can be performed with each routing protocol. For these reasons, the study over the security of P2P applications is a must today, and more studies need to be performed in terms of attacks and countermeasures to apply on P2P routing protocols.

The main topic of this thesis are attacks to routing mechanisms in P2P networks and possible countermeasures. In particular, we will start illustrating P2P architectures, paying particular attention on the mostly used routing protocols and the applications that make use of them. We will also highlight the weaknesses to which every P2P network is exposed, showing how attacks can be technically mounted and how they can be mitigated with the correct countermeasures. Then we will show a simulated real protocol, we will test some of the exposed attacks and we will present some new attacks. We will then propose some countermeasures to mitigate these attacks introducing some modifications to operate on the original protocol.

This thesis is organized as follows. In Chapter 1 we will introduce the P2P technology and some preliminary notations. In Chapter 2 we will illustrate some of the routing protocols widely used today by P2P networks, presenting their structure and their main features, and we will show how their routing mechanisms work. In Chapter 3 we will introduce some network attacks of which P2P networks (but also other kind of networks) suffer from. We will give an exhaustive definition of each attack explaining which weaknesses of the routing protocols they force and which are their points of strength. In Chapter 4 we will present some solutions proposed by different authors to the attacks shown in Chapter 3, evaluating each solution in terms of efficiency and effective efficacy. Finally, in Chapter 5 we will test some of the attacks exposed in Chapter 3 and we will propose some new attacks over one of the protocol presented in Chapter 2 using the Peersim simulator. We will then try to apply some countermeasures to these attacks, showing how they can be mitigated. At the end of this thesis we will finally present some conclusions.

CHAPTER 1

Preliminaries on P2P Networks

In this chapter we will first start with a brief introduction on the history and usage of P2P systems and we will then recall some definitions and standard terminology.

1.1 Introduction to P2P networks

With the term **Peer-to-Peer** (acronym *P2P*) we refer to a communication model in which each of the participants (or **entities**) is able to communicate with the other entities, and to share resources. Moreover, each entity is equal to the others in terms of capabilities and functionalities, i.e. it may act both as a Client, when requesting a resource to another entity, or as a Server, when supplying a resource to another entity. In other words there is no central server inside the network and all nodes cooperate in order to allow the entire network to provide its services in a distributed way. Formally:

Definition 1 [79]: "***Peer-to-peer*** *is an approach to computer networking where all computers share equivalent responsibility for processing data. Peer-to-peer networking (also known simply as peer networking) differs from client-server networking, where certain devices have responsibility for providing or 'serving' data and other devices consume or otherwise act as 'clients' of those servers.*"

Very often shared resources are files, thus an advantage of this technique is that the underlying network can deal with a large amount of traffic whose load is equally shared between all the entities. P2P networks are more scalable and resilient than normal Client/Server networks because they do not rely on central entities to perform actions and, more important, they are robust to faults because files are replicated and communication links are redundant. Formally:

Definition 2: *A network is* ***robust****, or fault tolerant, if it can work even in case of multiple malfunctioning of entities inside it*; *it is* ***flexible*** *if it can adapt to multiple and frequent changes to the network, such as addition and deletion of entities*; *it is* ***scalable*** *if entities can be added or deleted on demand*.

1.1.2 History of P2P networks

P2P networks began to spread in the last twenty years, however they were already used long time ago by public institutions, such as universities and public companies **[52]**. Their first usage was designed to exchange information within a Local Area Network or small Wide Area Networks. In late seventies Tom Truscott and Jim Ellis, two graduate students from Duke University developed **Usenet**, a network to exchange information within the Unix community, based on a P2P network

[113, 114]. Using Usenet, students between two different universities were able to exchange information, post news, comment posts and so on.

The P2P networks we know today, i.e. file sharing networks, were introduced in the late nineties with the advent of **Napster**, a network created by Shawn Fanning in 1999 with the intent of supplying music files whose parts were shared among users **[88]**. He observed how difficult it was to share music online but, on the other hand, many users in the Internet had these music files on their hard disks, thus the sharing had to be really simple by using the right mechanism. Thus he used the instant-messaging system IRC of Microsoft and Unix, combined with the advanced capabilities of search engines, to create a network in which to share music files. The main problem with this network was its centralized fashion. In fact, file descriptors were maintained in a central server, or a set of servers, to which users were able to send requests when needed. This made the ruin of Napster whose creator was accused of copyright infringement from Metallica, a famous rock band. In practice, the prosecution pointed out that, as there was a central point in which file descriptors were residing the administrator could have blocked illegal files diffusion instead of allowing their distribution for free. The process went on for about an year, after which Shawn Fanning had to pay 26 million dollars. Two years after, in 2003, Napster came back as pay service, together with many other P2P programs such as **Gnutella** and **Kazaa**, the second one which is now offline **[82]**. These new programs allowed file sharing to be totally distributed, thus making harder the identification of the downloaded content. Today P2P applications are used in many fields, in particular file sharing and video streaming. Regarding file sharing, the most popular programs are: Bittorent clients such as **uTorrent**, **BitLord**, **Vuze** or **BitComet [104, 112]**; eDonkey derived clients such as **eMule [81]**; Gnutella-based programs such as **Limewire**, **BearShare** and **FrostWire [112]**. More details about all these P2P programs will be given in Chapter 2. Questions about legality of file sharing or, simply, P2P networks usage are many and are constantly increasing **[77, 80, 106]**.
Recently, issues related to P2P networks have evolved from normal file sharing to instant messaging applications, live video streaming and television broadcasting **[112]**. The main issue with decentralized systems is trust because P2P networks, as dynamic systems, are commonly targeted by networking attacks. The main problem here is to localize malicious users in the network to block them and to avoid their actions in terms of not compromising the network. More details will be given in Chapters 3 and 4.

1.2 Notation

In this section we introduce some notation and we formally define what a P2P network is **[35]**.

A **Peer-to-Peer** system (acronym *P2P*) is formed by autonomous entities which are able to organize themselves and to share resources inside a network. Resources are used to provide some functionalities in a total – or partial – decentralized way. As **functionalities** we intend computational features or capabilities maintained by each entity in the network, i.e. file sharing, storing space, computing power and bandwidth.

With this intent we can give a more precise definition of P2P system different from the one given in the previous section:

Definition 3 [90]: "*A distributed network architecture may be called a* ***Peer-to-Peer*** *network, if the participants share a part of their own hardware resources. These shared resources are necessary to provide the service and content offered by the network. They are accessible by other peers.*"

Resources may be, e.g. processing power, storage capacity, network link capacity, printers, and devices are, e.g. file sharing or shared workspaces for collaboration. Entities are interconnected to each other. The system is able to self-organize to the constant change of the entities leaving unchanged the connectivity and the performance of the network without the use of a centralized unit. Entities in these kind of systems are usually called **peers**. Special peers that are both client and server are called servents. Formally:

Definition 4: *A* ***servent*** *is an entity that can act both as a Client and as a Server. The functionalities it provides are called* ***symmetric*** *since a peer can both provide or request a resource in the network.*

Formally a P2P network can be represented as a graph:

$$G = (V, E)$$

where V is the set of **nodes** (entities or peers) and E is the set of **edges** (links or connections between nodes). We assume $|V| = N$ and $|E| = M$. We call N the **size** of the network.
Connections among nodes in a P2P system are said to be transient:

Definition 5: *A connection inside a P2P network is said to be* ***transient*** *if it allows frequent insertions and deletions of nodes. The ratio between the number of joining and leaving nodes is called* ***churn****.*

Since each node can enter or exit from the network at any instant of time also resources associated to nodes are added or deleted dynamically from the system.
Insertion and deletion of nodes requires the exchange of different information. Each node has associated an **IP address** and a unique identifier (acronym **ID**). The IP address is dynamic, i.e. it may change at any new connection to the network. The ID is a special unique number calculated with a cryptographic hash function whose arguments are either the IP address, or the IP address together with the **port number**, that identifies the access point for an host to which messages are delivered by the routing protocol **[59, 64]**.

Definition 6: *A cryptographic* ***hash function*** *is a function that translates a variable length content in a fixed length string. It has the following properties: it is easy to compute; it is computationally infeasible to find two contents with the same hash; it is infeasible to generate a content that has a given hash; it is infeasible to modify a content without changing the hash.*

Hashing operations are made by hash functions.
Once a node wants to connect to the network it either provides to the other nodes a pair **(IP address, ID)** or a triplet **(IP Address, Port number, ID)** depending on the protocol.
Resources are identified via an unique ID, as for nodes, calculated by the same hash function used for calculating node IDs, but having as argument the name of the resource and its content. Every protocol, however, uses its own way to create identifiers for resources as we will see in Chapter 2.

We now introduce the concept of routing inside a P2P network.

Definition 7 [98]: "***Routing*** *is the process of moving packets across a network from one host to an another. It is usually performed by dedicated devices called* ***routers***."

In a P2P network each node acts as router and messages are forwarded from node to node until the destination is reached. The concatenation of all the nodes the message traverses is the path for that message. Every time we will make some routing examples we will use two nodes called source and destination.

Definition 8: *The* ***source*** *s is a node that starts a routing mechanism, by sending a message M to another node called* ***destination*** *d.*

Each message exchanged in the network by a source and a destination needs to traverse a path, that can be unique or not.

Definition 9: *A* ***path*** *in a P2P network is a set of nodes traversed by the message sent from a source to a destination. The path can be* ***unique****, if there exists a unique sequence of nodes connecting the source and the destination.*

Formally, if a node source s wants to send a message M to a destination d, M has to follow a path, i.e. a concatenation of edges $P = [\{x_1, x_2\}, \{x_2, x_3\}, \dots, \{x_{i-1}, x_i\}]$, where $x_i, z \in V$, $i: \{1..k\}$, $x_1 = s$ and $x_k = d$, $\{x_j, x_{j+1}\} \in E$, $j: \{1, \dots, k-1\}$.

Each node, finally, maintains a routing table.

Definition 10: *A* ***routing table*** *is a table with several entries containing references to other nodes in the network. A* ***distance routing table*** *contains the distance from the actual node position to a specified other node in the network. Given a graph G = (V, E) the* ***neighborhood*** *of a node x is the set of all the nodes connected by an arc to x, i.e.* $\{z_i \mid \{x, z_i\} \in E, x \neq z_i \text{ and } z_i \in V\}$*.The maximum degree D of G is the maximum number of neighbours of all nodes in G.*

The content of the routing table changes depending on the protocol used inside the network. More details will be given in Chapter 2.

Routing can be static or dynamic depending on the protocol.

Definition 11: *Routing is **static** if the routes through which a message can be sent are fixed. Routing is **dynamic** if nodes are inserted and deleted and routes between a source and a destination change over time.*

In P2P Networks the routing mechanism is clearly dynamic as there might be many events changing the network (node joins and leaves, crashes, faults, etc.).

Messages exchanged in the network have an expiration time, usually called hop number (the summary of the number of hops) or Time To Live.

Definition 12: *The **hop number** for a message M during a routing operation in a P2P network is the number of edges traversed from M to move from the source node s to the destination node d.*

Definition 13: *The **Time To Live** (**TTL**) is the maximum time of permanence of a message in the network, i.e. how many hops a message travels before being discarded.*

The choice of the next hop is usually demanded to an ad-hoc algorithm, such as the Prefix Matching Algorithm. Formally:

Definition 14: *Given a source node s and a destination node d, the **Prefix Matching Algorithm** chooses the neighbor of s whose ID has the longest common prefix with s.*

Each message delivery is also characterized by the latency.

Definition 15 [115]: *"In a network, **latency**, a synonym for delay, is an expression of how much time it takes for a packet of data to get from one designated point to another."*

Once a new node needs to login in the network, usually it contacts a random selected node already inside the network called the bootstrap node. Formally:

Definition 16: *A **bootstrap protocol** is a protocol that returns to the node joining the P2P network one or a set of nodes already inside the network. A **bootstrap node** is a node returned by the bootstrap protocol.*

Finally, sometimes we will analyze some events in which a peer is said to be behind a NAT.

Definition 17 [89]: "*A **Network Address Translation** or **Network Address Translator** (**NAT**) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside.*"

P2P systems can be centralized or decentralized depending on the presence of a centralized server **[35]**.

Definition 18: *A P2P system is **centralized** if each node connects to a central server, called **broker**, through which it can retrieve information about a resource or it can share a resource.*

Thus in this setting the network is a star (see Figure 1) and if the central node, i.e. the broker, fails files cannot be retrieved or exchanged.

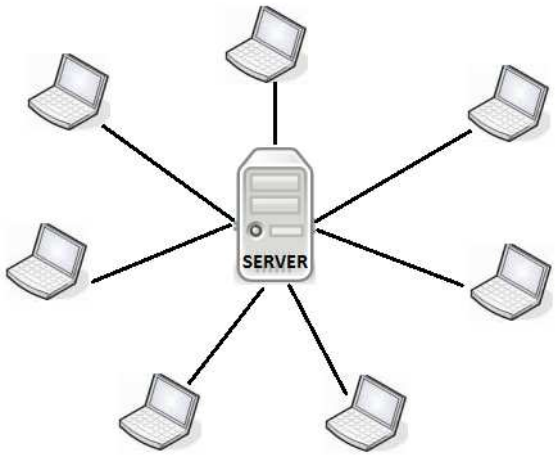

Figure 1 An example of star topology.

Definition 19: *A P2P system is **decentralized** if there is no central server and the information is distributed among the nodes.*

That is, once a peer needs to find a resource identified by an ID it sends a message to a subset of its neighbors (depending on the protocol chosen) requesting the resource. Then each neighbor contacts its own set of neighbors to locate the resource, and the process goes on until the resource is located. After this, the starting node receives the IP address and the port number of the destination node and the share begins (see Figure 2). In this kind of systems the service with which queries are computed and delivered, and how the source node receives the information of the destination node, depends on the underlying protocol used.

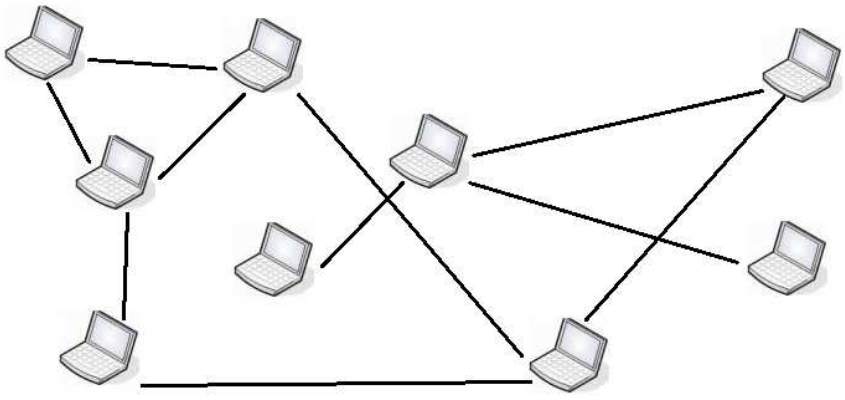

Figure 2 An example of P2P Decentralized Network.

Hybrid P2P Systems are intermediate architectures where nodes are organized in a hierarchical way ordered by dynamically created levels.

Definition 20: *An **hybrid P2P system** is a system in which there are both nodes and supernodes (superpeers). Superpeers are servers that index resources maintained in the network. Once a node*

wants to retrieve a precise resource, it contacts the superpeer which provides information about the node that stores the resource.

An example of this architecture is represented in Figure 3. Usually these solutions are used to improve the performance of the system, since queries and message forwarding are faster and the traffic in the network is lowered.

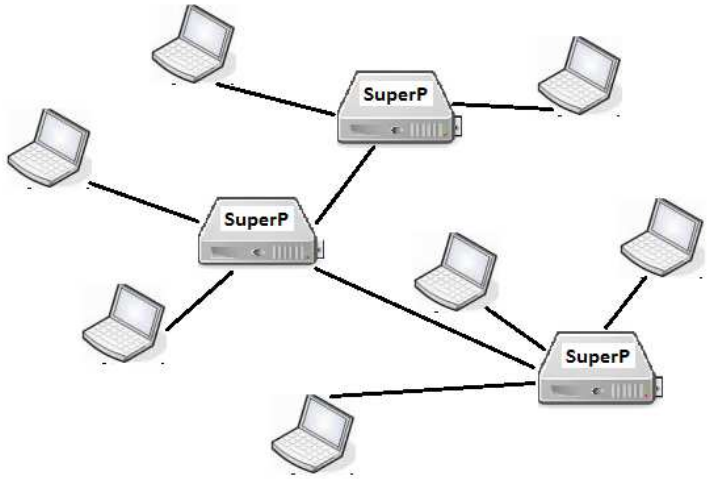

Figure 3 An example of Hybrid P2P Network.

The main purpose of today P2P networks is content distribution. In networks where resources are stored inside some supernodes, these supernodes are central point of failures. A **segmentation approach** is used to overcome this problem, that is the resource is split among nodes which collaborate and exchange their own parts in order to reconstruct the whole resource. Let us now show an example of this approach.

Example 1.1 – The segmentation method

Let us suppose to have the network of Figure 4 where node S is the supernode and the other are simple nodes. Let us suppose that nodes A, B and C request to S the file it stores. Since the approach of distribution is segmented, A, B and C will receive different parts of the file and will then exchange the remaining parts among them (Figure 5). At the end of this process nodes A, B, and C will have the entire file and can start providing it to other nodes that want to download it (see Figure 6). In this way the traffic directed to S is low since at this point some requests will be satisfied by A, B and C.

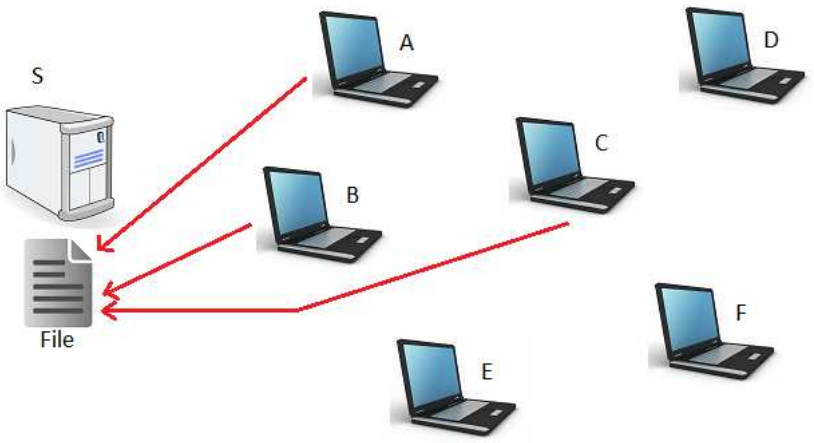

Figure 4 Starting situation for Example 1.2.

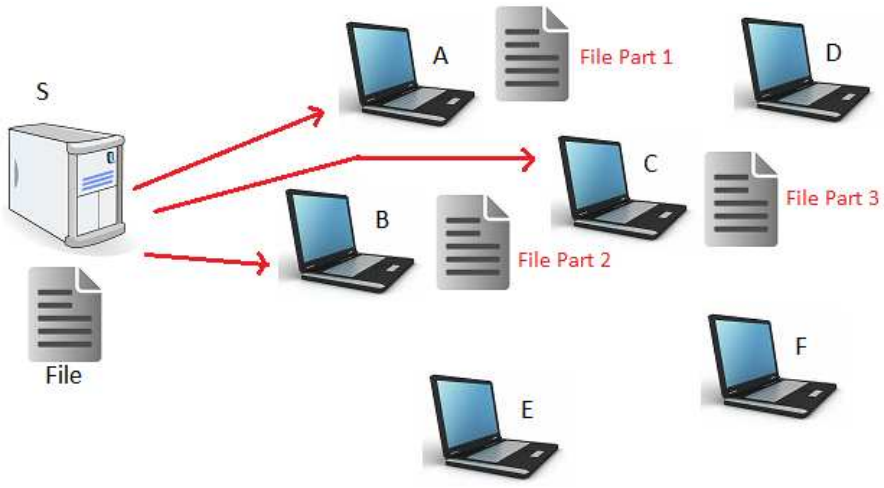

Figure 5 Intermediate state of the distribution.

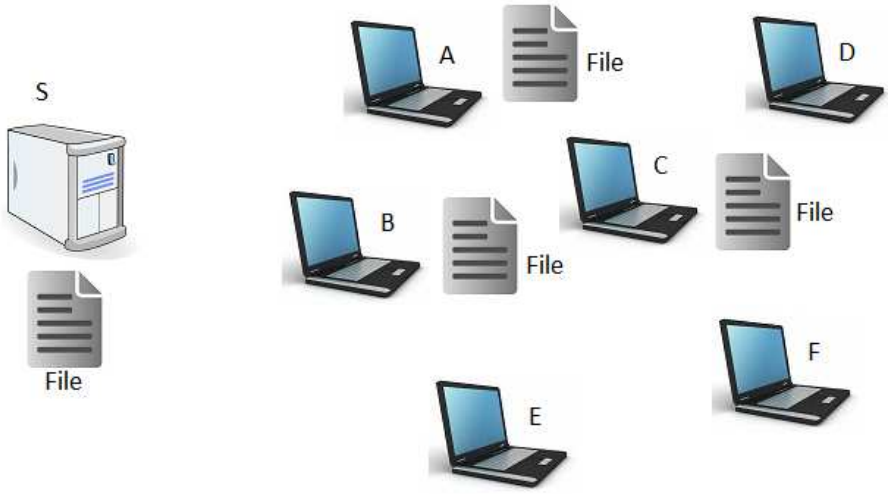

Figure 6 Final state of the distribution.

◊

An evolution of this system is the **P2P Storage Network** where a cluster of nodes uses its computational resource to supply a storage service for the whole network (see Figure 7).

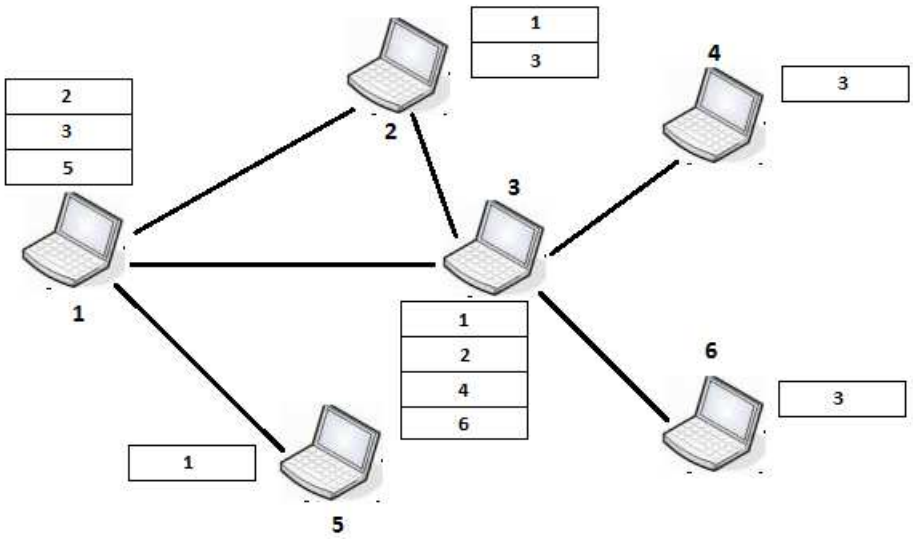

Figure 7 An example of Storage Network.

Example 1.2 – A Storage Network

Let us take as example Figure 7. At startup each node gives disposition for use a certain portion of its storage memory identified by an ID created with an hash function. Each node maintains a threshold for its storage area, that identifies the maximum amount of data that can be inserted in its own area. The maximum capability of the network is the sum of all the thresholds. A table is then maintained at every node and each entry locates a neighbor in the network maintaining a certain resource. The identification for resources is made via identifiers calculated by an hash function on the name of the resource or part of its content. Tables are initially filled with some hello messages propagated in the network, with which each node acknowledges the existence of some neighbors in the network, and stores its identifier. Once a search begins, a query is sent along the network, and each node contacts a set of its neighbors looking at those with the nearest ID to the destination. In Figure 7, for example, node 1 sends an hello message that is replied by nodes 2, 3 and 5. Node 1 stores the IDs for these nodes in its table, and the other three nodes iterate the process. Let us suppose that the resource maintained by each node have an ID composed by four digits, which the first is equal to the manager ID and the other three are random characters. Once node 1 wants to locate resource R identified by the ID 6acf, maintained by node 6, it propagates the query passing as destination the ID of R. In this example, resource R shares the first digit with node 6, thus node 6 is its manager and the search process proceeds looking for the node which shares the first digit with the resource ID.

◊

The tables presented in Example 1.2 are usually **Dynamic Hash Tables** (acronym *DHT*) in which each entry constitutes a correspondence between keys and values of entities inside the network, i.e. in the case of nodes they match the ID of the node to its IP address and port number. Before explaining how the DHT works, we define what an hash table is.

Definition 21 [76]: "*An **hash table** is a lookup table that is designed to efficiently store non-contiguous keys that may have wide gaps in their alphabetic and numeric sequences. Hash tables*

are created by using a hash function to hash the keys into hash buckets. Each ***bucket*** *is a list of* ***(key, value)*** *pairs. Since different keys may hash to the same bucket, the goal of hash table design is to spread out the key-value pairs evenly with each bucket containing as few key-value pairs as possible. When an item is looked up, its key is hashed to find the appropriate bucket. Then, the bucket is searched for the right key-value pair.*"

Keys created for the hash tables are non contiguous since each key is not followed directly by another, i.e. between two keys there can be a certain amount of space where other keys can be created.

Each entry in these tables are formed by a pair (key, value) and are grouped in certain containers called buckets (see Figure 8). In a P2P network these buckets identify a portion of the network where nodes inside it have a common prefix in their identifiers. This structure allows to identify a resource inside a bucket by hashing its key and searching the bucket that contains it. Once the bucket is found, the correct (key, value) pair is queried. To have an example of an hash table see Figure 8.

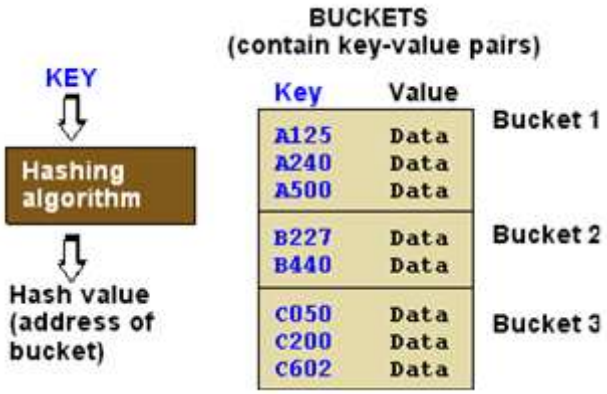

Figure 8 An example of Hash Table **[76].**

Example 1.3 – Mapping resource into hash tables

Let us suppose to be in a network where node A inserts a resource whose content has to be hashed in order to associate a unique identifier to it. The hash, in our example, is built over the name of the resource, namely MyFile.txt. The corresponding hash, part of the MD5[1] *coding of the name of the resource (without the extension), is then:*

7039d49e15fg4e164e2c07

and it will be inserted in the corresponding bucket, as shown in Figure 9, i.e. in the one where resource IDs start with ***70****. Each node knows that the resource mapped to ID* ***7039d49e15fg4e164e2c07*** *has associated a specified value that identifies the content of the resource (a number, the resource name, etc.). In this case the value corresponds to the name of the resource, that is "MyFile.txt". The value is returned once the ID is queried by some nodes in the network.*

[1] *MD5 is a cryptographic hash function created by Ronald Rivest in 1991 and part of the RFC 1321 standard* ***[107, 108]***

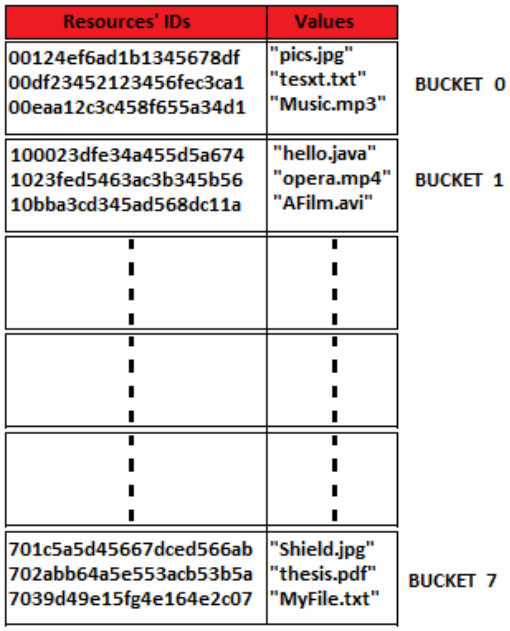

Figure 9 Example of hash table.

◊

From these structures evolves the concept of Distributed Hash Table.

Definition 22 [68]: *"A **Distributed Hash Table** (DHT) is a distributed and often decentralized mechanism for associating Hash values (keys) with some kind of content. Participants in the DHT each store a small section of the contents of the hashtable."*

This mechanism is widely used in P2P networks to provide scalable, fault-tolerant and flexible networks. We will see in the next chapter how DHT works and which are their main characteristics.

Finally P2P networks can be unstructured or structured **[42]**.

Definition 23: *A P2P network is said to be **unstructured** if links between nodes are established in an arbitrary way, without correlation between nodes and resources maintained by them.*

Briefly, this means that resources managed by some nodes may be stored at whatever position inside the network, not necessarily near the node that maintains them. To locate a resource a node needs to flood the whole network with a request to reach all the nodes. The disadvantage of this technique is that the traffic load inside the network is considerably high and the search efficiency is not so good.

Definition 24: *A P2P network is said to be **structured** if links between nodes are not arbitrarily established and resources are placed at precise locations*.

At each resource is assigned a key and peers are organized inside a graph that maps each of these keys to a specified node. This helps in precisely locating a resource in the network using the key of

the resource. In this kind of networks is usually applied the DHT mechanism, since each node knows where a specified resource is maintained in the network.

At this step we can start introducing **P2P protocols**, that are protocols defining the set of messages exchanged in the P2P network in which they are used. Messages are intended in terms of their format and contents. As we have previously explained, each protocol uniquely identifies a node through a unique identifier obtained via hashing its IP address and port number. Protocols are defined at the application level of the **TCP/IP** protocol, i.e. the level containing all the specifications for communications in a protocol-to-protocol environment, since they define how networks information are routed among nodes. Protocols are defined at a logical level on the so-called **Overlay Network [35]**.

Definition 25: *The **Overlay Network** identifies the way nodes are connected via their ID, i.e. the maximum level of abstraction of the network. This logical connections correspond to the graph behind the network, through which the routing mechanism works. At each logical link can correspond one or more physical links.*

Logical links correspond to edges in the graph. The Overlay Network is completely independent by the Physical Network and it can be structured in a hierarchical way and can include a central server. In the Overlay network information is exchanged via TCP connections. Thus the Overlay Network defines the structure of the network, i.e. if it is structured or unstructured, and the routing mechanism behind the network. P2P systems are classified looking at their Overlay Network, and with this network they can achieve scalability.
A Physical Network is the network through which nodes exchange data after locating them through the Overlay Network.

Definition 26 [63]: "*The **Physical Network** defines a system of computers that communicate via cabling, modems, or other hardware, and may include more than one logical network or form part of a logical network".*

Once a message have to be delivered, the source s queries the Overlay network to find the path from itself to the destination d. In this case, only the Overlay network is used, since s will forward the message through logical links in the overlay. After the destination is located, and s receives information about d, the Physical network is used to send the message from s to d passing several physical links, that can differ from logical links **[23]**. As example we can take a look to Figure 10 where Overlay (top) and Physical (bottom) topology for a network are represented.

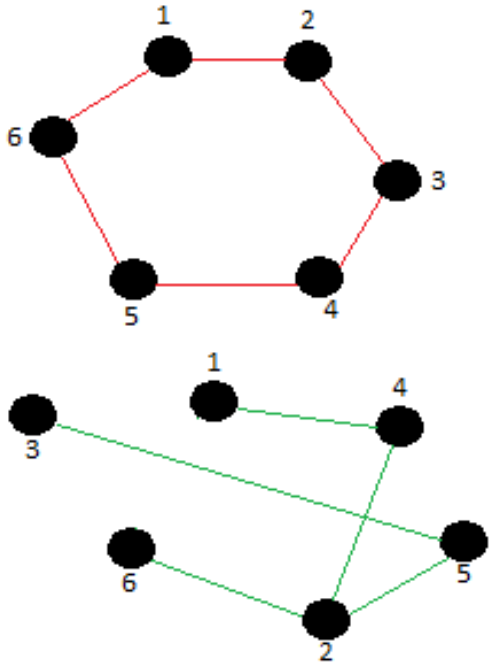

Figure 10 Overlay Network vs. Physical Network.

Example 1.4 – Overlay vs. Physical Network

Let us suppose to have the situation of Figure 10, and that node 1 needs to send a message to node 3. Node 1 uses the Overlay network to locate node 3 in the Physical network, i.e. to receive its IP address and port number from node 2 that is the nearest node to 3 that node 1 knows. After this, node 1 will use the Physical network to send the packet to node 3, passing nodes 4, 2 and 5 respectively until it reaches node 3.

◊

CHAPTER 2

Routing in P2P Networks

In this chapter we will describe different methods for routing in P2P networks. We will start with a general introduction and then describe different techniques in detail.

2.1 Introduction

As we have previously mentioned, a routing protocol has the aim of delivering messages from a source node to a destination node. All protocols have to deal with message forwarding, routing tables updates, contents indexing and nodes authentication in the network. What differs from protocol to protocol is the information maintained at each node and how these information, usually called **metadata**, is queried and provided in the network **[44]**. The lack or the presence of metadata inside the network identifies the type of the network, that is in unstructured networks there is no metadata stored and the only way of performing routing is by broadcasting the request, i.e. by sending it to all the neighbors and repeating this operation up to when the destination node is reached **[35]**. On the other hand, in structured networks nodes possess some precise information regarding the entity location, i.e. each node precisely knows which is the node to forward a request to when a resource is needed.

In this chapter we will introduce and explain different routing mechanisms in unstructured networks, such as Gnutella 0.4 **[35, 74, 102]**, in structured networks, such as Napster **[35, 78, 87, 109]**, Chord **[7, 12, 35, 40]** and Pastry **[37]**, Kademlia **[27, 31, 32, 35]**, KAD **[31, 35, 71]** and Tapestry **[50, 92]**. We will also introduce and explain routing in hybrid P2P systems, i.e. Gnutella 0.6 **[35]**. Finally, we will present the networks which are mostly used today, i.e. BitTorrent **[35, 54, 103]** and eDonkey **[35]**.

Before analyzing all the different protocols we need to introduce some parameters used to evaluate protocols:

- **Usability:** This feature deals with the type of queries that can be used inside the system and how a protocol is easy to use in the network. An example of this is the case of query usage: some protocols may allow the use of very complex queries while others may not.
- **Storage:** To maintain metadata inside the network the protocol needs to supply a storage space in which to maintain them. The protocol needs also to specify the amount of storage space allowed in the network, i.e. on each node.
- **Coverage:** This feature concerns the number of processed requests inside the network. A high number implies a good coverage.
- **Efficiency:** This is computed in terms of response time for a resource request. With this parameter becomes important the notion of latency.
- **Security:** Is the ability of a protocol to maintain information without corruption and to securely supply it to nodes inside the network.

- **Anonymity:** is the ability of a protocol to maintain secret the identity of a node that has requested some information inside the network **[26]**.
- **Scalability:** represents the ability of the system to grow or decrease on demand **[99, 100]**.
- **Reliability:** is the ability of the system to overcome faults in an efficient way.

In the following sections we will present all protocols pointing out their topology, i.e. the Overlay structure, how routing is performed, which information structures are maintained by each node and giving some examples of routing. We will begin with a brief explanation of what are the routing techniques used by P2P protocols.

2.2 Routing Techniques

In P2P networks, routing can be of three kinds: iterative, recursive and trace routing **[43]**. What differs among them is the way messages are forwarded and which node handles the communication. Assuming a source node s wants to send a message M to a destination node d:

- ✓ **Iterative Routing:** with this solution, s has total control over the message forwarding. Once s needs to send M to d, it checks in its routing table if d appears among its entries. If it appears, s directly sends M to d, otherwise it selects the nearest node to d to contact, via the metric used by the underlying P2P network protocol. The newly contacted node, namely b, iterates the process giving back the IP address and port number of d if it has them in its routing table. Otherwise b sends to s one or a set of contacts (depending on the specifications of the underlying routing protocol) for nodes nearest to d than itself. s contacts the nearest node to d among the contacts provided by b and the process iterates until d is reached (see Figure 11). The main advantages for this solution are that s can easily trace the routing path of M and quickly locate crashes in the network **[17]**. The main disadvantage in this technique is the high latency since the path to d is computed in different increasing hops.
- ✓ **Recursive Routing:** with this solution, s does not have total control over the message forwarding. Once s needs to send M to d, it checks in its routing table if it posses information about d. If it has them s directly delivers M to d. Otherwise s sends M to the nearest node (depending on the protocol metric) to d in its routing table. This new node, namely b, repeats the previous process, contacting s only if it possess information about d. If it does not have these information, it forwards the message to the node nearest to d that appears in its routing table, and the process repeats. S is contacted again only from the node that possess the IP address and port number of d (see Figure 12). The main advantage is that generally the latency is low, since messages are quickly passed among nodes in the forwarding path. The main disadvantage is that s does not control the entire forwarding phases and some errors or attacks can occur during them. This will be discussed in Chapter 3;
- ✓ **Trace Routing:** this is a result of combining the previous two techniques. In practice, once a node needs to forward the message, it sends both M to next node in the path and a message back to the source to communicate the next step.

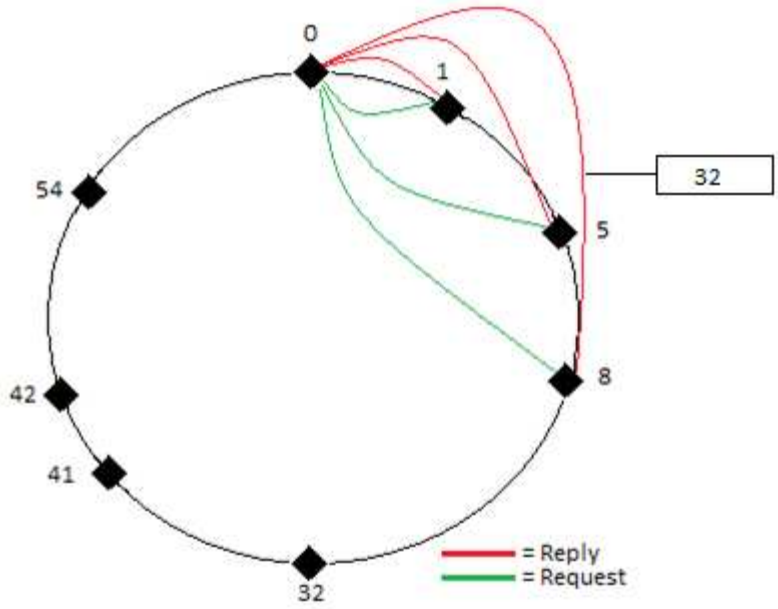

Figure 11 Example of Iterative Routing.

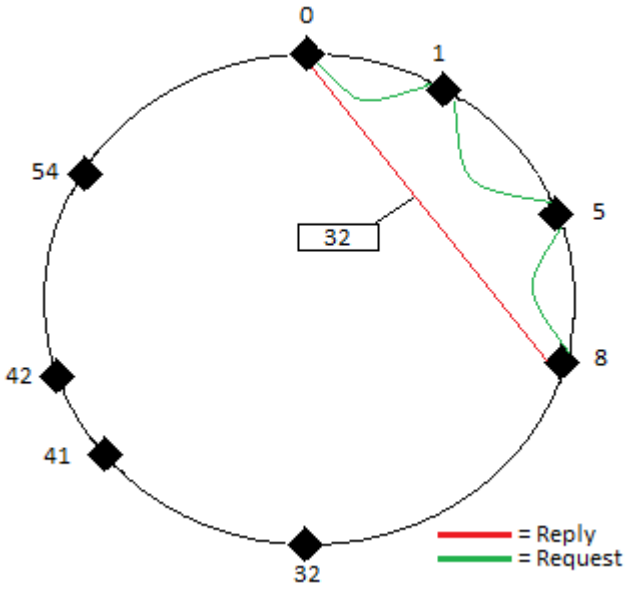

Figure 12 Example of Recursive Routing.

We will now outline the main P2P protocols widely used in today file sharing applications. Moreover we will point our attention on how they manage and update the overlay network, the join and leave of each node and resource management. Furthermore we will see how each protocol deals with message routing inside the network. We will start with unstructured networks, and in particular with the Gnutella protocol.

2.3 Routing in Unstructured networks

In this section we will introduce routing protocols for Unstructured P2P Networks. We will give a general introduction on the methodologies used by these kind of protocols to perform the

bootstrap phase, the authentication phase, the message forwarding and the lookup phase. We will also give an Protocol Evaluations based on the parameters introduced in section 2.1. As an example we will describe the Gnutella 0.4 protocol.

The Bootstrap phase As explained in Section 1.2, unstructured networks lack of a centralization point, so a bootstrap procedure for the network needs to be used. With Gnutella 0.4 this phase relies on some entities called **beacon servers**, that store a list of nodes inside the network that have a high probability of being available. These servers identify a centralization point only for the bootstrap procedure. Each node stores in a system file a list containing all the nodes contacted in previous sessions.

Resource Search Given the lack of stored metadata, nodes do not notify resources they want to share and paths to nodes storing resources are created dynamically. Moreover, a search mechanism does not exist so there is the need to provide a decentralized way of forwarding queries, and a system to uniquely identify messages exchanged by nodes in order to avoid cycles in messages exchanging. Once the source wants to locate a resource, it sends a query to all its neighbors with a specified Time To Live for each query. This allows the message not to be forwarded forever in the network, but the scalability is poor since if a node joins after the message is passed through its position it will not be contacted in the forwarding phase. Since the definition of a correct TTL is non-trivial, the network allows a peer to send the query with a low TTL and, if the resource is not found, it can send again the same query with an higher TTL. The process is iterated until the correct TTL is found. Even in this case the network is not much scalable, since the joining or leaving of nodes is not taken into account. Once the query or the message reach the destination, the positive answer is sent in **backward routing**, i.e. the message follows the path in opposite direction, and the content is transferred from the destination directly to the source.

Message Delivery In unstructured networks a neighbor to which to forward the message is sent on a random walk, to reduce the number of messages and the latency for each message.

Definition 27 [65]: "*A **Random Walk** is a mathematical model used to describe physical processes, such as diffusion, in which a particle moves in straight-line steps of constant length but random direction*."

To speed up the message delivery above described, multiple random walks are usually computed, where a set of neighbors are chosen randomly to perform the delivery. The number of messages clearly grows but the latency decreases. As before, to stop the message forwarding the TTL value is used.

In the next section we will analyze an example of routing protocol for unstructured network, namely the Gnutella protocol version 0.4.

2.3.1 The Gnutella 0.4 protocol

Gnutella is the first completely decentralized network that was proposed by Frankel and Peppers **[35]**. In this protocol resources are not maintained inside a central server but are maintained distributed among nodes in the network and, at a certain time, they are provided to nodes that

request them. In Gnutella any node (X) can see resources from a small number of other nodes, that are those queried during the search phase. Moreover, Gnutella allows users to exchange each kind of files, such as music, videos, photos, etc.

We will now illustrate the different mechanisms inside the network.

Bootstrap mechanism When a node j joins the network there are two possible situations:

1. J has never joined the network before, thus it contacts the beacon servers which in the last version of Gnutella 0.4 are called GWebServers and that provide to j the IP addresses and IDs of some nodes inside the network. These nodes are chosen looking at their lifetime (starting from the oldest), at the dimension of exchanged files (preferring nodes with an high amount of exchanged data) and at the last time they were contacted by the GWebServer (preferring fresh contacts). Each node stores these information in an internal cache that contains a hundred contacts refreshed with pong messages that will be received in the future by both GWebServers or other nodes in the network. The chosen nodes are then contacted by j by opening TCP/IP connections with them.
2. If the node has already joined the network in the past, then it launches the Gnutella client and the local cache is loaded to perform the bootstrap algorithm. J contacts the nodes inside its local cache, it waits an amount of K seconds (typically K = 5 seconds) and if no responses are received it tries to contact a GWebServer. If this node does not respond after H seconds (usually H = 10 seconds), another server is contacted and the process is repeated until a contact is made. J makes sure a server is never contacted twice during the bootstrap phase by maintaining references for already contacted servers during the actual session. Note that when a node leaves the network its cache is stored locally and it is updated with the latest information received by the network before the node lefts.

At this point, the authentication phase starts.

Authentication This phase concerns the network exploration through the bootstrap node at which j connects. Once connected, j sends a **ping** message to the bootstrap node that answers with a **pong** message, containing its IP address and port number that j will store in its routing table. The bootstrap node (b) will iterate the process by sending the ping to all its neighbors and will receive back pong messages. Each node accepts a maximum number of connections from other nodes, in order to not overload the traffic to its position. Each servent also determines a maximum threshold C of opened connections to other nodes during the bootstrap phase where $C \leq T$ (T is the amount of ongoing connections) and the remaining $C - T$ connections are used for incoming links from other nodes. When a node receives a ping message it increments the number of linked nodes and it decrements the TTL of the message, it then forwards the message to a subset k of neighbors (k is usually set equal to 4) except the sender. If a node receives an already processed message it ignores it, and if the TTL is equal to zero it process the message and stops the forwarding phase. To create a Physical network j then opens a set of TCP connections to the chosen set of nodes. An example of the authentication phase is outlined in Example 2.1.

Example 2.1 – Example of Authentication in Gnutella 0.4

Let us suppose node A of Figure 13 is the joining node. After contacting a beacon server, internal or external to the network, node A receives information on node B already inside the Gnutella network and sends a ping message to B, with a TTL of 7 (the standard value of the network).

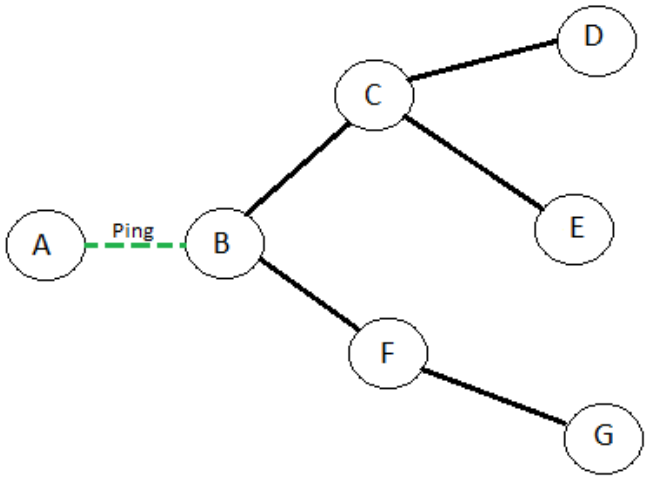

Figure 13 Authentication phase one.

Once B receives the ping message it decrements the TTL, that is now 6, and answers to A with a pong while it forwards the message to C and F. A receives the pong from B and inserts its contact in the local cache, as shown in Figure 14.

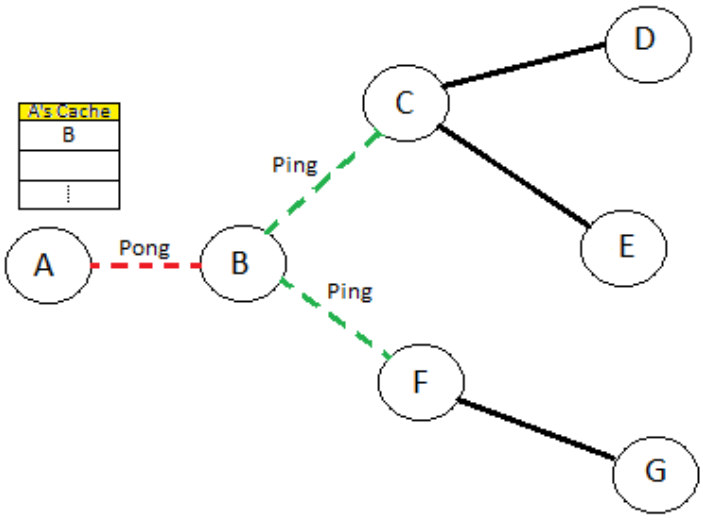

Figure 14 Authentication phase two.

The process goes on until the TTL for A first message expires. After this event, the internal cache of A is the one in Figure 15, i.e. it contains the IP address, port number, number of files, Kilobytes exchanged and hop number. This cache is updated periodically, when a new node joins the network. If this never happens, then the system is updated after y seconds, with updates forwarded to all the links in the network. The value of the parameter y changes with respect to the usage of the pong caching mechanism by the network, as explained further in this section.

	IP	port	F	Kb	H
B	10.0.0.1	4672	5	2657	1
C	10.0.0.2	4651	2	1672	2
E	10.0.0.8	5632	1	765	3

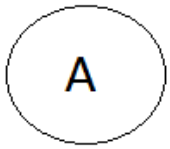

Figure 15 The cache of node A.

◊

Queries The forwarding of query messages, that are messages used by nodes to locate a resource in the network, works like the authentication phase. Once a node receives a query it checks if it possess the content specified in the query message. If so, it sends a query hit message up to the source node using the backward direction in order to decrease the number of connections.

Example 2.3 – The Query Process in Gnutella 0.4

Let us suppose to be in the network represented in Figure 16, and that node s starts a query to find the music file 'MyMusic.mp3' with an associated TTL equal to 7. Let us also suppose that node D, that stores the file, is not directly known by S. S sends the query to all its neighbors, that are A, B, C and E. They will check if some of their resources matches the keywords expressed in the query by S. If they posses such a file, they will send back a query hit message, otherwise they iterate the procedure, decrementing to 6 the TTL for the query message and then sending a query message to node F, G, H, I, J and L. At this point, once the message reaches D, D will send backward the query hit message corresponding to the query sent by S (Figure 17). The query hit will traverse the query path in backward direction reaching s that will now start the downloading phase.

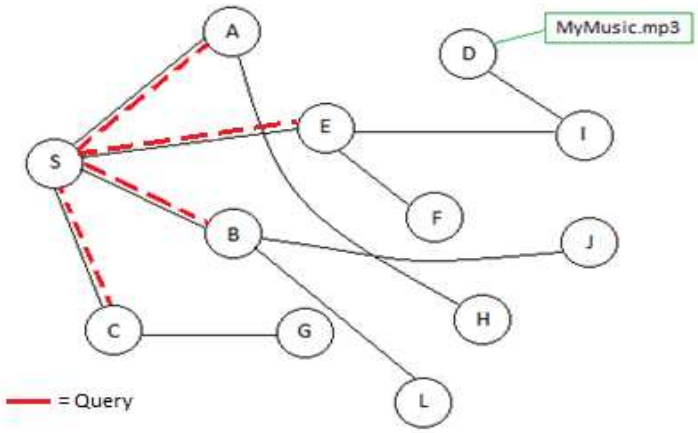

Figure 16 Query search started from S.

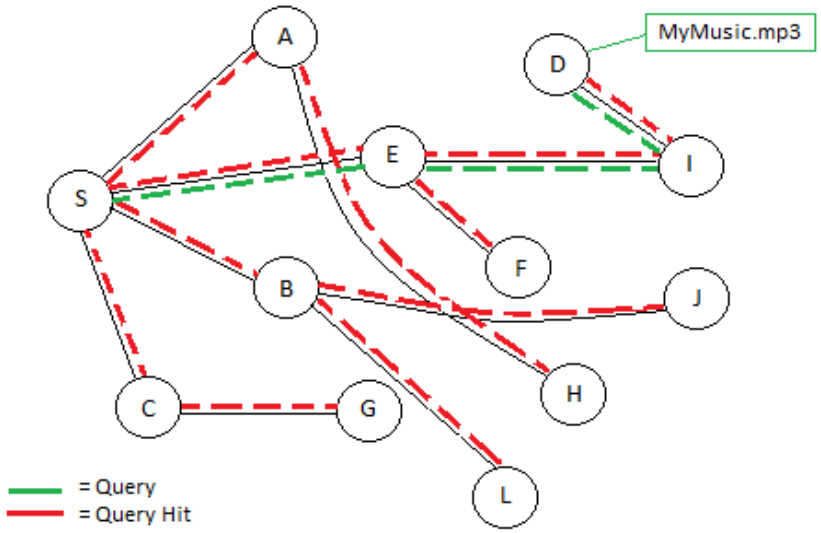

Figure 17 Query Hit response from node D.

◊

Downloading Phase Once the source node of a query message receives back a query hit message, it starts the downloading of the resource. The connection used for this phase is the **Content Transfer Connection**, used for data transfer and based on the HTTP protocol. It is a direct connection between the source node and the destination node and it is temporary. Sometimes the node that sends the query hit can be behind a NAT, thus it has to attach to the message a special flag that specifies that it is behind a NAT. In this case, the source node sends to the destination a special message called **push**. With this message the client on the destination side is able to open a direct connection to the source client and the downloading takes place. This is made because when a node is behind a NAT, its IP address is not the real one, but it is equal to the NAT IP. If the destination node will use the IP address of the NAT all the communications will be directed to the latter instead of the destination node.

Example 2.4 – The Push Mechanism

Let us suppose to be in the situation of Figure 18. Node D is the destination for the query message sent by S. As shown in the figure, node D is behind a NAT and it has to insert in the query hit message the firewalled flag, that is the flag that indicates to S that it is behind a NAT. If D does not send this kind of message, S will open a connection for the downloading phase with the NAT since it will receive the IP address of the latter device. After receiving the query hit, S will send the push message that contains the IP address of S and its port number, to which D will open a connection once received the push message (right side of Figure 18). The downloading phase can then start.

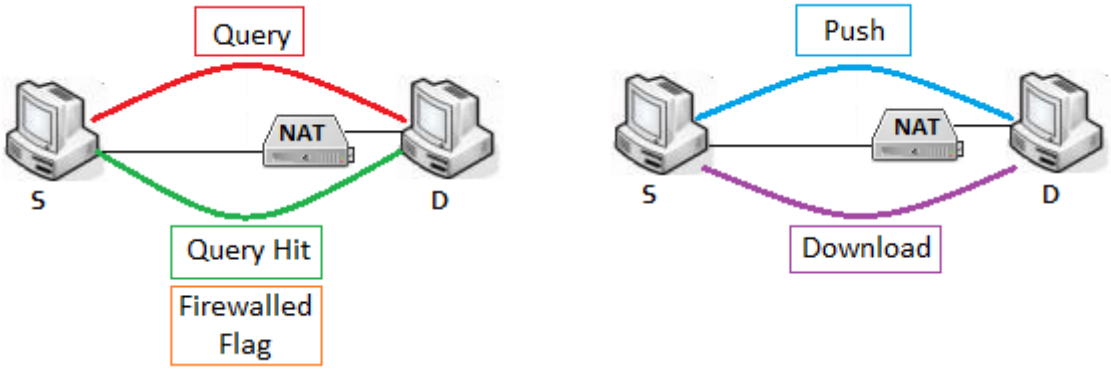

Figure 18 Evolution of a Push Message.

◊

Type of Messages Each message has its own identifier that allows to be quickly identified, to avoid loops in the forwarding process, to avoid duplicates and to help the backward routing. This is the **Gnutella Unique Identifier** (acronym *GUID*) used for messages sent along the network. The main problem concerns the definition of the memory permanence time of GUID messages, given that not all of them can be forever stored in the memory. For this reason each of them is associated with an interval of time. Moreover, each node stores a table containing information on different GUIDs. Once a node receives a request, it checks the GUIDs table to find the GUID of the message and if it is not founded this means that its time has expired and it has been removed from the table. For push messages the technique used is the same. GUIDs and pushes have not to be mixed because at each query hit can correspond multiple GUIDs. As we will see further in this section, push messages contain a field with the node ID that provides a resource, that is similar to GUIDs. If the servent ID and the GUID of a message are exchanged this can lead to errors in the network. Each message exchanged in the network is preceded by an header, that contains specification for the incoming message. An example of header for a message in Gnutella is represented in Table 1.

Bytes	16	1	1	1	4
Description	GUID	Payload Desc.	Time to Live	Hops	Payload Length

Table 1 A generic message structure in Gnutella.

The first field contains an identifier for the type of message, as those seen above; the second field describes the payload descriptor of the message, i.e. query, query hit, pong; the third field is the TTL for the message, while the fourth shows the number of hops of the message; the last field contains the length of the message payload to determine its length by which the system is able to identify the starting point and the ending point of the message, since there is no flag systems to determine the ending points for messages.

Once an header is received, a node expects to receive a message that has the same format as the one specified in the *Function* field.

Types of messages and their structures can be different, depending on their descriptor ID:

- ✓ **Ping:** a servent sends over the network a ping message to gain information about other available servents. Nothing is contained in the payload since the header contains already all the information for the case;
- ✓ **Pong:** is sent from a servent that receives a ping from another one. Its shape is the following:

Bytes	2	4	4	4
Description	Port	IP Address	File Counter	Tot. File Dim.

Table 2 Pong message in Gnutella.

- ✓ **Query:** it is used when a file is sought in the network. Its shape is the following:

Bytes	2	N
Description	Min. Speed	Search Query

Table 3 Query message in Gnutella.

The first field identifies the minimum length to communicate with the source servent, while the second contains the search string of length N bytes ending with a *NULL* character to mark the descriptor end;

- ✓ **Query Hit:** it is a kind of descriptor used only for positive answers to a previously received query messages. If the servent that receives the query message possess the resource specified in the *Search Query* field, and it has a speed high enough to interact with the source servent, specified in the *Speed* field, it creates the query hit message and sends it back to the source. The query hit message shape is the following:

Bytes	1	2	4	1	N	16
Description	Hits	Port	IP Address	Speed	Result Set	Servent ID

Table 4 Query Hit message in Gnutella.

The first field contains the number of possessed files that matches the received query, while the fifth field contains a list of MD5 hash identifiers for files specified in the first field along with their names and dimensions;

- ✓ **Push:** this message is used if the servent is behind a firewall that does not allow incoming connections. Its structure is the following:

Bytes	16	4	4	2
Description	Servent ID	File Index	IP Address	Port

Table 5 Push message in Gnutella.

The index of the file in the second field is obtained by a previously received query hit message.

The **Sachrifc Algorithm** is a special algorithm to control the message flow used by some Gnutella clients **[35]**. The aim of this algorithm is to reduce the number of messages exchanged in the network and the latency for each message. Moreover, messages are delivered in a different order with respect to the previous are explained, that is: push, query hit, query, pong and ping. The first two types are the most important for the network and the first type requires more information exchanging. Furthermore, the algorithm checks that a certain type of message does not dominate the entire network to prevent network congestion. Finally, Sachrifc enhances the power of query hits for rare file instead of common files to speed up their diffusion. The main feature of this algorithm is that it implies a vector of five cells per each connection, i.e. one cell for each kind of message. What changes between messages is that query hits are ordered in increasing level while other messages are ordered by decreasing level. Messages are forwarded using a **Round Robin** mechanisms that deletes messages continuously from the stack and sends them in the network. The set of messages to be sent is fixed by a network parameter. Moreover messages are sent through the network if they are not too old. The organization is **Last In First Out** (acronym *LIFO*), i.e. the last received message is sent first. This decision is to reduce the average time of persistence for messages in the stack.

Evaluation of Gnutella 0.4 Messages that are exchanged in the network are very numerous, so the scalability is not optimal for this kind of networks. To improve this factor a hybrid mechanism like superpeers or further caching mechanisms have to be added. The pong caching is the best way to proceed since a large amount of messages forwarded in the network are pong messages, so they can be simply cached to make each node able to supply them when a proper ping is issued. For each connection a node has a pong vector ordered by increasing timestamp in order to override old messages with new ones. The vector is updated at a certain fixed interval of time. This technique has some disadvantages. First of all, for each ping message several pongs are generated and this means to have a possible congestion at the source node. Second, each servent needs to store received packets in order to be able to respond to further inherent requests. This means that a certain node can be overloaded since nodes are unable to determine the network dimensions. However the main problem is the one concerning message identifiers. Theoretically speaking they are unique but in practice they are not and we can possibly have duplicates and possibly give wrong answers to legitimate messages. Another possible inconvenient is when a certain node that maintains a file leaves the network but after a certain time it joins again, resulting in a changed identifier for the shared file. This can lead to have fake references to files or, for a node, to not be able to retrieve the file again. Another problem is in terms of scalability for the network. Since the number of messages, e.g. queries and pings, grows exponentially we can express their amount in terms of a **geometric series**, as in the following:

$$\sum_{i=1}^{TTL} x^i = \frac{(x^{i+1} - 1)}{(x-1)} = \frac{\left(x^{TTL+1} - 1\right)}{x-1} \tag{2.1}$$

where usually TTL is equal to 7 and x is equal to 4, the latter representing the number of neighbors that are contacted by the forwarding operation. This is because each node forwards the message to x neighbors[2] in the network, and they will do the same to other x neighbors, having an exponential growth, that can be translated into a geometric series since the ratio between the current element and its predecessor in the series is always the same. Answers need not to be equal to the number of contacted nodes, since this number depends on the popularity of the file that is been queried. A solution to solve this problem is to introduce hybrid or caching mechanisms. In version 0.4 of the Gnutella protocol a pong caching solution is used. This mechanism reduces the amount of messages flowing in the overlay network.
To avoid duplicates of ping messages, it is enough to store a table of duplicates where each GUID is stored once it is encountered for the first time in its expiration interval.
For pongs this thing is slightly different because more than the GUID also the IP address and the port number of the communication have to be stored. For simple queries the solution is the same as for ping messages, while for query hit we need to store GUIDs and Server IDs since they can be split in more parts. To have a more robust system we can also store the total payload for the query hit. In flow control methodology the main goal is to limit the network overload, so to level the traffic inside the network and to decrease the latency for each message. If latency problems are verified then messages can be randomly deleted from tables or some connection can be closed, otherwise resulting in a bandwidth waste. The other side of the solution is to have a buffer for incoming messages but this will mean to have a great amount of memory used by servents and the risk of their block. A better policy will be the one that associates a stack to each connection, whose dimensions are greater than the biggest allowed message dimension. A maximum threshold is specified for its content and if it is exceeded, the flow control methodology is started, some messages are discarded, i.e. messages corresponding to congested connections, and the system iterates this proceeding until the traffic level is lower than the threshold.

Another mechanism to decrease the network traffic is the one concerning priority of messages that considers the number of hops used by each type of message inside the network to reach the destination. Queries and ping messages have priority inversely proportional to the number of hops performed to reach the destination, since they are more frequent and spread multiple times along the network, while query hit and pong messages have the priority directly proportional to their hop number, since they are more rare.

Complexity In unstructured networks given that flooding has to be used, the message complexity in a network of size N is given by N * (D – 1) in the worst case, where D is the maximum degree of the nodes. In Gnutella the servents fix a limit to the spread of a message to the TTL value, and each message is sent to at most 4 neighbors, thus it generates a constant number of messages.

2.4 Routing in hybrid networks

In hybrid P2P networks a hierarchical level is exploited, since some high-level entities named **ultrapeers** or **superpeers** are inserted. These special entities operate all the functionalities of the underlying network protocol forming an overlay for the network and providing an entry point for the network itself. The remaining nodes in this organization are called **leaf nodes**. The overlay

[2]*The message is sent to all the neighbors except the one which forwarded the message.*

network is a tree and leaf nodes connect to superpeers to use the network but do not participate in the forwarding operation of some types of messages. Superpeers, on the other hand, handle the forwarding of all the types of messages exchanged in the network (see Figure 5 of Chapter 1 for an example).

2.4.1 The Gnutella 0.6 protocol

This protocol was developed after the version 0.4 in 2001 **[35]** and it is at the base of some other commonly used protocols such as Limewire **[83]** and Bearshare **[53]**. The main goal of this protocol is to reduce the number of exchanged messages (compared to Gnutella 0.4) as well as providing a better performance level for the network. To enhance the performances of the network, those nodes that are not fast enough in terms of bandwidth are excluded from the overlay, even if they can access to it. In poor words, the network needs to be fast and searching operations have to be fast as well to give to the users a better tool to use and a good sharing experience.

Authentication With this protocol two types of nodes are identified: ultrapeers and leaf nodes. The first kind of nodes constitute an entry point in the network for the leaf nodes. Each ultrapeer maintains about 10 connections to the leaf nodes and only a few number of connections with the other ultrapeers. Their bandwidth have to be high as well as their connection speed. On the other hand, leaf nodes do not accept connections from other nodes but only from ultrapeers. Nodes with low bandwidth are connected to other entities with an higher bandwidth capability, such in the case of the Internet, to increase the speed of the network. Leaf nodes autonomously became ultrapeer if they have the requested abilities, e.g. a minimum level of uploading and downloading speed, not being behind a NAT, etc. This decision is made after the analysis of a distributed algorithm, called **UltraPeer Dynamic Tuning**, that checks if the current node possess or not the ability to claim itself an ultrapeer **[35]**. A peer can became an ultrapeer: if it is not behind a NAT; if it respects the upload and download minimal thresholds; if it has enough memory for storing routing tables; if it has an operating system capable to handle an high amount of sockets efficiently, a computing power high enough and a good future uptime, calculated after a proportion over the past uptime. The need of new ultrapeers is determined after the computation of actual online ultrapeers in the overlay and if this is the case, when a new node enters the network, it is informed with a special header for this need.

Once a new node A connects in the network it contacts an ultrapeer U, sending an header containing its information, i.e. its role (that is leaf node), in the field **X-Ultrapeer** that is set to false. On the other hand, U, that is an ultrapeer, has this field set to true. With this field, U maintains also other fields:

- ✓ **X-Ultrapeer-Needed:** indicates if other ultrapeers are needed in the network;
- ✓ **X-Try-Ultrapeers:** contains addresses from other ultrapeers and it is integrated with the field **X-Try**;
- ✓ **X-Query-Routing:** informs the peer connecting which kind of routing protocol it uses for queries;
- ✓ **X-Degree:** number of leaves managed by the current ultrapeer;

- ✓ **X-My-Address:** IP address for the ultrapeer.

while leaf nodes posses all the fields except *X-Ultrapeer-Needed*, *X-Try-Ultrapeers* and *X-Try*. Sometimes leaf nodes can change their state into ultrapeer and vice versa. An ultrapeer can determine if a new node, that possess the capabilities to became ultrapeer, can truly be declared an ultrapeer. A new node A sends a message to one of the ultrapeers (U) with the field X-Ultrapeer set to true specifying that it candidates to the role of ultrapeer. Once received, U checks if new ultrapeers are needed in the network, by querying other ultrapeers, setting the field X-Ultrapeer-Needed in the header of the answer to A with the correct value, i.e. true or false. If the two field, X-Ultrapeer on A side and X-Ultrapeer-Needed on U side, are equally true A became an ultrapeer, otherwise not. Let us now show an example on how an ultrapeer can became a peer.

Example 2.5 – Node redirecting

Let us suppose to have a node y that at a certain time is set as ultrapeer for the network. After some time, y has to change its state into leaf node, since there are too many ultrapeers in the network. Now, let us suppose that node z needs to contact node y to have information about a resource, but z does not know about the change of role of y. After y receives the connection message by z, it answers with a denied connection, attaching into the header the X-Try-Ultrapeers field containing addresses of other ultrapeers in the network. This event can occur when the system does not send routing updates in time after some roles changing in the network.

◊

Each ultrapeer possesses a set of k slots (where k is a system value), available for inserting addresses of other ultrapeers and n slots (another system parameter), for leaf nodes addresses. Once an ultrapeer receives a request by a leaf node it checks if it has some free slots to store its address, otherwise the ultrapeer refuse the connection; if the connection comes from an ultrapeer and there are some slots for leaf nodes still available, the ultrapeer is degraded to leaf node, otherwise it is inserted in the ultrapeer list. Sometimes an ultrapeer can refuse to serve incoming connections, i.e. because it has no more free slots, giving to the connecting nodes the addresses of other ultrapeers specifying them in the X-Try-Ultrapeers field of the answer.

Queries Hybrid networks need also to provide and indexing system for resources maintained in the network. In Gnutella 0.6 this system is called **Query Routing Protocol** (acronym *QRP*) in which each resource is identified by a set of keywords. Each ultrapeer maintains a QRP Table that is a vector of 2^{16} bits containing information about shared resources. These tables are built with routing vectors coming from each node managed by the ultrapeer. Information inside routing tables are constituted by an hash of the file name and those keywords identifying the resource. The hash function gives as output a value between 0 and 2^{16} and the corresponding position in the QRP Table is set to 1. Let us show an example of this.

Example 2.6 – Resource hashing in Gnutella 0.6

Let us suppose that node y inserts in the network a file named picture.jpg, that is described by a set of keywords specified by the user, i.e. photo, flowers and mountains. Once the file is inserted the node creates an hash of the name of the file, without its extension, as well as an hash for all the keywords specified for the file:

$$hash('picture') = 4\,, hash('photo') = 6, hash('flowers') = 10, hash('mountains') = 8$$

The hash function applied to each content will return the index of a specific position inside the QRP table. The corresponding position inside this table will be set to one indicating that the current resource is described by those keywords. The QRP table of y will then be updated as the following:

Value	*0*	*0*	*0*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*…*	*0*
Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*…*	*65535*

This table will be useful for the ultrapeer to which y is connected during the search phase.

◊

Once a node needs to query a resource, it specifies to its ultrapeer a set of keyword describing the queried resource. The ultrapeer hashes these keywords one by one and checks in the QRP Table of all the nodes connected to it if some of them satisfies the set of hashes, i.e. if the positions given by the hash function applied to the keywords of the search query matches those in the QRP Table of one of the leafs. In a system like this, the ultrapeer is not always sure that a complete match is indeed a match for the just received query since each leaf shares more files, thus the match can be given for more than one file. With Gnutella 0.6 a query can be answered with fake positives but never with fake negatives. Once a leaf node needs to send the table to its ultrapeer, it compresses it and subdivides the table into blocks. The ultrapeer stores the routing table using it has its own **Query Routing Table** (acronym *QRT*). Once a node x needs to find a resource, it sends its ultrapeer a query message with a set of keywords, the ultrapeer check via its QRT if a node managed by itself matches hashes contained in the query. At the same time, other connected ultrapeers are contacted by the starting ultrapeer with the search to find as many nodes that handle the searched resource as possible. Answers are handled as in Gnutella 0.4, with the backward routing from the destination to the source of the query via a query hit sent back by the destination. The file exchange is done with a direct connection among source and destination. To save memory and to enhance the search mechanism, an ultrapeer can combine its QRT to those maintained by its leaf nodes with the bit-wise or operation on the routing tables. Then, routing tables can be forwarded with a certain TTL associated and their content are updated with the hop number from the host that maintains a specified resource. In this case each node maintains another table that indicates the distance between its position and the host that maintains a specified resource, If this value is less than the TTL, otherwise is set to infinity. Each request is computed for its TTL value. Once computed a node checks in its routing table if some nodes providing the searched resource

are in the range of the specified TTL. If not, the corresponding position for the hash of the keyword in the contacted node routing table will be set to infinity.

Example 2.7 – An example of distance routing table

Let us suppose that a node x has a routing vector which at position $g_1 = 8$ has a 1 and that this one refers to a resource maintained by another node y connected to x. The distance routing table contains the minimal distance from a node to a resource, so at position g_1 of the distance routing table of x there will be the number of hops from x to y, if only node y posses a resource that is identified by a keyword corresponding to the value 8 of the hash function. If both x and y have this resource, the distance routing table of x will have a zero in the corresponding position. If only y has the specified resource, than at the position 8 of the distance routing table of x there will be the number of hops from x to y. If more nodes connected to x have the specified resource, the number of hops stored in x distance routing table will be the minimum among all the distances to these nodes. So, for example, if x, y, a and b have the resource whose hash function of the associated keyword 'bomberman' gives 9, the 9^{th} position of the distance routing table of x will be:

$$DRT_x = \min\{x, y, a, b\} = 0$$

because x possesses the specified resource.

◊

For the other messages Gnutella 0.6 uses the same mechanism used by Gnutella 0.4.

Protocol Evaluation The main advantage of this solution is the scalability, since the network can adapt very fast and without sensible problems to the join and leave of nodes. Another important advantage is the decreased traffic of messages in the network, as only ultrapeers exchange control messages inside the network and leaf nodes are never contacted by ping messages or messages to check the state of nodes. Finally, once a ultrapeer receives a ping from a leaf node, it sends back a pong message containing information on other ultrapeers in the network, thus the leaf node is able to remain connected to the network even in case of departures or crashes of its direct ultrapeer.

Complexity Compared to Gnutella 0.4, the query mechanism requires the exchange of a lower number of messages to retrieve a resource. Also the network update requires less messages going through the network, since the ping/pong mechanism is dealt by the ultrapeers as for the query system. In a network with U < N ultrapeers, the ping/pong mechanism requires 2 * U * (U – 1), or 20 * (D – 1) if D < U, update messages, that is lower than the N * (D – 1) of Gnutella 0.4. If a node detects the failure of its ultrapeer it requires on average L/2 connection requests to other ultrapeers, if L is the length of the list of other ultrapeers in the network.

2.5 Routing in structured networks

The main difference between structured and unstructured networks is the way information is stored and how it is retrieved. In structured networks a storing mechanism, completely

distributed, is provided to locate and retrieve resources. This enhances the scalability of the system and makes it fault-tolerant. What differs from unstructured solutions is that the required memory to store information about the resources is less, while the number of messages exchanged for the search operation grows. We can now model a structured P2P network as a directed graph of n nodes (Figure 19) **[7]**.

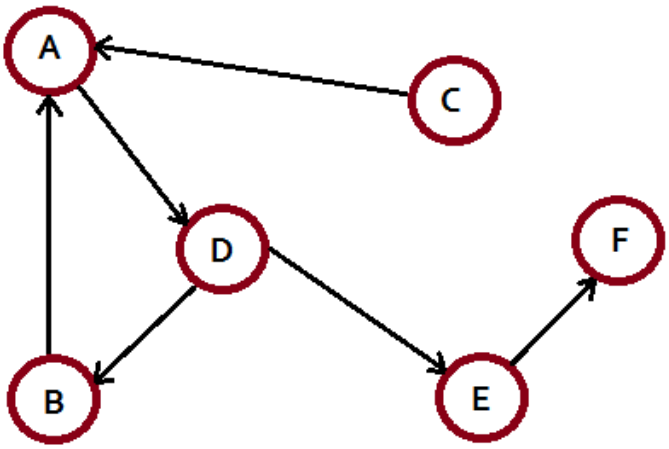

Figure 19 Example of Directed Graph.

In structured P2P networks the routing mechanisms is based on Dynamic Hash Tables, explained in the next section.

2.5.1 Dynamic Hash Tables usage

This solution provides an equilibrium point between a centralized solution and a complete decentralized solution, i.e. massive query flooding. DHT allows the system to eliminate the probability of false positives and to improve scalability, handling autonomously the join and leave of nodes **[18, 35]**. The main goals for DHTs are scalability, flexibility and reliability. The latter concerns networks characterized by high churn, where resources change frequently their location and the search process needs to be efficient. The main disadvantage in implying this solution is that complex queries cannot be used.

Basically each DHT-based protocol assigns a unique identifier to each node in the network as well as resources maintained inside the network. A common logical space for peers and resource IDs is made available in the network and each node handles a specific portion of this logical space, i.e. the resources contained inside it. Since each node can join and leave at any moment, mapping between nodes and resources can change at any time. Once a node (A) needs to find a certain resource (R) maintained by another node (B), it needs to send a message to B. For this purpose each node maintains a routing table based on correspondences between keys and values of resources maintained in the network and nodes that handle them. To define the logical space of nodes and resources, a system needs to hash the physical address of each node obtaining an identifier of a certain length that identifies uniquely the node, or the resource, inside the logical space. In poor words, the hash function provides a bridge between the Physical and the Overlay network, as in Figure 20. In this logical space an ordering technique is used, and operation are made modulo the number of bits constituting the identifiers.

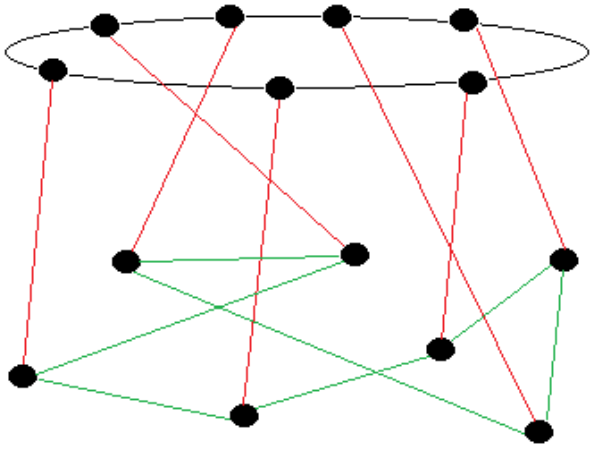

Figure 20 Overlay network vs. Physical Network.

After this preliminary operation, each node receives a portion of the logical space to handle, in which a certain number of resources are stored. Usually identifiers for resources are created after an hash function of the name or the content (or both) of the specified resource. Sometimes **redundancy**, that is the spreading of multiple copies of the same resource shared in the network, is included to improve the reliability of the system. The main attention to pay is to balance the load of resources handled at each node. In fact, if a node has to handle more resources than other nodes it will be probably flooded with a great amount of requests being possibly unable to solve all of them. Once a node A needs to find a resource R maintained by B, without knowing the address of B, it needs to forward a query in which it specifies the ID of R. The mechanism with which the request is forwarded, i.e. which will be the next neighbor contacted, depends on the routing protocol adopted by the network. Sometimes the routing mechanism chooses the next node basing the choice on the distance between the node ID and the destination ID in some terms of distance, i.e. longest prefix matching, Euclidean distance[3], etc.
DHT stores information on resources in two ways: direct or indirect storage.

Definition 28: *The storing mechanism of DHT is **direct** if the ID of the resource falls in the portion of the logical space handled by the node with the nearest ID. If, instead, the node stores the physical address of the location of the resource, the storage method is said to be **indirect**.*

Example 2.8 – Direct and Indirect storing mechanism

Let us suppose that node 12cd4e20 sends a query to node 30ab5d31 specifying the resource ID 30ab5d25, and that node 30ab5d31 manages the logical space with prefix 30ab5d. Since the ID of the resource falls in the area managed by node 30ab5d31, this latter will respond directly to node 12cd4e20 with some kind of query hit message and the downloading phase begins. Let us now suppose that the ID for the queried resource is 25cd3f10, not managed by node 30ab5d31, but that node 30ab5d31 possess the reference for the queried resource. Once computed the query, node 30ab5d31 will give back the physical address at which node 12cd4e20 will find resource 25cd3f10

[3] *The longest prefix matching gives the longest portion of the prefix that matches between two IDs while the Euclidean distance is the absolute value of the distance between two points in the space.*

and the downloading phase will begin after the source node has contacted this physical address (see Figure 21).

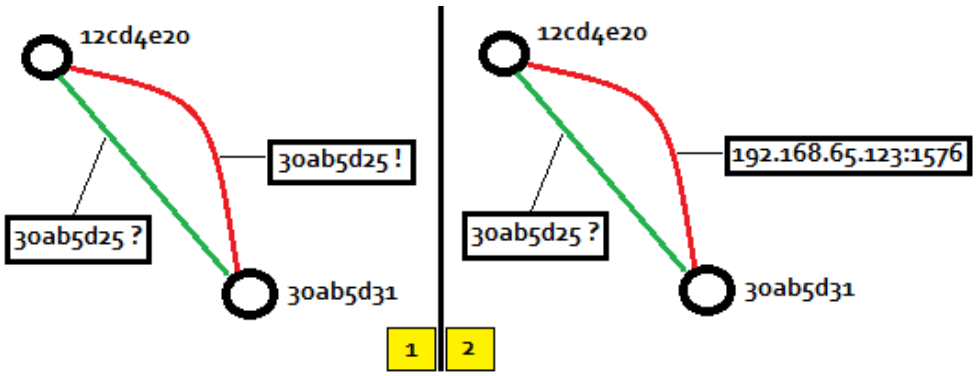

Figure 21 Example of direct (on the left) or indirect (on the right) storing mechanism.

◊

Once a peer receives the physical address of the node that stores a resource it downloads it using IP level connections (TCP or UDP).

When a node enters the network, it contacts a bootstrap node from which it receives an identifier calculated via the hash function specified by the protocol, and a portion of the logical space, and its presence is communicated to its neighbors, via a pair composed by a key (the outcome of the hash function) and the value (the content of the resource), to update their routing tables. If a node voluntarily leaves the network its logical space and resources are partitioned among neighbors, while the node is deleted from all the routing tables. If the node abruptly quits the network its resources are loss if no redundancy mechanisms are used in the network. In some protocols the alternative routing mechanism, i.e. usage of multiple paths, is used to avoid faults of nodes.

2.5.2 The Napster protocol

Napster, developed in 1999, is the first example of centralized P2P network. The system contains a centralized server that maintains references to files exchanged in the network. Once a node wants to download a file, it needs to contact the central server, called **broker**, that provides information about the node containing the resource. The overlay network topology is a star. The broker is able to recognize if a certain source node is behind a NAT or a firewall and, in this case, a **Push** procedure is used. Using a NAT or firewall protocols is useful to save IP addresses but would disable capabilities for peers to exchange information. A node in Napster is composed of a **coordinator** (that coordinates connections and communications with the lookup servers or brokers), a **connection handler** (that deals with incoming requests) and a **download/upload/push manager** (that handles direct information exchange among nodes or the case in which a node is behind a NAT). Generally, a message in the Napster network is expressed as in Figure 22. What changes between each kind of message is the **<Function>** field and the content of the **Payload** section, that varies in contents and dimensions with the message type.

Payload Lenght	Function	Payload
4 Byte	4 Byte	Parameters of the message

Figure 22 Header message in Napster.

Authentication The first message exchanged in the Napster network between the user and the server is the **New User Login** message, shown below:

User Nick	Password	Port Number	Link-Type	User e-mail

Table 6 Registration message in Napster.

This is used by a new user to register itself in the network. The server checks if the nickname is already used and answers with an ad-hoc message telling the user if the nickname is valid or not. Clear fields are **User Nick**, **Password** and **Port Number**, while **Client-Info** specify the version of the client used by the user and the **Link-Type** field is an integer representing the bandwidth of the user logging in. After this first step, the user can login to the server and share files. For the login process, the user sends a message like the following table.

User Nick	Password	Port Number	Client-Info	Link-Type

Table 7 Login message in Napster.

The server answers with an ack. Once the user is authenticated, it can share a file. Napster was originally developed to share only MP3 audio files. Once a user decided to share a file, a message as in Table 8 was sent to the server:

Filename	MD5 of File	Size of File	Bit-rate	Frequency	Time

Table 8 Notice of file sharing in Napster.

The field **MD5 of File** represents the MD5 checksum of a portion of the file to be shared, usually the first 300 Kbytes of the file, an hash function that has a low probability of incurring in collisions **[47, 107]**. This field has a length of at most 128 bits and it is used to uniquely identify a file inside the network. This field was the main reason behind the ruin of Napster, since the prosecution states that through MD5 fingerprints Napster administrator could have been able to block illegal diffusion of copyrighted files. The remaining fields regard the structure of the file such as its dimensions on the hard disk, the bit-rate, that is the number of bits used to codify one second of the audio file, the sampling frequency of the file, that is the number of samples per second, and its length in seconds.

Example 2.9 – A Client/Server interaction in Napster

*Let us suppose to have user **Johnny74** that wants, in order, to create an account, login and share a file in the Napster network. After downloading the Napster client from the Napster website, the user needs to create an account. It then sends the message in the format of Table 6 filled as follows:*

Johnny74	*passJohnny74*	*6754*	*56*	*johnny.74@gmail.com*

Table 9 Registration message for Johnny74.

The Napster server receives the message and checks if the user is already registered. Let us suppose the specified user is a new one, and that the Napster server accepts the registration, thus Johnny74 can login to the Napster server and send the message in the format specified by Table 7.

Johnny74	*passJohnny74*	*6754*	*"nap v1.5"*	*56*

Table 10 Login by user Johnny74.

The server now checks if the information corresponds to user Johnny74 and, in this case, it sends back a positive ack to the user, thus Johnny74 is logged in.

Now, the user wants to share a file named MyMusic.mp3. As we saw in Table 8, the user needs to specify a set of information on the file he wants to share. The right message the user needs to send is in the following form:

"MyMusic.mp3"	*03e8f89f0bc98e20315659f019a9483d*	*6754*	*196*	*44100*	*245*

Table 11 File sharing by Johnny74.

At this point the file MyMusic.mp3 is uploaded in the network and the users connected to Napster can download it from Johnny74 after receiving its contact by the server.

◊

Queries More than sharing, a user can search a file. For this purpose another kind of message is provided for clients:

Search Criteria	Max Results	Line-speed	Bit-rate	Frequency

Table 12 Query message in Napster.

In the first field the user specifies the search criteria for the searched resource, such as its name or dimensions. With the second field the user specifies the maximum number of results to be given back from the server. The third field specifies the speed of the connection of the resource keepers and the fourth field specifies the bit rate of the searched resource. The last field specifies the sampling frequency for the file. After this message, in case of hit, the server usually answers with the response message shown in Table 13.

MD5 of File	Size of File	Bit-rate	Frequency	Time	Nickname	IP Address	Link-Type

Table 13 Response message in Napster.

The message contains the complete name of the file with its characteristics (field 1 to 5) along with features of the host that maintains the file, such as the nickname of the user, the IP address and the bandwidth maintained by the host. After this response, the download phase starts through the HTTP protocol.

Example 2.10 – Search example in Napster

Let us now suppose that user Clive86 wants to download the file MyMusic.mp3 uploaded by Johnny74, and that Clive86 is logged in on Napster. So, Clive86 needs to specify the search criteria as expressed in Table 12 with the following message:

FILENAME CONTAINS *"MyMusic"*	**MAX RESULTS** *70*	**LINESPEED** *"AT LEAST" 20*	**BITRATE** *"AT LEAST" 128*	**FREQ** *"EQUAL TO" 44100*

Table 14 Example of search by Clive86.

*In the above example, the user specifies the file name MyMusic, how may results have to be shown (at most 70) the **LINESPEED** (at least 20), the **BITRATE** (at least 128) and the sampling frequency (equal to 44100). If some files satisfying the search are founded by the server, as in this case, it will answer with the following message:*

03e8f89f0bc98e20315659f019a9483d	*6754*	*196*	*44100*	*245*	*Johnny74*	*10.0.0.1*	*56*

Table 15 Answer from the server at the request from Clive86.

supposing Johnny74 has an IP address equal to 10.0.0.1.

◊

Protocol Evaluation The main problems with the Napster architecture is its scalability, since it relies on a unique server. A solution is to increase the number of servers in order to build a server farm, located at a precise geographic location. This solution resembles very well the case of hybrid P2P systems, even if in the latter ultrapeers are geographically distributed and they provides coordination functionalities and decentralization of network resources. Each server shares with others the database in which the index of each file is maintained. Moreover, to increase the balancing in terms of traffic directed to each server, some nodes called **lookup servers**, that are servers to search resources in the network, are included in the network. Briefly, once a peer wants to download a file, it contacts a lookup server and the latter forwards the connection to the broker that is less loaded with incoming connections. The set of servers is permanent and well defined, i.e. servers cannot be changed or integrated with new servers among peers in the network such in

the case of Gnutella 0.6. Despite these weaknesses, all modern approaches, such as **BitTorrent** and **eMule**, use basics from Napster to develop their networks.

Complexity The number of messages exchanged during routing operations in Napster is very low. For search operations or file sharing notifications each user needs only to send one message to the server, which answers with another message. On average, in fact, the number of messages needed for a complete search (query → query hit → download) is equal to 3*N (where N is the maximum size of the network), since the user sends a message to the server (at most at distance N), that answer with a query hit reporting the contact of another user sharing the queried resource and the first node contacts it to start the downloading phase. In normal cases, i.e. where there are no faults on the server side, contents are ensured to be found with the correct query.

2.5.3 The Chord protocol

The first protocol we chose to outline is Chord that is a protocol created at Massachussetts Institute of Technology (*MIT*) in 2001 by Ion Stoica, Robert Morris, David Karger and Frans Kaashoek **[8, 41]**. The main properties of this protocol are:

- ✓ **Workload Balancing:** Key identifiers are mapped with a consistent hash function in an equal way between nodes, naturally balancing the load.
- ✓ **Decentralization:** All nodes are equally important with a completely distributed protocol.
- ✓ **Scalability:** Adding more nodes increases the time to visit the network but only in the order of the logarithm of the total number of nodes, thus also having an high number of nodes the network is still feasible.
- ✓ **Flexible naming:** Key structures are not constrained and nodes can attach their name to some key to identify themselves to the network in a very flexible way.

This protocol precisely maps a key to a specific node. Identifiers for both nodes and keys are assigned through an hash function, and when a new node enters the network, the keys are redistributed among nodes to balance the load of the entire network. The applications that use Chord are informed whenever a key is moved from a node to another, and they are responsible for data authentication, making use of the protocol itself that is very flexible in this case. Each node in the network has a structure in which information about other nodes in the network are maintained. Information for the nodes in the neighborhood are made by the IP address, the port number and the network ID for each neighbor.

Chord is an example of DHT-based structured network in which information storage is performed using hash tables. Each node can share resources, or retrieve them, via use of the functions **put(key, value)**, to share a resource, and **get(key)**, to retrieve the resource. Node IDs are created via the SHA-1[4] hash function on their IP addresses and port numbers, and are composed of m bits (usually m is equal to 160) in a way such that with very high probability there are no collisions and IDs cannot be forged. Resources are mapped in the same logical space of node IDs with the same hash function over their content. The overlay network is organized in a clockwise

[4] *SHA-1 is a cryptographic hash function created by the National Security Agency (NSA) in 1993 and part of the FIPS pub 180 specification **[57]***

oriented ring composed by all the active nodes in the network with their own ID. Keys are assigned with consistent hashing. Formally:

Definition 29: *With **consistent hashing** we refer to the way resource are mapped inside a network. Resource are mapped uniformly in the logical space to balance node workload and to lighten update operations in case of join and leave operations of nodes.*

Resource Mapping In Chord resources are mapped to the node whose ID is the closest to the resource key that has an ID greater or equal to the key ID. An example of resource mapping in Chord is shown in Example 2.9, while in Figure 23 we can have a look to the inner topology of the Chord overlay.

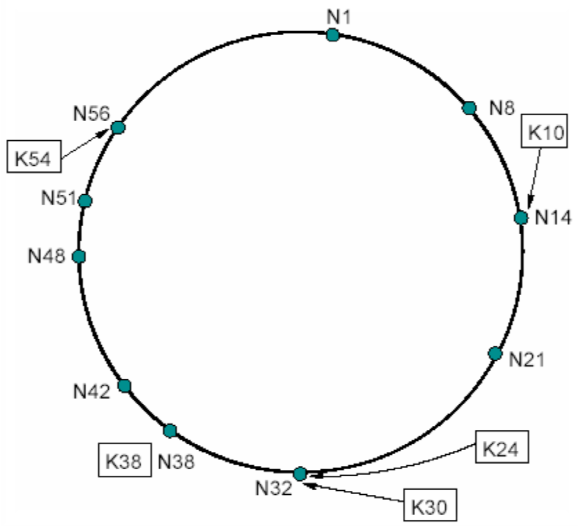

Figure 23 Chord Overlay Network **[41].**

Example 2.11 – Resource Mapping in Chord

Consider the ring of Figure 23. If a resource with key 54 is inserted in the network, this resource will be assigned to the first node encountered in clockwise direction in the ring, whose ID is greater or equal to 54. In this case, key 54 will be assigned to node N56.

◊

Routing The overlay network, i.e. the ring, is traversed in clockwise direction. A first simple example of routing in Chord is the one in which each node stores the reference to its successor node in the ring. When a key is queried each node has to pass the query to the successor node in clockwise direction and the search proceeds until the destination node is reached or the number of hops associated to the search message finishes. Clearly the execution time for a single search, in the worst case, is equal to the time needed to visit all the nodes. For this reason, each node has a structure called **finger table** that allows the search to be more efficient. Each table contains records to nodes from the current i^{th} node to the $(2^{i+1})^{th}$ entry in the ring, even if positions are not occupied by a node. Each entry of the table is composed by the triplet **(ID_{node}, IP, port number)** of the j^{th} node in the ring, with the first entry pointing at the current successor node. These

information are always refreshed when nodes receive search requests or updates. The search time is then lowered to the logarithm of the number N of nodes that can be inserted in the network ($\log N$). An example of finger tables is shown in Figure 24.

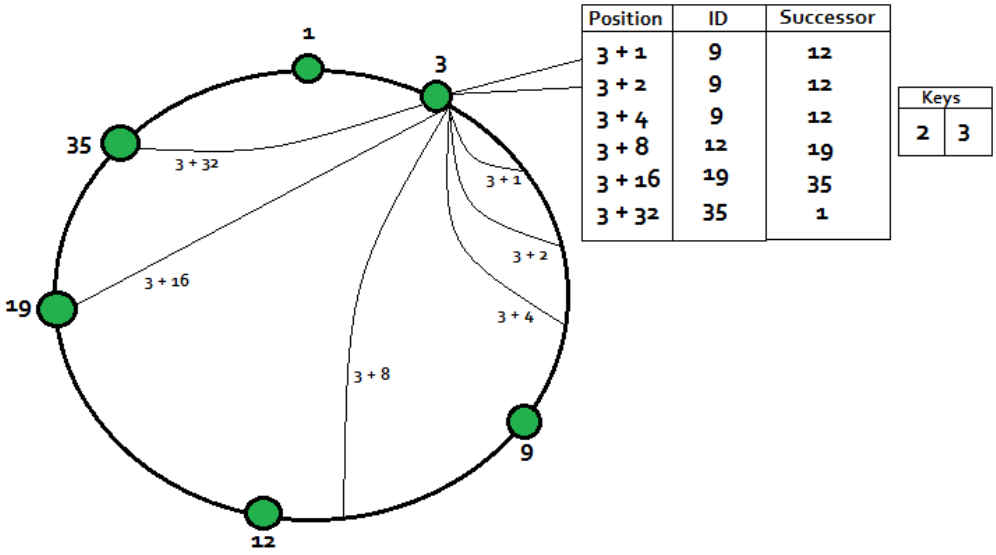

Figure 24 Chord finger tables.

The finger table maintained by each node contains also the reference to the successor node in the ring (the first entry of the table), the set of keys mapped in its portion of the logical space and a set of pins that points to the successors of the current entry (the *Successor* column in Figure 24) in the finger table for network consistency in case of join and leaves. To enforce the reliability of the system, the node posses also a pointer to the predecessor node in the ring in a different structure. The information about the predecessor of the current node is important in terms of finger tables update and in the joining phase, as we will see further. Example 2.11 shows how the search mechanism in Chord works.

Example 2.11 – Searching keys in Chord

Consider the ring in Figure 23, and assume that node N8 wants to locate K38 assigned to node N38. N8 consults its routing table and sends the query to the farthest node in the table whose ID is lower than 38, since the key for the searched resource is equal to 38. Let us suppose N8 has the reference of node N21 as the farthest. At this point, N21 forwards the query to node N38 who is the farthest alive node in the ring whose ID is the nearest to the specified key and has to posses K38. The reference for the node is passed in counter-clockwise direction to the source node, and the download is then handled with through the physical address of the destination.

◊

Authentication The attention is now moved to the join and leave of nodes from the network. In the first case, when a node joins the network it knows at least one node, the **bootstrap node**, that gives him information about the network and its new identifier. Then, the bootstrap node contacts

the network to find the predecessor of the new node, in order to place it in the ring. Once it has found it, the bootstrap node links the new node to its predecessor. At this point the network update follows two phases:

1. The predecessor gives to the new node the ID of its successor, that becomes the successor of the new node. The latter contacts the successor to authenticate itself, and the successor contacts the predecessor to have confirm of the new node. If all goes well, the successor registers the new node as its new predecessor, and the new node registers the new successors as its own successor. At the end, the successor sends a **stabilize** command to the predecessor to inform that it has to update its successor to the new node.
2. The finger tables have to be updated, thus the predecessor starts the update in backward direction in the ring to make aware the nodes until the $(2^{i-1})^{th}$ position (where i is the new node), that a new entry is in the network. The new node build its routing tables by querying the predecessor with the position from $Node_{ID}$ + 1 to log N (the maximum number of entries in the table).

During the first phase or after the second, the new node contacts its predecessor to take charge of the keys maintained by it. Each node also maintains information about its predecessor, that is important for the updating mechanisms. The leave mechanism works as the join mechanism, in opposite direction. A special case here is when a certain node fails abruptly. In this case each of its predecessors can update their table to delete it. Is also important that some notions about the successors of the failed node are maintained by its predecessors. For this reason, each node maintains a set of r successors in the ring, to be able to contact them if their direct successor fails abruptly. The parameter r is chosen high enough to be able to find some alive nodes in the ring.
Another big problem is the one regarding keys maintained by the departed node: if they have not be duplicated, they are lost. This can be avoided replicating keys in nearest nodes, in such a way to allow their retrieval even after the node departure. When a node searches a resource that was mapped into a failed node, the query is passed to one of its successors, using the information in the list of r elements. If replication is used, one of these successors will have the reference for the searched resource and will give it back to the source node. Let us see an example of node departures and how these events are handled.

Example 2.12 – Node failures in Chord

Let us suppose that we have the ring of Figure 25, and that node 7 has to forward a request for key 26 to node 11. Once forwarded the request, node 7 waits a certain timeout in which it is supposed to receive an answer from node 11. If this timeout expires, node 7 forwards the request to node 12, the first alive successor of node 11, changing the entry corresponding to node 11 with node 12 in its finger table, since node 11 did not answer. The process is repeated until node 7 finds the key or it does not have other nodes to contact, i.e. if the ID of the next node to contact is greater than the queried resource ID. In the latter case the network returns an error, since the resource cannot be located (this is a very improbable results for the search phase since the list of r successors is big enough to find a replica of the initial resource).

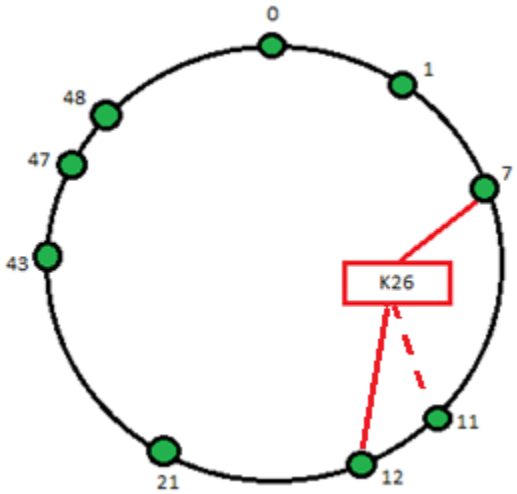

Figure 25 Example of node failure in Chord.

◊

The main goal of the network, in terms of robustness, is to maintain the ring intact and with no fragmentations. The main problem is the correctness of finger tables as we are sure that, after a join, only the predecessor and successor fields are correct. So, we can have a case in which they are not correct and the search of keys takes more time than the natural case or that there are some loops. The update mechanism, however, along with the join mechanism allows to obtain a successful search even in case of multiple joins. The case of multiple departures is controlled with the r parameter.

Protocol Evaluation The Chord protocol is robust if a certain number of nodes remains alive, but under a certain threshold (the 50%) the stability of the network is compromised. The main problems arise when a great quantity of data is maintained inside the network since each piece of data will have to be hashed and this will lead to a massive usage of computational resources.

Complexity Having N nodes inside the network, a routing operation is said to terminate on average in O(log N) steps, since each message gets closer in logarithmic way to its destination at each step. The finger tables maintained by nodes inside the network have log N entries each of 2^m bits, where usually m = 160.

2.5.4 The Pastry protocol

Pastry is a routing protocol for P2P networks created by Anthony Rowstron and Peter Druschel in 2001 **[37]**. It is based on DHT and has many similarities with Chord. The pair **(key, value)** proper of the DHT-based protocols are stored in a network of Internet hosts that have multiple connections. The network is robust because it is redundant and not centralized, so there are no single point of failures and, thus the risk of data loss is very low.

Nodes In Pastry nodes posses identifiers with 128-bit of length distributed in a uniform set of IDs created by an hash function applied to the public key of the node or to its IP address. The overlay is organized in a ring structure such as in Chord. Given a certain ID and a message, each node sends the message to the node nearest to the destination with an average delivery time equal to $\log_{2^b} N$, where N is the maximum size of the network and b is a fixed network parameter usually equal to 4. Finally, the power 2^b indicates the base of the digits of nodes and resources ID, i.e. 16. The system works until a certain number of failures for adjacent nodes is verified, typically l/2. Usually the number l is fixed to 16 by the system, and identifies the length of the **leaf set** of a node, explained further.

Routing Tables Each node posses a routing table that has size of:

$$\left(2^b - 1\right) * \log_{2^b} N + 1 \quad \text{bits} \tag{2.1}$$

having $\log_{2^b} N$ rows each composed by $(2^b - 1)$ bits, where the generic entry i at column j posses the first i bits equal to the node to which the table belongs. Tables are also updated in the logarithmic order in case of a join and a leave of a node and map node IDs to node IPs. Adjacent nodes are determined with proximity techniques such as the Round Trip Time (acronym *RTT*). Formally:

Definition 30 [66]: "***Round-trip time*** *(RTT), also called round-trip delay, is the time required for a signal pulse or packet to travel from a specific source to a specific destination and back again.*"

An entry in the routing table is left empty if no nodes with the corresponding portion of the ID is known in the network.
More than the routing table, each node posses also a set of node IPs nearest to itself computed over the l/2 IDs nearest to its own, upward and backward so to cover l nodes. This set is the previously mentioned leaf set.

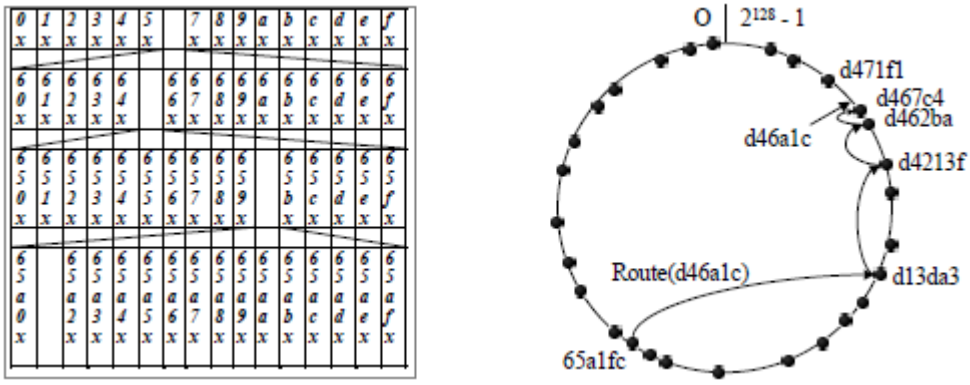

Figure 26 Example of routing table (left) and routing mechanism (right) in Pastry **[37].**

Routing To route messages, the source node needs to provide the key for a destination in the network. Then, the message is delivered to the node that has at least the first digit of its ID in common with the specified key. If no nodes with this peculiarity are found, then the message is forwarded to a node that shares the same number of digits as the source but that is numerically closer to the destination than the source. If neither in this case a node is found, so the current

node is the destination of the message. Let us see an example of routing mechanism in Pastry with Example 2.13.

Example 2.13 – Example of routing in Pastry

*Let us suppose to be in the overlay network such that in the right part of Figure 26. Let us take node **65a1fc** that needs to reach key **d46a1c**. To forward the message the current node needs to take from its routing table the node that shares at least l + 1 digits with the destination key, starting from left to right, with l representing the current level of the search. The node that satisfies this criterion in the table of 65a1fc is **d13da3**. Now, the latter node iterates the proceeding, looking in its routing table for a node that has an ID of the form d4*****. The choice falls to node **d4213f**. The process goes on until the request reaches node **d467c4** that stores the key d46a1c.*

◊

Authentication and Network updates The system is dynamic since neighbor sets and routing tables are updated in the presence of joins and leaves of nodes. In node joins, the bootstrap methodology is used also in Pastry. A new node (X) that wants to connect inside the network needs to know another node (A) already inside, i.e. through a bootstrap mechanism. Once contacted, A starts a search for node X inside the network reaching node (Z) whose ID is the nearest to the one of X. X receives back a table containing the nodes traversed in the path from A to Z and all nodes interested in the entry of X are contacted to add X triplet inside their routing tables. During the join phase also the routing table is built. With a similar procedure nodes are updated even in case of failures by other nodes exchanging a small number of messages, i.e. $\log_{2^b} N$ in the worst case, a property of Pastry called **short routes**. The main method is the comparison of the **Round Trip Time** (similar to the TTL) and calculates the shortest path comparing the time of two different paths used by two different messages that have to reach a common destination. The time sought is inside the time needed for the two paths to converge, another property of Pastry called **route convergence property**. When a node fails, other nodes can verify this event with the lack of keepalive messages from the failed node. Formally:

Definition 31: *A **keepalive** message is a message used by some protocols during the update phase of the system. These messages communicate to nodes receiving them that their neighbors are still alive. The lack of these messages means that neighbors are not connected anymore.*

This means that periodically each node sends to its neighbors a sort of message telling them that it is still in the network. If a node does not send these kind of messages for a certain period it is considered failed. When a certain time (T) passes, where T is usually equal to one hour, after the last keepalive message from node (X) was received, other nodes contained in the leaf set of X are notified in order to update their information. This kind of update is based on a propagation: nodes in the initial leaf set contact their own set, and so on.

Protocol Evaluation This protocol is surely quick in terms of routing mechanisms and message delivery. The ID assignment process is quite similar to the other protocols since the ID for a node is calculated over an information the node possess, such as its IP address. This can be a problem since each node can arbitrarily choose its own ID simply changing the IP address, as we will see in Chapter 3. However, Pastry is an high scalable system that delivers messages in a very short time and combine pros and cons of an architecture like Chord and an architecture like Tapestry, presented next.

Complexity Assuming that each routing table and leaf set is consistent and that there were no recent failures in the network, the routing mechanism is said to converge in at most $\log_{2^b} N$ hops. Moreover, considering the case in which the routing table is queried to select the next hop to which to forward the messages to, at each time the number of possible candidates for the next hop is reduced by a factor 2^b, thus the threshold in the number of hops for the message is enforced. This threshold becomes trivial only in the case that the destination is in the leaf set, since this means that the destination is only one hop away. Usually these two cases are common and the case in which the next hop does not compare both in the routing table and in the leaf set is quite improbable. If this happens, however, the system needs only an additional step on average to reach the destination for the message. The main disadvantage in the routing mechanism is that when many recent failures verify, the routing mechanism does not work very well and does not ensure the delivery of the messages. To lower as much as possible the probability of blocking for the routing mechanism an acceptable value for the length of the leaf set needs to be selected, such as 16 or 32.

2.5.5 The Tapestry protocol

Tapestry is a routing protocol created by Ben Y. Zhao, John Kubiatowicz and Anthony D. Joseph in 2001 **[50]**. The main goal of this algorithm is to provide a completely distributed and highly stable system through usage of statistic functions**.** Tapestry masks failure components, failed paths and attacked nodes, deleting them from the network. Moreover, communications between nodes are adapted to the several circumstances the network will face, i.e. instant failures. The peculiarity of this protocol is that each node is the root of its own tree, thus each routing table is an ensemble of spanning trees where the current selected node in the routing table is the root.

To gain an high stability, this protocol adopts massive data redundancy at different levels. At the lower level this means sending messages through different paths, reducing the standard deviation on communication latency. An alternative way is to use redundant links that are used only in case of failures, or to use data backups, or object replication in the network. A good choice concerns storing several copies of the same object in different nodes of the network, thus helping the persistency and the strength of data. To do all this, routing primitives and resources location techniques are combined, which means routing a message to a certain destination and ensure that the message reaches it, more than ensure that the final destination is the nearest to the searched resource. If a system like this exists the usage of a centralized point is superfluous but the service needs to be provided by an integrated device and not by a combination of distinct services or

routing infrastructures. Another problem concerns failures, since the network needs to work also in case of failures and information needs to remain highly consistent and available, because the network needs to be able to quickly recover from failures. Moreover, to avoid the contact by malicious endpoints a location system with cryptographic primitives is provided. Finally, the network needs to automatically organize itself given the dynamicity in presence of multiple joins, leaves and failures that can verify.

Routing As routing and location system, Tapestry uses a system similar to Plaxton with which each node posses an ID, usually of 160-bit, based on the hash of a random number with a certain length, calculated with the SHA-1 hash function **[50, 57]**. This mechanism assumes that each node maintains a data structure called **neighbor maps** to use as a routing map inside the network. The forwarding of a message is based on the longest suffix matching and, at each step, the message gets closer to the destination ID. The table is formed by multiple levels, where each level contains a number of entries equal to the value of the ID base. To find the next node to forward the message to, the current node needs to look at the next level in its neighbor map and to select the $(j + 1)^{th}$ entry in this level, that matches the next digit of the destination ID. The matching proceeds from the right to the left. Having a space of size N for all the nodes, the algorithm ensures that the message reaches the destination in at most $\log_b N$ hops if and only if the neighbor maps are not compromised, with b the base of the IDs. The main assumption of this protocol is that each node in the path of the message knows that at that time step, the suffix of the ID matches the destination suffix for at least l digits, where l is the current level of the neighbor map. This allows each node to maintain only a certain fixed constant number of entries at each routing levels, identified by a network parameter named b. In so doing each neighbor map will be of the constant size:

$$S_{Map} = Map_{entries} * Map_{number} = b * Log_b N$$

where S_{Map} indicates the size of the neighbor map, $Map_{entries}$ is the number of entries that can be stored in each row of the map and Map_{number} if the number of rows for the neighbor map.

Example 2.14 – Example of routing in the Tapestry network

*The routing mechanism in Tapestry works as in Pastry. Let us suppose to have the situation in Figure 27, where node **0325** needs to send a message to node **4598**. In Tapestry the next hop is selected with longest common prefix from the right to the left, so the last digit in each node ID is considered. In this case, the source controls at the first level of its routing table which node is the nearest with an ID like ***8. The choice falls over node **B4F8**, which increases the level trying to match **98, thus selecting **9098**. Then the next hop selected is **7598**, and finally the message reaches the destination **4598**.*

◊

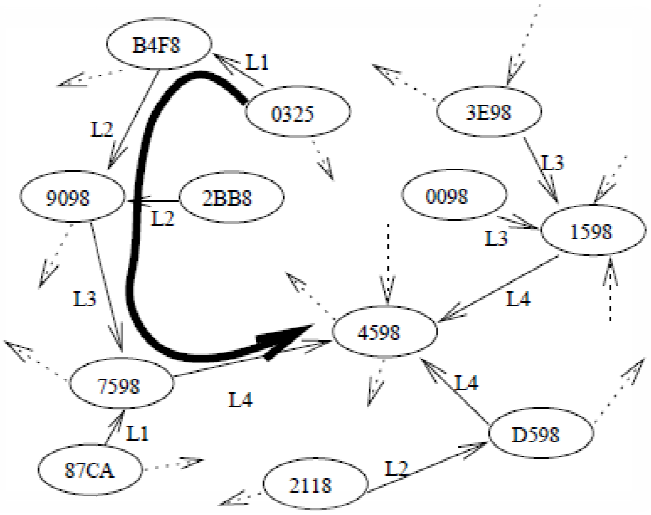

Figure 27 Example of routing in Tapestry **[50].**

Authentication Once a new node (A) needs to enter inside the network it contacts a bootstrap node (B). A joining procedure is then started, in which the bootstrap node sends in the network a sort of lookup message specifying the ID of the new node. The new node will receive back a scrap of its neighbor map from the route of the message (like in Figure 27), knowing that for each entry the new node shares a portion of ID, i.e. a specified growing set of digits in the suffix. From this initial scrap, the new node contacts its neighbors to receive their own neighbor map in order to check if the nodes registered in these new neighbor maps are nearer than the initial neighbors of A, and stores them if it is the case. This process goes on until A does not find node nearer than those already stored. After A has built its own map, it needs to inform the neighbors of its presence, thus it sends an alive message to its neighbors and to the set of substitute neighbors (the one created for network consistency). Once received the alive message, the neighbors of A stores its ID in the correct position of their neighbor map, and forward the alive message to all the nodes that can be interested in storing the reference of A. The process goes iteratively until no more nodes have to be notified.
When a node leaves the network, it needs only to inform its neighbors before departing or in case this is not done, its neighbors wait a certain timeout in which they contact the departed node in order to refresh its reference. If it does not answer within the time interval, it is considered failed and not contacted for further routing procedures. It is important to notice that the failed node is not removed from the maps of its neighbors but only marked as invalid. This will avoid the iterative update of the network in case its fault state is only temporary. Finally, in case that the underlining structure of the network changes, each node needs to send some ping messages to perform the update phase of the routing tables.

Queries When a node needs to find a resource inside the network, it can locate it with the routing mechanism and, once it has received the location, it can send a message to the server that stores and manages the searched resource. In the Tapestry architecture, each portion of the network is connected to a root node. When a server has to announce it has a certain object, it sends to the

root node a message containing the object ID along with the Server ID, identifying a couple **(O_{ID}, S_{ID})** stored not only at the root node but also at each node in the path from the server to the root. As we can clearly see, this root node identifies a single point of failure for the network since if it crashes all the information on the objects it manages can be lost or not properly propagated. The storing mechanism for the objects maintained by each server works in terms of message propagation. A message is sent from the server to the root node, informing the latter of the existence of the new object. Each node in the path from the server to the root will store the information in order to speed up an eventual search for the object. Once a certain node looks for an object that a server shares in the network, it sends a message through the network to reach that object. At each step the message gets closer to the root that posses the information about the object but, if the message reaches a node containing the reference for the server that manages the searched object, the source is redirected to this server. If no nodes that contain the reference for the server are found, the message reaches the root and the result will be the same. Concluding, roots are chosen with a deterministic and globally consistent algorithm that takes into account some global knowledge about the nodes in the network, such as their up-time and other useful information, to determine if they are able to take the role of a root node.

Improvements of Tapestry Tapestry evolves from Plaxton as an overlay infrastructure for the creation of scalable applications, that are also fault tolerant, in dynamic networks of big dimensions. Each entry in the routing tables points to a set of neighbors in the network and each node maintains a list of nodes resulting in the nearest set of neighbors for the current node. An improvement from Plaxton is that the information over the resource replicas coming from the various servers are all stored by the nodes contacted with the replication algorithm and not only those information regarding resources very near to the node positions. For this reason a distance metric is added to calculate the distance between nodes and resources. When requested, the resource is looked up specifying in which modality the search algorithm will give back results: proximity, fastest path or shortest one to the location of the resource, etc **[50]**. The main advantage of Tapestry is its system of failure recovery, since each node maintains a cache with some elements that are periodically refreshed or purged on the need. Failures may be encountered too: nodes or servers no more available, links that do not work anymore or corrupted tables. To detect these failures Tapestry operates on the timeout of the TCP protocol and each node sends periodically alive messages (similar to the keepalive messages of Pastry) to neighbors through UDP connections with which a node can receive free information on the validity of its table and on other nodes status. In addition to this, each node maintains for each neighbor a substitution neighbor and contacts it in case of failure of the first, since the co-failure is not so probable and this is the advantage. When a node is considered failed, it is marked as not valid instead of being deleted from the list and until the node is completely removed or restored, messages are forwarded through another path. Control messages are managed through a statistic function that specifies the maximum amount of messages to be exchanged in the network. To enhance the routing mechanisms, Tapestry provides multiple roots for a single sub-tree or object. To identify correct roots for the objects or the sub-trees some salient features from the IDs of the nodes are extracted and then hashed to provide an identifier for the newly elected root. These

values are then included in the contacting message by the server that maintains the resource. This increases the redundancy and the robustness of data in the network. More than this, data can be retrieved by multiple paths in the network and this improve also availability. For example, having s roots for a file the probability to have it available is $P = 1 – (1/2)^s$, because the event in which all the roots are failed is composed by several independent events in which the current S_i root is failed (with i: 1..s), thus the probability of each event (1/2, because the root is failed or not) needs to be multiplied to obtain the total probability ($1/2^s$). This value will be subtracted to one, since P indicates the complementary event in which all the roots are online. Finally, information on the resources are exchanged at a regular interval of time. This allows to have fresh information on resources and to be able to remove those that are not valid anymore. Only hardware failures can compromise seriously the life of the system.

Protocol Evaluation Tapestry is surely a strong routing protocol that provides an high fault tolerant system. A resource is located successfully with high probability, because of the high amount of replicas. The main disadvantage can be the high workload the network needs to bear, since many consistency operations are made to make the system work. Moreover, a node needs to bear an high amount of work as well, since many structures and algorithms are provided to the nodes inside the network to overcome possible failures or general problems. The principal disadvantage for Tapestry can be this high workload to both servers and clients. Another con is that alive messages are exchanged through the UDP protocol, that is not reliable and messages can be lost. Finally, the fact that some centralized points are used to provide resources, such as servers or root, is not good since they constitutes a single point of failure (even if information are highly replicated).

Complexity The complexity for the routing mechanism in Tapestry is $O(\log_B N)$ steps to reach a destination, where B is the base of the IDs assigned to the N nodes in the network, while each neighbor map posses $\log_B N$ rows each containing at most B bits.

2.5.6 The Kademlia protocol

Kademlia was created by Petar Maymounkov and David Mazières in 2002 **[35]**. This protocol is part of the third generation of routing protocols, and it is a compromise between centralized and decentralized networks, using DHT and providing the decentralization of P2P networks and the efficiency of centralized systems **[27]**. This is an overlay protocol that bases its functionalities on lower level protocols. With this the network structure is specified, as well as the communication methods and the information exchanging procedure. Specifically the network posses three basic properties:

1. **Decentralization:** Communication does not use a centralized point and search methods are used via keywords. All nodes divides the load coming from the resource storage duty.

2. **Scalability:** The network efficiency is assured also in presence of an high number of nodes.
3. **Fault tolerance:** The network also tolerates node joins, leaves and failures.

The protocol only needs to know the IP addresses and port numbers of nodes to calculate their IDs and to create a reserved UDP port for the communication, usually port 4672. The strength of Kademlia is in the fact that it provides a set of characteristics not offered by any other protocol seen until now, since it uses both the iterative and recursive routing and it implements the usage of multiple paths for each search activity. Finally, Kademlia uses a peculiar ID matching techniques for the routing mechanism, based on the suffix matching like Tapestry does **[91]**.

Identification System A node or a resource is identified in Kademlia with a 160-bit identifier, computed with an hash function applied to the couple **(IP address, port number)**, in the case of nodes, or to the content, in the case of resources. The hash of the resource will refer as **key** for the resource while the content of the resource is called **value**. The resource is then stored in terms of a **(key, value)** couple to the node whose ID is closest to the key, that will be the manager for the current resource. This will mean that a resource shared by a certain node (A) can be indexed to another node (B) simply because the ID of the resource will fall in the area managed by B. To choose the node in which to store the resource, the system uses a metric based on the XOR. In fact, the key and a node ID are XOR-ed together and the result gives a distance between them. An example is the following:

$$d(X,Y) = 10010110\ (XOR) 10101010 = 111100 = 60_{Dec}$$

The node having this value smaller than the others is claimed as manager of the searched resource. The same metric is also used when a node needs to contact another node during the searching phase. An important feature of the XOR metric is that it is **unidirectional**, and this means that if two messages are sent from different sources to a common destination, their path will converge to a common one.

Routing Tables Initially each node maintains a table with only one bucket that covers the entire dimensional space of the IDs. Once it has to record a new entry, the initial bucket is split, and this process is repeated every time a new entry have to be inserted in a new bucket. Generally, each node maintains in its routing table log N contacts of other nodes in the network. Each entry is made of three elements: the IP Address, the node ID and the UDP port on which the node is listening for incoming connections. Routing tables are organized by levels, where each level is a row and contains one prefix. Each row contains a series of k buckets (usually k is taken equal to 20) in which are contained those contacts that share the same prefix of the current bucket. The Overlay is a binary tree organized by node IDs and each node has at least a contact in its hash table for each sub-tree in the Overlay (see Figure 28). Finally, each bucket identifies a leaf of the binary tree.

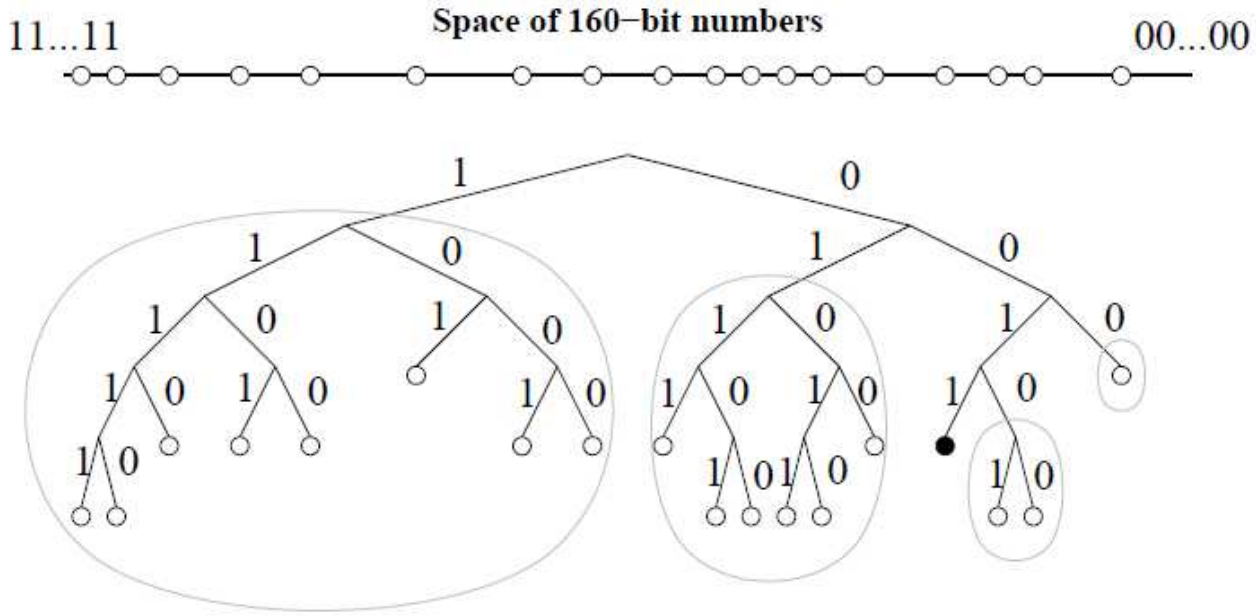

Figure 28 The Kademlia binary tree **[27]**.

Queries Once a node (A) needs to find a resource (R), it looks for the bucket corresponding to the sub-tree that matches the longest prefix of R. After this, it sends a message to contacts in the sub-tree to locate the resource or to find the closest node to it. The process is repeated until the resource is found and at each step the message gets closer to the destination pruning the number of candidates for the next hop of about the 50% in the best case. Kademlia routing tables differ from Pastry routing tables in terms of stored contacts. In Pastry the number of entries per each cell of the routing table was equal to 1 while in Kademlia this value is usually equal to 20. Moreover, the Kademlia network results more robust and allows to choose alternative paths for the routing mechanism, since each bucket is composed by more than one entry.

Authentication When a node wants to join the network it needs to contact a bootstrap node. An ID is then assigned to the new node and the bootstrap node finds where it is located in the network and assign it a routing table which covers the entire ID space, dividing entries in blocks by ID suffixes, that are the buckets. Buckets are ordered with an uptime associated to each node. At the top of each bucket there are nodes with a high uptime, while at the bottom are inserted nodes with a lower uptime. When a new node not contained in the bucket contacts the current node and the bucket is full, the receiving node contacts the first node in the bucket, i.e. the oldest one. If it does not respond, the old node is deleted and the new node is inserted at the bottom of the list. If the bucket has space, or it is empty, the node is inserted at the first empty position. This property is used because nodes with higher values of uptime are more probable to stay online in the future. Each time a node receives a request or participates to a search its table is updated, otherwise it is updated once per hour.

When a node logouts from the network it can do this in two ways: voluntarily or not. In the first case, the node decides to logout and partitions the portion of addresses it was managing, sending some copies of each couple <key, value> to nodes that have became the new managers for the resources, while the outgoing node is deleted from the routing tables. In the second case,

the node fails abruptly. This means that, without replication of information, each reference to keys or files are lost in the process. In real environments replication is made during the routing tables update.

Messages Each node in Kademlia uses four types of messages to interact with other nodes. These messages are the following:

- ✓ **Ping:** is used to ask a certain node if it is alive or not.
- ✓ **Store(key, value):** is used to store a resource to node(s) nearest to the specified key.
- ✓ **Find_Node(ID):** is used to find a node or a file possessing the specified ID. Once found the triplet associated to the ID is returned.
- ✓ **Find_Value(ID):** it works like the previous but this time it returns the value associated to the key instead of the triplet.

Each message is then identified by a unique ID to avoid loops, and to use the last two functions the system uses the prefix matching algorithm exposed in Chapter 1. During a search, nodes compare the prefix of the destination node to the bucket identifiers in order to find where it will fall. Each search is made in parallel, that is the contacting node sends a certain number of messages to find the current resource to a subset of its neighbors. The number of parallel searches is fixed to a system parameter α and usually it is equal to 3. This is used to make each request quicker and more probable to success. With this mechanism the system provides more paths to find the destination in order to avoid faults, but it does not ensure that the first path that reaches the destination node is the optimal one. Moreover, each request can be controlled by the source node, or its control can be demanded to further nodes in the path. In Kademlia, the network is ensured to be visited in at most the order of *log N*, where N is the number of nodes inside the network. The scheme behind this protocol is highly scalable but requires that the searched content resides in a node in the network in order to be found by other nodes. Finally, differently from Chord, each node can update its routing table by inserting new nodes in the buckets that are contained in the messages forwarded in the network.

Example 2.15 – Example of routing in the Kademlia network

*Let us take as example the situation of Figure 29, and that the black node, **0011**, wants to contact the red node **1110**. In its routing table the source node has stored contacts for node **1001** and node **1100** selected with the prefix algorithm. At this step, having to choose one of the two, it computes the XOR distance between destination and both nodes separately, in order to choose the nearest one.*

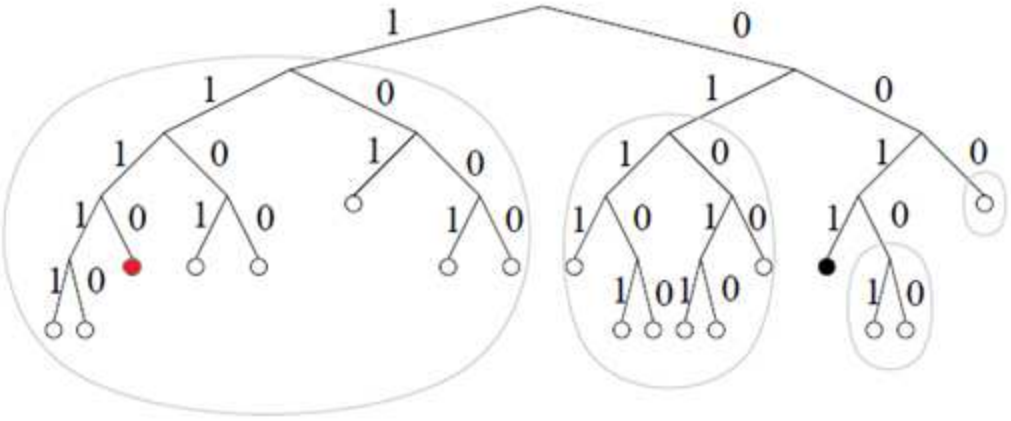

Figure 29 The initial state of the network **[35].**

The result for each operation is exposed by the following two operations:

$$dist(1110, 1001) = 5$$
$$dist(1110, 1100) = 2$$

Node 1100 is the chosen as the next hop, since its distance to the destination is lower than the one between the destination to node 1001. After this, if nodes 1100 knows the destination, it returns to the source contacts for the destination, otherwise the procedure is repeated until the destination is reached. If the routing is made in parallel way, both node 1001 and node 1100 would have been contacted, since the number of parallel paths by default is taken equal to 3 (the α parameter).

◊

With the iterative routing each contacted node gives back to the source a set of candidates, to which to forward the messages, that satisfy the proximity measures through the destination. Then the source orders these results by increasing the distance from its position and contacts the first α nodes to proceed the routing.

Protocol Evaluation Kademlia evolves from Tapestry and Pastry, creating a good merge between these two protocols. Its strength is on the routing of messages since each message is forwarded at least α = 3 times to three different nodes in order to achieve a good delivery ratio between the number of sent and delivered messages.

Complexity A routing operation takes a number of steps in the order of $O(\log_B N)$ where N is the size of the network and B is the base of the IDs assigned to the nodes. Buckets have a dimension of $B*\log_B N + B$, slightly smaller than Pastry but bigger than Tapestry **[24]**.

2.5.7 The KAD protocol

KAD is a type of protocol based on Kademlia and used in eMule to reduce bottlenecks and improve anonymity inside the network. It provides a system of publication and retrieval of information

based on UDP, while the download process is made through the TCP protocol. The presence of NATs or firewalls is handled automatically.

KAD ID Space In KAD the ID space is mapped over a 128-bit identifiers space with usage of the XOR metric and it allows to have 2^{128} identifiers in the network. The routing table system used in Kademlia is modified to allow the insertion of more nodes than the normal case, and join operations are made through nodes already in the network since the applications that use KAD provide a list of active nodes, differently from Kademlia. Nodes behind NATs or Firewalls are managed by **firewall checks** and **buddy peers** procedures, presented further.

File Sharing Once a node wants to publish a file, it needs to store the file on a local support and to use the DHT system to store a series of references to the file. It is not suitable to share the entire file in the DHT because it will mean to overload the network. A certain number of nodes are contacted with the identifier of the file and they maintain it to be able to provide the reference to the file when requested. The number of replicas to send is fixed to R, and the subset of nodes maintaining the references to the file is called **tolerance zone** for the file. KAD uses an auxiliary system of identifiers with MD4 that is added to the original identifying system for nodes, with fixed non uniformly distributed values not uniformly distributed. To each file or node some KadID identifiers are given and for each file two different couples **(KadID, list)** of pointers are published. The first, namely the **Location Information**, maintains the file identifiers as an hash and a list of logical identifiers of the nodes that publish and store the file. The second list, namely **Metadata**, contains a logical identifier computed from one of the keywords identifying the file and the logical identifier of the file. The first list is more important than the second one and it points directly to the client that published or stores the file, while the second one points to the previous list. Notes associated to each file from users are published in the same way, and they include the **rating** of the file and **comments** on it. Finally, since a set of keywords can correspond to different files, along with the hash of these keywords an identifier for each file is associated. Moreover, the same file can have more users that shares it, thus a logical list of these users is provided.

Routing Tables Since the logical network is organized as an unbalanced binary tree, routing tables are the leaves of this tree, corresponding to the buckets in Kademlia, and contain a set of nodes with the same prefix. The path from the root to a certain leaf is the common prefix for nodes inside the routing table. The path to the root maintains a set of distances that once concatenated along the path from a node to another gives the total distance between the two nodes. To have the ID of a certain node we need to subtract to the total distance the ID of the considered node. Each path on the tree corresponds to a certain set of distances. Leaves corresponds to buckets for routing tables and each of them contains nodes that have a distance equal to d from the considered node. Each level of the tree has two **routing zones** that contain the level of the tree and the zone index that contains distances in terms of routing zones from the considered to the right-most at the same level. Buckets become **routing bins** and each of them contains a maximum of ten contacts. An example of tree with routing zones is shown in Figure 30.

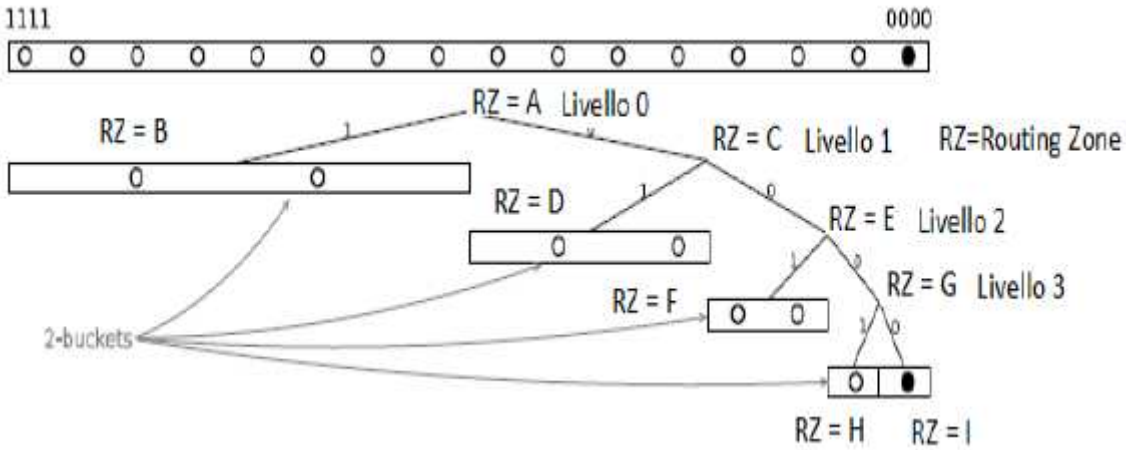

Figure 30 KAD tree and routing zones **[35].**

Example 2.16 – Example of routing in the KAD Network

Once a node (A) needs to locate a resource or a node, it performs a search similar to the one used by the Kademlia protocol. A starts two threads: the first one is aimed at finding those nodes near the target entity (in terms of prefix matching) inside the routing bins; the second one is aimed at querying those nodes found by the first one. The first thread returns 50 contacts, starting the selection from the bin nearest, in terms of prefix matching, to the destination ID and then selecting the second nearest bin, then the third and so on, until the number 50 is reached. The second thread then starts by contacting all the nodes returned by the first thread in parallel (with 3 parallel paths for each contact). In the message sent to these contacts are inserted the ID of the destination and the number of candidates requested to proceed with the forwarding of the message. The process is repeated until no nodes nearest to the destination than the current position are returned.

◊

Once a node needs to add new nodes inside its table, the operation is not the same as in Kademlia. To add a contact a node select the proper container and, if it is not already inside the bin, it is added on the tail. If the bin is full then the node controls if it can be expanded and, if it can, then the bin is duplicated and the new contact is inserted in the new one. The expansion of the bin have to be controlled to not add more contacts than allowed. The expansion can be made only to zones near the current node, allowing in this way a logarithmic routing such that in Chord. Once a bin is duplicated a leaf is changed to internal node and to this internal node two leaves are attached, representing the two new bins. The duplication is allowed only for certain zones of the tree and only for certain levels:

- ✓ If the level is lower or equal to 3 then the expansion is always allowed and routing zones with index higher than one are created. The duplication takes the tree to have four levels and then sixteen bins.
- ✓ If the level of the tree is higher or equal to 4 then the duplication is admitted only if the level of routing zones is lower than five. The subdivided bins are those rightmost on the tree, and contains contacts related to those zones that are nearest to the examined node.

The split mechanism does not go over the 127th level of the tree, if the length of the IDs is 128. The bin cardinality is from the right to the left.

The total number of contacts that can be stored in the tree is ten times the sum of leaves at level four, leaves on levels from 5 to 127 and leaves on level 128, since each leaf contains ten contacts. The insertion of new contacts implies the expansion of the tree. The tendency is to expand the tree to the right in order to have as many node as possible in close routing zones, thus the routing can be logarithmic such as in Chord or Pastry. Edges has a label depending on the side the tree is expanded: if the expansion is on the left the edge will be labeled with a 1 while if the expansion is on the right it will be labeled with a 0. An example of expansion is represented in Figure 31.

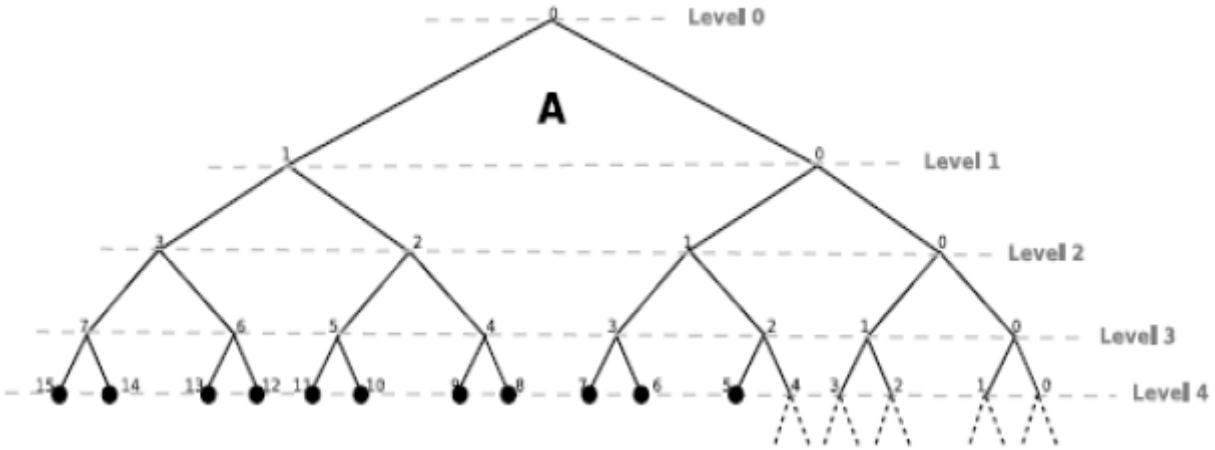

Figure 31 Expansion of a KAD tree **[35].**

To update the routing tables, two mechanisms are used: **trivial contacts deletion** and **new contacts search**. In the latter a node controls periodically if in its routing table there are some empty bins or some with empty positions to fill or if some of those that are full can be split. Then, a message is sent through the network containing the ID of the chosen bin to find nodes near the ID of the bin and those that answers are inserted in the bin. For trivial contacts deletion the mechanism controls the type of the contacts and their expiring time. The latter concerns frequency of pings for each node, that is with which rate the node have to be contacted with alive messages. Contact can be of six types, as shown in Figure 32.

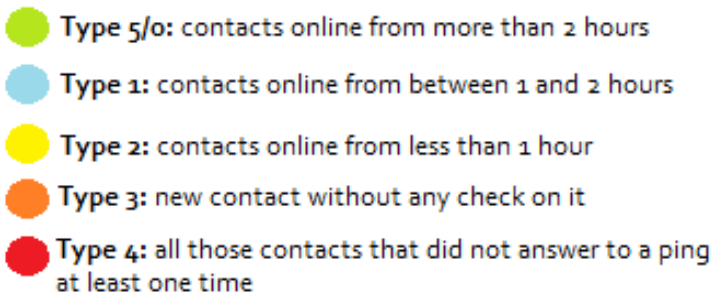

Figure 32 Type of contacts in the KAD network.

The priority is given to older contacts, since their presence until the current moment increases their probability of being up in the future. For all the others at a regular and short interval of sixty

seconds their expiring time is controlled and if it is expired they have to be contacted. If the contacts answer they are maintained at the tail of the list, their type is updated and their expiring time is reset. Periodically, those bins containing five or less contacts are controlled and sometimes merged in an interval of forty-five minutes usually.

Queries A Search in KAD can be made in several ways:

- ✓ **Lookup:** it is the search for new contacts to insert inside bins.
- ✓ **Sources:** if there are some files in the download list some sources are contacted for them, that are nodes supplying files.
- ✓ **Keywords:** is the common search we do on a search engine by specifying some words describing the file we are looking for.
- ✓ **Notes:** some notes are specified for the sought file.

Anyway an ID is specified when a search is provided, that can arrive from: the hash of the contacted file, the hash of a client, the hash of a note, etc. What we gain back is the set of nodes nearest to the contacted ID. Even if the number of nodes in the routing table is high we have a low probability to have the destination node inside it for the network dimensions. For each search a thread is started aimed at finding all nodes nearest to the provided ID plus another to query the found nodes to take or store the information. The first thread returns the first fifty contacts nearest to the provided ID by XOR metric, starting from the bin nearest to the goal. The search is parallel and it is stopped once no more node nearest to the goal are found from the previous step. Results are useful to proceed in the search process and to populate the routing tables. The search of sources is made on background for each files and terminates once each file has at least fifty sources. It is split in two phases:

1. In the first phase metadata are searched computing the hash of the keywords specified by the user. Given this hash, those keys in the tolerance zone for the specified hash are sought. The starting client gains a set of references for the files containing specified keywords and they are added in a download list. The routing mechanism is based on Kademlia and can be iterative or parallel.
2. The second phase works as the first but here the goals are sources for the files in terms of iterative and parallel routing in the tolerance zone.

Once a file is published, the node needs also to publish that it is a source for the file, the keywords that defines it and the notes associated to the file. Information related to the file are replicated in the tolerance zone of the associated ID.

Firewalls & NATs In case of client behind a NAT each server can verify and check this eventuality, and start the **buddy peer** procedure as follows. The node (P) identifies a contact (C) and sends to it a message containing the TCP port on which P waits for connections. C find the IP of P in the message and sends it back to P. This last verifies that the IP is the same of its own and if they are

different, than P knows to be behind a firewall or a NAT. At this point, P is given a low ID, and peer of this type is marked as **firewalled** and cannot receive requests for publishing or search, and KAD have to automatically find another client that is not firewalled to use as companion for the first. Each normal node can deal with a limited number of firewalled nodes. To have an uniform distribution on these buddies they are determined by inverting the ID of firewalled nodes and each node can receive at most one request to being buddy of some firewalled node. Once contacted, the buddy answers and a connection is opened between the two nodes (see Figure 33).

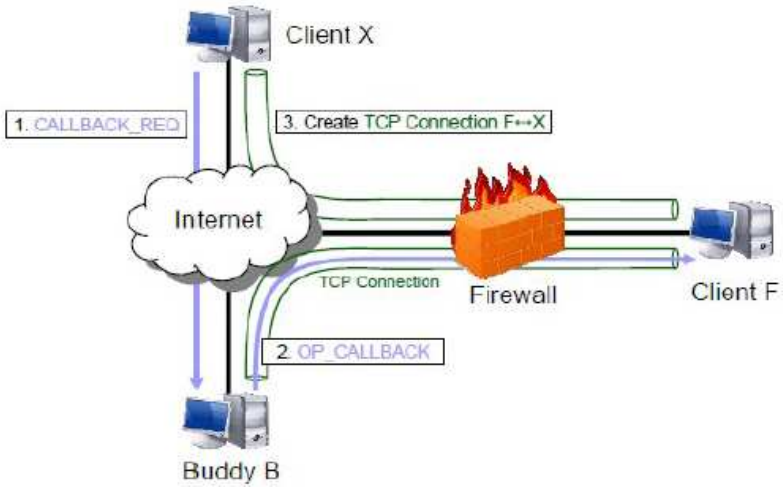

Figure 33 As the buddy system works **[35].**

Protocol Evaluation To protect the network from overload, file ID are mapped uniformly in the network in such a way that each node manages approximately the same number of files. The difference is among sources for the files, even if differences in interest of nodes provides a balance also for the sources. The main problem can reside in keywords, since one keyword can identify more than a file. A node can modify at most 60,000 metadata and if it manages a keyword with high references on files this value is reached quickly. Generally a node does not store metadata if it has already an high quantity stored. The overload is also in terms of replicas, fixed usually to 24 hours but in practice the hours to wait are computed over the formula:

$$7days * \frac{AV_{load}}{100} \tag{2.2}$$

where AV_{load} indicates the average number of entries stored at the current node for the current keyword. The node that has to share a keyword contacts the destination nodes for their load in terms of stored metadata, and calculates their mean. If this value exceeds a certain threshold the keyword is inserted in a **banned list** and nodes that maintains it are considered as **hot**. A certain time has to pass before the keyword can be published again.

Complexity The KAD protocol evolves from Kademlia, from which it inherits the complexity that is a message is said to be delivered in $O(\log_B N)$ where N is the number of nodes in the network and B is the base of the IDs assigned to each node. Routing tables posses at most ten contacts per routing bin so their dimensionality is equal to $10*N_{bins}$.

2.6 Routing in modern P2P networks

In this section we will describe two of the P2P networks mostly, i.e. BitTorrent and the eDonkey network used by eMule.
Let us first define what **Content Distribution Networks** are (see Figure 34).

Definition 32: *A **Content Distribution Network** (CDN) is a set of distributed hosts that cooperates to spread a great amount of data to the final users. These data can be of different types, usually constituted by multimedia contents.*

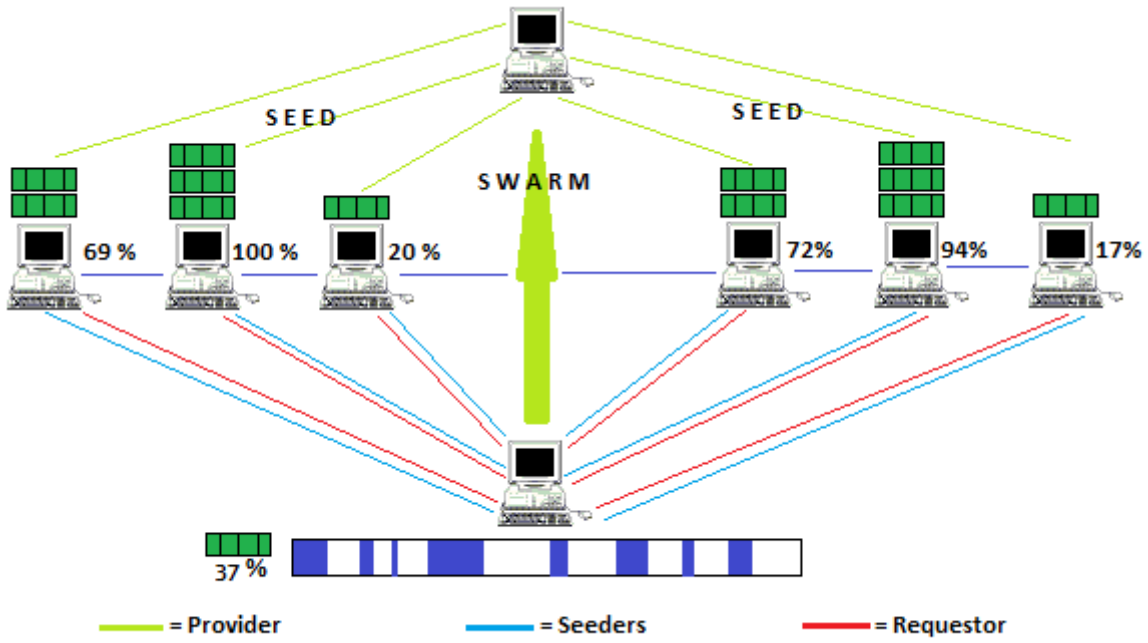

Figure 34 Example of Content Distribution Network.

2.6.1 The BitTorrent network

The first technology presented is BitTorrent, developed with Python in 2002 by Bram Cohen **[54, 103]**. The initial idea for BitTorrent was to provide a new way of encrypted content exchanging for corporate or enterprises. The idea was to break a certain file and to divide it in the network so that at the arrival of a request the workload is balanced through the entire network. This technology uses a network based on content distribution where a certain number of hosts shares a great amount of data in a cooperative way. This is useful to avoid central servers congestion. To improve this feature, also server mirroring in a way called **flash crowd** is used to manage a great level of traffic generated by an high number of connections to a Web server. The main goal of this kind of networks is to distribute contents workload in order to use all the download bandwidth through asymmetric connections: the upload duty for a certain file is distributed among all the nodes. The main goal of BitTorrent network is to distribute contents using external search mechanisms.

File Sharing When a host wants to share a file, a descriptor for it is created packed in a file with extension *.torrent* and published on an HTTP server. This kind of file is created over the **Bencode**[5] coding that gives flexibility and a good lightweight structure to handle **[54, 105]**. The descriptor contains a series of references to **trackers** that provides a coordination service between hosts that cooperates in sharing the file, the latter called **swarm** (Figure 34). On the other hand a node that wants to download a file receives the *.torrent* file and opens it with an ad-hoc client application, connects to the trackers pointed by the file acknowledging them for its existence. These trackers provides back a list of nodes that resides in the swarm and all the nodes connected starts exchanging parts of the file maintained locally. In this environment there are two kind of entities: the first one is represented by those nodes possessing the entire file to share called **seeders**, and the second represented by those nodes that possess only some parts or none of the file called **leechers**. Each file is subdivided into blocks of minimum length of 256 Kb that are subdivided into smaller blocks of 16 Kb with transfer made block-by-block. The file descriptor is created automatically or through the client to which the file name and the trackers addresses are specified. A descriptor can contain more than one file inside it and these will be subdivided and chained together. Having more small pieces of a file is good to improve the speed of the sharing but having less pieces means to occupy lower bandwidth, so a good trade-off have to be chosen.

From the tracker side, a subsample of nodes subscribed in the swarm for a certain file are sent to all the nodes interested in the file, composing each information by the IP address and port number of the nodes in the swarm. During the sharing activity the tracker continuously connects new nodes demanding the file, noticing the swarm of the newcomer. If a node notice that the number of seeders or leechers is behind a certain threshold than it contacts back to the tracker to have new contacts. All these operations are made through GET requests over the HTTP protocol. When a node contacts the tracker inserts the following information in the GET request:

- ✓ **Info_hash:** is a SHA-1 hash of the **info** dictionary of the Bencode coding as specified by the URL conventions.
- ✓ **Peer_id:** that is a 20-chars string allowing the almost unique identity of the user over the tracker list.
- ✓ **Port:** is the client port where it is listening;
- ✓ **Uploaded:** it is the amount of bytes sent to the other nodes from the beginning, codified in ASCII base ten.
- ✓ **Downloaded:** it is the amount of data downloaded by the client, codified in ASCII base ten.
- ✓ **Left:** specifies the amount of data remaining to complete the download in ASCII base ten. The value zero corresponds to seeders.
- ✓ **Event:** this is of three types: **started**, **stopped** and **completed**. The first one is used when the tracker is contacted, while the second closes the connection with the tracker and asks for being removed from the tracker list. The third communicates the passage from leecher state to seeder state since the download is over.

Other parameters can be included inside the request but they are optional. In the following we will outline these parameters:

[5]*With the Bencode coding data are expressed in a format more concise than their original shape. An example of Bencoded string is: <ASCII encoding of the length of the string>:<string data>, or in the format <key>:<value>* ***[54]****.*

- ✓ *Compact:* indicates to the tracker to use the Compact Announce modality.
- ✓ *IP:* is the IP address of the peer if it is behind a NAT.
- ✓ *Numwant:* specifies the maximum number of sources to be contacted by the tracker.
- ✓ *Key*: is the randomized string to better identify uniquely the client.
- ✓ *Trackerid:* this field is used if the tracker has communicated before its ID to the peer.

To this announce a tracker can respond with a message containing the reason for the possible contact failure or a warning message. If the client is using the Compact Announce modality the answer from the tracker containing the subsample of peers is formed by a string where each node occupies 6 bytes, 4 for the IP address and 2 for the port number. If the Compact Announce is not used the answer contains a set of dictionaries for each node in the answer. The dictionary is composed by:

- ✓ **Peer_id:** is a string of arbitrary value supplied by the node during the connection phase.
- ✓ **IP:** is a string containing peer IP in IPv4, IPv6 or DNS format.
- ✓ **Port:** is an integer containing the number of port used by the node.

As a convention, the number of nodes received with a request should not overcome the value 50. To count nodes connected to the tracker for a certain file, the tracker can count those connected to the announcer for the file. In practice the tracker makes use of a scrape script that is translated by each client from an announce link specified by the tracker itself and connected to the file. Once a client wants to have the scrap, it contacts the tracker with the GET primitive and specifies the *info_hash* for the sought descriptor. The server gives back a dictionary as follows:

- ✓ **Info_hash:** containing the identifier for the current descriptor.
- ✓ **Complete:** contains the list of all the seeders connected to the tracker.
- ✓ **Incomplete:** contains the leechers actually connected to the tracker.
- ✓ **Downloaded:** number of times the file was downloaded.

If with the GET request is not specified the *info_hash* for the descriptor, a list of all the descriptors in the tracker with the above information is returned.

Routing Nodes communicates through **Peer Wire Protocol** where each node chooses a number of elements from the subsample provided by the tracker and establish a TCP connection leaving some of them empty for incoming remote nodes. When a node is contacted it can refuse the connection in two cases: if the ID of the requesting node is not inside its subsample or if the requested file ID is not inside its subsample. Otherwise, once accepted the connection a list of files maintained by the requestor is created. Through these connections the message exchanging takes place over the file inside the network. From version 4.1.0 BitTorrent supports also the Kademlia architectures being able to handle also tracker-less peers.

Protocol Evaluation Problems with Bittorent concerns bottlenecks corresponding to the trackers positions. As we learnt a good intuition was to add a multi-tracker system in the network and, as done in last few years, to adopt the DHT technology. For the multi-trackers case each descriptor contains a certain number of trackers for a file and the final aim is to balance the incoming workload between all trackers. This improves also the information redundancy since each tracker posses the same information. In the second case, a node acts also like a tracker. An hash of the file inside the IDs space is assigned to each of these node-trackers as well for the swarm list. Another possible solution is to use gossip algorithms where nodes exchanges continually information about contacts in the network in order to relieve trackers and to improve network reliability.

Complexity The BitTorrent network ensures the location of popular contents and the download performances are very good for this kind of contents. Sometimes a user can encounter problems while trying to upload some contents, since the network can refuse the uploading for some time. The node needs to send a query message for a resource to the tracker to receive the list of nodes that share the parts of the queried file. Once it has the list, the node starts opening a parallel number of connections equal to the number of parts in which the file is subdivided. Supposing the queried file is subdivided in 10 pieces and that each piece is shared by 5 nodes each, the source node opens 50 connections to all these peers. Each connection needs an handshake between the parts involved, a request for the part and the response for this request, thus each connection involves 4 messages: 2 for the handshake and 2 for the downloading phase.

2.6.2 The eDonkey protocol

The eDonkey protocol is used in softwares like eMule and it is a protocol for content sharing of all types. Servers that use this protocol have the final aim of locating file sought by the users, as in the case of Napster. These servers are inserted dynamically in the network and DHTs are used, with the KAD network based on Kademlia.

The eD2K network uses an extension of the eDonkey protocol with a client/server architecture and TCP or UDP protocols. The eDonkey protocol manages interactions between client and servers and also between clients. Each client implements some strategies like the **credit system**, the **stack managing** and the **corrupted files handling**. Each client has a pre-loaded list of active servers and selects one to connect to through the TCP protocol. To perform the search mechanism a client opens hundreds of TCP connections with other clients being able to download the same parts of a file from different hosts and to share these parts even if the download is not completed. UDP connections are used to refresh the servers list and to verify the state of nodes in the download list of other nodes. Each server maintains a list of descriptors of files shared by the nodes and sends them to the clients a series of files or nodes actually connected to the server. There are no physical storage of files or communication between servers and these can be used as buddy in case of peers behind NAT or firewalls.

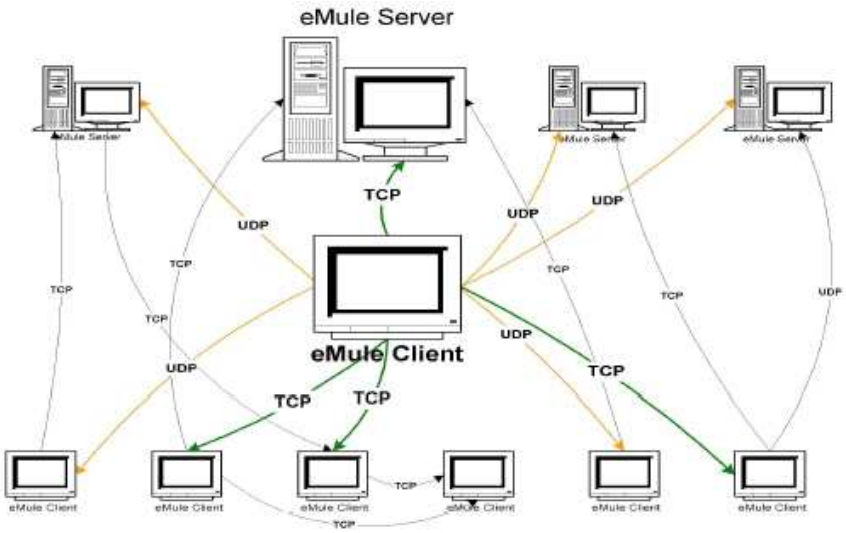

Figure 35 An example of the eMule network using eDonkey **[35].**

Authentication Each user is identified with an **User ID** of 128-bit, used by the credit system and that is assigned the first time the client is started and is never modified again. Clients posses their own **Client ID** of 32-bit that is valid only for TCP sessions and can be high or low with respect to the connection type in the network. Files are identified by an MD4 128-bit string assigned by the system once the file is being shared. It becomes uniquely identified by this ID in the network and can be queried by the users.
Once a client connects to a server it chooses only one and this choice can be changed only by the user during its uptime. Two TCP connections are started from the server to the client and back, and at this time the server chooses if the client will have an high or low Client ID. This choice depends integrally from the fact that the client is behind a NAT or not. If the connection is not refused, the high ID is assigned over the IP address of the client with a special formula as explained in the following example.

Example 2.17 – ID assigning in eD2K

Let us suppose to have a requesting client with IP address like 153.24.168.12. Its new ID will be: $153 + 256 * 24 + 256^2 * 168 + 256^3 * 12$.

◊

Once the connection is established the client sends a list of files to be shared and receives back a list containing the servers known by the first one. With this list, the server sends also a list of sources for all the files in the client download list with a low level of sources. If two clients (A) and (B) are connected to the same server and B has a low ID the file exchange happens through the server. Briefly, A asks to the server a **callback** that consists in a new connection from A to B through the connection already open between B and the server. On B side, a new connection is open with A and the exchange process begins.

Queries Looking for files, the client sends a query to the server that answers with a list of file satisfying the query parameters. Once received, the client communicates to the server the list of

file to download in order to receive the associated sources. The UDP communication is used only for keepalive messages that consists in a random number echoed by the server that if does not respond after a certain time is considered offline. Other uses for the UDP connections are those to increase the number of sources for a file and to increase the query hit rate that happens in communications between servers in the client list.

Routing Communication can happen also between clients, with the major interaction using TCP different for each triplet **<file, client1, client2>**. When the connection starts an handshake is made between the two clients via a secure identification. After this step, the client that wants to download a file communicates to the other this purpose, and the latter inserts the requesting node in an upload list, unique and organized by priority. The number of downloads is limited in order to save the bandwidth of the client. If the stack of node (A) is empty then node (B) is served immediately, otherwise B request is inserted at the bottom of the list of A and the latter receives a message with its position in the list. The priority system for clients calculate their value with the product between client scores and waiting time and then normalizes this calculation with 100 in order to have a percentage. The score of a client is calculated with respect to the credits maintained by the contacted client for it. The download phase begins when the client reaches the first position in the list. Another priority system is created once a file is shared and the sharer sets a priority for this file, automatically or manually. Each file possess also a stack rank that indicates how many users are before the current in the download list. It is important to notice that the downloader can download only a certain part of the file from one supplier and for each connection only three parts of the file can be downloaded from the downloader.

Protocol Evaluation As we will see in the end of this paragraph, a well known problem in file sharing applications is the one concerning **Free Riders**. To avoid this problem, the network inserts a credit system to motivate the resource sharing. When a client uploads a file to another one it is regarded by a non-global credit from the receiving side, in the range of [1, 10]. Clients that receive connections store the credits associated to the incoming clients. The mechanism is provided for a secure file exchange and a secure identifying mechanism to prevent illegal requests for credits by clients without them. All credits are maintained in a configuration folder for a maximum time of 5 months, deleting those that are expired at the first useful connection. Credits are assigned with the following formula:

$$\min\left(\frac{TRB_{P2} * 2}{TSB_{P2}}, \sqrt{TRB_{P2} + 2}\right) \tag{2.3}$$

where TRB stands for **Total Received Bytes**, TSB for **Total Sent Bytes** and P2 is the source node to which to assign credits. In some cases the value 1 is assigned if TRB is lower than 1 MB and if TSB is zero the credit goes to 10, so clients that gives more data than received are privileged.

Complexity The eDonkey network ensures to the user to locate precisely a popular content and that the download speed for this content, as well as the sources retrieval process for this kind of contents, is very high. On average a user needs only to send a message to retrieve the contacts for

a resource he wants to download and the server answers with another message, from which the user selects the host from which to download the resource, thus 2 * N messages in the worst case.

2.6.3 An eDonkey based network: Overnet

Another kind of network based on eDonkey and KAD is **Overnet**. It is based on DHTs with algorithms similar to those used by Kademlia. Once a node enters the network it is rewarded with a 128-bit ID and when a resource is sought using a key of 128-bit as well, a node sends UDP messages to the neighbors of the key calculating their proximity with a specified metric.

Authentication & File Publishing Once a node connects to the network it tries to connect to at least one node in its local cache and once found, it sends a search message to fill its routing table. Once it has enough contacts it informs them of its existence in the network. At the end of this phase it can start publishing information over files it wants to share. This procedure comes in two phases:

1. After computing the hash of the file the node sends a triplet composed by the hash, the IP address and port number for the file over the table to nodes nearest to the file ID.
2. In the second phase, the node extracts some keywords from the file and hashes them in a 128-bit key length that it will send through the table to the nodes near the hash of the file. All the data are passed with iterative routing.

Queries During the search phase, the node that wants to download the resource inserts a certain amount of keywords that will be hashed and sought for correspondence in the routing table. Once a node receives a query pointing to a set of keywords it maintains, it will respond with a query match that consists in the identifier of the file and the associated metadata, more than the set of identifiers (once per keyword). After this, the source node maintains only those titles that matches the entire query and selects some nodes from its routing table to start the download with TCP connections.

2.7 Conclusion

What emerges from this chapter is that routing is not a simple affair, when talking about P2P networks. In fact, each protocol presented in this section has its own way to route information and its own way to store contacts of the network at each node. Some of them are highly secure, while some others are not.

The main goal of this chapter was to prepare the reader to the next chapter, in which P2P networks attacks are presented. This kind of attacks aims at disrupting the routing service provided by a protocol in different ways, and at subverting the voting mechanism and resource availability in the network. In the next chapter we will outline all these attacks giving some examples over some of the protocols presented in this chapter.

Free Riders To conclude this section we will analyze a well known problem for P2P applications, namely the problem of **Free Riders**. These are users that do not participate actively to the network life and to the content distribution using it freely and occupying all their bandwidth for the download and leaving nothing for the upload. These kinds of users are real because all files are resources and as resources they cannot be blocked for some users. Some countermeasures have to be taken in order to diminish the level of free riders in the network but solutions to be supplied at client-side are not good because through reverse engineering they can be exploited.

CHAPTER 3

Attacks in P2P Networks

In this section we will introduce some network attacks of which P2P networks suffer from. We will outline their structures, characteristics and strength points and, after their detailed description, we will present some examples of real applications of these attacks in P2P networks. We will start by presenting their taxonomy and correlation.

3.1 Attacks Taxonomy

P2P attacks are grouped by their functionalities and properties, and several families of attacks can be discovered in P2P environments **[49]**. In Figure 36 is represented a taxonomy of P2P attacks.

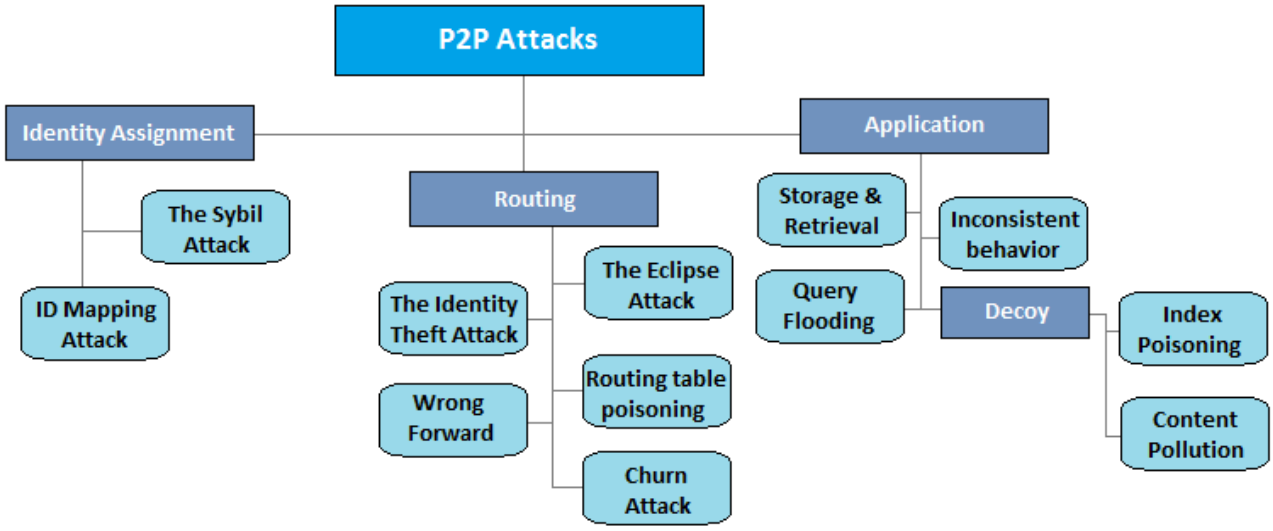

Figure 36 Taxonomy of P2P Attacks.

Let us outline each family to see their properties and behaviors:

- ✓ **Identity Assignment:** In a P2P network every entity is supposed to have a unique random identity to participate to the network activities. With these attacks a malicious user tries either to have multiple identities (*Sybil Attack*) or to have assigned a specific ID, possibly next to the value of the key of a resource or a to node inside the network it wants to take control of (*ID Mapping*).
- ✓ **Routing:** The main functionality for a P2P network protocol is to continuously update the network in order to record joins and leaves of nodes as well as storing new routing table values on each node. These data have to be correct in order to permit the retrieval of correct information during the lookup phase. A possible attack that compromise the correct update of routing tables consists of sending along the network fake routing tables to force nodes to delete valid information and store incorrect one (*Table Poisoning*). Moreover, a malicious node can arbitrarily block the information passing through its

position or send them to invalid nodes in the network to create a wrong routing (*Wrong Forward*). A third method could be claiming to have a resource that is being queried in the network, not having it in reality. This attack consists, on the attacker side, to steal the identity of the node that manages the queried resource in order to take its place and fool other nodes during the search phase (*Identity Theft*). Another method is to exasperate the churn. If a malicious node is able to insert and delete a great quantity of nodes in the network, this could cause routing congestion given the continuous update of the routing tables (*Churn Attack*). An ultimate way to block the location of a specified node or resource is to insert multiple fake nodes around the desired location, via Identity Assignment attacks in order to potentially take control of a specified section of the network and blocking all the requests directed to the blocked entity (*Eclipse Attack*).

- ✓ **Application:** These attacks are at the application level, aim at blocking accesses to some resources. A malicious node could claim to posses resources or capabilities it does not have, or vice versa, fooling other nodes (*Storage&Retrieval Attacks*). A similar attack consists of inserting fake information in the network, i.e. unreadable or corrupted files (*Decoy – Index Poisoning*). This is typically used by film and music producers to prevent the illegal download of their products. Finally, to flood the network a node can send multiple fake queries to other nodes creating routing congestion (*Query Flooding*). These attacks are made at the application level because a malicious user needs the applicative that interacts with the P2P network or part of the tools provided from the applicative to perform them, i.e. the search engine for the resource location to perform the *Query Flooding*.

As we can clearly see, these attacks are widely connected to each other. This means that from one attack we can mount another attack simply using the effects of the first one. We present now some examples:

- ✓ **Identity Assignment → Eclipse:** To mount an Eclipse Attack a malicious user needs to control a certain number of nodes in the network. To have the control on these nodes he needs to mount an Identity Assignment attack, such as the Sybil attack.
- ✓ **Identity Assignment → Storage&Retrieval Attack:** Some P2P protocols maps newly inserted resources to nodes that are somehow closer to their identifiers. A malicious user may want to have a complete control over some of these resources, and this can be done by assigning identities of nearest nodes to the desired resources to control them. Dangers from this attack are basically two: first, a resource that was previously inserted by a trusted node is now blocked; second, a resource that was mapped in a certain area now is pretended to be in another, because the malicious node fools the other nodes that will believe this thing.
- ✓ **Table Poisoning → Eclipse:** Having the control on a portion of the network, it is possible to insert fake routing information for nearest nodes in the routing tables mechanism. This can be dangerous in the case of repeated updates to all the nodes. If a malicious user is able to send fake information in terms of contacts authenticated in the network to the other

nodes inside it, members of the latter group will update their routing tables with these fake information and, in the worst case, they can cut themselves out of the network. This will end up in an "indirect" Eclipse attack, since the node itself will not be able to send keepalive messages to its neighbors since it contains fake contacts in its routing table.

- ✓ **Eclipse → Identity Theft:** If a node (A) is controlled by a malicious user, each of its resources and information are controlled and exposed to other malicious nodes, that can claim to be A. We know that resources in P2P networks are assigned to certain areas, controlled by a node that has the closest ID to the ID of the area. If a malicious user can take the identity of a node controlling a certain area where some resources are mapped, it will become the manager of those resources being able to use them on its wish, such as masking them to the rest of the network.

As we can see from the relationships above, chains of attacks are clearly possible. For example, we can start from a Sybil attack to mount an Eclipse attack, and from this situation we can go further and mount an Identity Theft to realize a Storage&Retrieval Attack. Moreover, from all the classes of attacks shown in Figure 36 we can mount a **Denial of Service** (DoS) or its **Distributed** variant (*DDoS*) that aims at blocking some resources or to move them to other locations, simply by changing entries in routing tables of nodes in the network during the update phases. This attack can be performed by a single node (the simple DoS) or by a group of nodes (the DDoS). In the latter, the attack is made in a distributed way since each node of the group will perform a Denial of Service: for the same resource in order to contact more nodes at the same time; or for different resources to block a wider portion of the network.

It is now interesting to see how these attacks can be used, that is which weakness of each network protocol allows to mount them. We will outline some known weaknesses of P2P routing protocols to see which kind of attacks can be mounted.
Before proceeding, we need to define a particular kind of node we can find in P2P networks: the malicious node. Formally:

Definition 33 [5]: "*A **malicious node** is a node that can forge arbitrary IP packets with arbitrary contents, but is able only to examine packets addressed to itself.*"

When we will present examples on attacks mounted on P2P networks we will refer to the source of the attack as the malicious node.

3.1.1 P2P protocols weaknesses

We will now present some weaknesses of the protocols introduced in Chapter 2, and we will underline which kind of attacks can be mounted in those networks that use them.

Gnutella The Gnutella protocol has not basic security defenses against network attacks, since it bases its security on the behavior of nodes in the network, since they are supposed to act following the rules of the protocol. This assumption exposes Gnutella to a wide amount of attacks that can be used to subvert its functionalities **[73]**. For example, the Gnutella protocol is very weak

against the Denial of Service attack, since neither the Gnutella protocol or the Gnutella clients posses defenses to thwart these attacks.

Chord As we saw in Section 2.5.3, the Chord protocol has a singular way to update finger tables when a node enters the network or departs. When a new node joins, there is no guarantee that the insertion went well. Another problem is that when the insertion finishes, the predecessor of the new node starts the backward update for node inside its finger table. Nothing prevents the new node to depart after the update starts, thus for a certain amount of time the finger tables of its neighbors will contain wrong values, having wrong routing specifications for some nodes. Moreover, we know that resources are mapped in a systematical way. With a good ID Mapping or Identity Theft attacks a malicious node can take advantage of a specific resource, in order to block it. This can be a good spot from which to mount an Eclipse attack as well.

Kademlia The first thing to say is that the hash function for nodes and keys IDs is not completely robust against collisions. This means that, with a range of 160 bit for the identifiers, collisions are difficult to see but their probability is not equal to zero. Moreover, Kademlia deletes nodes from the routing tables after they do not respond to a keepalive message. This is bad, because if a node is subject to an Eclipse attack it is deleted even if it is a valid node. Moreover, since routing messages are exchanged by UDP packets without a handshake, an attacker can poison the destination routing table by simply spoofing the IP address of the source in an ongoing communication **[4]**. Since we can encounter many times the term spoofing, we will address what "to spoof" means. Formally:

Definition 34 [67]: "*The word **spoof** means to hoax, trick, or deceive. Therefore, in the IT world, spoofing refers tricking or deceiving computer systems or other computer users. This is typically done by hiding one identity or faking the identity of another user on the Internet.*"

Finally, when a node authenticates in the network and sends its ID and IP address, each node receiving some information does not control the effective validity of these information.

KAD In the KAD network the main weaknesses is the notion of node identity and authentication. This means that attacks that aim at subverting the identities of some nodes in the network are highly usable in this network, i.e. the Sybil attack. Moreover, KAD does not supply any mechanism to bind a host to its own ID **[45]**. This is a big blunder since it is a technique used by the system to avoid the high time needed by users to fill their routing tables. Since in KAD every node routing table depends on its ID, the average time to build one is around seven minutes **[41]**. Using the mechanism of persistent IDs (that do not bind hosts to IPs) and routing tables, the system allows nodes to wait a time much lower than the average. Moreover, with this solution, also NAT traversal is avoided and this saves some of the time needed for the communication. This design was inserted in order to provide a repair mechanism for KAD users that leaves the network from a certain location and joins back from another, having a different IP address for each new join. This means that once it re-joins the network, after it fixes its routing table, it communicates to the

other nodes its new location. This can be a good spot to retain more addresses in the network and to mount some Routing and Denial of Service attacks.

Application weaknesses To conclude we can make a consideration also on those applications that allow user to share files. This because not only P2P network protocols present weaknesses but also these applicative can have some problems, called bugs. Formally:

Definition 35 [58]: "*In computer science a **bug** is an unexpected error from a program that causes a malfunction. This can be a coding error and sometimes cannot be simply identified, because it is not said that the program will block after the bug action.*"

The presence of bugs in an application is a common thing, but they are potentially dangerous. If an application for file sharing is widely used and it contains bugs, each consumer that uses it identifies a potential point of failure for the network.

3.2 The attacks

In this section we will analyze in deep detail all the attacks outlined in Section 3.1, beginning from the most common Identity Assignment attacks.

3.2.1 Identity Assignment attacks

In this section we will explain how Identity Assignment attacks works and how they can be used in P2P networks.

3.2.1.1 The Sybil attack

This kind of attack aims at inserting some malicious nodes inside the network **[33]**. We can think to a Sybil as a valid node with a fake identity. To clarify what we mean when talking of valid or fake nodes we give a formal definition of these terms:

Definition 36: *We say that a node is **valid** if it acts following the rules specified by the P2P protocol and it is **fake** otherwise.*

What distinguishes valid nodes from fake, i.e. Sybils, is their identity. Formally:

Definition 37: ***Sybil nodes** are pseudonyms and do not identify a real user, while valid nodes have a valid identity that identifies a real user.*

The main goal of this attack is to subvert the voting mechanism inside the network.

Definition 38 [21]: "*A **voting mechanism**, or **reputation system**, computes a score for a set of objects and publishes it to other entities inside a certain network. The rating for an entity is given by the opinion of other nodes on the one considered. Each network can use its own reputation algorithm to give scoring.*"

Note that also P2P networks can be affected by the Sybil attack. Let us now suppose to be inside a network based on the Chord protocol. We know that finger tables are updated during query forwarding to each node. If between two valid nodes there is a malicious one, it can forward messages with fake updates or with fake information. The effect is even more dangerous when the level of churn is high, because the number of inserted and deleted nodes is high as well and the system fatigues to recognize valid and fake nodes. Moreover, a massive usage of the Sybil attack causes a high churn. The problem here is that Chord is highly vulnerable to Sybil, as Pastry **[36]**, because a malicious user can choose its own node ID via spoofing several IP addresses from the network, where this is possible. This can be made since Chord creates ID with a hash function on the IP address of the entering node.
After understanding how a Sybil attack can be mounted on a P2P network, let us see some example of its usage.

Example 3.1 – To buy or not to buy?

*Let us suppose to be in a network in which users exchange information about products they buy, i.e. features, pros and cons of the product and personal opinions **[10]**. This system is clearly a voting mechanism, even if there are no such credits exchanged by users. In this kind of system, trust is given on comments over products purchased by the users. To have a practical example, let us take as product a portable MP3 player. Usually inside those pages presenting evaluations on purchased products there are technical tables in which the features of the object are explained. Along with this, an area for comments in which people can exchange opinions on the object is usually available. Often, before being able to post a comment, a user needs to register to the service. This kind of registration does not usually ask for some important information to the users, except for their e-mail addresses or their geographical locations. A malicious user can simply register many fake users simply giving fake information and post fake comments on the objects. If someone visits the page of the MP3 player and reads the comments to evaluate the purchase or not of the object, it can be fooled by some fake comments inserted by the malicious user, being not able to recognize if they are fake or not. Moreover, some comment mechanisms allow to post comments "on the fly", i.e. without previous registration but only inserting a name and an e-mail address.*

◊

The following is an example of Sybil attack inside the KAD network used by eMule **[45]**. Before doing any evil operation, the malicious node uses a crawler to spy the zone, in order to discover nodes already inside it to contact further. Formally:

Definition 39 [60]: "*A **crawler** is a program that visits Web sites and reads their pages and other information in order to create entries for a search engine index. The major search engines on the Web all have such a program, which is also known as a **spider** or a **bot**.*"

In the case of P2P networks the crawler gather information about the nodes inside the network, such as their IDs or IP addresses. Let us now present a Sybil attack in KAD **[39]**.

Example 3.2 – Sybil attack in KAD

Let us suppose to have a certain resource inside the zone Z identified by a prefix of 8 bits with the remaining 8 bits used for identifying nodes inside Z. Now, some malicious users intend to attack Z by inserting multiple Sybils inside it. In our case the crawler spies the network to gain contacts of nodes inside the network. Knowing the nodes participating in the network it can introduce at most 2^{16} Sybils, having 8 binary bits for the zone identifier and 8 for node identifiers. Let us unveil in detail how the spy works:

- ✓ *First, the spy crawls the zone Z to know about node IDs inside the zone to not create duplicates and to discover available IDs.*
- ✓ *Then, with multiple requests, the spy sends an authenticating message to those nodes discovered in the crawling phase in order to poison their routing tables with entries that point to the Sybils. If the buckets corresponding to the Sybils are not full, entries are added properly.*
- ✓ *At this step, later in time, when one of the Sybils receives a forward request it gives back a set of Sybil nodes near the requestor if the destination is in zone Z, ignoring it otherwise.*
- ✓ *Requests are all stored for further analysis.*

◊

The aim of this Sybil attack is only to spy which kind of contents are sought and used from users inside the network. But it can be also used to make some real and greater damages, such as disrupting the network combined with other attacks such the Eclipse Attack.

3.2.1.2 The ID Mapping attack

As explained before, this attack aims at obtaining a specific ID in the network, because in DHT-based environments usually resources are mapped in some areas corresponding to the ID assigned to them by the hash function. If a node can choose its own identifier, the attack will be simple to implement **[5]**. Ideally, most of the P2P protocols introduced in Chapter 2 have an ID space with 2^{160} possible IDs so the usage of an ID Mapping Attack is not so simple. Usually however, nodes are not too much in the network and the ID space is quite empty. This means once selected a target

node, obtaining an ID closer to this target is not so difficult. The main problem here is replication that is used in some P2P protocols. In such protocols, such as Kademlia, resources are replicated to the k nearest neighbors of the initial location of the resource. Thus, a malicious node cannot simply insert itself near the location, since at least other k – 1 nodes will have information about the target resource. So, to perform a good ID Mapping attack a malicious node needs to perform also a Sybil attack. This is not so difficult, since in P2P networks anonymity is a must and, with the lack of a central authority controlling who is connecting to the network, a malicious node can create multiple identities, thus realizing the Sybil attack. With this explanation we can underline three types of attacks to mount in order to perform a good ID Mapping, explained in the following two examples **[5]**.

Example 3.3 – Total resource control

Let us suppose to be in a DHT network, i.e. Kademlia, and that a malicious node wants to take control of a certain ID in the space. Just to recall, the ID space in the Kademlia network is composed of 2^{160} entries since the length of each identifier is of 160 bits. Generally a node cannot choose its own identifier since this will mean to have serious security issues and potentially many collisions, thus selecting a specified ID in the space will be quite infeasible. Substantially, the malicious node can try to achieve its evil goal thinking that in common situations, the network is not fully populated and many entries in the space are free. This allows the malicious node to get an identifier very close to the target one, thus having the possibility to success in its attack. In DHT-based network resources are mapped with couple ***[key, value]*** *where the key identifies the ID of the node nearest to the resource, i.e. its manager. Obtaining an ID very close to the identifier of the resource means to possibly be its manager, thus having control of it. The malicious node, however, cannot take only one attempt to reach its evil goal, thus it needs to obtain more IDs near the location of the target. This can be achieved by mounting a Sybil attack by which the evil user inserts multiple nodes in the area of the target. Let us now suppose that the malicious user wants to take control of all the locations around the target ID, i.e. a resource it wants to obscure from the network, and that the ID space is composed by M_{ID} entries, with identifiers spacing from 0 to M_{ID} – 1. Let us also suppose to be interested in the case in which the malicious user achieves its goal at the first attempt. To produce a general case for this attack, we need to focalize our attention on the concept of 'nearest' in the network, since each DHT-based protocol has its own meaning of nearest. In the case of this attack the definition of nearest is taken as the node with the lowest ID that is greater than the ID of the target for the attack. Moreover, supposing that the entities in the network (resources or nodes) are uniformly distributed, choosing one or the other target produce the same results in terms of probability of success for the attack. This also means that if the variable (A) describing the success of the current attempt in the ID choosing mechanism is uniformly distributed, the probability that an ID in the space is chosen is equal to:*

$$p(ID_r) = \frac{1}{M_{ID}} \tag{3.1}$$

where ID_r is the currently chosen random ID and M_{ID} is the maximum number of IDs in the space. Since there are some positions set free in the space, the number of nodes currently existing in the network is set to $M_c < M_{ID}$. The malicious user needs to select an ID inside the space, thus comprised in the interval [0, $M_{ID} - 1$], greater or at most equal to the target one. If the target ID is set equal to b, the probability to choose this kind of ID is set to:

$$P(B \geq b) = (1 - \frac{b}{M_{ID}})^{M_c} \tag{3.2}$$

In this case the variable describing this kind of operation, namely (B), is a discrete random variable and its distribution is like the following:

$$p(b) = \left(1 - \frac{b}{M_{ID}}\right)^{M_c} - \left(1 - \frac{b+1}{M_{ID}}\right)^{M_c} \tag{3.3}$$

To succeed its attack, the malicious user needs to take the ID immediately lower than the limit outlined by variable B. This means to have variable A lower than variable B, i.e. A < B, thus having a distribution like the following:

$$S = \sum_{T=0}^{M_{ID}-1} \sum_{a=0}^{b-1} \frac{p(b)}{M_{ID}} = \frac{1}{M_{ID}} \sum_{T=0}^{M_{ID}-1} \left(\frac{b}{M_{ID}}\right)^{M_c} \tag{3.4}$$

where S represents P(A < B). This is because variables A and B are independent, thus their probability have to be multiplied to retrieve the success probability S. The difficulty here is in the process to calculate Formula 3.4, since that value is very high to calculate. With a more accurate look, anyway, we can notice that Formula 3.4 is not influenced by the parameter M_{ID} since its value is very high (equal to 2^{160}), so we can translate Formula 3.4 into the following:

$$S \sim \frac{1}{M_c + 1} \tag{3.5}$$

The speech just made have to be contextualized in a space where IDs are continuous, since it will be very unfeasible to compute such a probability in a discrete environment, since the hash function maps an ID to a specific position and this function is not surjective or injective. In poor words, selecting a random ID with a fixed value is very difficult in discrete environments since the distribution of values is not clearly known. In the continuous case this probability is still difficult to calculate but trying to get an ID near the target one is less difficult, thus is a definition that can be used in this case of environments. Knowing this, the variable B possesses a density and a cumulative distribution equal to the following:

$$f(b) = \frac{d}{db} F(b) = \frac{M_c}{M_{ID}} * (1 - \frac{b}{M_{ID}})^{M_c - 1} \tag{3.6}$$

$$F(b) = P(B \leq b) = 1 - P(B \geq b) \tag{3.7}$$

With these new formulas the probability S of success is translated as well, and it is like the following:

$$S = \int_0^{M_{ID}} \frac{1}{M_{ID}} da \int_0^{M_{ID}} \frac{M_c}{M_{ID}} (1 - \frac{b}{M_{ID}})^{M_c - 1} dT \tag{3.8}$$

This time the probability S can be simplified exactly to the ratio between one and the number of currently active nodes plus one. This is important since we can rewrite the probability success only respect to the number of currently active nodes.
Until now we referred to this attack considering only the case in which the malicious users achieve its goal using only one attempt. In real cases more attempts are needed in order to have the desired ID in the space. Considering this case means to consider a geometric distribution depending on the value of the probability of success S. In our case the malicious user wants to control all those nodes that posses relevant information on the target ID, i.e. its k nearest neighbors. So, the expected value of the rate of success (R) for this attack is like the following:

$$EV(R) = \frac{k}{S} \sim k * (M_c + 1) \tag{3.9}$$

Studying the relevance of this attack and its computational requirements we can state that the only two parameters that influence this attack are the rate of replication for resources in the network, i.e. the parameter k, and the number of currently active nodes in the network, i.e. the parameter M_c. It is important to notice that the replication parameter increases the probability of success of this attack.

◊

Example 3.4 – Attacking more resources

In Example 3.3 we presented the case in which the malicious node wants to take control of all the nodes that knows the position of a determined resource. Let us suppose that this time the malicious node is not interested in taking control of all the k nodes that has the information on the location of resource (T), but that it is only interested in controlling only a part of this sample. The amount of nodes selected during the evil operation is fixed to a ratio of the original k nodes, identified by the value of a parameter F that varies between zero and one. The amount of desired nodes is thus:

$$N = (1 - F) * k \tag{3.10}$$

This time the malicious node reaches its goal in selecting the correct node to control if no more than N nodes active in the network possess an ID lower than the one generated. This event is described by a discrete random variable (C) while the probability to create and ID greater than a generic value (c) is equal to:

$$P(C > c) = \sum_{j=0}^{N} \binom{M_c}{j} * \left(\frac{c}{M_{ID}}\right)^j * \left(1 - \frac{c}{M_{ID}}\right)^{M_c - j} \tag{3.11}$$

where M_c is the number of currently active nodes and M_{ID} is the maximum number of nodes that can be inserted in the network. We can then generalize the distribution of variable C with the following:

$$p(c) = \sum_{j=0}^{N} \binom{M_c}{j} * \left[\left(\frac{c}{M_{ID}}\right)^j * \left(1 - \frac{c}{M_{ID}}\right)^{M_c - j} - \left(\frac{c+1}{M_{ID}}\right)^j * \left(1 - \frac{c+1}{M_{ID}}\right)^{M_c - j}\right] \tag{3.12}$$

because setting the value of N to zero we obtain the exact value of Formula 3.3 since we will describe the event in which the malicious node wants to control all the nodes that knows about T. We can also translate Formula 3.4 in the actual case, since a speech similar to the previous can be made also in this case for the parameter N. The Formula 3.4 can be translated into the following:

$$S = \sum_{c=0}^{M_{ID}-1} \sum_{j=0}^{c-1} \frac{1}{M} * p(c) = \frac{1}{M_{ID}} \sum_{j=0}^{N} \binom{M_c}{j} * \sum_{y=0}^{M_{ID}-1} \left[\left(\frac{c}{M_{ID}} \right)^{M_c - j} * \left(1 - \frac{c}{M_{ID}} \right)^{j} \right] \quad (3.13)$$

Even in this case the value of S does not depend from the maximum number of nodes that can be inserted in the network, but from the number of active nodes and from the number N. Formally:

$$S \sim \frac{N+1}{M_c + 1} \quad (3.14)$$

Clearly the value expressed by Formula 3.14 is higher than that expressed in Formula 3.5 since the latter is increased by a constant with value N + 1. This means that the attacker has a higher probability of success in this case respect to the case of Example 3.3. Moreover, the number of attempts needed to the attacker to reach its final goal is diminished since the total number of attempts to have an ID that follows the distribution of variable C now depends also from factor N. Formally:

$$EV(R) \sim \frac{k-N}{N+1} * (M_c + 1) \quad (3.15)$$

To have a real example for this attack (and also for that presented in Example 3.3) if we assume k = 20 as in default cases for the Karelia protocol, and F = 0.6, the attacker only needs to attempt:

$$T_A = \frac{8}{13} * (M_c + 1) \quad (3.16)$$

*tries respect to Formula 3.15, while this number grows to 20 * (M_c + 1) for the case of Example 3.3.*

◊

3.2.2 Routing attacks

In this section we will introduce those attacks that aim at blocking the routing mechanism behind a P2P protocol. We will give a definition and some examples of these attacks starting from the natural evolution of the Sybil attack: the Eclipse Attack.

3.2.2.1 The Eclipse attack

As we saw in Section 3.1, the main goal of this attack is to block or make unavailable a certain resource or node by inserting multiple nodes in the target area **[38]**. To mount this attack, a malicious node needs to have preliminary knowledge about the environment around the node or resource it wants to block. This can be achieved by inserting some malicious nodes using the Sybil attack and use these Sybils to collect information about the area where the target resides, i.e. its neighbors or the number of nodes inside the network. When the malicious node knows its

neighborhood it contacts its neighbors sending them its authentication credentials. The neighbors register the malicious node in their routing tables and thus the node will be contacted once a resource is queried. The important thing to notice is that malicious nodes aim at blocking a certain entity in the network. When they acknowledge the position of a determined resource, they can force the insertion of multiple Sybils in the target area being able to potentially block it. This can be done by refusing to forward correctly a query that is directed to the blocked resource, i.e. dropping it or simply forwarding it to a location very far from the real destination. The same can be done for a single node instead of a resource.
As we can think, the resource or the target node for this attack is *eclipsed* (from here the name of the attack) from the network, and it will be unavailable. Moreover, even if defenses against a Sybil attack are mounted, the Eclipse attack is still feasible. Let us take a valid node inside the network and suppose that when the update of the routing tables starts this node acts as a malicious one and sends fake tables to other nodes. If this action is not blocked, and many nodes act like this one, then the network is widely subject to the Eclipse attack (see Figure 37).

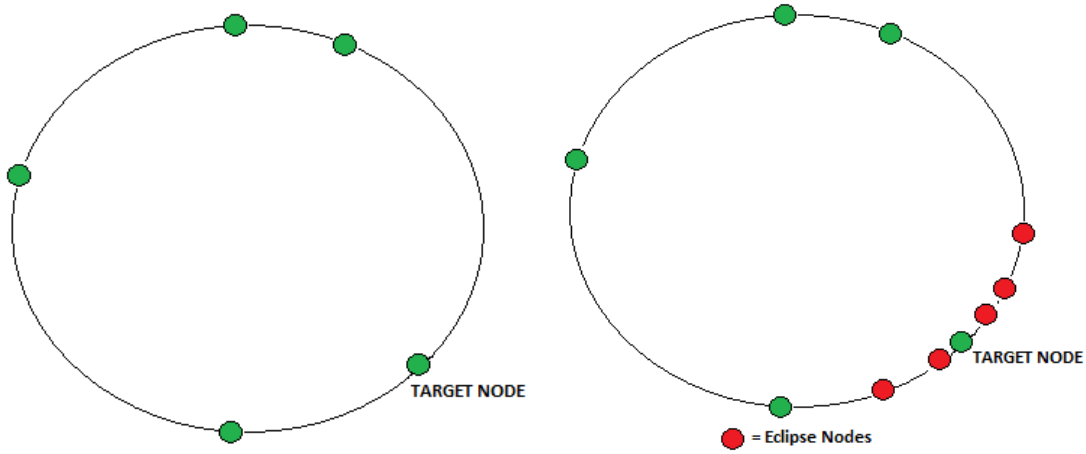

Figure 37 An ongoing Eclipse Attack.

This attack gains its strength to a lack of control in unstructured networks that usually do not provide restrictions in neighbor selections. Usually, when a node connects to the network asks to a bootstrap node its ID and starts to build its routing table. We can have at least two cases of Eclipse success: the first is when the bootstrap node is evil and gives fake information to the new node, and the second is when a malicious node is inside the path used by the bootstrap node to build a routing path for the newcomer. In both cases the new node has fake information on the network. In structured networks the success is more difficult because they maintain a stricter and clear graphical organization than unstructured networks, to success in finding a point in space with a finite number of hops. The main vulnerability is the way in which a node chooses its neighbors, since in most structured networks the choice of neighbors is made via a certain calculation of distance between nodes. Basically, neighbors are chosen by prefix similarity and this opens a wide road for Eclipse attacks to be mounted.

Example 3.5 – Eclipsing the KAD protocol

*Before going into details of the attack on the KAD protocol **[39]**, we need to give preliminary assumptions on the environment where the Eclipse Attack will be mounted. First of all, we need to say that before mounting the below attack some Sybils need to be inserted into the network. From the attacker side, we need to know that it can control only end-systems and no corruption or misrouting on IP layer is used. Furthermore, the cost of attacks for the malicious node is only in terms of bandwidth consumption, while it has enough computational and storage resources to process messages and store states from the network. We will suppose to have two main phases in the attack: the first is preparatory where the malicious node spies the network to know about nodes inside it and to send them fake routing entries in order to force them to contact other malicious nodes, i.e. previously inserted Sybils. We also specify a final purpose for the attacker. He knows that in a specific part of the network there is a resource of critical importance, and he wants to block all the contacting nodes from reaching it.*

*The attacker possesses a number of H hosts identified by an index h: {0, ..., H – 1}. These hosts respect the rules for a P2P network since they are equal in computational power and bandwidth availability. To produce a routing table for each host, the attacker creates a set containing the triplets for each malicious host, composed by **(h, IP_h, $port\_number_h$)** and this table is copied to all the H hosts. Let us now suppose that the network creates ID composed of 128 bits, thus the number of nodes in the network are at most 2^{128}. To gain an ID inside this space, the malicious user needs to create H IDs, with the following formula:*

$$ID_h = \frac{2^{128}}{H} \tag{3.17}$$

in order to introduce each of the H malicious hosts in a different portion of the network, to not have two malicious nodes in the same portion. After each node gets its own ID, labeled as H_i, the creation of the neighbor maps is started and each node queries its neighbors to have their contact and stores them. At this point a problem arises. If only neighbors of the current Hi malicious node are polluted with fake routing table entries, they can simply get valid contacts by their own neighbors. For this reason, the malicious node needs also to pollute some entries in the routing tables of its neighbors. The malicious node sends an update request to its neighbors pretending to be one of their entries in their routing tables, thus the neighbors update the reference. This can be done since in KAD network there are no authentication controls, thus a node can present itself with a fake ID to the others. Moreover, the malicious node needs also to pollute the routing table entries of those nodes known by its neighbors. This can be done by receiving the contacts in the routing tables of its neighbors and sending the same fake request explained before to them. This operation must be done since the contacts in the routing tables of the neighbors of the malicious node can contact its neighbors during the update phase.

Now, everything is ready to start the attack. We notice that if a malicious node simply drops a query or does not forward some of them, it will be deleted from other node tables and this is bad because it would not be part of the network anymore, thus it cannot perform attacks. Moreover,

the Kademlia network, from which KAD is derived, is robust against query dropping, since each message is sent through α paths (α is usually equal to 3) ***[45]****. But, looking in deep detail, we know that Kademlia, and KAD as well, is weak in terms of choice of the next hop. In fact the closest node is chosen to forward a request. This means that whatever node that can inserts a fake ID in the network, will intercept a query to a specific node, and this is the case. One of the H malicious nodes can intercept a request claiming to be the destination node, or it alternatively gives back a set of α nodes near the destination. Now we can see what happens in different sceneries for this attack. In KAD the routing protocol for queries terminates when a query submitted receives 300 or more responses to its set of keywords. When a malicious node receives a query, it can send back to its sender a set of more than 300 fake nodes making the request forwarding to stop. The sender does not know that the responses are fakes, and thus thinks it has reached the destination. However, recent versions of the software that use the KAD protocol, e.g. versions of eMule greater than the 0.47a, are strong against this kind of attacks, thus the only way to perform such an attack will be to try a* ***Dictionary Attack****, even if the size of the dictionary has to be very consistent because the attacker needs to reverse the hash value of the requested resource, as in Example 3.4 for the ID Mapping attack. The main problem for this attack is timing: in fact, the malicious node needs to answer to the requestor before the root node maintaining the resource. This can be achieved by the attacker simply knowing the value of the resource key transported by the query. We know that KAD reasons in terms of XOR to calculate nodes proximity. This is made as well for keys and query are spread in a certain interval δ (the tolerance zone presented in Section 2.5.7) depending from the value of the key. The malicious node needs only to solve the value of this interval and place some malicious nodes inside this interval.*
The result is given by:

$$XOR(ID_R,\ K) < \delta \tag{3.18}$$

where ID_R is the ID of a node that stores the information for the current resource. The only thing remaining is to set the entry value for this resource such as it matches [h, Ip_h, $port\_number_h$] where h is the ID of the malicious entity calculated by K mod H. So, the real resource is never reached and the real keeper does not receive requests and can be totally cut out from the network indirectly.

The attack just proposed is a good point of start for Identity Theft attacks and Index Poisoning attacks above all.

◊

3.2.2.2 The Wrong Forward attack

This kind of attack is a special case of routing attacks where a malicious node inserted into the network acts as a normal node. When it receives a request, it forwards the request to a different destination than the one specified by the sender. In practice it forwards the message to a destination farer than the normal one. This can lead to loops in the communication or even to heavy network clogging.

Example 3.6 – Far away routing

Let us suppose that a valid user starts a query to find a specific resource. Let us also suppose to have a malicious node interacting inside the network, whose path is between the source of the query and its destination. The query is forwarded correctly along the network until it reaches the malicious node. At this point, the node acts to block the correct forwarding of the query by not forwarding it to the nearest bin to the destination, but to another random bin. The malicious node aims at sending the query as far as possible from the right destination, in order to increase the path of the query.

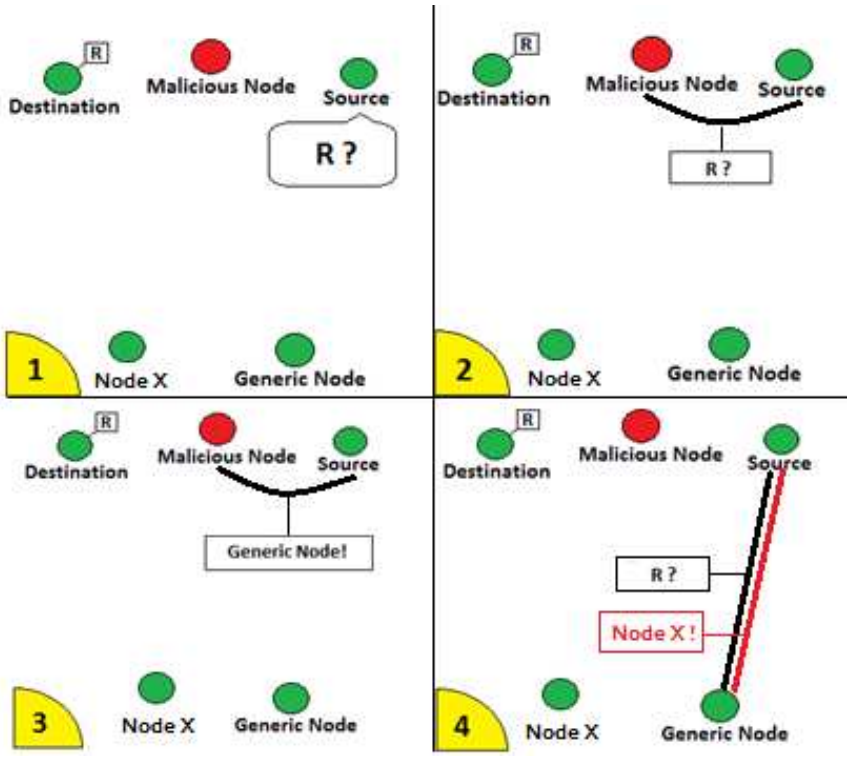

Figure 38 The evolution of a Wrong Forward attack.

Let us suppose to have a starting situation like the one represented in Figure 38 part 1, and that node Source is searching resource R that is managed by node Destination. Node Source sends a query requesting R to one of its neighbors, determined by the routing protocol behind the network that is the Malicious Node (see Figure 38 part 2). The latter act as malicious and forwards the query to a node very far from the node Destination, and let us suppose that this choice is node Generic Node (see Figure 38 part 3). At this point, the node Source will contact Generic Node to retrieve R or a contact that knows about it (see Figure 38 part 4). Let us now suppose that Generic Node does not know about Destination but knows about Node X that is the nearest node to R it knows. Source will then contact Node X and the process goes on, until it reaches Destination.

◊

The evolution of this attack could take the entire network to fall in loops or queries forwarded for a long time reaching no point of success. More, if the malicious user can operate with other

malicious nodes, this attack can be combined with Poisoning Attacks and Identity Theft, because the first malicious node could forward the query to a second malicious node that claims to be the root of the resource without having it.

3.2.2.3 The Table Poisoning attack

We know that in a P2P network where Dynamic Hash Tables are used, nodes are mapped in a certain space with IDs created by a hash function. The main goal of the Table Poisoning Attack is to make unavailable a certain node by spreading fake information in the network to other nodes. In practice, a malicious node could send to its neighbors a fake routing table entry containing its ID and a combination of IP and port number of another host, claiming to be the latter. Those nodes that receive this information do not control the validity of the combination neither the existence of the node in the network, since anonymity is provided with P2P protocols. They simply add this entry to their table and contact the node whenever they need it. The result is that, if the node is inside the network it is flooded with network requests and it is unable to answer or the communication goes into a loop. If it is not inside the network, it cannot compile requests and it can fall in an error state or reply with an error message. If we enlarge the case to hundreds of requests we can clearly state how a real case of attack can be dangerous. This attack is sometimes prevented by deleting those nodes that answers bad or does not answer to some requests. This countermeasure is also useful to estimate the success of the attack in terms of amplitude.

Example 3.7 – Poisoning Tables

A possible scenery for the Poisoning attack is the one in which malicious nodes subvert values inside routing tables. If we think to the KAD case, we can simply try to figure an attack for Table Poisoning. Once the attacker has crawled the network, it inserts itself inside the network and then starts poisoning it. This means that it contacts all its neighbors, crawled in the preparation phase, and sends them fake information. Just to think, an attacker may want to block the forwarding to a certain destination. He receives from this destination the IP address and port number by simply contacting it. After this, he can take its ID having the IP or simply by the weakness of KAD that does not restrict this thing. Then, it sends back to its neighborhood the new entry composed by the ID of the real node, and an IP and port number different from the real one, that can be an host outside or inside the network. In the second case, it may want to redirect the traffic through other malicious nodes just to control it or just to disrupt the network. After this, all nodes in the neighborhood start updating with their own neighborhood, making the poisoning more effective. Combined with an Eclipse attack, where malicious nodes block requests coming from good entities to poisoned ones, this can be a good point from which to start a DoS attack.
As we saw, the poisoning attack is really simple to implement, and it is more effective having Sybils inside the network or an Eclipse attack already mounted. We say this because we will see that countermeasures against Sybil are widely used but Eclipse attack is still feasible over these countermeasures.

◊

Example 3.8 – Poisoning Gnutella

Let us suppose to be in the situation represented in Figure 39, where a malicious node enters the network and wants to build its routing table in a Gnutella network. As saw in Chapter 2, a node acknowledges the existence of other nodes by sending a ping message at which each receiving host replies with a pong, that contains information on the current node. Among these information, there are IP addresses and port numbers.

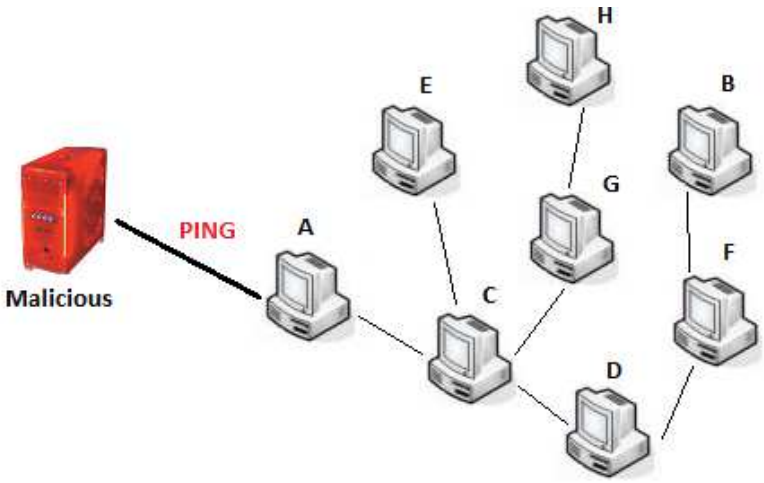

Figure 39 Starting situation for Table Poisoning.

Let us also suppose that the TTL associated to the Ping message from the malicious node is equal to 4, so it will be forwarded to node A, C, D, E, F, G and H. After this, the malicious node will receive back pong messages from A, C, D, E, F, G and H and stores them inside its routing table along with the number of hops. Initially the remaining field, such as the number of files and dimensions exchanged, will be empty. Let us now suppose that a new node joins the network, near to the malicious one, and sends the ping message. Let us also suppose that the ping has a TTL of 4 as well. The situation is now the one in Figure 40.

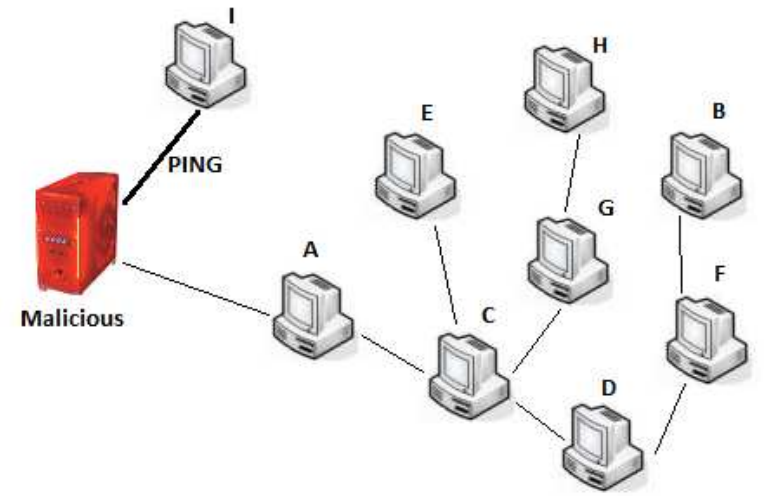

Figure 40 Evolution of the Table Poisoning.

Now, the malicious node that wants to perform a Table poisoning, sends back a fake pong message, having as arguments the IP address, port, etc., of node F, for example. Node I that receives the pong from the malicious node stores the information inserted in the pong, that are all fakes, not knowing that all of them identifies node F instead. Is important to notice that, since the TTL of the message sent by node I is equal to 4, it would not receive the true information of F, since it is at a distance 5. From this situation, a Denial of Service or an Eclipse attack can be mounted.

◊

3.2.2.4 The Spam Attack

The concept of spam is sometimes misbehaved in the Internet, since all the processes that involve the sending of a high amount of messages between two endpoints are considered as spam. We instead define it as a massive amount of messages sent along the network to some users with commercial (or non) purposes. If we think about it, we think first to those e-mails offering new products by some agencies. This kind of problem is an actual one today and every private or public institution could use this method simply managing a mail server. Protections against these techniques are widely proposed by some producers to mail servers and ISPs. In a P2P network a malicious node inside it could send many messages to several destinations, varying from queries to simple updating messages, thus it is realizing a Spam attack **[9, 73]**.

Example 3.9 – Spamming in Gnutella

Let us suppose to be in a Gnutella network, where a certain number of nodes are active and among them, one is a malicious node that wants to perform a Spam attack. In this example we will start with the situation represented in Figure 41, where there are five active nodes plus a malicious node.

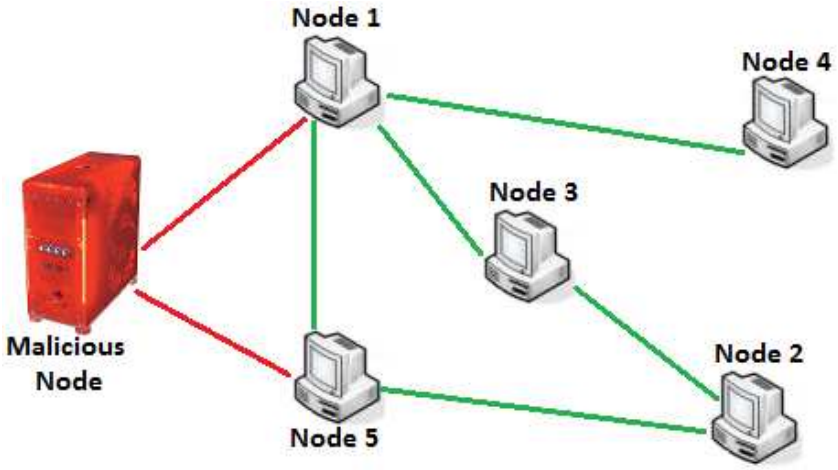

Figure 41 Spamming attack in Gnutella.

After a while, Node 3 starts a query by demanding all those contents that match the keyword 'soccer'. Node 3 sends the query to Node 2 and Node 1, since the mechanism is query flooding because in unstructured network there are no hints on where files reside. Node 2 will propagate

the query to Node 5 while Node 1 will propagate it to Node 4 and to Malicious Node. At this step, the Malicious Node could use the Spam attack. This means that it can say to know where the resource resides, by answering the query with the query hit message reporting the IP address of the node that is sharing the specified resource (Figure 42).

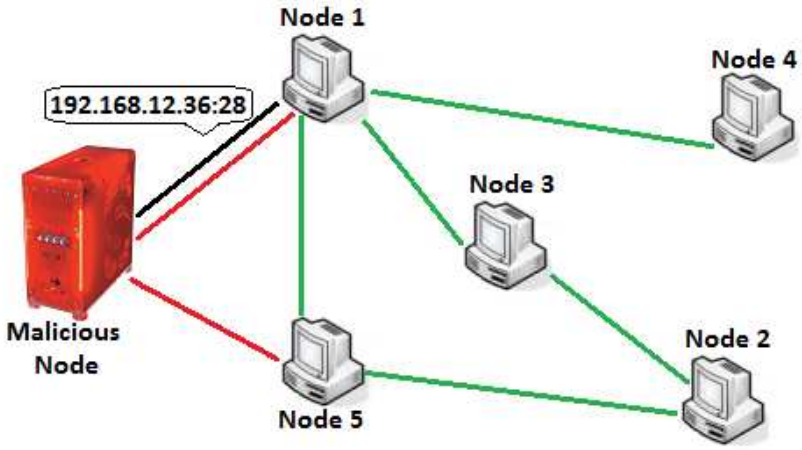

Figure 42 Evolution of the Spamming Attack in Gnutella.

Now, Node 1 will send back the query hit to Node 3 that will further contact the node passed by the Malicious Node. After a while, also Node 5 will contact the Malicious Node, and it will be answered with the same fake contact as for Node 1. The result of the attack is poor in the case of only one query, since only Node 3 will contact the fake destination in the process. The true success of this attack will be in a large scale network with hundreds of nodes inside it. Let us suppose to have this kind of network and that there are at least one thousand nodes contacting the Malicious Node for the resource. This will translate into a wide spread of fake information in large scale and, more than this, nodes that receives the fake contact from the Malicious Node will provide it to other nodes searching for the same resource. This attack can also be translated in a Denial of Service or an Eclipse Attack, since the resource is somehow blocked and the fake contact will be loaded enough of requests to disrupt its service.

◊

3.2.3 Application attacks

In this section are presented attacks that interact on the application level, i.e. using functionalities supplied by P2P applications. We will present the most used attacks of this group starting from the Index Poisoning Attack.

3.2.3.1 The Index Poisoning attack

When we talk about indexes in P2P networks we need to think at correspondences between resources and locations. Briefly, these correspondences are made between the hash of the

resource and the IP address and port number of the node or entity that manages it. In other P2P systems, such as **Voice over IP** (acronym *VoIP*), users are mapped within geographic areas. In many of today system indexes are distributed, and the resource is maintained by several nodes in a complete way or subdivided in some parts. If a malicious user wants to disrupt this mechanism needs only to fake the system, sending fake correspondence between resources and hosts managing them. This practice is called Index Poisoning and aims at directing the traffic to a resource through an invalid destination. Simply, a malicious user could envelope an information on a resource containing the **[ID, IP, port number]** triplet of a fake host and pollute the network with it. The result is that each node that receives the contact inserts it in its table, not controlling the validity of it because it would mean to flood the network with control messages. The only thing the attacker has to know is the hash of the file that is simple to retrieve observing requests on the network. After the introduction of the hash, a node could ask to the fake resource keeper to have the resource, opening a TCP connection to it. When the fake host receives the request it could act in several ways: not responding, closing the connection or maintaining it open. The scenery is more critic if we add the fact that requests are replied after a short time and that nodes querying are in the order of hundreds **[20]**.

A similar way to attempt this attack is to use a kind of host called **zombies** that are hosts without a determined role in the network that send TCP SYN requests to some other hosts. This kind of connection implies the authentication from the sender, i.e. the zombie, and then the handshake between the parts involved in the communication. If the zombie closes the connection before completing the request, the receiver maintains the connection open waiting for response. If we multiply these requests by hundreds of times we got the desired effect. This, however, is a usage proper for DoS attacks. Sometimes this kind of attack is used also by producers to block illegal diffusion of copyrighted material in the network. An attack of this kind is called **Content Pollution** since the resource name is the correct one but the content inside the resource is a fake. This is commonly used by producers because they aim at discourage P2P network users to download illegal content by sending multiple fakes in the network. An example of poisoning attack is explained in the following.

Example 3.10 – An example of Index Poisoning

As we know, Index poisoning techniques aim at blocking the availability of determined resources or files inside the network. We also know that files are recorded in a distributed way and this means that each node is responsible of managing references to where files and resources are kept. Let us recall the example made with the Eclipse attack. An attacker inserts itself and more fake nodes inside the network with the final goal to block a certain resource. The proceeding could be in two ways: the first by blocking all queries going to the specified destination, i.e. with the key hash value, and the second by blocking only those queries containing specific keywords. The purposes are different. With the first we block all the queries directed to the attacked resource while with the second we block only a certain number of queries, but some are still forwarded. The second, in addiction, is stealthier than the first because a user can think to its interaction as a momentarily technical problem. Combining the two leads to a more dangerous attack: blocking the keyword search engine. In practice the node will block all the possible couple of keywords associated to a

certain file in order to disrupt the entire search engine. When the attacker crawls the network, he learn some notions on queries submitted by taking keywords used in them to make a list of most used queries and produce couples of keywords and resources. At this step, when it connects to the network, it inserts references to couples of keywords and files previously created in high quantity, and refresh them at a constant rate. The insertion is simply made by giving to neighbors the triplet ***[i, IP$_i$, port$_i$]*** *where i is calculated always with K mod n.*

To clarify what explained until now we can look at Figure 43, where the node S starts a query with which it wants to find resource R in the network by specifying a set of keywords for the search. In networks that use KAD, or Kademlia-like networks, keywords are hashed and these hashes points to the location of a resource characterized by the inserted keyword. Let us suppose to be in the case that the malicious node wants to block all requests for resources responding to the keyword soccer. Let us also suppose that node S sends a query with the keyword soccer.

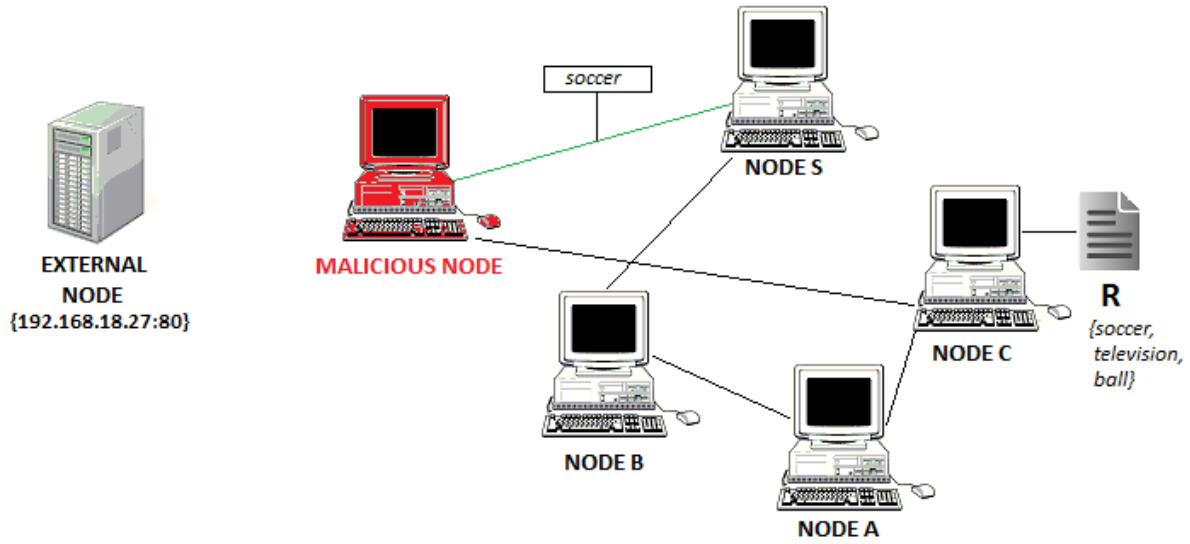

Figure 43 Starting situation for the Index Poisoning Attack.

Now, the malicious node recognizes the keyword soccer that is one of the hot keywords, i.e. one inserted in the ensemble to block. At this point, it performs its evil action, by giving back not the address of node C, maintaining the resource with keyword soccer, but the address of External Node, that is 192.168.1.27:80 (see Figure 44).

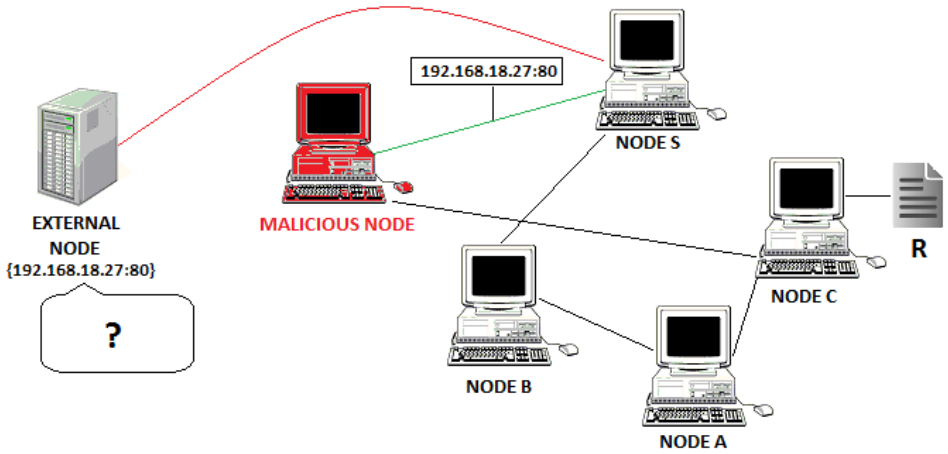

Figure 44 Final situation of the Index Poisoning Attack.

After receiving the contact of External Node, S contacts it. External Node, being whatever node not in the P2P network, receives the request but it cannot satisfy it. The reason is because the request can be not understandable by the node, or because it comes from a network different from its own, etc. At this step, node S is not able to locate the resource R with keyword soccer. The attack seems not so bad with only one host requesting the resource. But if we enlarge the number of requests to ten thousands and the fact that each request can be repeated more times at fixed intervals of five seconds, we can make the calculation on the number of requests that are moved away from the correct resource.

◊

3.2.3.2 The Query Flooding attack

Another kind of attack where the load in the network is taken to the limit is the Query Flooding. Actually, this attack is performed by a malicious node that starts a search inside the network. Doing this, it specifies a valid entry for a resource and it requests this to its neighbors. When the query is forwarded, it waits a certain period of time and sends it again. The proceeding goes on for several minutes, in order to have a massive amount of messages running through the network. If a set of queries are forwarded for a different set of resources, than the network is widely clogged and a bigger number of nodes are touched. This attack is usually made to block functionalities in the network and occupy resources such as bandwidth and availability of nodes to compute requests. The most important thing is that also valid nodes could operate involuntarily this kind of attack simply requesting a resource multiple times, thus its discovering is quite difficult. Moreover, this attack is commonly used in unstructured networks where queries are forwarded to all neighbors because the source does not know where a specific resource resides. With the following example we analyze a possible scenario for this attack.

Example 3.11 – Flooding the network

Let us suppose to be in the situation shown in Figure 45. Node A is our malicious node that wants to act evil. In practice, node A will start sending query in the network at a specified interval of time,

every time the same or every time a different query. Let us also suppose that query are forwarded to the first three nearest node to destination location. At each step, so, three more nodes are contacted from the query forwarding. Let us finally suppose that A point a destination very far from him, i.e. node W, in order to propagate the query as far as possible.

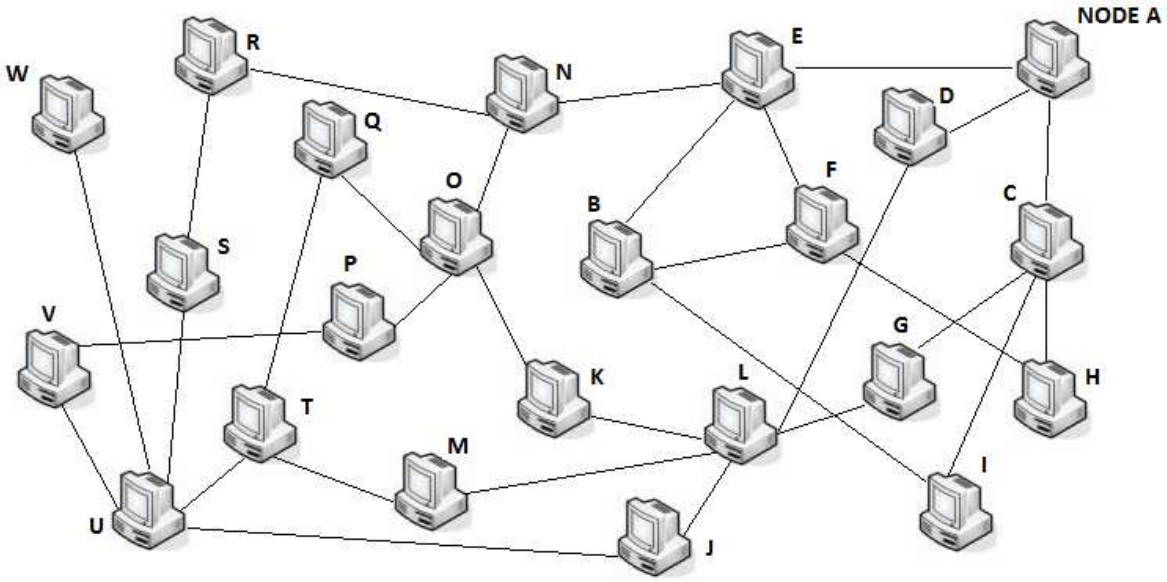

Figure 45 A possible scenario for the Query Flooding.

Let us consider the first step in which node A forwards the three queries to nodes C, D and E. These three nodes look in their routing tables and forward the message to nodes G, H, I, L, F and B. At this point A sends another query for node Q, and so on. At a certain point, the number of message will be high in the network, constituted by query messages, thus A continues to send them and they are being forwarded, and query hit messages. In the example only 23 nodes are available, but if we think to a network composed by one million nodes with about a tenth acting as node A in this example, we can quickly have a clogged network.

◊

3.2.4 The (D)DoS attack

In this chapter we directly or indirectly encountered the Denial of Service attack **[16, 29]**. More precisely, we saw some starting point from which to mount a good DoS attack in a P2P network. Just to recall, the primary goal of a Denial of Service attack is to disrupt a service, that is making unavailable for a certain amount of time a specified source of service. Preferred targets of this attack are the upload and the download bandwidth of nodes, and it is usually mounted after some Index Poisoning or Table Poisoning attacks because the attacker needs to poison routing tables references in order to include the target host inside the routing tables of nodes inside the network. The target host can be or not be part of the network: in the latter case the discovering of a Denial of Service is simple, since the target node will be discovered, but usually the Denial of Service is mounted with targets outside the network. This attack, differently from others, can be mounted in a distributed way. This variant involves more malicious nodes in the spreading of fake references to target hosts and it is the most used variation of the Denial of Service. We already

saw a good point of start for these kinds of attacks, by mounting a Sybil attack followed by an Eclipse and some Table or Index Poisoning or an Identity Theft attacks. Even if this seems a lot of work to do, if we manage to reach the second step where the Eclipse attack is mounted and working, we will be able to run a Denial of Service in no time. Let us see two examples of this attack, one concerning resource blocking and one applied in the KAD network.

Example 3.12 - Fake Resourcing

Let us suppose to a have a certain set of malicious nodes inside the network. The insertion is made in the way we exposed for the Sybil and Eclipse attacks. Now, a malicious node wants to crash a certain server, inside or outside the P2P network. It takes the IP address and port of this server and starts its attack. First of all, it has to find some popular resources to use as mirrors for its attack, in order to operate an Index Poisoning. After crawling the network in the preparative step, he knows about a certain file exchanged in a massive way along the network. It takes the ID of this file and starts sending triplet containing [ID_f, IP_N, $port_N$] where f is the file and N is the node who is supposed to manage f. Neighbors who receives this entry do not contact N to be sure it is online or inside the network or if it really maintains f, simply add the entry on their tables to use it when necessary. Now, Let us suppose a query is started somewhere in the network and it points to the attacked file. Once the query reaches the poisoned neighborhood, it is forwarded to node N that is supposed to maintain the requested file. At this point node N can compute the request and respond in several ways:

- ✓ *First it can simply drop the request, because it comes from an unknown network or because it is not up for those kinds of requests.*
- ✓ *Second, it can return an error message because it is not able to answer the query.*
- ✓ *Third, it can terminate the connection.*

In all of this three cases, a TCP connection is open with node N and it can stay active in case two, because node N waits for the requesting node to send back another request, since in most P2P protocols, requests can be repeated in time for a certain number of times and at specified intervals. If we multiply this request for about twenty times with interval of one second between them, we can think the problem is not so critical. In fact the server has only to compute twenty requests in about twenty seconds. But if we enlarge this problem to a set of ten thousands requests, we can clearly state that node N can be highly clogged. The server receiving the requests opens ten thousands connections to some nodes only to send back error messages or not being able to respond. These occupy the server that is no more able to do its job. If we add also the fact that the server can be inducted into routing tables to be contacted during query forwarding, the damage is even bigger. In fact, with this action the attacker adds a block inside the query forwarding mechanism because the server does not forward it correctly not knowing what to do.

Another possible scenery is the one where the attacked node is inside the network. This changes the final because the node, after receiving multiple requests, may crash and its neighbors can delete it from their routing tables. If we think about it, in those networks where resources replication is not made, this means losing all the resources managed by the crashed node. In both

cases, the service is disrupted and is not always so simple to block a Denial of Service in these terms. Moreover, to use DoS we need to perform an Index or Table poisoning. The next example will introduce such an attack on Overnet.

◊

Example 3.13 - DoS in eMule

In this example, Index Poisoning and Table Poisoning are used in order to disrupt the network and to clog it as well. The overlay network is one like Overnet exposed when we talked about eDonkey. Two simply sceneries are outlined: one using Index Poisoning to pollute the network references to resources and Routing Poisoning combined with the previous. In both attacks the node is supposed to crawl the network to gain IP addresses and port numbers of nodes connected to the network. For the first attack, let us suppose the malicious node wants to insert some fake references for a resource, make them point to a certain node Z not necessarily inside the network. After it has entered the network, it contacts its neighbors by sending them an authentication message via UDP packets including the IP and port number of node Z, attaching to them a hash for a certain resource in the network. As for Kademlia and KAD, from which Overnet inherits its behavior, the nodes receiving the UDP packet does not check if the node residing to the given IP address and port is a correct one or if it maintains the specified resource. After the entry is stored in neighbors tables, when some nodes search the specified resource they will give back as response the address of node Z. As we saw in the brief introduction, the node Z is then flooded with requests and TCP connections with the result of being unable to stay online anymore.

Another way to pollute the network is the one which inserts in neighbors tables a fake reference to a node, making other believes it is inside the network. To do this, the malicious node sends via UDP packet a reference to node Z, the target for the attack, composed by the ID, the IP and the port number. Neighbors that receive the contacts control if the node is near them to insert it inside their tables. If it is, it is inserted and contacted whenever a query has to pass through it for closeness. This means that the node, being not able to compute the query, has to drop it or going in an error state, after as always opening a TCP connection with the sender. If multiple requests are issued to it, it can go into a crash state.

◊

Let us now focus on some evolutions of the attacks seen in the previous section and how they evolve with a Distributed Denial of Service application. Considering Example 3.10, in which a malicious node forwards a query to a node outside the network, we can develop a Denial of Service. First of all, when the node communicates to the source node that the searched element is at the fake address, Node S stores this information and tries to contact External Node. Let us suppose that during the second operation another node sends a query to Node S with the keyword *soccer*. Node S will respond with the address of External Node, since Node S thinks it has the right resource to share. If we think at a spreading in large scale of the wrong information we will have many nodes trying to contact massively the External Node with the final result of

blocking the External Node to work. A possible snapshot in which all the nodes, except for Node C that has resource R, have a wrong reference to External Node is represented in Figure 46.

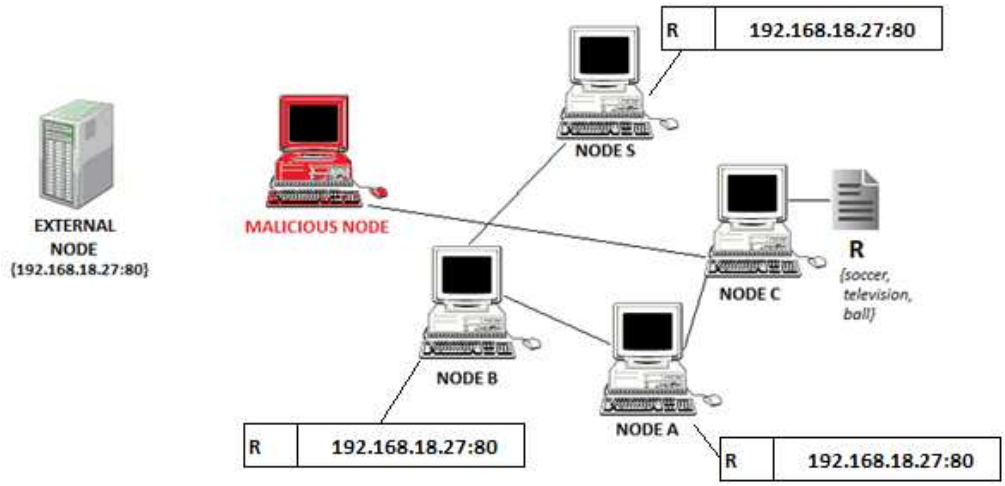

Figure 46 A good spot for Denial of Service.

Once another node except those in the figure above contacts them to have information about R, they will forward the request to External Node.

3.3 Conclusion

In this section we outlined some of the attacks that can be mounted on a P2P network in order to disrupt its services **[3]**. Other attacks are usable in this kind of environment to provoke some damages inside the network. One of this is the **Inconsistent Behavior**, in which a node behaves following the rules with its neighbors while it refuses to behave well once contacted by remote nodes **[49]**. Others are the **Identity Theft**, in which one (or some) malicious node claims to be another node, inside or outside the network, and forces other nodes to store this fake information in their local storage. This attack is similar to the Table Poisoning or the Index Poisoning. With the **Churn Attack** malicious nodes join and leave frequently the network. This can be dealt by routing protocols with a low amount of the churn ratio, i.e. the ratio between the number of joining and leaving nodes, but in the case of a very high amount this can be dangerous. Problems here reside in the update mechanisms of routing tables when a node joins and leaves the network. For example, when a new node joins the network an update phase is started, in which all the node routing tables in the path from the bootstrap node to the nearest neighbor of the joining node are updated with the information of the newcomer. If this joining node exits during the update phase, the message is forwarded until it reaches the destination but the joining node is no longer inside the network. This can lead to have a high traffic of trivial messages in the network and a Table Poisoning. If we multiply this action for a thousand nodes and we add the event in which the node exits with the new update we can have a large scale attack. Usually this attack is not used so much because P2P protocols are strong against it since the join and leave of nodes is daily life. Countermeasures in this sense cannot be clearly adopted since the join and leave operations are

allowed for all the nodes at whatever time **[42]**. Some of all these attacks, such as the **Decoy**, are useful for producers since they can block the illegal diffusion of copyrighted material in order to discourage the users to download illegally their products.

In the next chapter we will outline some countermeasures for the attacks shown in this chapter, pointing out which solutions are usable to block or to reduce the effect of these attacks.

CHAPTER 4

Countermeasures against P2P Attacks

In this chapter we will analyze some countermeasures and general solutions applicable to the attacks shown in Chapter 3. Some solutions are applicable to block more than one attack while some other solutions are strong against one attack but not against others. After presenting these countermeasures we will compare their applicability and their cost.

4.1 Introduction

Let us first analyze how the attacks shown in Chapter 3, force some weaknesses in P2P networks **[34]**. In P2P networks all nodes are supposed to cooperate but this is not always true, since some nodes inside the network might misbehave by not following the rules, i.e. maximizing their own profit to the detriment of other nodes in the network. This means that a node, malicious or not, can use a wide amount of system resources for its own purpose without giving the right portion of its own resources to other nodes. An example of this can be found in **free riders** presented in Section 2.7, or in the action from a user of maximizing the download bandwidth of its host while decreasing as much as possible the upload bandwidth to the network. The first problem is difficult to state since every resource in the network is considered as public for all users and blocking it for certain users is not possible **[35]**. This problem cannot be dealt with simple solutions, since resources have to be available at every moment and for every node in the network and nodes cannot be modified in order to block this unnatural behavior.

Centralized Networks In centralized networks, a possible point of failure is the centralized server (or the set of centralized servers), but also in some decentralized or hybrid networks, such as Tapestry, since once a resource is shared in the network, a root node is contacted with the information for the shared object and the server that shares it. If this root fails, the information cannot be retrieved. This thing can happen also in Chord, where nodes maintain references only for their successor and predecessor. In fact, a node that maintains references only for its successor in the ring is clearly weak. That is, if its successor crashes without any worries its predecessor would not be able to find another node as its successor and the ring will be fragmented.

Anonymity Another problem can be stated for anonymity in P2P networks. Since one of the main goal of a P2P protocol is to provide anonymity for its users, is not always simple to state who is in the network and if its generalities are true. Let us think at the example of a Table Poisoning. If a malicious node sends fake information about itself, i.e. an IP address and a port number different from its own and pointing to another node, this information will spread through the network causing other nodes to have wrong information about a portion of the network. This is possible because once a node receives the contact from another does not check if the information received is true, since this kind of check would possibly create network congestion.

Routing Some protocols do not use multiple routing paths to forward messages, thus if a node in the path crashes, the message cannot be forwarded through that path. DHT-based protocols avoid this thing by allowing multiple paths for the message forwarding. At this point, however, the problem of anonymity arises again since a node cannot know if it is subject to an Eclipse attack or if that messages will be forwarded to the destination or will reach the true destination.

Resource Diffusion Also resources exchanged in the network can give problems for the reliability of the system. In certain systems, resources are not always available since it is possible that some nodes that manage a resource fail, thus the resource is no more available. Moreover, for certain kinds of software, many versions exist and the user is not always sure to have the latest version or an old one. Along this, also the fact that some users rename the resources with incorrect names can lead to download the wrong resource. Finally, sometimes malicious users inserts in the network some kind of applications with malicious code inside them, called malwares. Formally:

Definition 40 [62]: "*Short for 'malicious software', **malware** refers to software programs designed to damage or do other unwanted actions on a computer system.*"

If a node inserts a file with some malicious code inside it and gives it the name of a popular resource exchanged in the network, many users will download it and, once run, they will be in trouble. Defenses against these kinds of attacks can be provided by software external to the P2P network, such as Antivirus or other similar tools.

We conclude this introductive section saying that not only malicious nodes can pose a threat to the network. Along with the hardware side of the network we need to take into account also the users using the P2P technology. Sometimes trusted users or good users are not interested to do something malicious but some of their unintentional behaviors create some problems in the network. For example, a user can wrongly set permissions for the P2P file sharing application to have access to a portion of the hard disk in which are maintained personal files. This can lead to the diffusion of these files in the network. As always, machines are not intelligent and many times are humans saying them what to do: if they act wrong is human fault.

4.2 Defenses against Identity Assignment attacks

In this section we will outline some countermeasures for those attacks classified as Identity Assignment: the Sybil attack and the ID Mapping attack.

4.2.1 Defenses against the Sybil attack

Blocking this kind of attack is not so simple. If we think about it, in the case of Example 3.1, no one can check if a user posting a comment is a trusted user nor if what it is posting is right or not. If a user follows the rules to authenticate in the network, no one can block him. We now illustrate some possible countermeasures.

Centralized Authorities Some common solutions involve inserting a central authority in the network. Formally:

Definition 41: *A **central authority** is a central node in the network designated to solve functionalities of controller and dispenser of certificates to join the network.*

As an example we can think about Certificate Authorities in the Internet, which supply certificates of authenticity for Web sites and similar. The main idea in P2P networks is to insert some special nodes, possibly outside the network, as central authorities for nodes joining in the network. These central authorities assign to the joining nodes their ID calculated with a hash function, thus IDs cannot be compromised.
As we can simply think, using a central point (inside or outside the network) to provide IDs is not so good, since it goes against the central characteristic of a P2P network, i.e. its decentralized nature. Moreover, inserting a central authority constitutes a central point of failure for the network. If a malicious node can reach the IP address and port number of the central authority, i.e. simply contacting it once joining the network, it can perform a Denial of Service using as target the central authority. So, this kind of solution is not so good even if it is the most efficient one.

Cryptographic Puzzles Another kind of solution to discover Sybils in the network is to make each of them solve some complex operations **[12, 61, 85]**. If a user is logged in the network with two identities on the same machine, he would not be able to solve two complex operations with a unique machine. Most used proposals are those which use cryptographic puzzles to prove the truth of an identity that involves considering computational resources as the discriminating factor between nodes. Once joining the network a node receives from the bootstrap node a puzzle to solve, i.e. a mathematical problem. Generally these problems are not so immediate to solve, and the joining node needs some time to solve it. The bootstrap node sets a timeout until which the joining node needs to send back the correct answer. If it does not, the bootstrap node can state that the node is inadequate to join the network and excludes it. The timeout is generally set to an average time for all the kind of hosts connecting to the network, because mobile users, such as phones, cannot solve a puzzle as fast as a personal computer. The important thing is to contact all the identities at the same time with the puzzles. Another solution of the same branch is to send to each entity a set of data of a certain dimension, i.e. a dimension that cannot be stored twice by a single node presenting multiple identities. In both cases, the problem here is that the bootstrap node randomly chooses to be malicious and can facilitate other malicious nodes to join the network. Moreover, in case of the Churn Attack the network will be congested quickly with a large amount of joining nodes when bootstrap nodes stays on a wait state for answers from joining nodes.

Stronger Identification A more difficult solution to realize is to attach in the identifier of a node its IP addresses **[34]**. This solution blocks the event in which a user inserts multiple identities from one host since each of these entities will have a portion of the identifier equal to the others. This is a difficult solution to implement since it adds an additional level in the complexity of the network

since the ID of each node has an extra field to be managed. Moreover, nothing prevents users to insert malicious nodes from different machines or, better, from virtual machines.

Gossip Some solutions use gossip as a possible way to discover a Sybil in the network.

Definition 42: *We refer to **gossip** as a protocol that allows a node to exchange periodically and at fixed timings some information on the network.*

In our case the gossip algorithm can be used to ask to other nodes in the network if a certain node has to be trusted or not. The main problem of this solution is that some malicious nodes inside the network can poison the gossip and give fake information about another malicious node under exam, i.e. forcing the examiner to accept a malicious node without knowing it is malicious. Another problem is that, with a large set of neighbors to contact, the number of messages to be exchanged is high and the network can be congested. This amount can be polished by using a **rumor mongering** protocol in which a node that receives a message checks if it has already seen it and if this is the case it does not forward the message **[35]**. This prevents the spreading of redundant messages. Another similar evolution of the gossip is the following. If a node (X) inside the network accepts a node (Y), which X trusts, with identity ID_y and on its hand, (Y) accepts node (Z) with identity ID_z, Y works as voice for Z to X because X trusts Y so it is inducted to accept Z, trusted by Y. If no controls are made to limit this thing an obvious weakness resides in the fact that if X trusts Y and Y is a malicious node, it can fool X by making it accepting other malicious nodes trusted by Y. A possible solution can be combining local knowledge, i.e. the nodes that X trusts, with global knowledge, i.e. which nodes trusted by X are trusted also by other nodes, weighting more the first source of information.

Guarantors Other solutions concern inserting an association mechanism through which at each node is assigned a trusted peer as guarantor for its trust **[15]**. The guarantor is chosen in order to have a low probability to sample a Sybil node. The idea is to assign the guarantor near to the new node, basing this choice on the similarity of their IDs. This should translate into a low probability to find two Sybils in a row. The similarity between nodes is determined by a small set of digits in the prefix or suffix of nodes IDs. The main problem is to determine the length of this prefix or suffix given that the designed guarantor needs to know the size of the network at a specified moment in time. This can be achieved by querying the network or the neighbors, but can translate into having a wide amount of messages, thus congestion. Once the joining node (J) enters the network it sends a message to its neighbors. The neighbors that receive the message sign it with their own digital signatures, in order to prevent some malicious node to sign it with no authorization, and send it back to J. After this process ends, J computes the minor distance among all its neighbors and chooses the nearest to be its guarantor. The main thing to say is that each joining node has a different guarantor respect to the other nodes. A possible problem here is that if two Sybils that are neighbors are encountered during the process, they can subvert the mechanism and force the joining node to unconsciously act as malicious, or facilitate the joining of other Sybils. The good thing of this solution is that the parameter for selecting the guarantor varies along with the length

of the network, so it is not always the same and malicious nodes cannot forecast which node will be selected as guarantor for them or for other nodes. The main problem, more than the amount of messages exchanged in the network, is that the search method for guarantors is equal to a **partial collision attack** over the hash function used to create nodes IDs. This is obvious because nodes look for the guarantor with the longest common suffix or prefix, thus will be difficult to recognize if an attack or the correct search mechanism is going on. A more useful implementation of this solution is to select a set of two nearest neighbors to take as guarantors. Another possible problem of this solution, more than those problems already presented, is that the chosen guarantor has to control all transactions that involves peer J. This means additional computational workload for the guarantor, even if each guarantor is assigned to only one node. Moreover, there are no guarantees that the reputational method for the network is maintained intact.

Payment Solutions Finally, another branch of solutions have been presented in the past years **[19]**. These solutions regard authentication via payment of some fees and authentication through trusted devices. In the first case when a user wants to join the network needs to pay a monetary value to the network for each identity he wants to authenticate. This discourages users to create more identities, thus decreases the level of Sybil attacks attempts even if it does not block them for sure. In the second case, a trust device is provided for each user connecting to the network, in a way very similar to the centralized authorities case. In this case the problem resides in the fact that no one prevents the attacker to purchase multiple trusted devices, even if the cost for each of them is high.

Evaluation of solutions The only good solution that will work without sensibly changing the network and providing the higher level of security for nodes inside the network remains the use of a central authority. This solution is problematic since inserting a central point of failure in the network goes against the nature of P2P networks, but it ensures that each entity will have only one identity and the possibility to have a Sybil attack going on is quite unfeasible.

4.2.2 Defenses against the ID Mapping attack

Basically, each protocol that allows a user to choose its own identifier is exposed to ID mapping attacks **[5]**. Usually node IDs are calculated with a hash function on their IP addresses and, sometimes, their port numbers. These information are strictly related to the nodes, thus they can manipulate them and choose the ID on their need. Many programs exist that allow a user to change its own IP address **[55]**. Even blocking the selection of the IP address is not so good, since the user can always select the port number on its need, even if this number is limited respect to the IP address.

Stronger Identification A solution is to add, along with the IP address and the port number, some information that the node cannot change. Usually, a malicious user cannot take control of a certain position in the network effortlessly. This means that if we take the IP address of the node and its port number, blocking the IP address would let the malicious user only to change its port

number, i.e. modifying the default settings of the P2P application. However, the number of identifiers that can be generated changing only the port number is very low compared to the entire logical space of the network, thus a malicious user needs to fatigue a lot to take control of a certain position in the network. Moreover, limiting the ID assignment in the space can lead to problems inside the network, since it is possible that the bootstrap mechanism turns into a deterministic one, thus malicious users could forecast where their nodes will fall.

Central Authorities Another solution is, as the case of the Sybil attack, to use some central authorities to invoke once a node wants to enter the network. Pros and cons of this solution have been already analyzed before and will not be discussed again here.

Variable Identifiers Coming back to the shape of nodes and resources identifiers in the network, we can have different options **[5]**. In the case of resources, once they are shared in the network they receive an ID that maps their position in the logical space of the network. We can add another identifier to them that represents the identifier corresponding to that resource at a given time in the network. This means that if a node wants to locate the specified resource it can search it with its starting identifier or by computing the temporary identifier composed by the current time in the network and the original identifier belonging to the resource. This translates into having an underlying system dedicated to movements of resources in the network. This means that at a certain time interval resources are moved from their actual location to another. Such a mechanism can lead to have two big problems: the first concerns the update of tables, since references for all the resources have to be refreshed and the amount of messages exchanged in the network will be very high; in the second case, if the resources are moved all to their new location at the same time, the workload for the network could be too much to handle and the network will thus crash. In DHT-based systems, a mechanism to move resources in case of join or leave by nodes is already available and the system can use it to perform this solution. The only problem remaining is that a resource, for a limited period, will be addressed with two identifiers. This means than once the resource is moved to its new location it cannot be found anymore to the old location and if during the moving process a node queries the old identifier, it will be directed to the old location, unable to find the resource. This can be bypassed duplicating the query for each resource, i.e. sending it to the old and new location. This will increase the number of messages going through the network but the number will be not so high also because these will be special events, not the rule. If we think at the case of two queries forwarded through the network in Kademlia, they are sent to the three nearest neighbors in case of replicated paths. This will lead to have other three queries to the three nearest nodes to the destination among all, and so on. The reasoning on using a change in nodes identifiers is the same. The cost is on updating routing tables at each round of IDs that can lead to a consistent load of the network.

Evaluation of solutions these two latter solutions, i.e. the rotation of IDs determined at a certain interval of time, can be good applied on DHT-based networks while using central authorities will be the best solution for the other kind of networks, as for the Sybil case. The reason is that in DHT-based networks, a system for moving a resource from a place to another already exists and it is

provided from the routing protocol. The main problem here is the high amount of messages exchanged by nodes and the high computational workload needed from the network to use the update mechanism.

4.3 Defenses against Routing attacks

In this section we will outline some defenses adopted to cope with routing attacks, presenting some mechanism to improve the security in the routing mechanisms inside P2P networks.

4.3.1 Defenses against the Eclipse attack

The natural evolution for Sybil Attacks is the Eclipse attack. As we saw in section 3.2.2.1, this kind of attack is still feasible when Sybil countermeasures are adopted. In the following we will outline some countermeasures to adopt against them, trying to evaluate them in terms of feasibility and robustness.

Constraint neighbors selection To avoid the Eclipse attack we focus the attention on routing tables which are usually attacked. A solution could be to constraint the selection of neighbors in nodes neighborhood **[38]**. With this solution neighboring tables are subject to constraints that mean that the network fixes certain points in its space trusted as valid points to be contacted whenever a node has doubts in trusting some other nodes. These kinds of valid points act as certifiers and make able other nodes to select their neighborhood properly. Even if this solution prevents Sybil Attacks and reduces greatly the effects of the Eclipse attack, the selection of neighbors is very rigid and techniques such as **proximity neighbors selection** cannot be used. A good solution is to look at the number of links a node is maintaining active and, if this number is higher than a system threshold, it is considered as malicious. Unstructured networks usually do not fix a constraint in terms of neighbors selection and when a node enters the network, through a bootstrap node and a random walk, it fills its routing table and neighbor table as well. It is clear that if a malicious node is in this phase the tables are invalid. Structured networks, on the other hand, maintain a fixed graphical representation of the network just to locate precisely an object with a certain number of hops. Usually each node or key has a unique ID of m bits and chooses its neighbors respecting some proximity logics. Basically they are chosen on prefix similarities and this opens a field for the Eclipse attack to be used. Using constrained tables in environments that use DHT, the Eclipse attack is prevented under two conditions: the first that each node has a unique ID not forgeable, given by an entity outside the network; the second that each node can identify other nodes near some specified points in space to know if they are online. With the latter, the probability to contact a malicious node and that a query is pointing to a non-valid endpoint are equal. To ensure these things a massive amount of messages are exchanged and an equal amount of ID checks are made with the final result to overload the system. Furthermore, constraining neighbor choices means in a loss of flexibility of the algorithm. A different shield against Eclipse resides in the choice of neighbors based on minimal network latency. This helps in choosing malicious nodes in a number lower than valid nodes, thus reducing the Eclipse effects. This

countermeasure assumes that malicious nodes cannot manipulate the response time and, due to network density, a great amount of nodes can fall inside the time interval, things that do not reduce the attack very well.

To cope with these problems nodes choose their neighbors comparing their internal connection grade. A malicious node mounting an Eclipse attack is supposed to have a higher level of internal connections grade, and each node has to reject it knowing this thing. But this is not enough since some malicious node can force others to raise their grade. We need to control also the external grade to be sure to identify malicious nodes. We fix also a threshold for both these values and nodes having that malicious nodes are excluded because have these values higher than the thresholds. In the particular case where all nodes has equal grade, the fraction of malicious nodes in the neighbor set is equal to:

$$\frac{m}{1-m} \tag{4.1}$$

where m is the fraction of malicious nodes in the network. We can fix the outgoing degree of connections for each node with an arbitrary parameter OD_c. The total amount of correct nodes in the network is fixed by V = N * (1 – m), since N are the nodes actually active in the network and (1 – m) is the fraction of correct nodes in the network. Since each of them posses an OD_c, we have a total outgoing degree equal to TOD_c = V * OD_c, and we can also insert a threshold for the ingoing degree with ID_c = OD_c * k, where k identifies the length of the neighbor set of each node (with k ≥ 1). Inserting these two thresholds forces the malicious nodes to use outgoing degrees of connection of other nodes to mount the Eclipse Attack. If we label the ratio of correct outgoing degree used by malicious nodes with m' we have that:

$$m' \times N \times (1-m) \times OD_c \leq N \times m \times ID_c \text{ and } m' \leq \frac{m \times k}{(1-m)} \tag{4.2}$$

thus depending on the length of the neighbor set fixed by k. For this reason, we need to insert a threshold for ingoing and outgoing degree of connection forcing each node to respect these values, thus blocking the number of malicious nodes to Formula 4.1. The next problem regards how to enforce the limit, because we need that each node can control other nodes validity. For this purpose a new set of neighbors is inserted, called **backpointer** that contains references to nodes that store the current node in their neighbor set. Periodically each node (X) anonymously challenges others in its backpointer to send back their own backpointers. If these are not sent back or X does not show up in some of them, the wrong answering node is removed from the neighbors list. To be sure that answers are not fake, X inserts a token within the request signed and forwarded back by contacts with their digital signatures. To enforce anonymity the message from X is sent through an intermediary node (Y). At this point we can have three possible cases. Let suppose that X wants to challenge node (Z) through Y:

1. **Z is evil and Y is not:** node Z can decide to answer or not, not knowing X identity.
2. **Z and Y are both evil:** Y sends to Z the identity of X and Z creates an ad-hoc valid response containing X. The latter does not figure out the dare.
3. **Z and Y are both valid:** Z gives its answer to Y that forwards it to X and the control always succeeds. X does not notice differences between this case and the previous.

4. **Z is valid and Y is evil:** Y can answer in place of Z or drop Z response making X to decide to block it.

The next step is to fix a certain criterion to block node Z. In fact, X can mark Z as suspect if it does not answer, or if it returns a set that does not contain X or if it returns a set greater that the allowed dimension. For the first two cases X can wait and redo the control, maybe changing the intermediate each time. Moreover, given that in the fourth case Y is evil, X has to fix a random interval in which to repeat the control, in order to block Y to take the place of X in the process or blocking Z if it wants to perform a connection analysis on X to figure out the random behavior, such in the first case where Z is evil.

To determine when a malicious node is identified a probability has to be defined. The main goal of a malicious node is to maximize its internal grade. Let us fix also the probability that the intermediate is evil equal to *f* and that a node is considered malicious if it does not respond to at least k < n challenges sent by the starting node. Given that due to the different timing and intermediate used each time challenges are independent from each other, all calculations are reduced to a Bernoulli function[6]. Furthermore, a correct node fails the challenge only if the intermediate is evil. So, the probability that a correct node is considered malicious in an n-challenges request is equal to:

$$\sum_{i=0}^{k-1} \binom{n}{i} f^{n-i}(1-f)^{i} \tag{4.3}$$

Fixing a limit for *f* and taking k in such a way the probability is small enough we can minimize the case in which the correct node is considered as malicious.
If a malicious node is on the challenged size, it can respond generating a set of r elements to fill its backpointer. So, if we define with r the overloading ratio, the probability to have the random sample to be accepted as valid is equal to 1/r, with complete anonymous requests. Moreover, the malicious node can respond with probability c and does not respond with probability equal to 1-c. So, for each challenge we have four particular cases:

1. With probability *f* the intermediate colludes with the malicious node and the latter passes the challenge.
2. With probability equal to *(1-f)*c/r* the answer contains X and X passes the challenge.
3. With probability *(1-f)*c*(1-1/r)* the malicious node does not give an answer containing X and does not pass the challenge.
4. With probability equal to *(1-f)*(1-c)* the malicious node does not respond.

Connections degree Other solutions imply the control on ongoing connections to a specific node. This means that an average threshold is defined for ongoing connections per each node and, if a node passes this threshold can be considered as malicious. After checking if nodes respect this parameter, malicious nodes can be kicked out from the network. The problem of these solutions is that is not always possible to determine if a node is really mounting an Eclipse attack or if it is only a popular node in the network, i.e. the manager for a very popular resource. Moreover, a node

[6]*The Bernoulli trial is an experiment that can give only two outputs: success or failure. It was named after the Swiss Mathematician Jacques Bernoulli* ***[115]***

can be under attack from a Denial of Service, and this translates in having to deal with a great amount of connections to deal with.

Certification Services Some other solutions use the same basics of countermeasures for the Sybil attack **[25]**. A certification service is introduced in the network and it aims at generating the IDs for nodes connecting in the network and binding them to their users. To bind these IDs the service creates some **tokens** based on a public key for the current node. Before entering the network, the joining node needs to contact the certification service node sending to it the user identifier and its public key. The server answers with a message containing the new identifier for the node, the user ID and the public key, this last couple sent before by the joining node. The three elements returned by the server are contained in a packet that represents the authorization code for the joining node, created after a signing procedure involving five entities: the node ID, its public key, the user ID, a timestamp representing how long will the token and the private key last for the certification server. The latter maintains a table in which each user ID is paired with its authentication code in order to provide to the same user always the same identifier in the expiration time interval. At the end of the joining procedure, the bootstrap mechanism is started. To enforce the solution, once the joining node (J) contacts the bootstrap node (B) they exchanges their ID and a nonce randomly created. After this preliminary step, J asks B to place it in the network by using a signature composed on the ID of the node J (if it is sending a message to B, or vice versa), the nonce received from the other node and an hash code of the request for the join of J or for the response from B. All this procedure is used to avoid the risk of man-in-the-middle attacks:

Definition 43 [84]: "*A **man in the middle attack** is one in which the attacker intercepts messages in a public key exchange and then retransmits them, substituting his own public key for the requested one, so that the two original parties still appear to be communicating with each other.*"

Evaluation of solutions The most valid solution among all those presented in this section is surely the one that uses Certification Servers. The main advantage of this solution is that a node cannot decide which ID to choose, since the certification server will give it an ad-hoc ID determining its authentication code. Moreover, even if the attacker succeeds in poisoning the table of a good node inside the network, it will be able to poison the table with its own identifier, since it cannot retrieve other nodes authentication codes. On the other hand, the Sybil attack is not blocked definitely since each node can send multiple requests for IDs passing different user authentication codes. This can be partially blocked by requesting a human interaction for the generation of the authentication code. Anyway, the highest problem is that the certification server constitutes a centralized point of failure. If it crashes in the network, the joining process will be blocked and, moreover, it can be exposed to the Denial of Service attack, since each node will know its IP address once contacting it to receive their authentication code.

4.3.2 Defenses against the Wrong Forward attack

The main problem we can face talking about routing attacks is for sure the wrong forwarding of messages or query by malicious nodes. As we exposed in Chapter 3, a malicious node that wants to perform a routing attack can choose to forward messages in a wrong way, so to block the routing mechanism provided by the P2P protocol **[14]**.

Stronger Routing Mechanisms Basically we can consider three ways in which messages are forwarded in a P2P network:

1. **Random:** With this protocol each routing operation is taken as random, from the ID assignment process to the choice of the next hop in a message forwarding process. Formally, also the entries maintained in the routing table of a node are randomly determined. Once the message forwarding process starts, the source node controls in its table if the destination appears inside it and, if this is the case, the source will directly contact the destination. If the destination does not appear in the routing table, the message is forwarded to a randomly chosen node from its routing table, repeating the proceeding until the message reaches the destination.
2. **Chord:** In Chord nodes ID are given computing the hash value of nodes IP addresses, mapping them in a logical space represented by a ring. With this protocol, routing tables are strictly related to the node ID, since they are filled with a number of nodes that are comprised in the interval constituted by the current node ID and the node whose ID is equal to 2^{i+1}, if i is the current node. The relationship between each node ID and its routing table is represented by the following formula:

$$R = \{R_i \mid R_i = (succ(i + 2^m), IP_i),\ m: 0, 1, \dots, i-1 \tag{4.4}$$

   In Formula 4.4 each entry for the routing table R_i is calculated inserting the successor of the current node, calculated at a distance 2^m from the starting node, and its IP address. The routing mechanism works as before, but if the node is not in the routing table the source chooses the nearest node to the destination (S).
3. **Diversity:** The initial point for the routing mechanism and the ID assignment process are as in the Chord case. The basic difference is on the routing tables where to the entries of the routing tables are appended also the path coming from the bootstrap graph for the node. This path helps identifying malicious nodes without using a centralized entity, and indicates connections between the nodes in the graph. If we examine the generic nodes N_A and N_B, the forwarding mechanism with this variation can work in three different ways:

   1. The node that receives the message locally stores the bootstrap path of it.
   2. The receiving node N_B sends to the source N_A its routing table from which the latter chooses the next node to forward the request basing its choice to the previous cached history, with respect to a certain algorithm, depending on the underlining protocol.
   3. Previous steps are iterated until the destination is reached.

Before presenting the security model we need to make some assumptions on the network regarding the maximal number of nodes fixed to 2^N, with ID of n bits. The network is assumed to be static that is it does not consider insertion or deletion of nodes during the routing process. In routing tables there are no repeated entries or the current node ID. The maximum routing path is defined as L_{max}, and it is equal to N. Finally, the fraction indicating the amount of malicious nodes is equal to m and goes from 0 to 1.

For the security model some definitions are used as well **[14]**. The first regards the **regular path** that indicates a routing path that does not pass through malicious nodes. To select this kind of path, before the message is sent, the routing algorithm generates a certain number of candidates paths and chooses the one with the higher **regular path probability**. Given that the probability can be faked by malicious nodes in the path or by the paths length, we need to define also the concept of **conditioned regular path probability**, used to determine the influence of malicious nodes in the path. In our case (A) determines the event "*The current normal path is lower than* L_{max}", even containing malicious nodes, while (B) is the event "*A regular path is generated"* **[14]**. The conditioned probability is then:

$$P(B\,|A) = \frac{P(A \cap B)}{P(A)} \tag{4.5}$$

thus Formula 4.5 measures the influence of malicious nodes in the path. Now we can move our attention to the previously unveiled routing algorithms:

- ✓ **Random:** the probability that the source node finds the destination for the current message in its routing table is expressed by:

$$P(A) = p = \frac{(2^t - 1) - t}{(2^t - 1)} \tag{4.6}$$

where t is the number of entries in the routing table of the source node and each routing operation is made randomly. Now, considering the event A we can determine the probability of the complementary event as:

$$P(A') = p^t \tag{4.7}$$

since if the source node does not find the destination inside its routing table it will have to forward the message to one of the entries in the table. The process goes on until the destination is found, thus the source node contacts all the t entries in its table. The intersection between event A and event B generates a new event like "*The* k^{th} *routing table contains the destination without malicious nodes in the associated path*". The probability associated to this new event is:

$$P(A \cap B) = \sum_{k=1}^{t} p^{k-1}(1-p)(1-m)^k \tag{4.8}$$

where t is the number of entries in the routing table of each node and m is the fraction of malicious nodes in the network. From Formula 4.18 we can calculate the conditioned probability of regular path with the following:

$$P(B\,|A) = \frac{P(A \cap B)}{P(A)} = \sum_{k=1}^{t} \frac{p^{k-1}(1-p)}{1-p^t}(1-m)^k \quad (4.9)$$

since P(A) represents 1 – P(A') and m is the fraction of malicious nodes in the network. Formula 4.8 describes a probability composed by independent events, thus its value is composed by the product of the probability of event A and event B for all the k tables.

- ✓ **Chord:** with this protocol the length of paths is always less than L_{max}, thus the probability associated to the event (A) is always equal to 1. Assumed this, we know that the maximum limit for the routing table at each node is equal to $2^{\log N} - 1$. We need to calculate also the clockwise distance between the source and the destination with the following:

$$d = \sum_{k=0}^{\log N - 1} a_k * 2^k \qquad (a_k = \{0, 1\}) \quad (4.10)$$

where a_k is the choice of the k^{th} path. The number of paths of length k resulting from the above formula is equal to C_n^k, since they are simple combinations of n = log N elements taken by groups of k. From this we can calculate the probability of the intersection between event A and B like the following:

$$P(A \cap B) = \sum_{k=1}^{\log N} \frac{C_n^k}{2^{\log N - 1}}(1-m)^k \quad (4.11)$$

that is also the conditioned probability initially sought, since P(A) is always equal to 1 and m is the fraction of malicious nodes in the network. Finally, in Formula 4.11 we have the product between the number of paths that satisfies Formula 4.10 normalized to the maximum number of entries a finger table can have and the fraction of valid nodes the routing process encounters at each of the k forwarding steps. The probability in Formula 4.11 represents the value of the conditioned probability of Formula 4.5.

- ✓ **Diversity:** here the difficulty is to formally represent the history of messages previously seen, thus we need to reason in terms of internal states of nodes. In this case we can deduce that the conditional probability will sensibly decrease with the increasing of the network size. Furthermore, this probability is influenced by the fraction of malicious nodes inside the network. Being a fixed value, the probability is strongly affected by two factors between which we need to find a good trade-off:

    1. **Network congestion:** if we identify with *e* the number of bits occupied by the couple [ID_{Node}, IP_{Node}], that are respectively made of 32 and 160 bit, we have that using n = 160 nodes the network traffic will be at least: (n + 1) * e = 161 * 192 = 30,912 bits.

2. **Space usage:** each node needs to store its routing table and also other information on the other nodes, and the history of received and forwarded messages. This will imply the usage of apposite storing mechanisms to avoid the saturation of the available storage space of a node.

Definitely, via using these notions we can determine which routing protocol is the best for our case through regular path search. In the future, stronger assumptions have to be made in order to look how real protocols behave and other algorithms that take into account node cooperation have to be developed.

The SeChord Protocol We saw in Section 4.1 that Chord is highly weak against routing attacks. Some solutions to fix these weaknesses were proposed, aimed at modifying the structure of the overlay network in order to obtain a more reliable network. The first solution consists in the **SeChord** protocol **[30]**. The main assumption for this security protocol is that each node cannot choose its own ID and this implies a central authority certification in the network (CA). Some nodes are also assumed to be compromised lookup requests can be redirected to malicious nodes by other malicious nodes. SeChord is effective against three different attacks:

1. A certain node does not answer to a lookup request coming from other nodes.
2. A malicious node forwards the lookup request to another node different from the right destination in order to create routing problems.
3. A group of malicious nodes cooperate to move away some requests.

The malicious node is supposed to possess a valid routing table containing information on other nodes in the network and another table containing position of other malicious nodes nearest to the entries in its routing table. To prevent the attacks outlined above, some statistical measures are supposed to be maintained in the network, such as the mean distances between nodes in order to identify an improper addressing of information inside the network. In Chord a certain property stands: given two nodes, the source (S) and the destination (D), if the message from S to D transits through a node F between them, the following inequality has to be true:

$$dist(S,F) \leq dist(D,F) \leq dist(S,D) \tag{4.12}$$

where *dist(A, B)* indicates the distance, in terms of identifiers, between node A and node B. If this is not the case, then an incorrect routing is acting between S and D. The underlining algorithm is modified in order to insert a reference in each node that points to the previous and the successive node in the ring. Furthermore an iterative routing is used instead of a recursive one, so to block the source to have complete control on the ongoing communication.

In Chord a generic node (A) maintains a finger table containing log N entries that consist by the information of the nodes at fixed distances, calculated by the following formula:

$$f_i = (ID_A + 2^i) \bmod 2^m \tag{4.13}$$

where m is the length of the node identifiers in bits, and i varies from 0 to ID_A + 1. Taking a candidate node (X) that has to be inserted in the finger table of the current node, because it satisfies the calculation of Formula 4.13, we need to make a check. A candidate node needs to satisfy a constraint in order to be inserted in the finger table of the current node, that is the distance between the candidate and its immediate predecessor needs to be lower or at most equal to the average distance between a generic finger table entry and its predecessor. If X is a valid entry for the finger table, this constraint is respected. However, sometimes this threshold can be too strict to be applied and a more elastic threshold is considered, equal to the average distance between an entry and its predecessor plus a portion of the standard deviation of the average distance. Formally the threshold will be:

$$T = AV_D + \alpha * SD_D \tag{4.14}$$

The introduction of this new threshold not only allows to have a more elastic bound, but also to control the presence of false positives or false negatives for the test. If node X satisfies the constraint expressed by Formula 4.14, then it will be inserted inside the finger table, otherwise not, thus preventing the insertion of fake information inside the finger tables. This constraint also blocks a node to arbitrarily forward a message to a destination different from those inside its finger table, thus preventing a Wrong Forward attack.

An important feature of Sechord is the **backtracking next hop selection** algorithm, exposed in the following. Once a node (A) needs to forward a message, it checks in its finger table which node is the nearest to the destination, i.e. the one with the higher ID among all the entries but that have an ID lower than the destination ID. If this node is not responding, node A contacts its immediate predecessor. Node A proceeds contacting other nodes getting closer to its own position every time a node in its finger table does not answer. If node A ends to contact the entries in its finger table without finding a suitable next hop, it demands the forwarding to its predecessor and the algorithm is repeated from this side. Node A stores the contacts of those nodes that fail in responding in a temporary list, in order to avoid contacting them again during the same routing operation. Along with this list, each node also stores the routing table of every newly contacted node in the current forwarding process and removes it at the end of the process. This process increases the average number of hops needed for routing operations, but ensures with a high probability to reach the destination for every routing operation.

Sechord uses a routing algorithm that consists in a modified version of the routing protocol of Chord. The main additions are the verification of the hop number for the message and the insertion of the backtracking next hop selection algorithm. Two structures are introduced with this new protocol and are maintained by each node: a **routing stack**, that contain new nodes discovered during a routing operation, and the black list explained previously, that contains all the nodes that are offline or that has no entries that can be contacted in their finger tables. The routing stack is formed by several entries, each of which contains a triplet **[ID_{node}, $finger\_table_{node}$, $index_{node}$]** where $index_{node}$ represents the current backtracking index for the current node. This index represents how many steps the algorithm has performed since it has encountered that node. The routing stack initially possesses only one entry that is the node that creates the routing

operation, having an index equal to zero. Once the node needs to select the next hop, it checks in its routing table for the node satisfying the proximity constraint. Once selected, its record is inserted at the top of the stack and both indexes are updated: the first node now posses an index equal to 1 and the next hop possess an index equal to zero. The selected next hop is verified with the rule in Formula 4.14 before being selected as next hop and it is not selected also if it is already inside the routing stack. At this point three possible sceneries can verify:

1. The node found is the destination because its ID matches the destination specified by the message.
2. If the node supplied is in the blacklist it is discarded and the process restart from the previous step.
3. If the node is not in the two previous cases it is contacted to get its routing table and a new entry in the stack is created.

When calculating the statistics, each node is able to calculate the length of an hop given that it possess the address of its successor and predecessor in the path and cannot modify them because they will be checked by a validity control. Servers that control new nodes entrances are supposed to be valid and that a level of trust is maintained between them. During the update phase of node IDs a check on the new IDs have to be provided in order to be sure that the new nodes do not differ from the previous too much and that the lists of successors and predecessors have not changed too much as well. Performances are computed in terms of malicious nodes discarding the lookup requests, the number of nodes that misroutes requests and the creation of sub-rings in the network by groups of malicious nodes. A ratio between correct lookups and total lookups cardinality is created and the number of average hops in the connections is counted.

The RChord Protocol Another solution for the Chord overlay is the **RChord** protocol **[11]**. As we learnt in previous sections, Chord is very good in terms of features and join/leave mechanism for nodes, but it is very fragile against those attacks that aim at increasing the average path length of messages. Many attacks can mitigate the correct usage of the protocol but the most dangerous are those attacks that redirect the queries far away from the final destination.

The main feature of Chord is its unidirectional routing. In fact, messages are forwarded only in clockwise direction **[11]**. The main weakness is that a malicious node can forward the query at a node that is some positions away from the destination, in order to force the system to forward again the message in clockwise direction and traverse the whole ring. An alternative to this behavior is to make available the forwarding of queries also in counter-clockwise direction. Introducing this mechanism, however, is not so simple. First of all we need to know how to add the so called **reverse edges** (the counter-clockwise links) and then we need to know also how to use the routing mechanism with them. For the first problem sometimes is useful to use the mirror of the routing table of each node. In practice we calculate a second routing table as we calculate the first one but in backward direction. The reverse routing table is not computed with the same length of the first because not so many nodes are required and thus we avoid a high overhead of communications having less nodes. A possible mechanism is the deterministic one, where each

node posses information over its nearest neighbors and can provide an edge pointing to them. Another system is to use the **Mirror Algorithm**. This algorithm gives back the exact mirroring of correct nodes in the finger tables. It receives as input the size of the network, i.e. the number N of nodes inside the ring, and the desired number of reverse edges R. To determine the reverse node for the generic i^{th} entry in the finger table, the algorithm solves $(i - 2^k) \bmod 2^m$, where k varies from zero to R – 1 and m is equal to the logarithm of the network size, identifying the first k predecessor of the current node i. A third example of algorithm that determines counter-clockwise nodes is the **Uniform Algorithm**, an extension of the mirror algorithm. We know that each node possess references to log N finger neighbors in clockwise direction that are uniformly distributed in the ring. So the main idea to find the reverse edges is to divide the space of dimension equal to log N, in R sections. As before, the k^{th} reverse neighbor for node i is computed with $(i - 2^k) \bmod 2^m$ but k changes as follows[7]:

$$k = \frac{(p+1)*m}{R+1},\ p = 0, 1, \dots, R-1 \tag{4.15}$$

With this formula the portion of the ring preceding the current node is partitioned in R + 1 spaces and reverse edges are selected in one of these portions. This ensures to have edges from different portions, thus a wider reverse finger table. The final algorithm that can be used is the **Local-Remote Combination Algorithm**. This algorithm tries to combine the information for local and remote edges, and these are made to cooperate. Remote edges identify nodes very far from the current position while local edges identify nodes very near to the current position. Moreover, local edges are selected in a higher amount respect to remote edges and their total amount is determined by the number of wanted reverse edges. The performances increase considerably if these two kinds of edges cooperate. In practice the R edges are inserted and the system determines how many local and remote edges need to be inserted at the current position. The i^{th} entry has always the k^{th} reverse edge determined as $(i - 2^k) \bmod 2^m$, having k varying as follows:

$$k = \left(\log N - 1 - \frac{p}{2}\right) * (p \bmod 2) + \frac{p}{2} * (1 - r \bmod 2),\ p, r = 0, 1, \dots, R-1 \tag{4.16}$$

where p/2 is rounded down. The Formula 4.16 allows selecting local edges with lower values of p, while it allows selecting remote edges with higher levels of p, while r needs to be an even number in order to have higher values.

An improvement of this algorithm is to add a random selection for remote edges. Motivations to this approach are given by the fact that random nodes make more difficult the usage of very disruptive attacks and the performances of the system will improve. What changes here is the method by which the remote node is selected. It is determined with a probability proportional to the distance between the current node and the remote one. Is clear that farthest nodes has higher probability of being selected but also nearest has some probability, even if very low.

[7] *To have the correct value for the parameter k we need to round down the value returned by Formula 4.14.*

The probability with which the farthest node is selected will be:

$$Prob(j) = \frac{(i-j)\ mod\ 2^{\log N}}{\sum_{q=1}^{\frac{n}{2}-1} q\ mod\ 2^{\log N}} \quad (4.17)$$

where j is a candidate node in the reverse neighbors set for being selected as reverse edge, i is the position of the current node and n is chosen high enough to facilitate the selection of farthest nodes. The probability described in Formula 4.17 is valid when the output of Formula 4.16 is odd and less than R, the number of reverse edges. The structural features of RChord are the same of the Chord algorithm, i.e. in case of leaving by a node. When a node quits the network the first node in reverse direction on the ring takes care of the resources maintained by the departed node. To modify the routing mechanism we need to notice that when a node needs to forward a message, it forwards the packet to the node nearest to the destination among its neighbors. In presence of reverse edges we need to insert a compatibility system to use them as well. This can be made simply making the algorithm consider also reverse edges and picking the edge that links the nearest node to the specified destination. A problem in these terms is that the number of reverse edges is usually lower than the nodes in the finger table. Failure in considering this thing can lead to accept nearest reverse nodes that need more hops to reach the destination. An idea is to make counter-clockwise and clockwise nodes cooperating to reach the best routing. The problem is that including all possible cooperative schemas leads to lack in robustness of the system. So the algorithm for the routing mechanism needs to consider also the number of hops needed for the message to pass from the current node (I) to the nearest node (J), in clockwise or counter-clockwise direction. The system is highly compatible with the original Chord routing mechanism because, lacking on reverse edges, the algorithm chooses only the nearest node in forward direction as Chord does.

Authors for this proposed modification of Chord evaluates their work by comparing different algorithms that choose reverse edges, and then drew some conclusions:

- ✓ The resilience of the RChord system is better than the normal Chord system. If we fix the success rate of an attack as P_r and we let it increase and apply the different algorithms to the network we will see that the performances increase as well. The authors make two tests with the above probability equal to zero and 0.3. In the first case, the performances in the increment of resilience for the RChord system for the Uniform, Local-Remote and Local-Remote Random algorithms are around 11, 12 and 13% respectively, while in the second case they are around 44, 53 an 43%.
- ✓ The best algorithm seems to be the Local-Remote, because it improves better the resilience of the RChord system, getting the best scores. The Mirror Algorithm is the worst while the Uniform Algorithm performs well with probability P_r lower than 0.3.

4.3.3 Defenses against Table Poisoning attacks

Here we present some useful countermeasures valid both for the Table poisoning and the Index poisoning attack. Malicious mechanisms involving search operations of nodes and values can derive from an incorrect routing mechanism or by an incorrect update phase. Commonly used solutions to block this event imply the redundancy concept. If the routing mechanism is badly organized, some solutions are those seen in the previous paragraph and the main idea for other solutions is to use different paths to transfer files or to made searches instead of using always one determined path. In the case of Table Poisoning the proposed solutions are different.

Duplicate Tables With this solution two kinds of routing tables are used: one like those used in Pastry and Tapestry that have entries ordered by increasing prefix or suffix matching with selected areas in the network, and one that possess entries in terms of increasing distance from the current position, such as the routing tables of Chord **[4]**. The first type of routing table is used when the node needs to perform a routing operation in order to have a fast routing and good performances. The second table is used if the first is corrupted, but deals to have lower performances. The first table is created after the bootstrap phase that involves more than one node in order to avoid the contact by malicious nodes or to lower as much as possible this event. Once the joining node receives back the table scraps by the bootstrap nodes it selects those entries that are closest to its own position and constitutes its first table. The second table is created by making multiple searches in the network for some IDs with a value higher than the current one. This produces a path that is then stored inside the second table.

Secret Tables With this solution each node inside the network possesses a secret value that is known only by itself and marked as V_A if A is the current node **[16]**. Once A receives a new contact by node (K) in the network, communicating that node (N) has just entered the network, A waits to insert N in its routing table and contacts this node, discarding the information received by K. Once node N replies, A can store its information after verifying the validity of the announcement. The passages for this insertion are basically three: (1) node K sends to A a notification composed by the triplet [ID_N, IP_N, $port\_number_N$], (2) after receiving the notification, A sends to N a message composed by the current timestamp and the one-way hash of the notification message along with the timestamp and its secret value V_A and (3) once N receives this message it replies with its own notification, composed by its triplet, and the hash sent previously by node A. To make the comparison, node A compares the hash of the current notification along with the timestamp sent by node N and the secret V_A, with the hash created to contact node N in step (2). If they are equal, then node N is authenticated and inserted in the routing table of A. The security here is based on the fact that knowing the secret V_A is very difficult, thus only node A can compute the hashes with V_A.

Evaluation Both the solutions exposed in this paragraph are good to be used against a Table Poisoning attack, in terms of efficacy. The first solution ensures to have at least one of the two routing tables always working and routing operations can always be performed. On the other hand, the routing mechanism is changed since the two routing tables need an ad-hoc mechanism

each to be used. This increase the complexity of the routing protocol as well as the possibility to have clogging inside the network. Moreover, the creation of the first table is very expensive in terms of communicative need, since many bootstrap nodes have to be contacted in order to build the table. This can lead to have many messages flowing inside the network and the possibility, in the case of multiple joins at the same time, to have a similarity with a denial of service. Finally, each node is not sure that among the bootstrap nodes there are not malicious nodes, thus their routing tables may be corrupted in part.
The second solution, that involves some kind of cryptography, is probably the better one in terms of efficacy, since each node can mark its messages to the others with its own secret. Problems here are in terms of volume of traffic inside the network. When a node receive the contact of another node to be stored in its routing table, it needs to check if this contact is valid or not by sending to this one a message. Multiplying this event for multiple nodes, means having a great amount of messages flowing in the network. Moreover, in the bootstrap phase the number of exchanged messages is equal to the number of the nodes returned by the bootstrap node, and this can be dangerous. On the other hand, a malicious node can send multiple times a message containing fake information about a node, with the IP address and the port number of a node (V) outside the network. Once the receiving nodes start the check phase, node V will be flooded with messages and being external to the network it will be unable to answer to them, thus resembling a denial of service. The same reasoning can be made for a node inside the network that is a more dangerous situation. Finally, the fact that each node needs to have a secret at its location means that some kind of central authority needs to be inserted in the network and this identifies a possible central point of failure. The creation of the secret cannot be demanded to the bootstrap node a new node contacts once connected in the network, since it can be malicious and can store the secret of the new node for its evil purposes.

4.4 Defenses against Application attacks

In this section we will introduce some defenses applicable at the application level to block those attacks appertaining to this branch.

4.4.1 Defenses against Index Poisoning attacks

Just to recall, this attack concerns the poisoning of the file indexing system. To prevent this event, several countermeasures are adopted, all falling in the field of cryptography **[15]**.

Merkle Trees The first solution proposed concerns Merkle Trees. Formally:

Definition 44 [96]: *"A **Merkle Tree** is a construction introduced by Ralph Merkle in 1979 to build secure authentication and signature schemes from hash functions (e.g. SHA1)."*

Using a consistent hash function, robust against collisions, a Merkle Tree is able to produce the cryptography of a large amount of data. The tree is organized as in Figure 47, where each leaf node contains the hash of a data set in a set of files. Internal nodes represent the hash of their children.

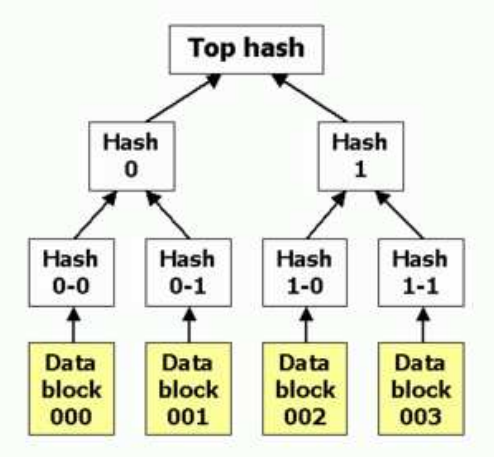

Figure 47 An example of Merkle Tree **[72]**.

Each element shared in the network is introduced in a leaf node and its authentication is performed recursively applying a hash function for each internal node from the starting point, i.e. the starting leaf node, to the root node for the tree. In this way, each element can be recognized in presence of fakes, since the hash resulting from the process will be different from the one represented in the tree. The problem here is that the root needs to be assigned to a trusted entity inside or outside the network. An evolution of Merkle Trees are **Tiger Trees**, that uses a binary hash tree having for each internal node two leaf nodes and apply the Tiger hash function to produce the hashes **[1, 116]**. As for all the solutions that imply the use of a hash function, even this case is sensible to collisions. A hash function is said to be theoretically (and ideally) strong against collisions, but in practice some of them are not. The Merkle Trees allow the user to use several hash function, but some of them are weak against collisions, i.e. the MD5 algorithm or the SHA-1 hash function.

RSA Accumulators Another kind of cryptographic solution is to use RSA[8] accumulators and an exponential modulation with a zero knowledge protocol **[16]**. Formally:

Definition 45: *A **zero knowledge** protocol is a methodology which involves two parties: a certifier and a verifier. The method implies that the certifier can prove to the verifier that a statement is true without knowing anything else about the statement, i.e. only that it is true.*

The network is divided into groups of nodes in which each node generates an exponent n_i. After this, all the nodes belonging to each group selects a seed (S) and a modulo (N), different for each group, to create the RSA accumulator expressed in Formula 4.18.

$$RSA_{Acc} = (S^{n_1 \cdot n_2 \cdot \cdots \cdot n_n}) mod\ N \tag{4.18}$$

Every group then generates an auxiliary value using Formula 4.18, removing the i^{th} exponent from all those generated by the nodes in the group. The auxiliary value is like the following:

$$AUX_i = (S^{n_1 \cdot n_2 \cdot \cdots \cdot n_{i-1} \cdot n_{i+1} \cdot \cdots n_n}) mod\ N \tag{4.19}$$

[8] *RSA is an asymmetric cryptography algorithm introduced by Whitfield Diffie and Martin Hellman **[101]***

with n the total cardinality of the current group, i.e. the number of nodes contained in the group. To verify the accountability of each node in the group, the i^{th} node presents to the verifier a couple composed as **(AUX_i, n_i)** and the verifier checks if Formula 4.19 is respected with the following test:

$$RSA_{Acc} = AUX_i^{n_i} \tag{4.20}$$

This methodology works for node identities checks but it does not block the insertion of fake contents inside the network, i.e. fake resources or routing table entries. Its efficacy is based on the fact that knowing the value (AUX_i, n_i) is a proof at zero knowledge, thus the current node needs to be trusted. It can be a good way to determine if a node can insert a certain content by checking if it can be trusted by the group instead.

RIEP A third methodology consists in three phases and recalls the usage of Merkle Trees **[16]**. In the first phase, when a node wants to join the network it contacts a bootstrap node that creates some values used for the authentication of the joining node. Once the node joins the network and wants to share some resources, they are inserted in the network without any additional information. Doing this, only the original sharer can modify the resource or its content. In the final phase, the initial sharer authenticates itself by solving a Merkle Puzzle and some zero knowledge proofs to store the resource. The main problem with this solution is that to build the tree the network needs a certain number of centralized authorities and that a malicious node can send messages in the network in place of a non-member.
Finally a public key cryptography system is proposed. This method implies that files are encrypted under a public key constituted by the nodes IDs and a verification system to check whether the exchanged information comes truly from the source node or not, based on the use of cryptographic functions. The system works as explained in the following. Having a record for a file F composed as in Formula 4.21, the system is able to determine if F is maintained by a certain node N whose ID is equal to P_{key}.

$$R = \left(Hash_F, P_{key}\right) \tag{4.21}$$

Here R represents the couple (key, value) for a DHT system, where the key is the hash of the file and the value is the ID of the node maintaining it. This describes a transparent trust system among nodes, called **Reliable Index Exchange Protocol** (acronym *RIEP*) and explained in the following. RIEP replaces the index format proper of the network with another one more secure, represented in Formula 4.22:

$$R_{RIEP} = \{R(Hash_F, P_{key}), Sign_{key}\left(Hash_F, P_{key}\right), K, Sign_K\left(P_{key}\right)\} \tag{4.22}$$

where $Sign_{key}\left(Hash_F, P_{key}\right)$ is the signature of the publishing node using its ID as key, K is the certifying authority for the network and $Sign_K\left(P_{key}\right)$ is the signature of the node ID with the key of the certificate authority. The first operand of Formula 4.22 is equal to Formula 4.21 and represents the normal association between the hash of a file and the ID of its manager. The RIEP protocol provides a system by which each resource publisher can sign with a private key the resources it wants to share. For this reason, each publisher contacts the certificate authority that

is designed to dispatch private keys for the signature process. The publisher (P) sends a message encrypted with the public key of the certificate authority (K) to the latter, containing its node ID and a randomly created numeric token (T). As answer, the certificate authority will send a message encrypted with T to P, containing the ID of the node, its new private key created by K and the signature of P ID, signed using the private key of K. After receiving the private key, a node can publish its resource with the method exposed with Formula 4.20 that is called **Security Enhanced File Index Format**. The system of private keys is also used when a node wants to query a resource in the network. Once a node (A) needs to locate a resource (R) in the network, it sends a query to its neighbors in a number that depends on the routing protocol used by the network. If among the neighbors of A there is the node B and A forwards the query to it, A will sign the query with its own private key. If B possesses the information on the location of R, it will send back this information in clear and a copy of the same information signed with its own private key. Formally, A sends the query message to B containing its ID, the query containing the hash of the searched file and the current timestamp, along with the signature of the previous two elements, signed with its own private key. Node B will answer with the information for the searched resource and the same information signed with its own private key, to provide credibility. In the case that node B does not possess information for R it will forward the query to other nodes. In this case, the message forwarded is made with the same content of the query hit message. If B has to forward the query to a node (C), it will contact the latter with a message containing the entire query from node A plus a signed version of the same query, signed with its own private key. The signature verification uses a simple mechanism. Once a node receives an information on the resource it is querying, it verifies if the signed content was created with the corresponding private key. This action bases on the fact that the signature was forged with the private key of the publisher, and the latter was created on its own with the public ID of the publisher, i.e. its public key. So, the public key of the publisher needs to be equal to the public key of the node that provides the searched content. If it is, the information is reliable, otherwise it is not.

Evaluation of the solutions With this latter solution attacks are sensibly lowered, but not blocked. In general cases a malicious node can launch attacks directed only to its position or to a bunch of IP addresses authorized for communication in past moments. The main problem here is the great amount of computational power needed for the RIEP system and requested to each node in the network. Moreover, a centralized authority is not a good solution because it will constitute a central point of failure for the network, as we saw in the case of the Sybil attack. However, this solution is a good starting point to prevent the Denial of Service attack as well as an Index Poisoning attack.

4.5 Defenses against (D)DoS attacks

In this section we will give some advices on defenses usable in P2P networks to prevent or to block Denial of Service attacks and their distributed variation. These attacks are usually at the routing level but they can be mounted also at the application level.

4.5.1 Defenses

As we saw in Chapter 3, the Denial of Service attack can aim at different goals and can imply different other kind of attacks. However, the mainly attacked mechanisms by a DoS attack in a P2P network are the routing and the file indexing mechanisms.

Trust Systems The first solution proposed regards a credit mechanism, similar to the voting scheme we presented in Section 3.2.1.1 **[46]**. Credits here are assigned to nodes depending on their behavior in the network, evaluated with some specifics:

- ✓ Each transaction or operation in the network has a different level of importance and credits are assigned by weighting each operation. For example, an operation that gives a low amount of credits is the manumission of information in the network.
- ✓ The more a node provides data and information in the network and the more credits it will receive. This has two advantages: a node that provides a high amount of data will gain a high level of credits and will reach higher levels of prestige in the network.
- ✓ To be sure that a node does not receive a high amount of credits and live of rent, an expiration time for credits is inserted.
- ✓ Finally, the entire underlying credit system needs to adapt itself to the fast change of nature in nodes behavior, i.e. the system needs to be highly scalable.

Each node (X) can discover its amount of credit C(X) after a transaction. Formally:

$$C(X) = \alpha \cdot \sum_{c=1}^{T} Sat(X,c) \cdot Cre[m(X,c)] \cdot T_{fact}(X,c) + \beta \cdot C_{fact}(X) \qquad (4.23)$$

where: *X* is the current node, *T* is the number of network operations accomplished by X, *Sat* is the satisfaction level for node X gained by the transaction *c*, *Cre* is the credibility factor coming from the other node involved in the transaction, T_{fact} consider the factor coming from the content of the transaction c for the node X and C_{fact} takes into account the community factor for node X. Finally, α and β are two weight factors normalized for the system, that varies between 0 and 1. These two parameters regulate the significance of the information coming from the community of the network (β) and the feedback evaluation (α). Their sum has to be equal to one. This formula comes from a system called **PeerTrust** that provides a trust system for P2P networks, similar to the credit system **[48]**. What changes here is the meaning of the parameters in Formula 4.22. The satisfaction parameter is translated into the representation of the level of importance associated to each operation in the network, as explained above. The operation importance is represented from the content factor that gives high penalties for the manumission of data in the network. Finally, the total amount of operations accomplished by the current node takes into account also the expiration time for each transaction, so only a small subset of accomplished transaction will be considered in the calculation. To enforce the system, credits are considered in four different types:

1. **Control Credits:** These kinds of credits are received or lost from a node if it helps or not the system to control the behavior of other nodes or the consistency of data exchanged in the network.

2. **Data Credits:** these are credits earned by providing data, respect to the amount of data provided in the network by a single node.
3. **Tampering Credits:** if a node does not change data in the network or does not intercept data direct to other nodes, its amount of tampering credits will be high, very poor otherwise.
4. **Disconnecting Credits:** if a node does not frequently (or in a high amount) disconnect other nodes that are downloading some resources from it, the amount of disconnecting credits will be high.

The most important types among all the four are the Tampering and Disconnecting credits, in order to calculate the importance of single transactions. So they can be ordered by descending importance as: tampering, disconnecting, data and controls. The countermeasure against the Denial of Service here is represented by the Disconnecting Credit system, since if a node L connects to a node J and then L disconnects from J, both L and J will be penalized for the operation. This system is good in case of malicious nodes that redirect connections from good nodes to fake nodes, by which the good ones need to disconnect, i.e. because they do not find the searched content or the connection falls. Moreover, with this also the problem of free riders is well mitigated. Finally, each node can select its set of neighbors by connecting to those which have a higher amount of credits among other possible neighbors. This system is provided by the difference between the levels of credits maintained by the current node and its neighbors. If the difference is lower than a certain threshold defined by the system, the current neighbor will be selected as neighbor by the current node. On the other hand, if a node detects that one (or more) of its neighbors are decreasing their credits level, it will disconnect from them.

This solution, called **Ripple-Stream**, was initially proposed for P2P video streaming networks, where the poisoning of contents is more common than in file sharing networks. However, a solution of this kind can be used also in file sharing networks, since a voting scheme can be actuated to all the routing protocols encountered in Chapter 2, but the performances can be poor. The main problem with this solution is that frequently the network is highly congested and, as we said previously, a congested network or a congested portion of the network can resemble very well a Denial of Service attack, thus the system cannot recognize an attack from a real congestion problem.

DD-Police Networks that are highly exposed to Denial of Service attacks are those that use query flooding mechanisms, such as unstructured networks. Just to recall from Chapter 2, in unstructured network a node that is searching for a resource queries a set of neighbors to retrieve the resource. The query forwarding process goes on until the resource is located or the Time To Live associated to the query expires. The main idea here is to make nodes to cooperate with each other to highlight malicious nodes that are misbehaving in the network, i.e. those which are performing an attack. This idea is at the base of the **DD-Police** that make nodes cooperate with each other in a range r from their position **[22]**. The solution implies three steps that involve neighbors in the network:

1. **Neighbor contacts exchange:** as we explained in Chapter 2, each P2P protocol uses a table to maintain information about neighbors in the network. Two neighbors with DD-Police are intended to exchange these lists at fixed intervals of time. The initial problem concerns determining the frequency of these exchange operations, because the churn of a P2P network is very high and nodes contacts changes frequently. On the other hand, exchanging too frequently these lists will cause congestion in the network. The safest solution is to exchange neighbors lists at fixed intervals of time. The table poisoning is prevented since at each exchange the two neighbors check the entries in the received list for consistency. If one or more entries are not consistent, then the node that advice this inconsistence will disconnect from the inconsistent node informing its neighbors of the fact. Neighbors in each list are intended to be neighbors in the overlay network, i.e. connected by logical links.
2. **Query Traffic Monitor:** for this part each node maintains two lists for the queries in the network. One list, called *Incoming(j)*, contains all the queries received by the neighbor (J), while the second list, called *Outgoing(j)*, contains all the queries sent to the neighbor (J).
3. **Recognizing Malicious Nodes:** this part is the most important one and concerns the discovering mechanism of malicious nodes inside the network. For this purpose, a new message type is inserted in the network to communicate to the neighbors that a node is considered malicious from another. The system defines a maximum threshold for incoming and outgoing queries for all the nodes in the network. Once a good node (G) denotes that another node (M) surpass this threshold, it can consider M as malicious. At this step, node G will inform its neighbors that node M is suspected to be malicious, by sending to each of its neighbors a message containing its IP address, the IP address of M, the current timestamp and the number of incoming and outgoing connections for the node M. These last two fields correspond to the M^{th} entry in the *Incoming* and *Outgoing* query lists of node G in the last minute. The message is forwarded by the neighbors of G if it was not already sent from them in the last five seconds, as specified by a system parameter. A node is considered as malicious if it has sent more queries than the maximum allowed in the last interval of time reference, e.g. a minute.

With the following example we can see in which cases a node is considered suspicious by another.

Example 4.1 – Recognizing malicious nodes

Let us suppose that we are in the network represented in Figure 48, and that a maximum query threshold is specified in the system and it is equal to 100. Let us now suppose that, after a certain amount of time, node G advices that node M exceeds the threshold for outgoing connections, sending more than 100 queries to node G. As we said before, node G can consider M as a suspicious node.

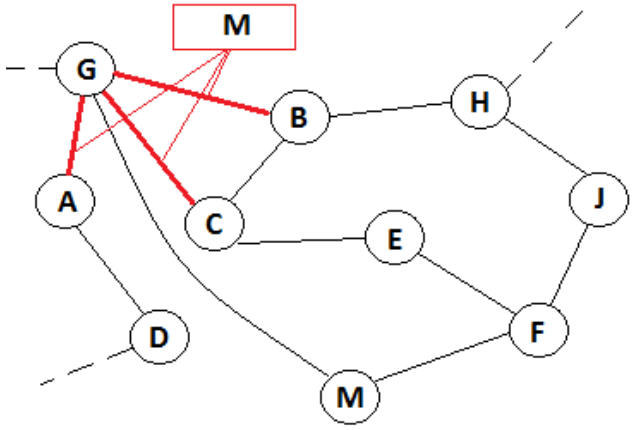

Figure 48 An example for malicious actions.

At this point, node G sends to its neighborhood the suspicion message containing all the parameters expressed before. Once the nodes in its neighborhood receive the message, they check if they have already sent this type of message in the last five seconds. If they have not, they will forward the message to their neighbors. At this step, node B, that is a neighbor of both G and C, receives again the suspicion message and will not forward it, since it has already forward it in the last five seconds as specified by a system parameter (see Figure 49).

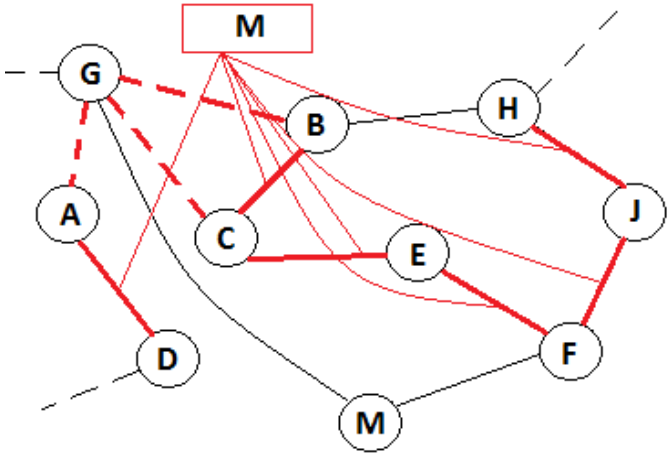

Figure 49 Evolution of the malicious action.

◊

To determine if a suspicious node is indeed a malicious node, its number of outgoing query is compared to the number of received query. These values can be retrieved by the Query Traffic Monitor and the Neighbor Contact Exchange explained before. If this comparison highlights that the difference, normalized to the maximum query threshold, is lower than or equal to one, the node is not marked as malicious. If the difference is greater than one, this means that the marked node is indeed malicious and has to be considered a node which is mounting a DoS attack. The main problem is to fix the correct threshold in the system: if it is fixed too high some malicious nodes will be considered as good ones, while if the threshold is fixed too low some good nodes can be marked as malicious.

Evaluation of the solutions In my opinion, all these solutions do not take into account an ongoing Eclipse attack. If some malicious nodes are cooperating to attack the network, they can mount a Distributed Denial of Service by sending each one a query less than the threshold to other nodes, overloading these latter. Another possible scenario is that a set of malicious nodes are inserted in the same neighborhood and launch many queries, possibly an amount higher to the maximum threshold. Once a good node in the neighborhood notices the malicious behavior of a node M, it will send to the neighborhood the suspicion message. Once the message reaches the other malicious nodes, they can cover the malicious behavior of M by telling the sending node for the suspicion message that they had sent to M the large amount of queries to be computed. So M will be considered as good. Moreover, a malicious group of nodes can perform an Eclipse attack directed to a victim node (V). If V is a popular node, that maintains a very popular resource, the malicious group can decide to cut it out from the network by sending to its neighbors a suspicion message. In the message they will insert a fake number of outgoing connections from V, larger than the maximum threshold, and an amount of queries directed to V that summed up gives a number lower than the number of queries specified in the outgoing field of the message. Once contacted back to have confirm about the suspicion from other good nodes in the neighborhood, they can answer that they have not sent the large amount of queries denoted in the suspicion message, thus the good node will be considered as malicious and cut out from the network. The damage is even bigger if no replication mechanisms are provided in the network. To conclude we can say that a singular malicious node that tries to fool the system cannot succeed, or they cannot succeed for long, but a group of malicious nodes can probably succeed.

4.6 Conclusion

In this chapter we outlined some of the defenses that can be applied to P2P networks in order to protect them against those P2P attacks exposed in Chapter 3. In the next chapter we will show an example of attacks using a P2P simulator called Peersim, trying to figure out the effectiveness and the degree of success of some of the attacks exposed in Chapter 3 and some new ones. We will try to propose a solution to some of the attacks by inserting some defenses inside the protocol.

CHAPTER 5

Simulating old and new routing attacks

In the first part of this chapter we will test some of the known attacks to the Pastry protocol, and we will propose some countermeasures. We will then introduce some new original attacks and we will try to enforce the original protocol in order to mitigate their effects. To simulate all the attacks we will make use of Peersim, a P2P network simulator proposed in **[28, 94]**.

5.1 Introduction

P2P networks are usually composed by a great number of nodes, thus designing such networks and providing services is a non-trivial problem. There are many aspects that have to be taken into account when developing a P2P network: the high number of subscribed nodes, the presence of NATs and/or firewalls inside the network, some differences in the internal structures among nodes, the dynamicity of the entire system and so on. Thus the testing phase for this kind of networks is non-trivial since all these many factors have to be observed and the observation phase in real networks is non-trivial as well. For this reason, in the last years software to test and evaluate P2P networks, such as **Grid5000**, **Planet Lab** and **Peersim** have been proposed **[28, 75, 94]**. The main need was to have a simulator with which to initialize and test a P2P network that was also able to do some tests repeatedly and to work even in large networks. In this thesis we have used Peersim, a simulator that gives to its users a high scalability and allows a high dynamicity proper of P2P networks. In the next section we will introduce this simulator.

5.2 The Peersim Simulator

Peersim is an open source simulator, developed at the Bologna University by Mark Jelasity, Alberto Montresor, Gian Paolo Jesi and Spyros Voulgaris **[28, 35, 94]**. The main goal of the creators of Peersim was to build a completely dynamic and highly scalable system to test the functionalities and properties of P2P network protocols **[35]**. After some time, they decided to make it public under the GPL open source license. The simulator is written in Java language, and comes with two basic engines: the **event-driven** engine, that supports the transport layer simulation and is more realistic, and the **cycle-based** engine (run by the event-driven engine) that does not make use of the transport layer but only of the results coming from the execution of each protocol at each node on every cycle of the system. The actual developing of the simulator is now at the University of Trento under the supervision of Alberto Montresor and Gian Paolo Jesi and their work is partially supported by the Napa-Wine project **[86]**.

Features One of the main goals of the Peersim simulator is to give to the final user a highly scalable system. This property is provided by the usage of the cycle-based engine, since it does not make use of the transport layer, thus the simulation becames faster and the user can add a higher number of nodes respect to the event-driven engine. Another important feature that Peersim

possesses is the high configurability, since each run is created over a configuration file in which a user can specify several parameters with which to run an experiment. Each simulation is made by using a set of classes as the **core** of the simulation. These classes specify the behavior of the simulator and a set of components to be inserted in the simulation process. The number of nodes inside the simulated network can be in the order of a million. Peersim provides also **graph abstractions** since the overlay network can be translated into a graph and some evaluations are given, such as the network dimensions and the connectivity among nodes **[35]**. The main disadvantage of this simulator is that each simulation is conducted in a single host, so the multi-core simulation is not used, thus this can lead to slow the simulation flow and to have bad performances.

The Simulator Each simulation is handled by the kernel of Peersim that is a central point containing minimal functionalities to make the simulation work. From the user side some specifications can be included and added to this primary kernel, such as components or parameters useful for the simulation. Each new component inserted by the user can be simply replaced by another component, since each simulation can be specified by a configuration file. The simulation is made in single threads, and can be of two kinds:

1. **Cycle-based:** this type of simulation creates an overlay network composed by a set of nodes. Each node is considered as a container in which a certain number of protocols are running, and nodes are considered one at a time with every protocol running at each node to see how they work. The routing mechanism is performed without taking into account the latency or problems during the delivery phase, such as message dropping or loss, since the transport layer is not simulated. This kind of simulation is usually employed to test the properties of the overlay network that are independent from the transport layer, such as the constant grow or decrease of nodes that affects the number of neighbors per each node. Moreover, the number of nodes used in this simulation is higher than the other type of simulation and it is in the order of 10^7.
2. **Event-driven:** this kind of simulation can be run along the previous simulation, and provides a more realistic test, modeling the routing system among nodes such as the exchange of messages between a source and a destination. Events are intended as the processing of messages received by other nodes. Once the joint simulation is used, some protocols are scheduled with the cycle-based simulation while other are used only when a certain event arises. Using the event-driven simulation deals to have a lower number of nodes in the system, in the order of 10^5.

Structure The architecture of Peersim is represented in Figure 50. The main component of this architecture is the **Network** vector in which references to all the nodes in the simulation are contained. Each node is created cloning the first node and each of them is identified by its own ID assigned by the system. Moreover, every node contains a set of protocols that define the behavior of each node during the simulation and each protocol is identified by a **Protocol Identifier** (acronym **PID**). A particular protocol is the **Linkable** protocol that provides the set of links a node possess in the network, i.e. the set of neighbors.

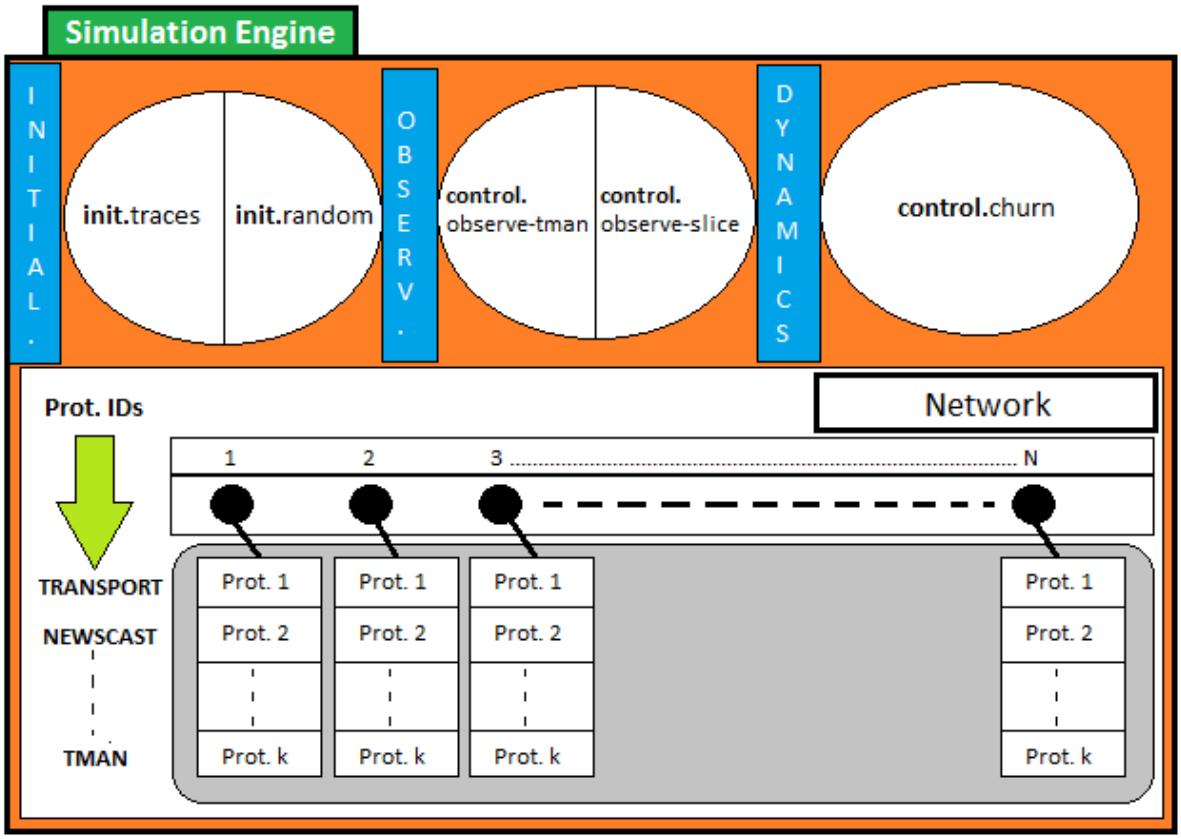

Figure 50 Peersim Simulation engine.

At the top of the figure we have the **Controllers** that are special classes that manage the initialization, the observation and the dynamic phase of the network. In the lower part of the figure there is the vector of nodes of the network, with all the node IDs and, in the grey part of the image, the stack of protocols they run.
Using the cycle-based simulation, the interface **CDProtcol** is implied and defines a set of protocols that each node executes sequentially at each cycle of the simulation. The user needs only to fill the *nextCycle()* method in the CDProtocol interface that will constitute a cycle of the simulation process, i.e. it will contain the parameters used by the simulation engine. After the simulation invokes *nextCycle()*, the current node in the Network vector is popped from the list and the protocol is run, selecting it with its own identifier from the configuration file.

Configuration Each simulation needs to specify some **global parameters**, such as the number of nodes inside the network, the number of cycles the simulation will perform, etc. Moreover, the configuration file needs to specify which protocols and controls the engine will perform during the simulation process. The configuration file can simply be a *.txt* format file containing some strings indicating the parameters for the execution. An example of a configuration file is given in Figure 51.

```
1   # PEERSIM EXAMPLE 1
2
3   random.seed 1234567890
4   simulation.cycles 30
5
6   control.shf Shuffle
7
8   network.size 50000
9
10  protocol.lnk IdleProtocol
11
12  protocol.avg example.aggregation.AverageFunction
13  protocol.avg.linkabe lnk
14
15  init.rnd WireKOut
16  init.rnd.protocol lnk
17  init.rnd.k 20
18
19  init.peak example.aggregation.PeakDistributionInitializer
20  init.peak.value 10000
```

Figure 51 Code for a Peersim example.

The important directives in Figure 51 are lines 3, 4, 6 and 8, respectively: the **random.seed** is a parameter used by the simulator to replicate exactly the same results per each simulation creating a behavior that is taken as pseudo-random; **simulation.cycles** is a parameter that indicates the number of cycles the simulation will perform; **control.shf** that is a random method by which nodes are popped from the Network vector to execute their protocols. The specification **Shuffle** means that at each iteration the order of the nodes is different; **network.size** indicates the number of nodes initialized in the network. An example of initialization for a protocol is given by line 10. The declaration **protocol.lnk IdleProtocol** creates a protocol of the type *IdleProtocol* with name lnk. This label will be used further to interact with the protocol *IdleProtocol* for the current execution. The keyword **init** is used to initialize a value inside a variable, such as at line 15 with the variable WireKOut. To specify the order by which protocols are executed the user can specify the directive **order.protocol A B C** to tell the system that the right order of execution for the protocols inside of each node is A, B and then C. Controls are executed after the protocol C. Finally, the WireKOut protocol creates the overlay for the network by connecting randomly all the nodes inside it.

Interfaces In the Peersim simulator there are several interfaces used for different scopes. An important one is the **Linkable** interface that defines the system of neighbors. With this one a node can add and search a neighbor in the network or receive its degree. To remove some nodes from the network the user needs to specify an implementation for the original system since the initial interface does not deal with this functionality. Another interface is the **Control** interface that constitutes a set of controls used in the network to maintain a global knowledge of it. Controls can be of three different types: **Initializers**, **Observers** and **Dynamics** (see Figure 50). The first types of controls are executed at start-up of the simulation and are used to define the initial state of the network that is the initial state of nodes and the initial overlay. After this phase, the other two controls are executed periodically and respectively concerns the average degree of each node and their join and leave from the overlay network.

Example 5.1 – How the cycle-based engine works

Let us suppose to initialize the Network vector with one node maintaining three protocols, labeled as P1, P2 and P3. Let us also suppose to define two periodical controls, such as an Observer (O) and a Dynamic (D) control. The entire life of the simulation based on 5 cycles, is shown in Figure 52. First the initialization is executed and then all the protocols and the global controls. The only things the user can define over the controls are the order with which they are executed. In our example the order of execution is: Initialization, P1, P2, P3, O and D.

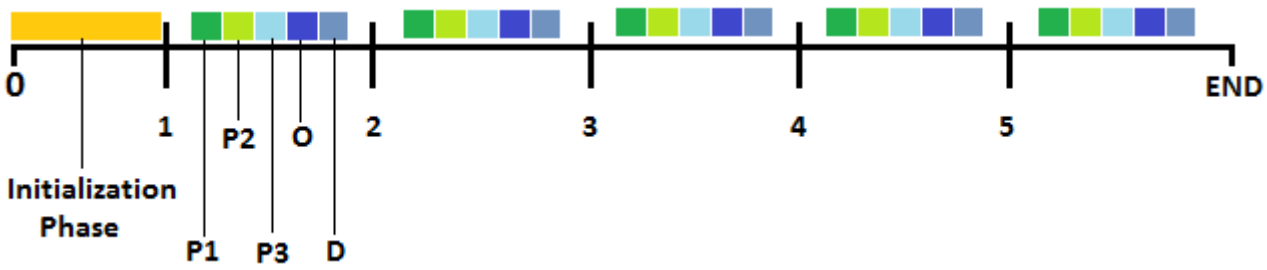

Figure 52 Example of the Cycle-based engine.

◊

Using the event-driven simulation, the system is taken as a set of cooperating nodes whose functionalities are used to provide a certain service. An example of this solution is the calculation of how many petrol pumps a gas station will need to serve twenty clients that have to fill up their tanks. At each step, the system is controlled for its state. Formally:

Definition 46: *A **state** for the system is defined by the values of the variables used in the system at a certain time.*

With the event-driven, basic entities are events that consist in some actions that modify the current state of the system, e.g. the ending of the supply of gasoline. With this variant the time is taken into account and handled by a clock, and it can be incremented **constantly** or with the verification of events. In the first case, a fixed interval of time is defined and the system progress with this interval. If an event happens in the middle of two intervals, it is moved at the beginning of the next nearest interval, thus providing a not so accurate simulation. Using the verification of events in place of the time, the system builds a stack in which all the events are inserted, ordered with their timestamp ordered from the oldest to the latest event in time, and executed in this order. In both the two variants Peersim considers as reference for the intervals of time the **tick**, that is a virtual instant of time more or less equal to 200 milliseconds. These ticks mark each execution of the simulation and the instant in which an event occurs. The user can specify the tick in which an event has to occur and events can be **periodical**, such as system updates, or **asynchronous**, such as random events like join and leave of nodes.
Along with the event-driven model we can specify some protocols that use the cycle-based engine. If this is the case, some protocols will be scheduled at a certain tick in the simulation and will be executed periodically at each interval of time specified by the user. When an interval associated to

the protocol expires, the engine will execute the code in the *nextCycle()* method. For each event, the user needs to specify the node and the protocol in which the event will take place, its delay before being scheduled by the Peersim Scheduler and the object that describes the event. If the event is periodical it is used as a cycle-based protocol, since the user needs to specify the first tick of execution and an interval at the end of which the protocol is executed again. Each node will have some code that indicates how to behave once the event is notified. The code is inserted in the *processEvent()* method of the **EDProtocol** interface that handles the event-driven engine. An example of events is the messaging system. Once a node needs to send a message to another node, the message is scheduled with its own timestamp, i.e. the tick in which the destination will receive the message. The *send* and *receive* operations are handled by the *processEvent()* code at the source and destination side respectively. Finally, the event-driven engine is used along with a simulator of the transport layer on which routing protocols are based. With this engine two basilar protocols are used: the **Uniform Random Transport** that provides a reliable system of message delivery with a random latency among the messages, and the **Unreliable Transport** that uses the random latency as before but does not provide a reliable delivery system, since messages are lost with a small probability. Other protocols can be specified by implementing the Peersim interface named **Transport**. To terminate the simulation using the event-driven engine the events stack needs to be empty or each event needs to have a timestamp greater than the end time specified by the user for the current simulation.

5.3 Simulation of the attacks

In this section we will test some of the attacks exposed in Chapter 3 and some new attacks using the Peersim simulator. The tests will be made over Pastry one of the protocols presented in Chapter 2, that is used e.g. in PAST, an archival storage extension for Pastry, SCRIBE, a group communication environment with event notification and in the Herald Project of Microsoft, to create a live event notification system **[37, 93]**. We will first introduce the structure of the protocol created for the Peersim simulator, some examples of its usage and then the old and new attacks.

5.3.1 A general overview of MSPastry

To test the new and old attacks to this protocol we have extended **MSPastry**, a Pastry implementation developed by Elisa Bisoffi and Manuel Cortella available at the Website of the Peersim project **[2, 94]**. The main differences between MSPastry and Pastry are in the way the network maintenance techniques and the routing mechanisms are used, the latter made much more reliable in this version. This version includes a system of correct lookup search that controls if the next hop for a message is truly the correct one, blocking incorrect behaviors of the routing mechanism.

Classes Several classes are available for this release. The first class is the **Message** class that defines the structure of a message in the protocol. This class is defined both for messages among nodes and service messages, i.e. those messages that are not visible by nodes. This class also

provides useful statistical surveys to classify the results. Connected to message delivery are the classes that concerns routing tables, defined by the **RoutingTable** and **Leafset** classes that define the routing tables and the leaf set of each node. This last class is combined with the **CustomDistribution** class that assigns random IDs to nodes inside the network. Node IDs can vary between 0 and 2^{128} as specified by the original protocol. After the ID assignment, the **StateBuilder** class is called to create the initial routing tables respecting the consistency properties of the network. An important class is **Turbulence** that simulates the join and leave of nodes and evaluates the performances even in case of high churn. The message delivery is handled by the **TrafficGenerator** class that chooses two random nodes and makes them exchanging messages during a simulation. Finally, the **MSPastryProtocol** class is the descendant of the EDProtocol class of the Peersim simulator and defines the structure of the MSPastry protocol, such as the routing mechanism with the *route()* method, thus the engine used is the event-driven one.

Setup The MSPastry package needs the basic packet of Peersim to work. To use the protocol we need to download both the Peersim package and the Pastry package **[94]**. Then we need to extract first the Peersim package in a folder that we will name **peersim-1.0.5**, and then we can extract also the Pastry package inside the latter folder. After this, we can open a terminal and point it to the test folder (see Figure 53).

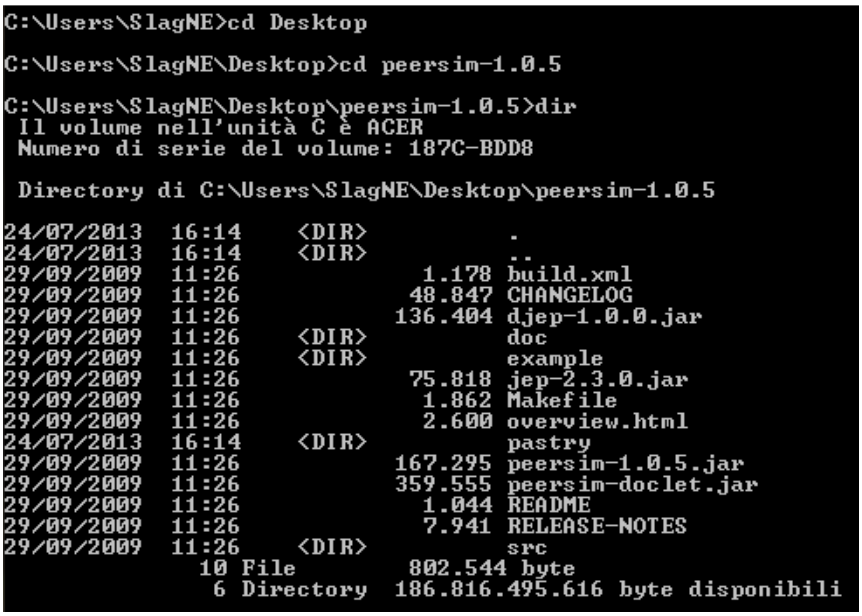

Figure 53 Output from the command line.

The next step is to add the path for the Java Compiler already installed in the system. Peersim authors recommend to use a version of the JDK equal to -or higher than- the 1.5.0. If this is not our case, i.e. the version of the JDK is lower than the 1.5.0, we can download it from the Oracle website **[81]**. Then, we can add the path to the Java Compiler inside the Computer folder, by clicking the right button of the mouse and select the voice *Property*. After this, on the menu on the left side of the newly opened window, we select *Advanced System Settings* and then the *Advanced* slice at the top. We then click on the button *Environmental Variables* and in the first text area, labeled with *Variable of the user*, we select the string *Path* and click on *Modify*. Now, we append

at the end of the string the value **;C:/Program Files/java/jdk1.6.0_06/bin** if the current version of the JDK is the 1.6.0 release 6[9]. After confirming the change, we need to reboot the system.

Execution On the terminal, that is pointed to the location of the test folder, we need to type the command to call the Java Compiler with the command **java –cp** where the directive –cp specifies that a class-path is needed to execute the java command, thus we need to specify the class-path for all the tools we will need to run the current simulation. If we take a look at the pastry folder inside peersim-1.0.5, we see that the classes for the execution of the package are contained in the folder **classes**. Along this folder we need to specify the three **Java Archives** (acronym **JAR**) in the test folder, namely *peersim-1.0.5.jar* (the Peersim core), *jep-2.3.0.jar* (the Java Expression Parser) and *djep-1.0.0.jar* (the Documentation for the JEP). The command until now will be like:

```
java –cp “peersim-1.0.5.jar;jep-2.3.0.jar;djep-1.0.0.jar;pastry/classes”
```

After this we will need to specify the path for the simulator of the Peersim package and the path for the configuration file for the current simulation, such as the one specified in Section 5.1.1. Let us suppose that the configuration file is named *example.cfg*, located in the *pastry* folder, and that the simulator is in the *src/peersim* folder. The complete command to run for the simulation will then be:

```
java –cp “peersim-1.0.5.jar;jep-2.3.0.jar;djep-1.0.0.jar;pastry/classes”
                peersim.Simulator pastry/example.cfg.
```

The system will run the configuration file and after few seconds will start to print the output. Since the output is composed by several rows, we need to save the output in a *txt* file. Let us suppose to put the *output.txt* file in the *pastry* folder. The command above changes by appending at the end the following string:

```
> pastry/output.txt
```

and we will obtain the output in that file. The output is composed by all the execution results specified in the configuration file except for some system outputs such as the time of execution.

Reading the Output The results for the simulation are composed by all the routing operations made by nodes in the network, such as lookup messages or other kind of messages. Let us look at one of the lines in the file:

```
[37e2-].route([type=MSG_LOOKUP][src:][dest:b6fc-][m.id=0]): [nexthop:b06b-]
```

This line specifies that the node with ID **37e2** is sending a lookup message (from the directive **MSG_LOOKUP** in the *type* field) to the node with ID **b6fc** specified in the *dest* field. At the end of the line is specified also the next hop the message will perform, that is node **b06b**. If we look further in the file we will find the line corresponding to the message forwarding by node b06b. Messages can be identified by their ID contained in the field **m.id**. When a message is delivered, a file in the output is inserted as the following:

```
[rr] Delivered message [m.id=0] [src=dest=b942-[in 1515 msecs] [3 hops]
```

[9] *The path for the Java Compiler may change depending on the operating system in use.*

The above line indicates that the message which ID is 0 and destined to node **b942** (in the field *src=dest=*) was delivered in 1515 milliseconds and required 3 hops. Messages can be of two types according to the actions performed by nodes in the network: MSG_LOOKUP and MSG_JOINREQUEST that indicate a join request by a new node in the network. Since the transport method used by the simulation is the unreliable one, it is possible that some messages are not delivered.

5.3.2 Simulating known attacks

In this section we will implement some of the known attacks to the Pastry protocol inside MSPastry: the **Wrong Forward** attack, the **Message Dropping** attack and the **Identity Theft** attack. In order to do this we will implement some code that will allow the nodes in the network to act as malicious. We will evaluate the rate of success of some attacks on the Pastry protocol and, in Section 5.3, we will propose and implement some known countermeasures. The visual representation of the effects of each attack will be obtained by using the **R** statistical software for statistical analysis on data **[110]**. Assuming that N is the population of active nodes in the network during the test, B is the value of the base used by the protocol to the ID system of the network and M is the fraction of malicious nodes in the network, for each of these attacks we will run different simulations with N=500, N=1000 and N=5000, B=2, B=3 and B=4 and M=N/8, M=N/4 and M=N/2. To recognize the malicious nodes against the good nodes we will introduce a new state for them. In Table 16 are represented all the possible states for a node in the Peersim simulator and their values.

State	Value	Description
OK	0	*The node is up and running*
DEAD	1	*The node is offline*
DOWN	2	*The node is temporarily unavailable*
MALICIOUS	3	*The node is malicious*

Tabella 16 Node states in the Peersim simulator.

Before running the simulation, we need to compile the configuration file, with the parameters **SIZE**, **random.seed** and **simulation.endtime** changed with respect to the original one (see Figure 54).

```
 8  SIZE 1000
 9  K 5
10  MINDELAY  500
11  MAXDELAY  900
12  CYCLE 500
13  TRAFFIC_STEP 1000
14  OBSERVER_STEP 2000
15  TURBULENCE_STEP 4000
16
17  # ::::: network :::::
18  random.seed 110489
19  simulation.experiments 1
20  simulation.endtime 1000*60*5
21  network.size SIZE
22  protocol.3mspastry.B 4
23  protocol.3mspastry.L 32
```

Figure 54 The configuration file for the attacks.

Wrong Forward This attack aims at increasing the message complexity of the routing algorithm. The malicious nodes in the network send messages to a different destination from the correct one, chosen at random. The effect of this attack can be measured in terms of number of hops performed on average by the messages sent through the network with respect to the number of malicious nodes inside it. Figure 55, 56 and 57 represent the results of the tests with N = 500, B = 4 and M = N/8, M = N/4 and M = N/2 respectively. We expect to find the average hop number for the message delivery system higher than the theoretical value, fixed to $\log_{2^B} N$.

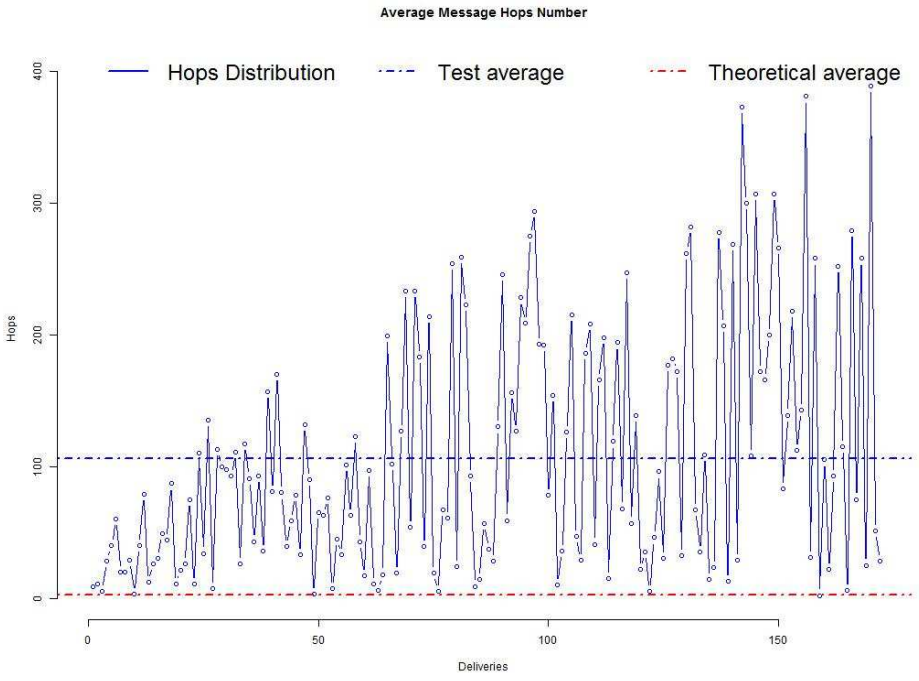

Figure 55 Results for the Wrong Forward simulation with N = 500, B = 4 and M = 63.

The red dotted line represents the value of $\log_{2^B} N$, thus with N = 500 and B = 4 the logarithm is equal to 2.24. The broken line connecting all the points in each figure represents the number of

hops made by each message delivered in the network, while the dotted colored line represents the average hop number of the messages.

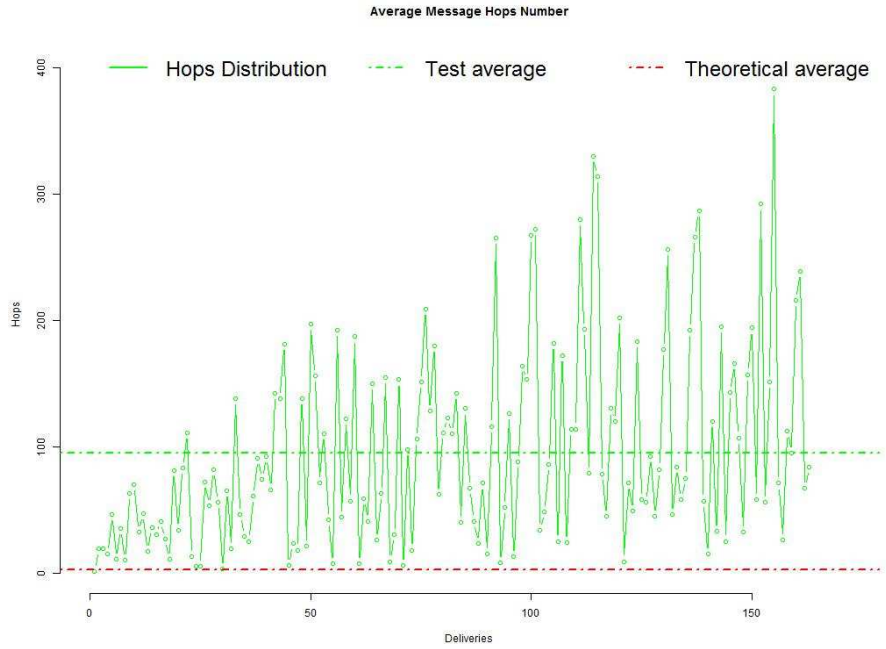

Figure 56 Results for the Wrong Forward simulation with N = 500, B = 4 and M = 125.

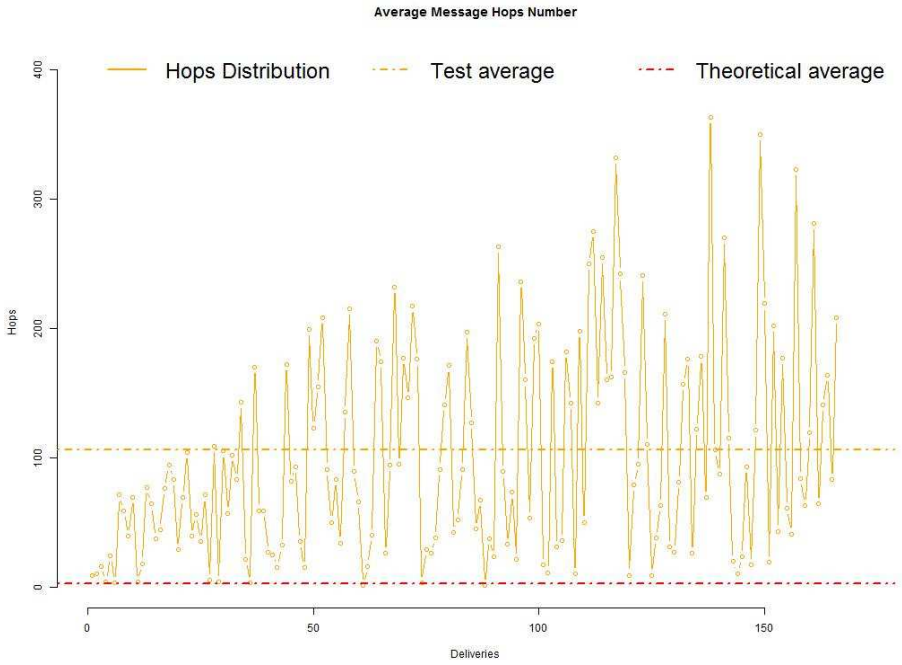

Figure 57 Results for the Wrong Forward simulation with N = 500, B = 4 and M = 250.

As we can see from the figures above, the situation does not change so much increasing or decreasing the number of malicious nodes in the network. The only difference is in terms of the average hop number per each simulation, that is higher for M = 63 and M = 250, while it is slightly lower for M = 125. To see if this is true, we can take a look at the results for the network

composed by 5000 nodes and with B equal to 4. This time, the red dotted line indicating the theoretical value for the logarithm is fixed to 3.

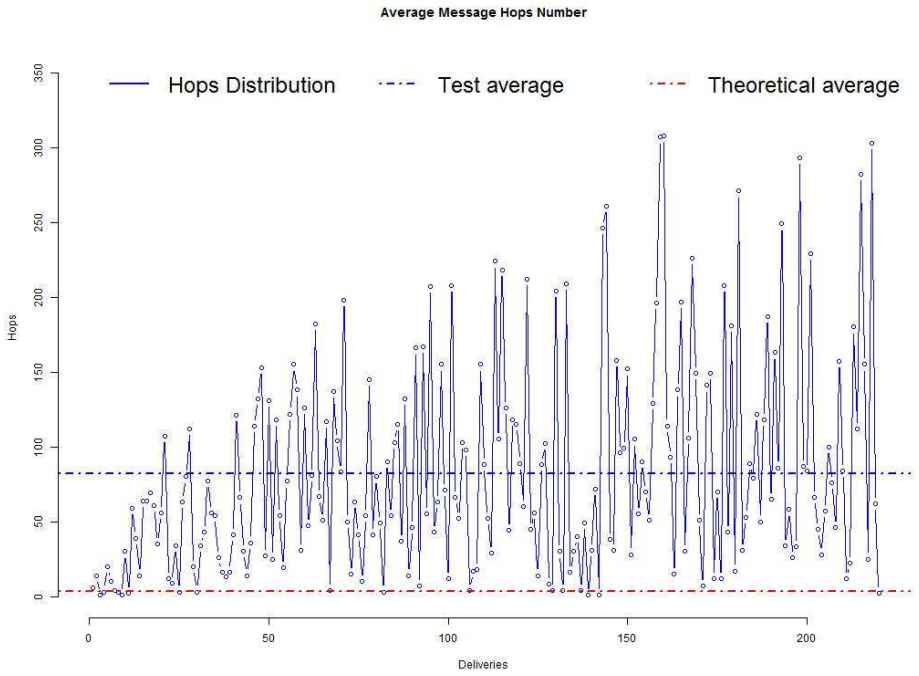

Figure 58 Results for the Wrong Forward simulation with N = 5000, B = 4 and M = 625.

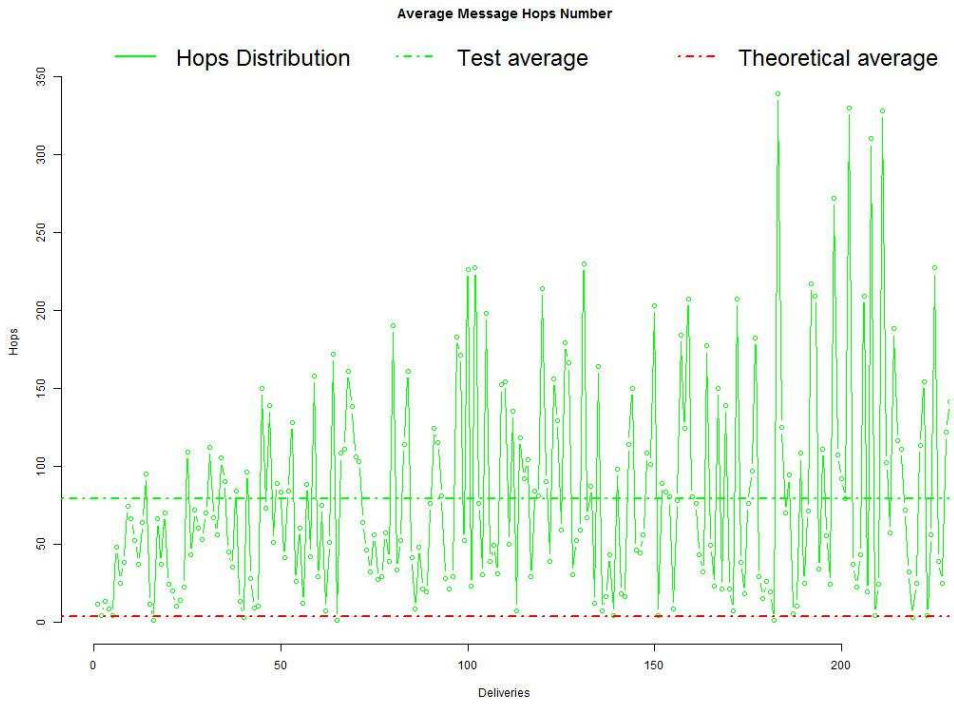

Figure 59 Results for the Wrong Forward simulation with N = 5000, B = 4 and M = 1250.

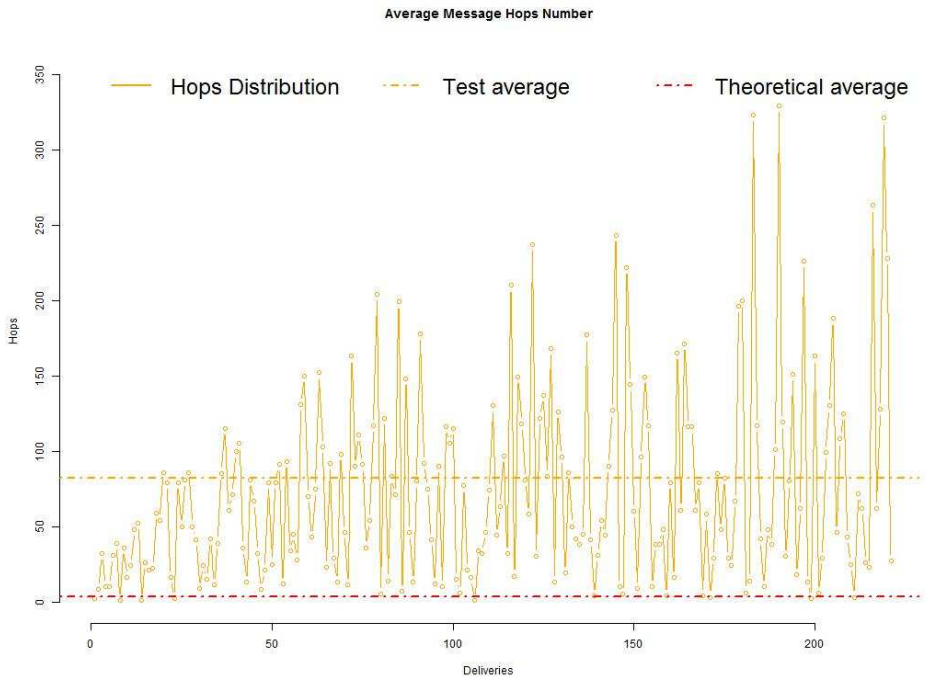

Figure 60 Results for the Wrong Forward simulation with N = 5000, B = 4 and M = 2500.

From these graphics we cannot ensure that the number of malicious nodes inside the network affects the average number of hops made by the messages sent through the network. The behavior is quite the same as the previous case and this means that the efficacy for this attack is a fact even with M = 1/8 of malicious nodes in the network. We can also take a look at the two variants of these tests in which B changes. Let us take a look at the same tests with B = 3, where the red dotted line this time is fixed to 2.95.

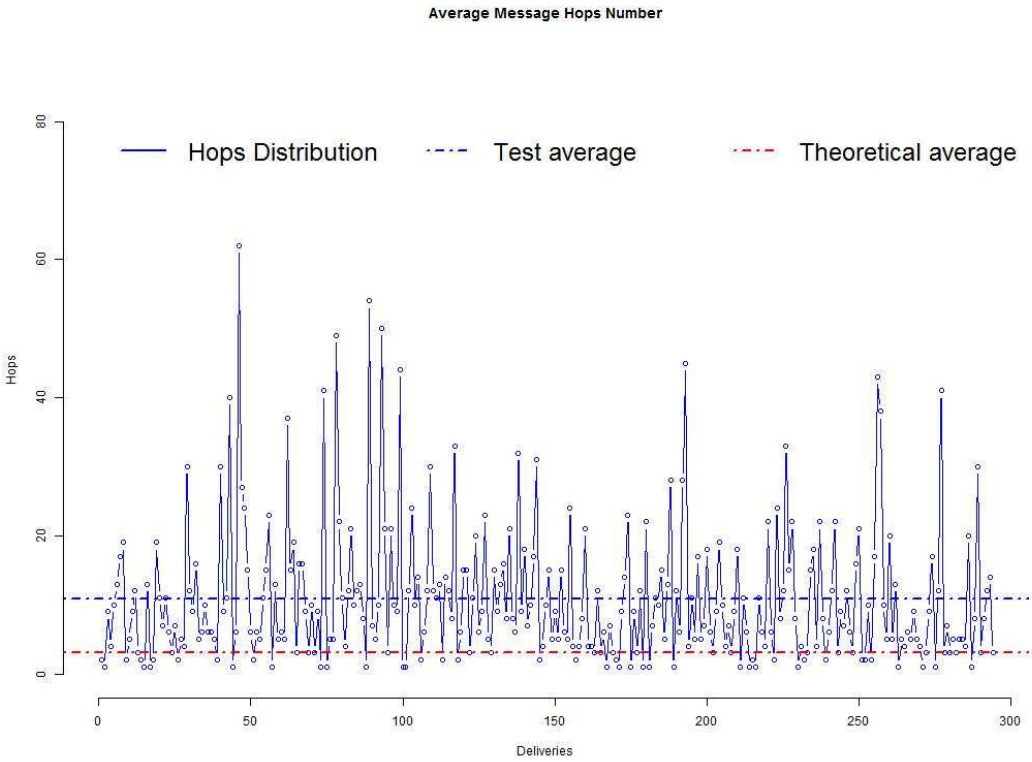

Figure 61 Results for the Wrong Forward simulation with N = 500, B = 3 and M = 63.

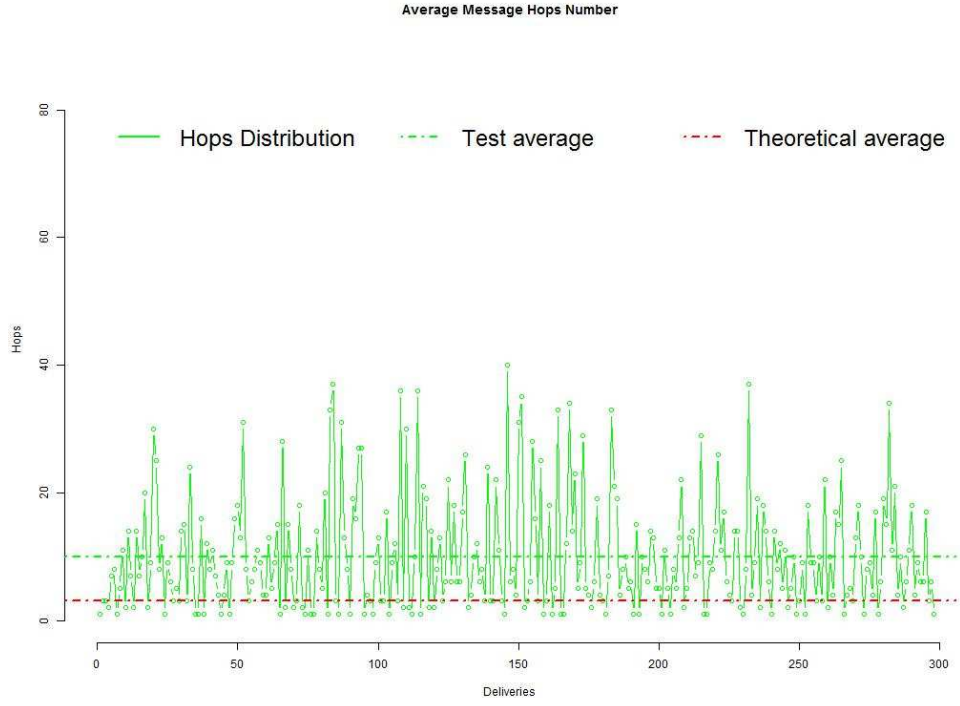

Figure 62 Results for the Wrong Forward simulation with N = 500, B = 3 and M = 125.

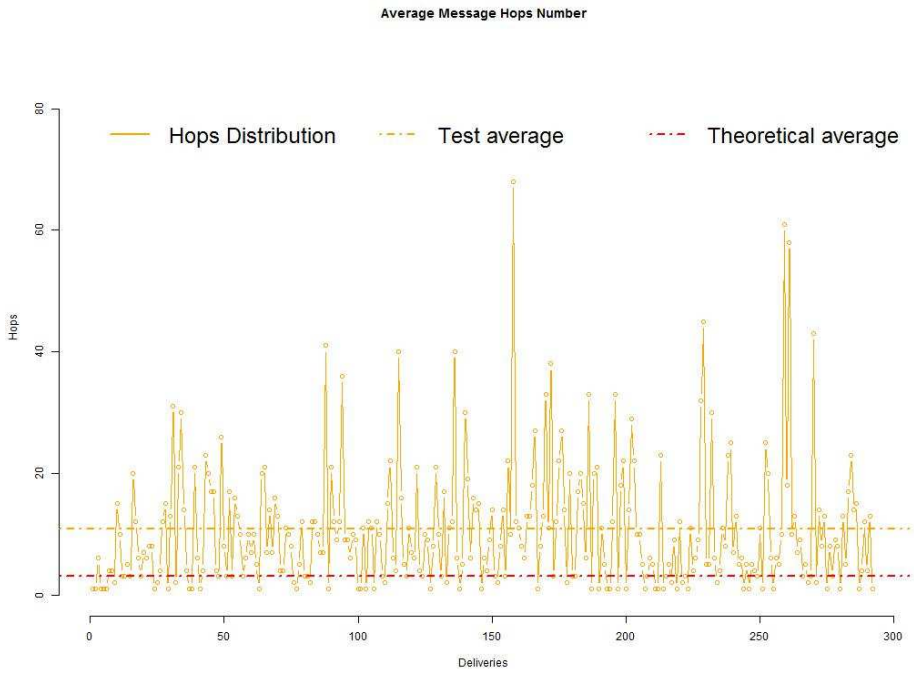

Figure 63 Results for the Wrong Forward simulation with N = 500, B = 3 and M = 250.

In the case with a population N = 500 results are quite different from before. The maximum number of hops in this case is around 70, with M = 1/8 and M = 1/2, while the average is near 10 for each test. Even in this case, the number of malicious nodes does not affect the value of the average but the value of B influences the number of hops made by the messages, since they make a lower number of hops in every case. Let us take a look at the behavior of the same test with N = 5000, and the value of the theoretical average equal to 4.09.

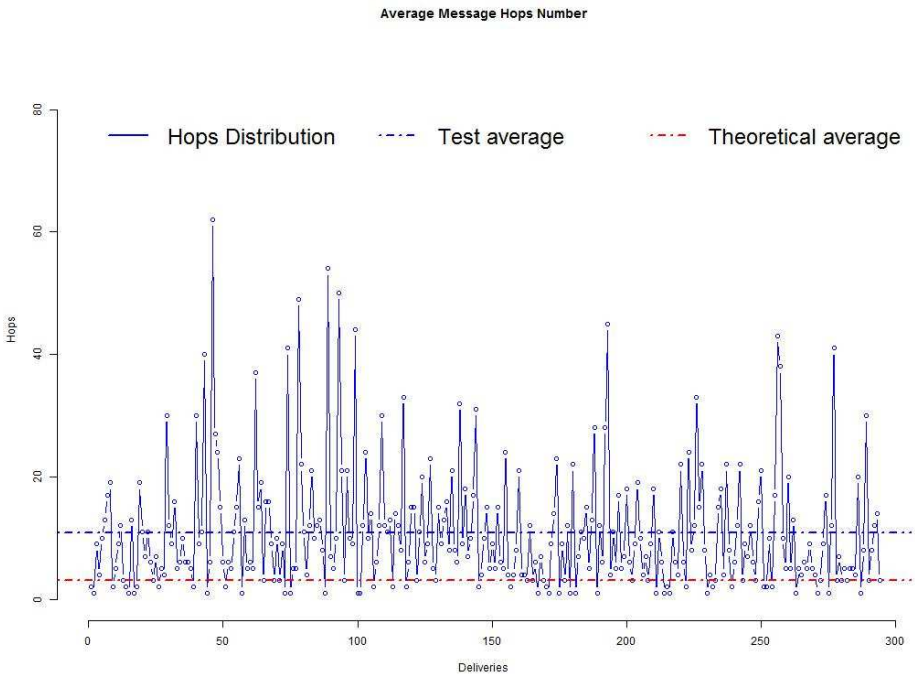

Figure 64 Results for the Wrong Forward simulation with N = 5000, B = 3 and M = 625.

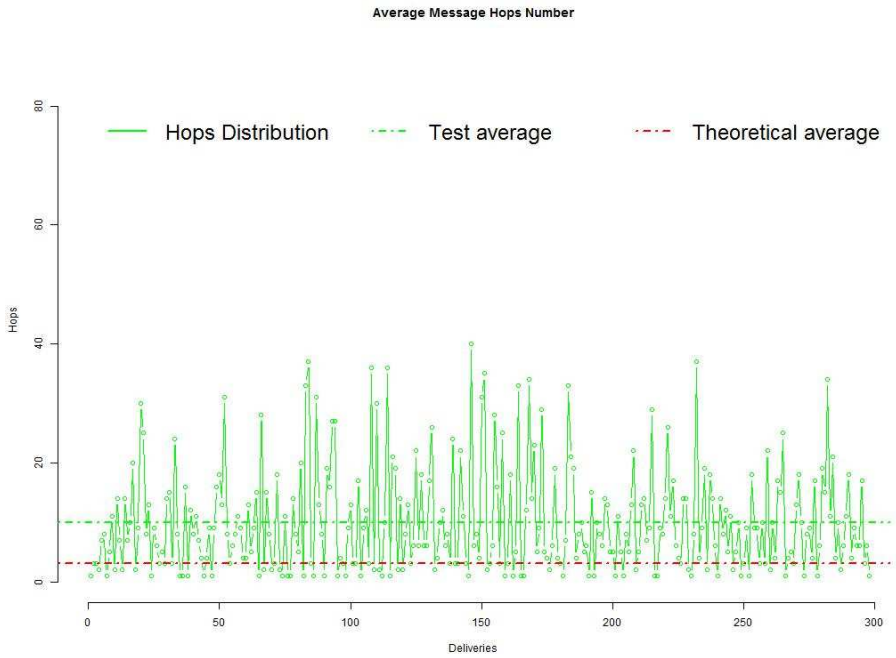

Figure 65 Results for the Wrong Forward simulation with N = 5000, B = 3 and M = 1250.

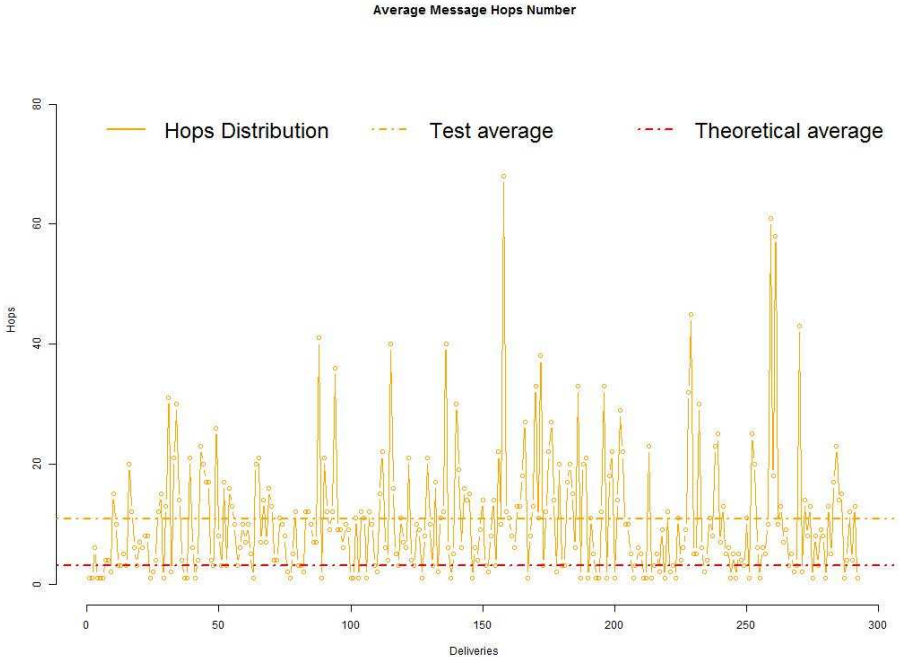

Figure 66 Results for the Wrong Forward simulation with N = 5000, B = 3 and M = 2500.

Even in this case the behavior is similar to the previous test. The first important result is that the behavior of the attack does not change with the increase of M, since for each test the average of the number of hops made by the messages is equal with the same population. An important results is that with B = 3, the average number of hops made by the messages is lower than that observed with B = 4, since with B= 4 an N = 500 the average number of hops is around 100 while with B = 3 and N = 500 the average line is fixed to 10.

To conclude, the increase on the number of malicious nodes, with respect to M = 1/8 * N, seems not to influence too much the behavior of the protocol. The decrease of B only affects the average hop number value that increases.

Message Dropping With this attack a malicious node that receives a message drops it and interrupts the forwarding process for that message, thus meaning that not all the messages will be correctly delivered in the network. To see the effects of this attack we will then count the number of sent messages versus the number of correct deliveries. We start presenting the results with N = 1000 nodes and B = 2.

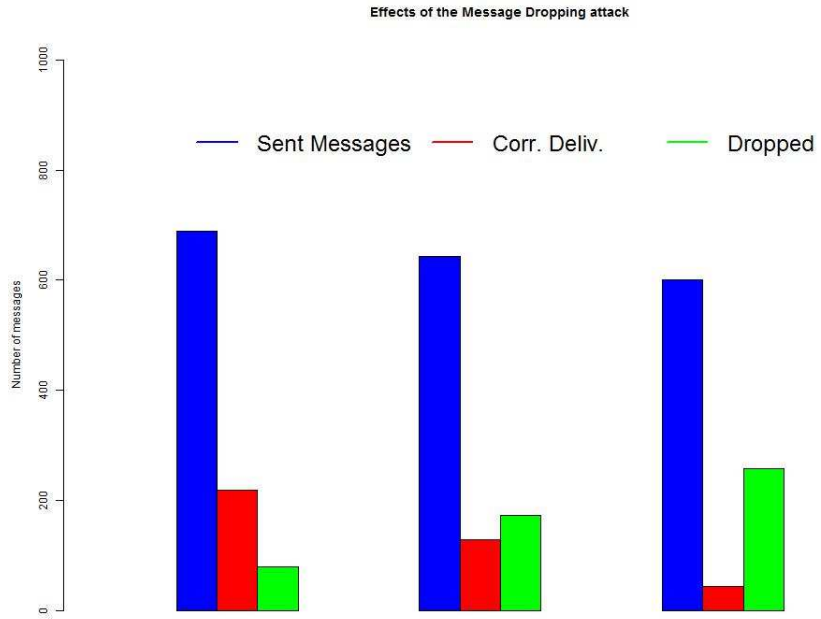

Figure 67 Effects of the Message Dropping attack with N = 1000, B = 2 and M = 125, M = 250 and M = 500.

From left to right, histograms are grouped by value of M, i.e. M = 125, M = 250 and M = 500. As we can see from the graphics, the number of currently active malicious nodes in the network affects both the number of correct deliveries and the number of dropped messages. To see if the parameter B influences in some way these results we can take a look at Figure 68 in which B is taken equal to 3.

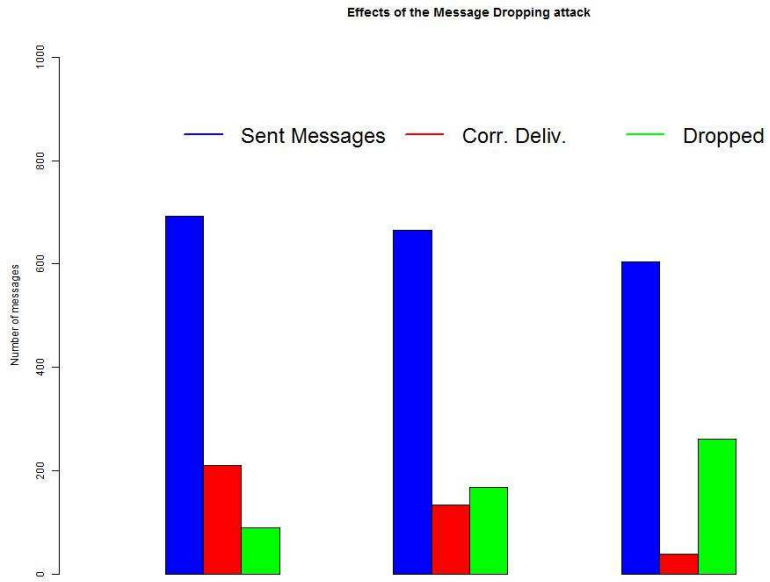

Figure 68 Effects of the Message Dropping attack with N = 1000 and B = 3 and M = 125, M = 250 and M = 500.

The results are very similar to the previous case, thus the value of B does not influence the effects of the Message Dropping attack. Let us see if the number of nodes inside the network, i.e. the sum of malicious and good nodes, affects the results, by setting it equal to N = 5000 and N = 500.

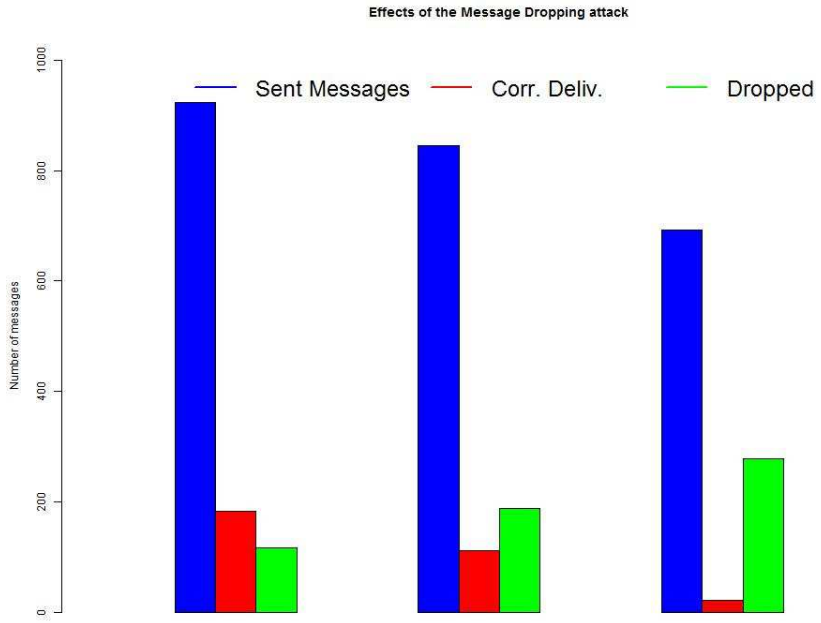

Figure 69 Effects of the Message Dropping attack with N = 5000 and B = 4 and M = 625, M = 1250 and M = 2500.

Figure 70 Effects of the Message Dropping attack with N = 500 and B = 4 and M = 63, M = 125 and M = 250.

From the graphics above we can say that the behavior of the attack remains quite similar for both the two cases. The only difference is in the number of correct deliveries versus the number of dropped messages in Figure 69 with respect to Figure 68, even if the difference is not so marked. This means that only the number M of malicious nodes affects the efficacy of this attack with the Pastry protocol. With respect to the population fixed to 1000, the percentage of dropped messages over the total number of delivered messages is 12% with 125 malicious nodes, the 25% with 250 malicious nodes and 35% with 500 malicious nodes.

Identity Theft This kind of attack is at the routing level and the malicious node that performs it claims to be the destination of a received message, even if it is not. This means to have some messages that are not forwarded correctly and that some messages in the network are blocked before reaching the real destination. The source node will not notify this event, since no confirmation on the receiving of the message is sent from the destination node to the source. On the side of the malicious node, this attack can be of no use or at least harder in the case that the protocol encrypts the content of the messages, since only the source and the real destination of each message are able to read the content. We can compare the number of sent messages versus the number of correct deliveries with respect to the increasing number of malicious nodes per each test. Figure 71 represents the results for N = 5000 and B = 4.

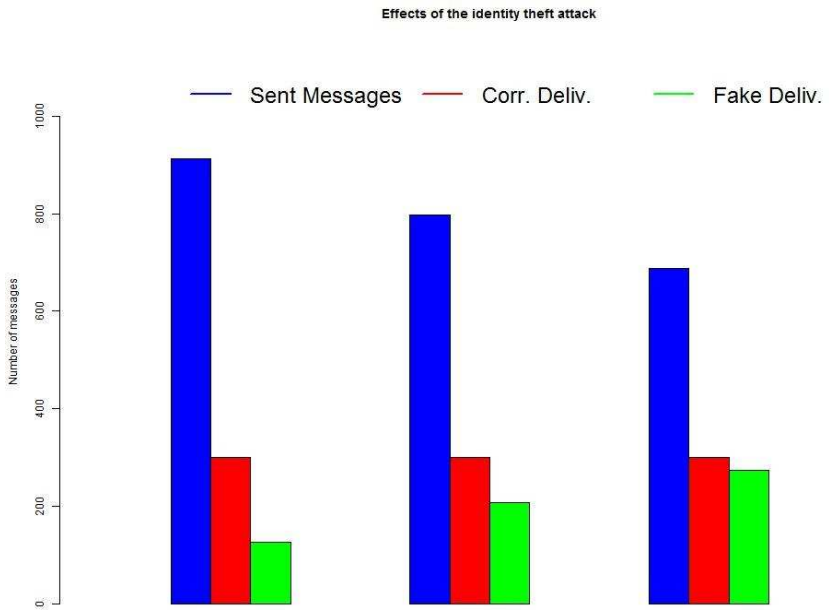

Figure 71 Effects of the Identity Theft attack with N = 5000 and B = 4 and M = 625, M = 1250 and M = 2500.

The number of malicious nodes in the graphics grows from left to right, i.e. M = 625, M = 1250 and M = 2500. The total number of deliveries is represented by the red bar and the green one is the fraction of fake deliveries. As we can see, increasing the number of malicious nodes increase the number of fake deliveries but decrease the number of messages sent through the network. We will now show the effects of changing the value B.

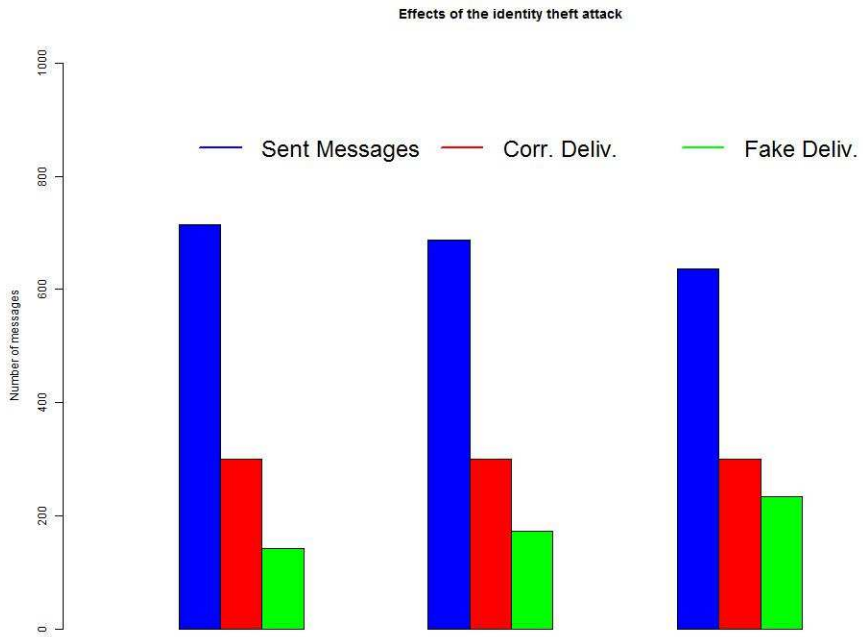

Figure 72 Effects of the Identity Theft attack with N = 5000 and B = 2 and M = 625, M = 1250 and M = 2500.

In this case, the value of B was set to 2. We can see that the fraction of fake deliveries with M = 1250 and M = 2500 is quite the same as the case with B = 4, while with M = 625 the number of

fake deliveries is smaller than that with B = 4. We can take a look also at the case with B = 3 to say if this parameter affects the efficacy of the attack.

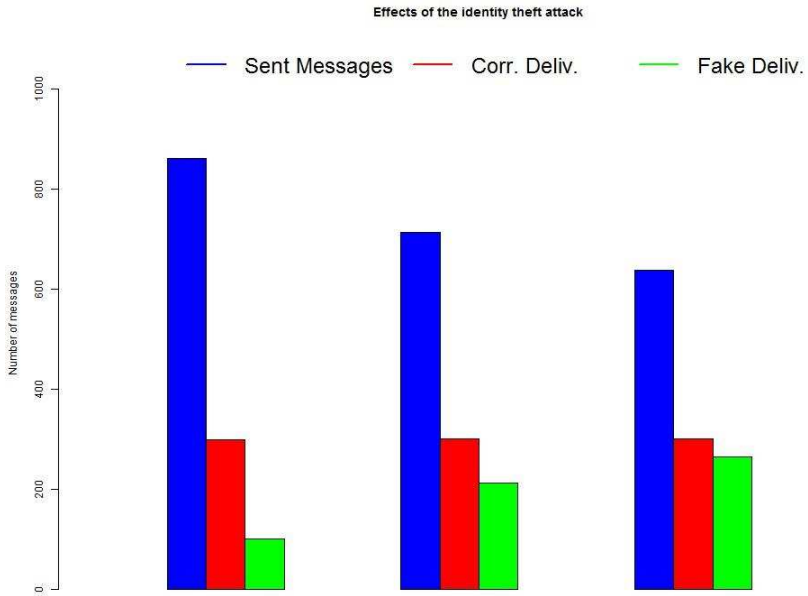

Figure 73 Effects of the Identity Theft attack with N = 5000 and B = 3 and M = 625, M = 1250 and M = 2500.

Even in this case the fraction of fakes deliveries follows a behavior similar to the case with B = 2, thus we can say that the parameter B affects in some way the efficacy of the attack, i.e. by lowering it the efficacy of the Identity Theft is lowered. Finally, we can see if the effects of the Identity Theft improves changing the number of nodes in the network, fixing the population to N = 1000.

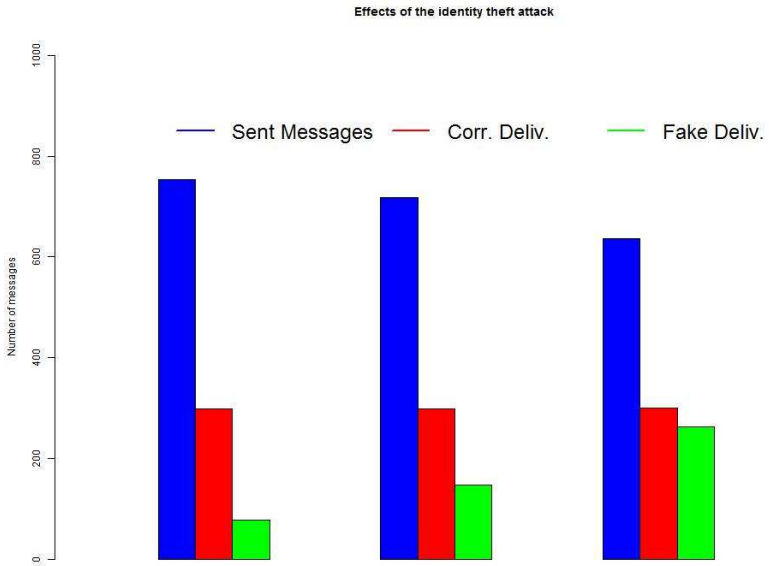

Figure 74 Effects of the Identity Theft attack with N = 1000 and B = 4 and M = 125, M = 250 and M = 500.

In Figure 74 we have the case in which the number of nodes N = 1000 and B = 4. The behavior seems to be the same as the case with N = 5000 nodes and B = 4 in Figure 71, thus the number of

nodes does not affect the efficacy of the identity theft. The only difference is in the number of fake deliveries but this is natural since the populations for the two tests are different, thus there are more nodes acting in Figure 71 that in Figure 74. Finally, taking as example the case with N = 1000 nodes, the fractions of fake deliveries are 20% with M = 125 malicious nodes, 50% with M = 250 malicious nodes and 90% with M = 500 malicious nodes.

Efficacy Evaluation The attacks just presented aim at disrupting the routing mechanism used by the Pastry protocol. Their efficacy can be graded in terms of damages they infer to this mechanism, in terms of average number of hops made by a message or if a message reaches the true destination. The Wrong Forward is very efficient in the Pastry protocol but it is clearly noticeable since it increases consistently the average number of hops made by the messages sent through the network. The Message Dropping is probably the most undetectable one, since the source cannot verify if its message arrived to the destination or not. Moreover, the fraction of dropped messages is not so consistent, thus the system cannot be sure that an attack is going on if, as in this case, it relies on an unreliable routing protocol. Finally, the Identity Theft attack is as stealthy as the Message Dropping, since the source node cannot check if the message reaches the correct destination, but it can be detected with more facility since the fraction of fake deliveries is very high. To conclude the study over the known attacks used against the Pastry protocol we can draw a conclusion in terms of effects of these attacks, imprinting a chart by which we can give a score to each attack. Ideally an attack inside a P2P network should be quite invisible, efficient, quick and requiring a small time to be written. A grade between 1 and 3 is given, in which 1 means poor and 3 means great. In the following we give a table in which all the attacks are evaluated under the just expressed grid (see Table 17).

Attack	Invisible	Efficient	Quick	Speedy	Av. Grade
Wrong Forward	1	3	3	2	2.25
Message Drop	3	3	3	3	3
Identity Theft	2	3	3	2	2.75

Table 17 Evaluation of the known attacks.

The best attack among all the three is thus the Message Dropping. In particular, without any ad-hoc control it is very invisible and efficient since it reach its goal in no-time and is very quick to mount inside the network, i.e. the performances of the network are not changed dramatically. So, this attack has to be considered the best against the Pastry protocol.

5.3.3 Simulating new attacks

In this section we will present some new attacks never tested before on the Pastry protocol. In Chapter 3 we have seen that P2P networks are exposed to certain weaknesses exploited by malicious users inside the network. In Pastry, messages are not formatted under a specific format, thus in theory a malicious node can arbitrarily send a malicious content inside these messages. This form of attack is very similar to a Trojan Horse, because it presents itself as a valid entity of the network, i.e. a valid message, but it contains some evil code. In the following we will present three theoretical variation of this attack: the **Colonizer**, the **Silent Murderess** and the **Sniffer**.

Before introducing all of them, we need to specify the parameters for the tests. We will test these attacks varying the parameters N, B and M as the test made for the case of known attacks. We will exclude the dynamicity of the network, i.e. we will analyze the network in an interval of time in which it does not change. The number of nodes can only be changed by the action of the set of malicious nodes. Messages are supposed to be free in content, i.e. that valid nodes send normal messages while malicious nodes send messages with an evil content that changes from attack to attack. The content of each malicious message is taken fix and equal to a special string. In real environments, this string can be ideally replaced by some malicious code that has to be run from the application using the routing protocol at the destination side. When a valid node receives a message containing the malicious code, it will act accordingly to the actions described by the code. This will resemble very well the virus we can find in an operating system that, once ran produces malicious actions. Finally, we modified again the simulator to introduce the new state POISONED with value 4 to recognize those nodes poisoned by the Sniffer attack.

The Colonizer This attack is like a virus and aims at spreading itself through the entire network. Every malicious node sends an evil message to another node at each cycle of the simulation, containing the string **colonized** that says to the destination to start acting maliciously. This means that the destination will become a malicious node and will start propagating the virus inside the network. This attack can be a starting point for large scale Denial of Service or an Eclipse attack. To realize this attack, we need again to modify the **TrafficGenerator** and **MSPastryProtocol** classes. At first we will modify the **execute** method by inserting a piece of code that selects all the malicious nodes inside the network and makes them sending one malicious message each per each cycle. In the second class, we will modify the **receiveRoute** method, by controlling the content of all the received messages. Once the destination receives the message, the malicious code starts running and the destination become a malicious node. The conversion of the state of the node can be seen as temporary, i.e. that the good node is only used for a certain time by a malicious endpoint to perform evil actions, or permanently, i.e. the good node becomes malicious for all the duration of its permanence in the network. If the evil message reaches a malicious node, the effect is idempotent since the destination is already a malicious node, thus it does not change its state. To see the effectiveness of this attack, we can make the same comparison made for The Silent Murderess checking how many messages are needed for the set of malicious nodes to colonize all the valid nodes in the network. In Figure 75 shows the results for a population of N = 1000 nodes and a value of B = 4.

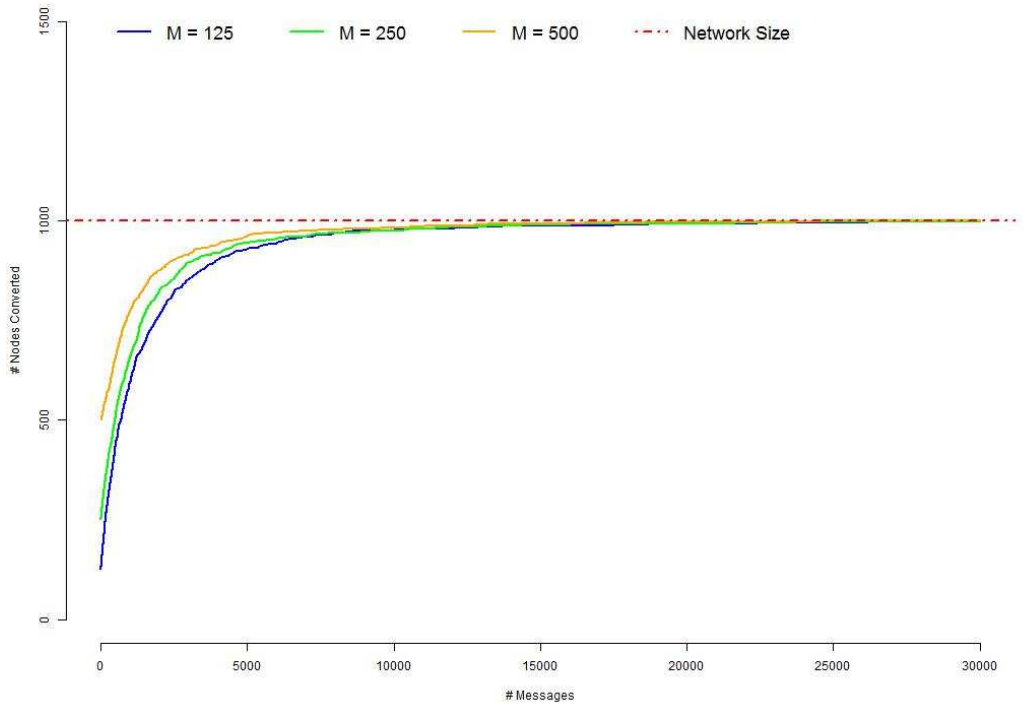

Figure 75 Effects of the Colonizer attack with N = 1000 and B = 4.

In all the three cases, in which the number of malicious nodes is M = 125, M = 250 and M = 500, the number of messages needed for the set of malicious nodes to colonize all the valid nodes is near 30,000. This means that a good effort needs to be made by the set of malicious nodes to conquer the entire network, even if their number increases at each cycle. We can run the same test with the value of B = 2 to see if it affects the efficacy of the attack.

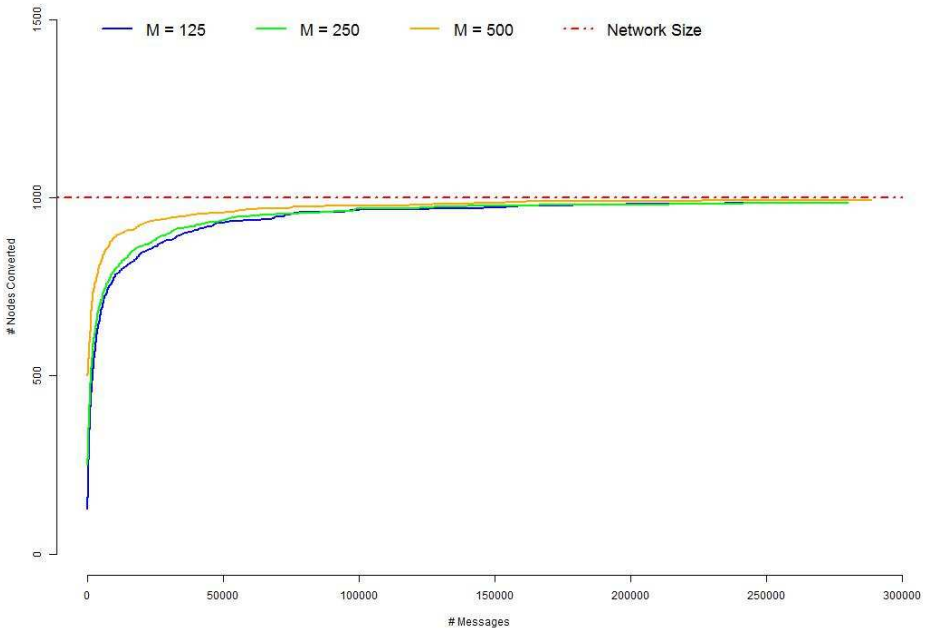

Figure 76 Effects of the Colonizer attack with N = 1000 and B = 2.

As we can see, with a lower number of B each variation of the attack will need a higher number of messages to colonize all the nodes, around 250,000 that is ten times higher than the previous case. We can also take a look at the following graphics in which N = 500 to see if with less nodes the behavior is the same.

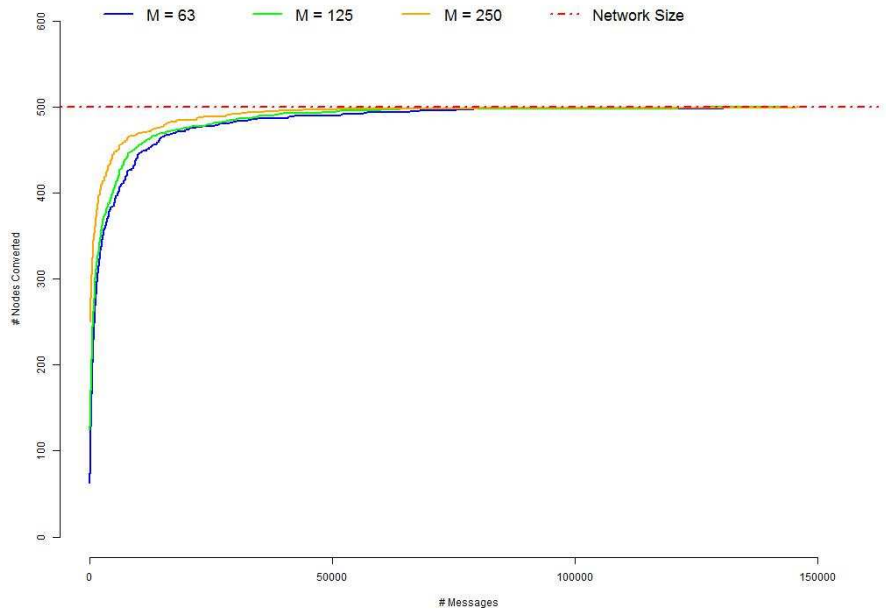

Figure 77 Effects of the Colonizer attack with N = 500 and B = 2.

With B = 2 and N = 500 nodes, the behavior is quite the same as the previous case. This means that the population of the network does not affect the efficacy of the attack.

The Silent Murderess This is an extreme case of application of a Trojan inside a network. With this attack every malicious node in the network sends at each cycle of the simulation a message containing the string **killyourself** to another host in the network. If the message reaches another malicious host, the message will not be computed by the destination. Otherwise, if the message reaches a good node, the latter will be disabled instantly. The force of this attack is that it cannot be detected by another node other than the destination, since only the source and the destination for a message can read its content. Moreover, without proper precautions, the destination node cannot be able to block the action of the malicious code. To realize this attack, we modified the **TrafficGenerator** and **MSPastryProtocol** the classes. In the first one, we will add to the **execute** method a piece of code that selects all and only the malicious nodes in the network and makes them sending one message each containing the evil string. Since the generation of the destination node is random and determined by the underlining protocol, we are not sure to reach a good node in all the tries. In the second class we will modify the **receiveRoute** method by which a node handles the receiving of a message. Once the node opens the message just received, it checks its content and if it is equal to the evil string, it will disable itself. We can evaluate the impact of this attack in terms of nodes disabled during the entire simulation, comparing the results among all the simulations in terms of number of malicious nodes in the network. We start showing the effects of the attack with a population of N = 1000 nodes and the parameter B = 4.

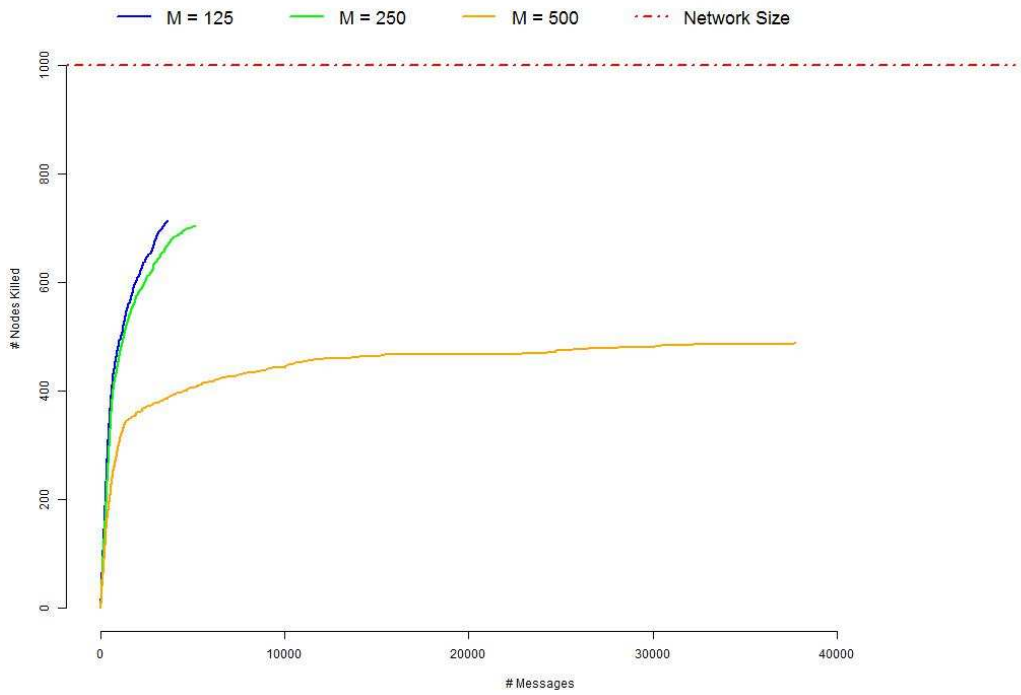

Figure 78 Effects of the Silent Murderess attack with N = 1000 and B = 4.

In the graphics above, the blue line represents the effects of the attack with M = 125, the green line represents the effects of the attack with M = 250 and the orange line the effects of the attack with M = 500. As we can see, the first two cases rapidly reach the number of malicious nodes in the network, thus they rapidly disable all the valid nodes with less messages than the third case. This latter takes more messages to reach the level of malicious nodes in the network, probably because there are less valid nodes to disable than the other two cases. We can test if the parameter B affects the efficacy of the attack by looking at Figure 79.

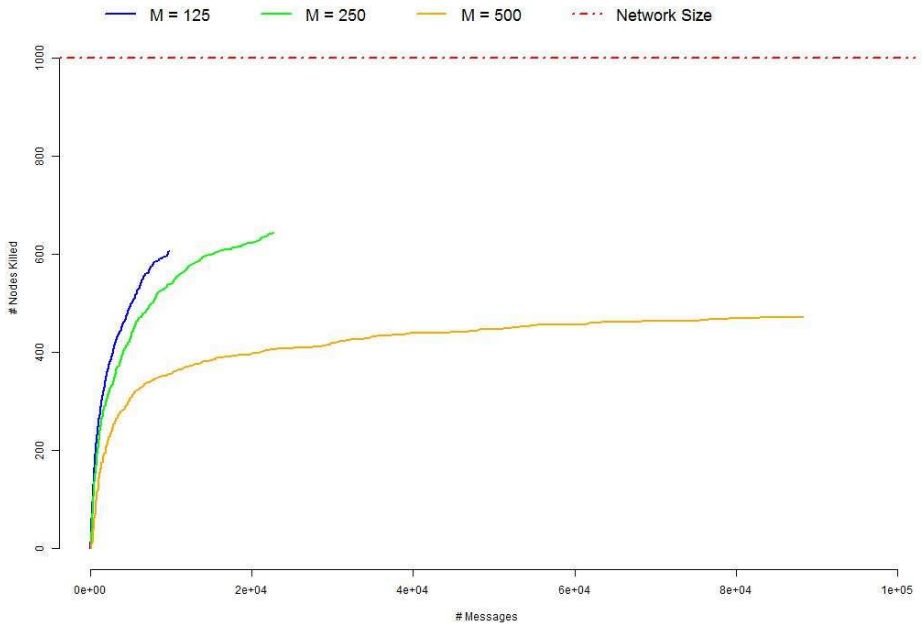

Figure 79 Effects of the Silent Murderess attack with N = 1000 and B = 2.

As we can see, the behavior of all the three cases is slightly different from before, since the curves are more elongated and they reach their maximal value in a longer time. This is a symptom that the parameter B affects in some way the efficacy of the attack. In all the three cases each attack reaches the number of malicious nodes slowly than before. As final test we can look at the same attack produced in a network of N = 500 hosts.

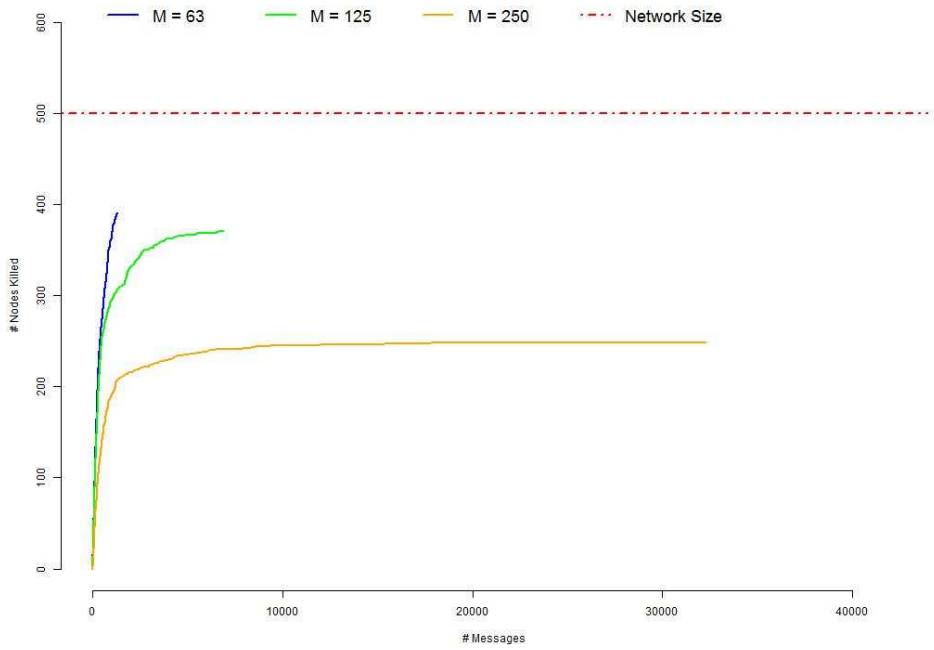

Figure 80 Effects of the Silent Murderess attack with N = 500 and B = 4.

Here, with B = 4, the behavior is similar to the case in which the network had 1000 nodes. The main difference is that the blue line descends more rapidly than the first test, like the green line does. The orange line conserves its initial behavior, reaching rapidly the number of malicious nodes and maintaining it constant. We can take a look at the same case with B = 2 to see if even in this case this parameter affects the results.

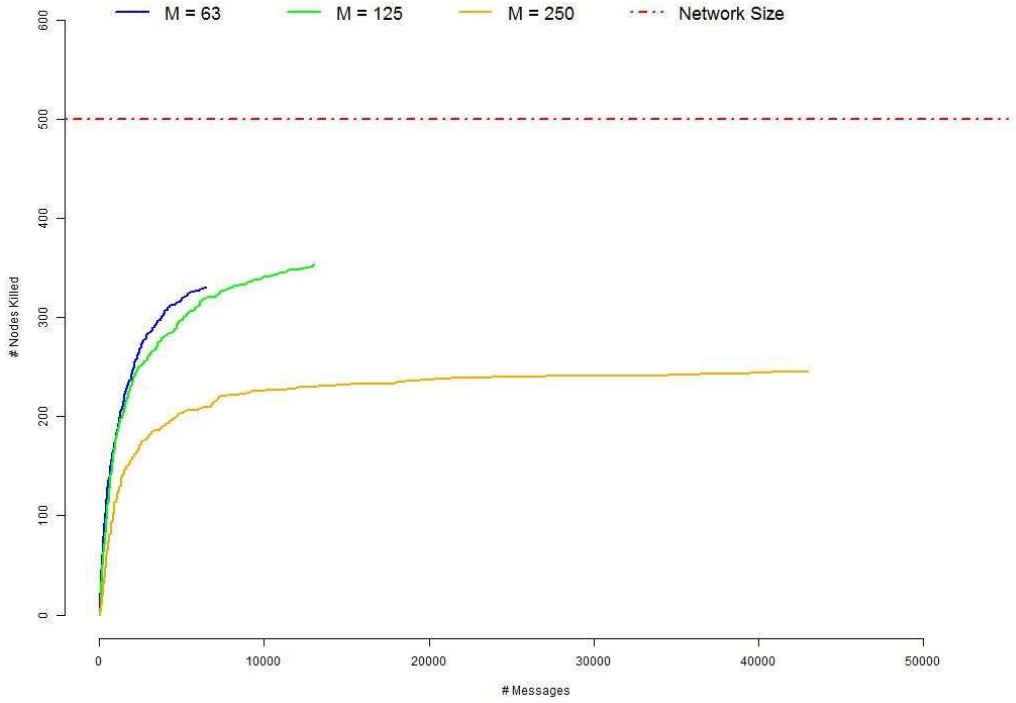

Figure 81 Effects of the Silent Murderess attack with N = 500 and B = 2.

Looking at the graphics above, we can say that also in this case the parameter B affects the efficacy of the attack, since each of the three variants presents a descending behavior less steep than the case in which B was equal to 4. This means that both the number of malicious nodes and the value of B affect the efficacy of the Silent Murderess in terms of messages needed from the set of malicious nodes to disable the valid nodes. With the increase of the fraction of malicious nodes, the attack will need a lower number of messages, while with lower values of B the attack will need more messages to succeed.

The Sniffer The third new attack uses the same behavior of the two previous attacks, in order to spy a certain good host in the network. A malicious node envelops a message containing the malicious string **noticeme** that, once received by the destination, creates a connection between the two hosts. The connection is not physical, but once the poisoned node receives a message, it will read it and send a copy to the malicious node. This can be a good workaround to avoid the fact that only the source and the destination for a delivered message can read its content. The malicious user will thus be able to read each message sent to the attacked node, and this can be very bad in real environments. To use this attack, we modified yet again the **TrafficGenerator** and **MSPastryProtocol** classes as for the two previous cases. This time, we will also add a new structure created by each node that is first poisoned by a malicious node that will contain all the

contacts of the malicious nodes that have contacted the current node. This structure will be useful for the poisoned node once it will have to send a copy of a received message. Malicious nodes are not interested by the poisoning mechanism, since if they receive such a message they will discard it. The actions performed by a node once received an evil message are:

1. Creation of the new structure and insertion of the source of the message just received;
2. Changing its state from 0 to 4, that means that the current node is poisoned;
3. Turning back on listening for incoming messages;

Once the poisoned message starts receiving the first message after the malicious one, it will start sending a copy of it to all the malicious nodes that has contacted it. To avoid an eventual cryptography of messages, the poisoned destination will first decrypt and then send the copy of the message, since without this specification the attack will be of no use. For this attack we only tested the case in which B was equal to 4, since the simulator was not working with B = 3 or B =2. We can evaluate the effects of this attack in terms of number of nodes poisoned, number of messages copied to malicious locations and scale-effect of the attack increasing the number of malicious nodes in the network. We will first present the results for the case of a population of N = 500 nodes.

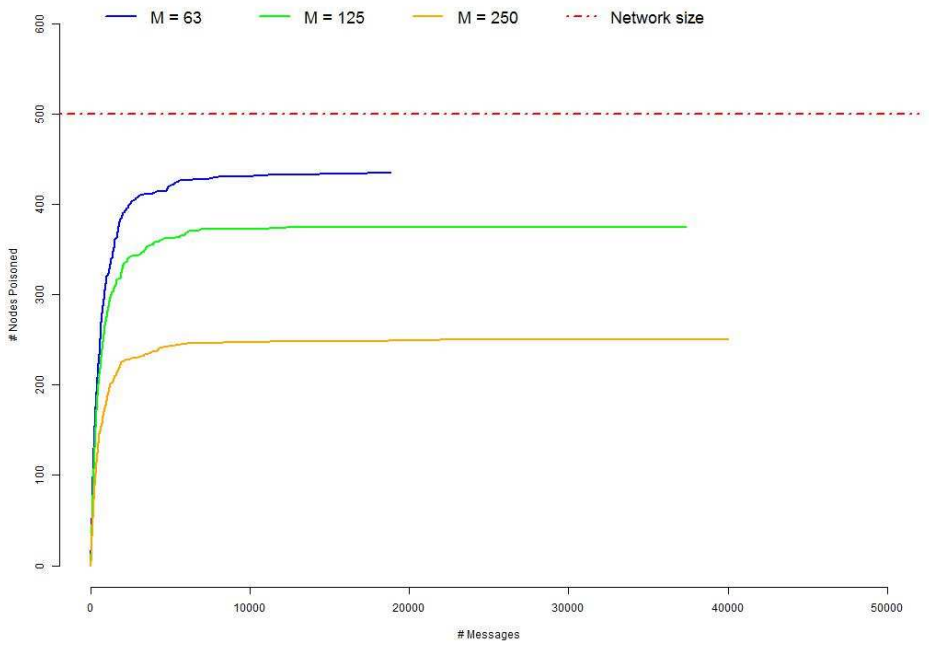

Figure 82 Effects of the Sniffer attack with N = 500 and B = 4.

As the previous examples, the blue curve represents the attack made by M = 63, while the green curve represents the attack made by M = 125 and the orange curve represents the attack made by M = 250. Finally, the red dotted line represents the size of the network. All the curves represent the number of poisoned nodes during the execution of the simulation, measured on the y-axis. On the x-axis we have the number of messages sent from the malicious nodes to poison the good nodes. As we can see from the graphics, the growth of poisoned nodes is sudden in all the three

cases. Moreover, for all the three tests the number of messages needed to realize the scope of the attack is around 20,000. We can take a look to the cases in which the population was equal to N = 1000 and N = 5000 to see if this factor is important.

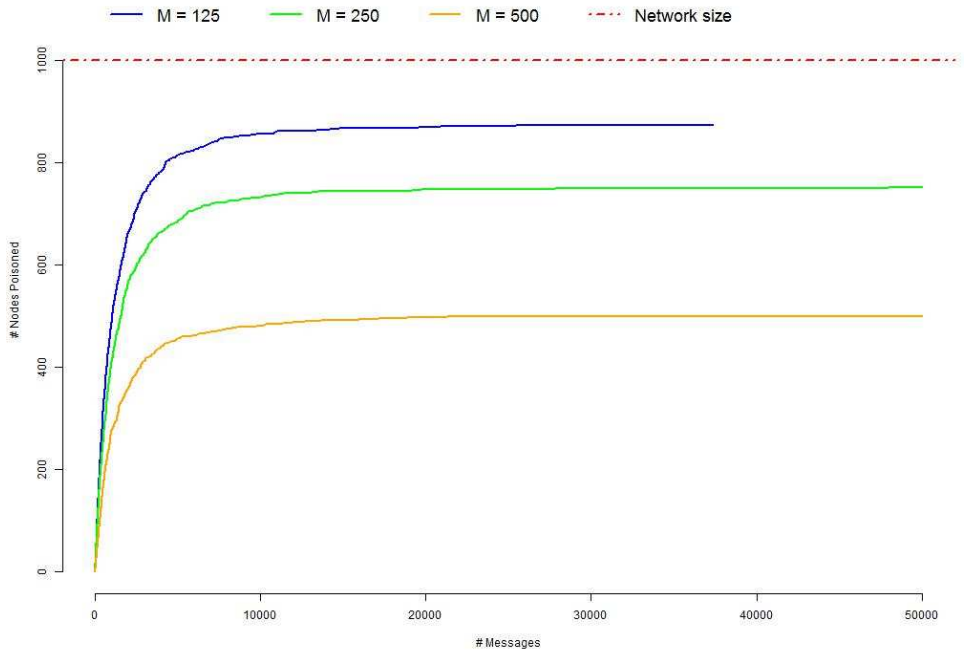

Figure 83 Effects of the Sniffer attack with N = 1000 and B = 4.

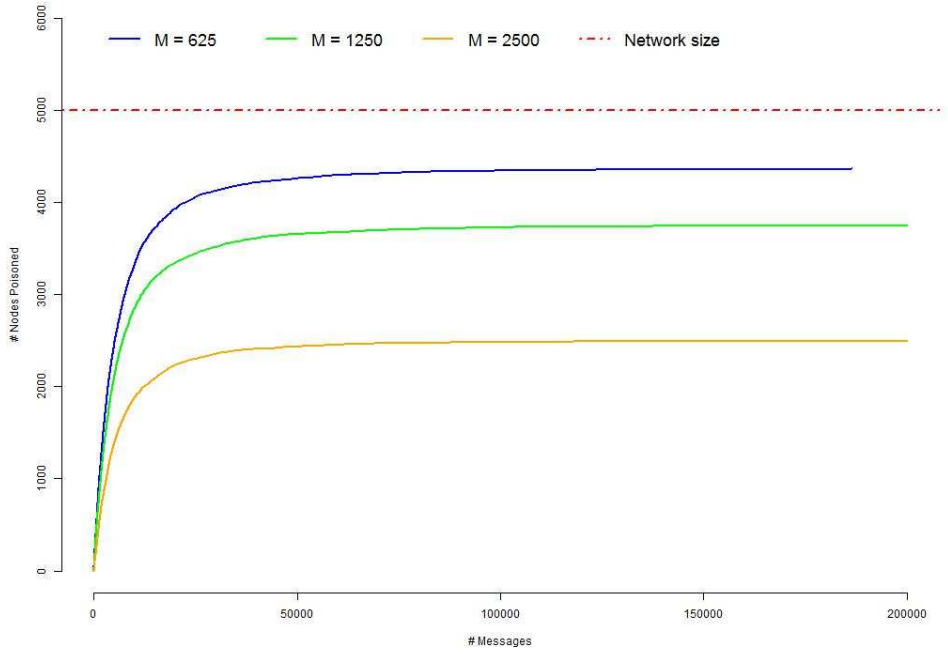

Figure 84 Effects of the Sniffer attack with N = 5000 and B = 4.

Results are not different from before in the last two cases, and this means that the population of the network is not a factor that affects the efficacy of this attack. As single method of comparison,

we can compare the number of messages sent through the network, the number of malicious messages and the number of messages copied to malicious endpoints for all the three cases.

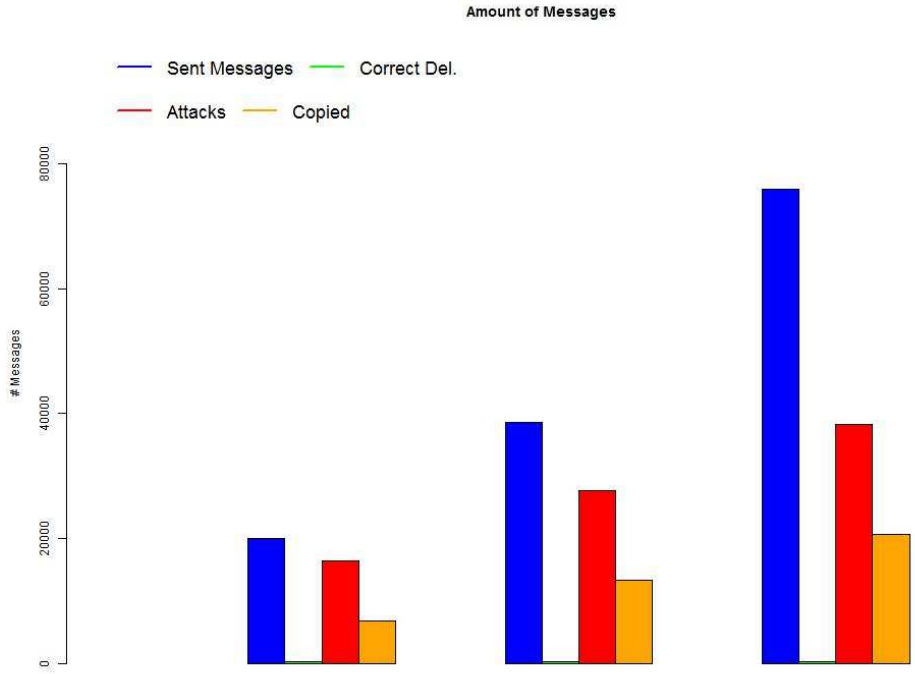

Figure 85 Number and types of messages sent from malicious nodes with N = 500 and B = 4.

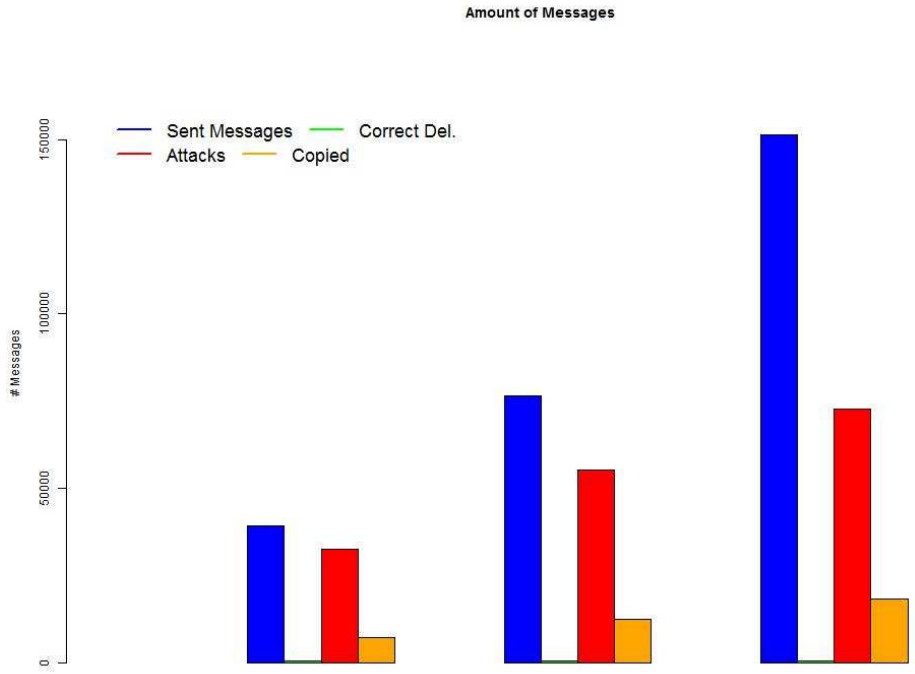

Figure 86 Number and types of messages sent from malicious nodes with N = 1000 and B = 4.

The blue rectangle represents the total number of messages sent through the network, the green rectangle represents the number of correct deliveries, the red rectangle the number of messages containing the malicious code and the orange rectangle represents the number of messages copied to malicious destinations. In all the three graphics the number of correct deliveries is different from zero despite the intuition we can have from the graphics.

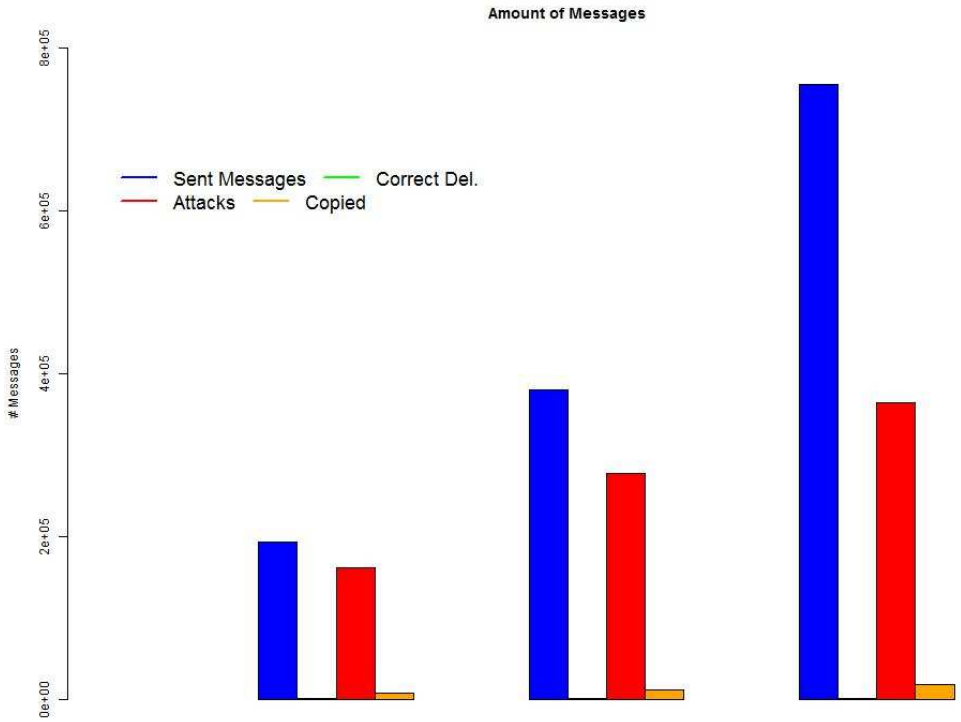

Figure 87 Number and types of messages sent from malicious nodes with N = 5000 and B = 4.

Taking as example the Figure 85 in which the population of the network was equal to 500, we can see that the fraction of attacks is consistent but most of them were lost during the process, since the number of copied messages is not so consistent. This means that many attacks reached a malicious endpoint, thus they were discarded. The behavior of this attack highlights a massive amount of messages sent through the network and a small number of messages copied to malicious endpoints. While in the case of N = 500 the ratio between malicious messages and copied messages is equal to 1/2, in the other two cases this ratio falls to 1/4 in the case of a population of N = 1000 nodes and to 1/10 with a population of N = 5000 nodes. This means that the population of the network affects the efficacy of the attack in terms of hit ratio, since less copied messages are sent to malicious endpoints with a larger population.

Efficacy Evaluation Assuming that a malicious node can send messages with some malicious code inside them, these three attacks are very strong against the Pastry protocol and with all the other protocols. A routing protocol does not check the content of each message before travelling it to the destination, also because a message is intended as secret, thus readable only by the source and the destination of the message. These three attacks assume no pre-formatted structure for the messages inside the protocol such as for the case of Gnutella or Napster (see Chapter 2), thus

this allows a malicious node to insert an arbitrary content inside them. To conclude we can give a chart on the evaluation of the new attacks as made for the case of the known attacks, with Table 18.

Attack	Invisible	Efficient	Quick	Speedy	Av. Grade
The Colonizer	3	3	3	3	3
The Silent Murderess	3	3	3	3	3
The Sniffer	2	3	3	3	2.75

Table 18 Evaluation of the new attacks.

The Colonizer and the Silent Murderess attacks seem to be the best since they scored the maximum grade on all the columns. The Sniffer takes a score equal to 2 in the Invisible column since it increases the number of outgoing messages to the destination of the attack.

5.4 Countermeasures against the attacks

In this section we will present some countermeasures adoptable against the attacks shown in Section 5.2. For the known attacks we will only revise some of the countermeasures presented in Chapter 4 talking about their applicability in Pastry, while for the new attacks we will present some countermeasures and test their efficacy on a network of 1000 nodes with 1/8 of them acting as malicious.

5.4.1 Countermeasures against known attacks

In this paragraph we will revise some countermeasures presented in Chapter 4 talking about their applicability on the Pastry protocol. We will start with the Wrong Forward attack.

Wrong Forward A valid countermeasure for this attack can be to enforce the routing mechanism in order to prevent that a message can be forwarded to a node too far from the real destination or such that the distance between the current node and the destination of the message is increased. A solution of this kind, that can be extended to the case of the Pastry protocol, is the one implemented for the Chord protocol, namely **RChord** presented in Section 4.3.2. With this solution, the routing is performed both in clockwise direction and in counter-clockwise direction in the ring. Once a node receives a message it checks if the destination was already surpassed by the message and, if this is the case, the message is sent in counter-clockwise direction to the node nearest to the destination known by the current node. This will sensibly decrease the average number of hops made by the messages sent through the network, but it will not identify the malicious node. Another possible solution is to check the distance from the current node to the previous hop and see if this distance is greater than the one between the previous hop and the destination of the message (in MSPastry this can be done since each message maintains the track of the forwarding process inside it). If this is the case the previous hop acted maliciously, thus it has to be marked as compromised. This a solution like the **SeChord** protocol presented in Section 4.3.2. Finally, another possible solution is to add an extra field to each message containing the Time To Live in order to block its diffusion whenever this TTL expires, i.e. whenever it reaches the

value zero. This will lead to have fewer messages forwarded incorrectly but also to assume the possibility to have fewer messages that reach the destination during the process.

Message Dropping Thinking on the effects of this attack, i.e. that messages are dropped once they reach a malicious node, we need to enforce the routing algorithm in order to increase as much as possible the probability to reach the destination. For this purpose we can add to the Pastry protocol the multi-path system implemented for Kademlia, presented in Section 2.5.6. In the latter protocol, messages are sent through α different paths, with α that is a system parameter usually fixed to 3. Along with this also the routing tables and leaf sets need to be modified. The reason for this is that the routing mechanism in Pastry selects the nearest node to the destination to which to forward the message, thus it cannot select α times the same node. We need to modify the routing tables and leaf sets in order to contain per each cell at least three contacts, in order to send the message to these 3 entries. With this, the probability to encounter a malicious node in the path performed by the message decreases with the increasing of the parameter α. On the other hand, the number of messages sent through the network will be higher than the normal case and this can lead to have network congestion.

Identity Theft In this case, discovering such an attack without any acknowledgment on the receiving of a message is quite impossible. For this reason, a possible solution to block the effects of this attack is to introduce an authentication system such that presented in Section 4.3.3 when talking about the countermeasures applicable against the Table Poisoning attack. Once a node (A) needs to send a message, it produces a message containing the current timestamp (T) and the one-way hash function of the triplet **[{ID_D, IP_D, $port_D$}, S_A, T]**, where ID_D is the ID of the destination, IP_D is its IP address and $port_D$ is the port number and S_A is the secret value of node A, not known by any other nodes. Once the malicious node (M) receives the message it replies to A with its own credentials, i.e. its ID, IP address and port number, the timestamp T and the hash previously received by A. When A receives back the message from M, it checks if the hash of the received credentials along with the timestamp inside the received message matches the hash previously produced. If the two values are not equal then the destination node is not the real one, thus it is a fake. However, a malicious node can retrieve the contact of another node in the network simply contacting it during a routing operation thus this control can be jumped from the malicious node. A good countermeasure against the Identity Theft can be a stronger identification, and the insertion of the multipath system presented in the previous paragraph, in order to decrease as much as possible the contact of malicious nodes in the network.

5.4.2 Countermeasures against new attacks

The new types of attacks are all inside the branch of the Trojan Virus that we find in software applications. For this reason, an antivirus-like system needs to be implemented inside the protocol to make it able to recognize malicious contents when they are encountered during routing operations. To realize this countermeasure, we will modify the **MSPastryProtocol** class and, more precisely, its **receiveRoute** method. Once a node receives a message it scans the content of the message and, if it is not equal to a pre-formatted content specified by the system, it is discarded and the source node is marked as evil, i.e. its state is set to 2 and it will not be contacted again for

routing operations. To make the system being able to recognize malicious contents, we need to define a fixed format for all the kind of messages we can have in the Pastry protocol. They are represented, along with the new fixed format, in Table 19.

Type of Message	Fixed Format
MSG_LOOKUP	*Automatic Lookup Message from the System*
MSG_JOINREQUEST	*This is a Join Request*
MSG_JOINREPLY	*Replying to a Join Request*
MSG_LSPROBEREQUEST	*Probing the Network*
MSG_LSPROBEREPLY	*Replying to a Probing*
MSG_SERVICEPOLL	*Requesting a Service to the System*

Table 19 Types and formats of messages for the antivirus fix in MSPastry.

The system is now able to block evil contents in the messages, thus we can run the tests. In Figure 88 are represented the results for the fix over the Colonizer.

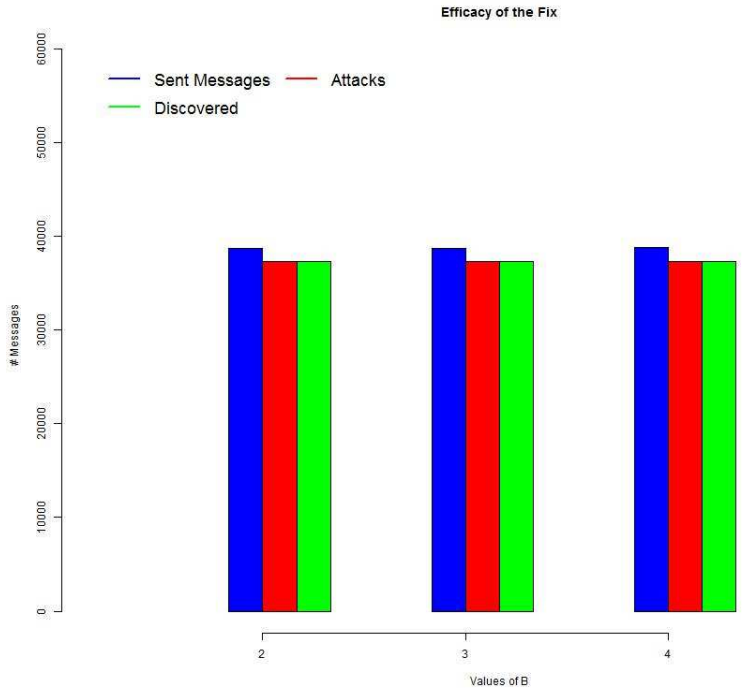

Figure 88 Efficacy of the antivirus fix against the Colonizer attack.

The blue rectangle represents the total number of sent messages while the red line represents the number of malicious messages and the green line the number of discovered malicious messages. In the case of the Colonizer the system works well with all the values of B, represented on the x-axis. Moreover, all the malicious messages are blocked. Let us take a look if it works also with the other two Trojan attacks.

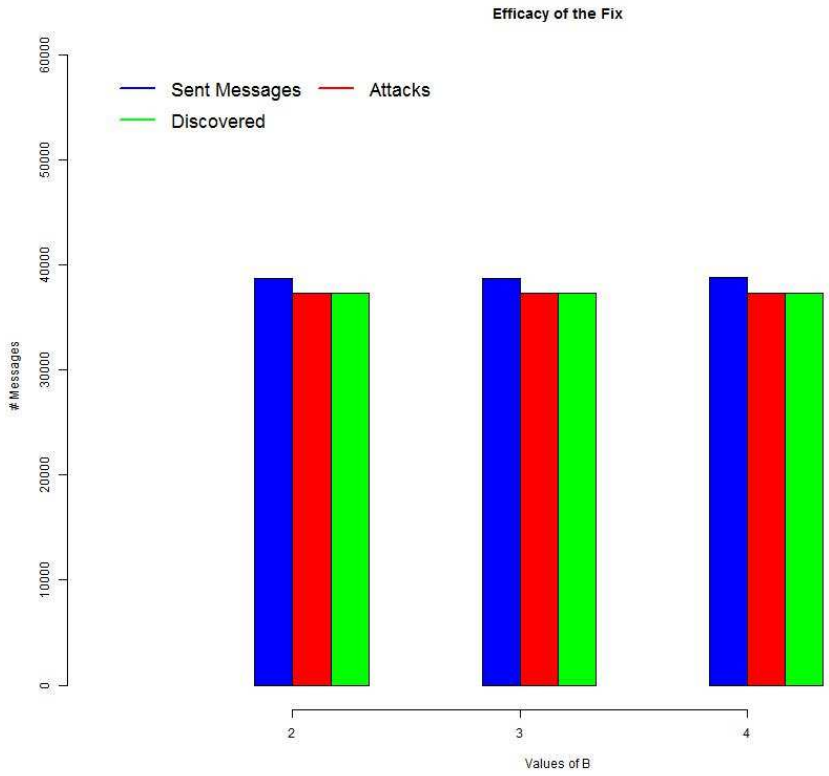

Figure 89 Efficacy of the antivirus fix against the Silent Murderess attack.

As we can see the system works well also against this attack since it blocks all the malicious messages sent by malicious nodes. As last test we can have a look at the results for the fix with the Sniffer attack.

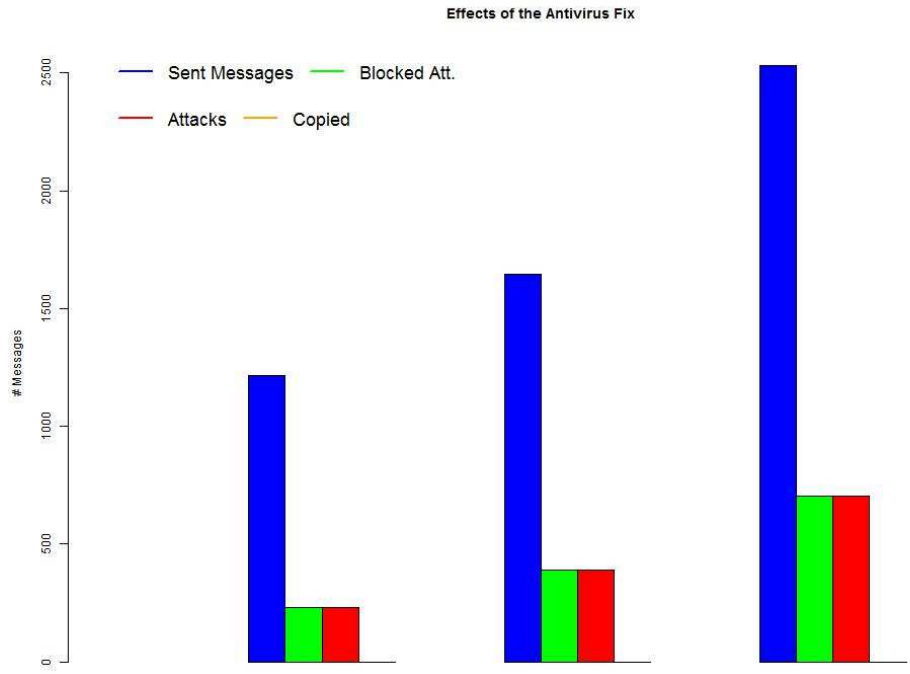

Figure 90 Efficacy of the antivirus fix against the Sniffer attack.

In the case of the Sniffer we varied the number of malicious nodes instead of the parameter B. Even in this case, the fix is good and the number of copied messages is equal to zero. For the case of the Silent Murderess and the Colonizer, the value of B does not affect the results of the tests.

Fix Evaluation Having to deal with arbitrary messages, the only thing a good node can do to discard evil messages and retain good messages is to open them in a secure environment before reading their content. An antivirus system is good to realize this countermeasure, since its job is to read the content of the messages to compare this value with a pre-formatted content. To have a visual representation on the effects of the detection of malicious nodes, we can take a look at Figure 91. Here are represented the decreasing number of malicious nodes in the network during the simulation of the Sniffer with a population of N = 1000 nodes. As we can see, the number of malicious nodes in the network decreases dramatically to zero in a short time, meaning that the antivirus discovered all the messages sent by malicious endpoints to good hosts during the simulation.

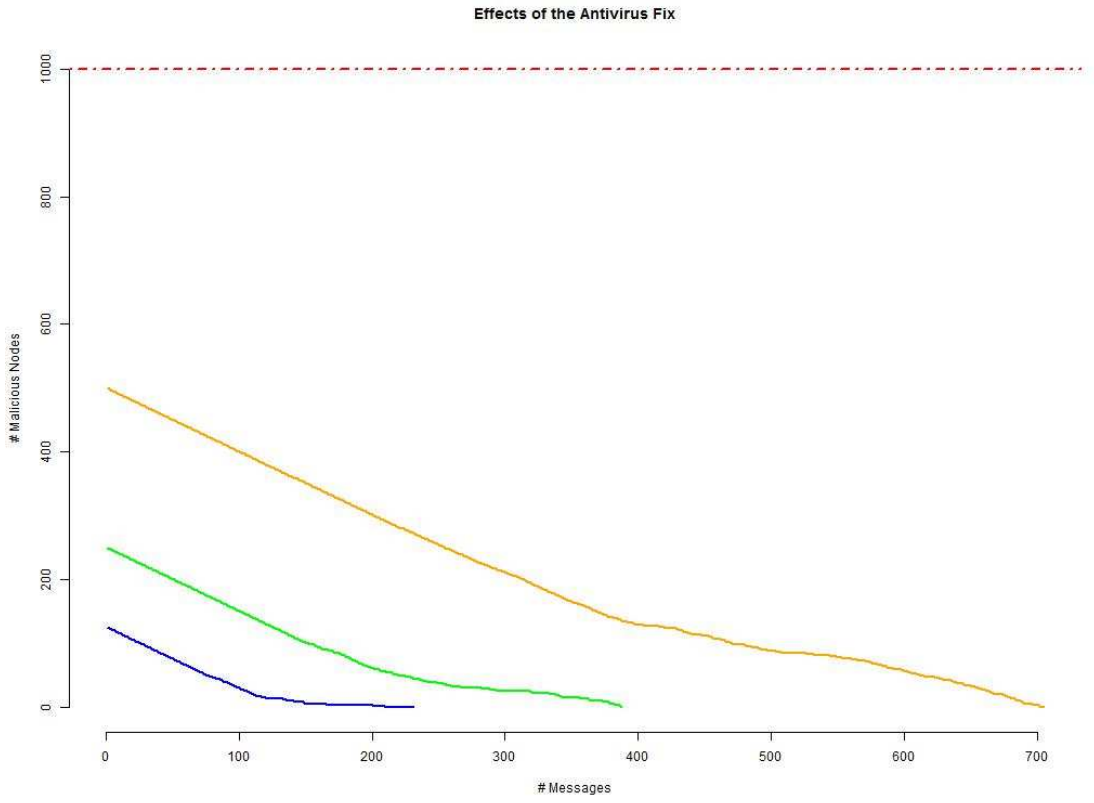

Figure 91 Decreasing of malicious nodes inside the network with the antivirus fix for the Sniffer attack.

5.5 Conclusion

In this chapter we presented the Peersim simulator, giving a deep presentation of its functionalities and its main features. We outlined how the simulator works and how it can be used to simulate a real P2P network. We tested its performances over the Pastry protocol, with the release of the MSPastry protocol by Elisa Bisoffi and Manuel Cortella. We have shown how this protocol works and its main features, we added some new parts to the code in order to test the weaknesses of the underlying Pastry protocol. We evaluated three of the attacks presented in

Chapter 3 and discussed the possible countermeasures. In the second part of this Chapter we introduced three new attacks never tested before on the Pastry protocol, outlining their structure and focusing the attention on the weaknesses of the protocol they use to produce some misbehavior in the network. We evaluated these attacks and presented some countermeasures.

Concluding, we can say that all the tests were made in a simulated environment, far from the real environments we can find every day in the Internet. However we feel that the obtained results shall give a good initial intuition over the security problems of real P2P networks.

CONCLUSION

The main goal of this thesis was to introduce P2P networks and their structures, outlining their main features and weaknesses. These structures were presented together with the protocols used to realize the routing mechanisms. Another goal was to introduce some attacks to these architectures, to describe them precisely and to present some countermeasures to use against their effects. The final goal of this work was to present a simulation of some of the attacks unveiled in previous chapters, and some new type of attacks, outlining their structure and features, showing also how they can be stopped to preserve the initial working state of the network. To make all our tests we used Peersim, a powerful simulator that allows to create a mimic environment in which to test every protocol running on a P2P network. Some of the results obtained during the test phase were surely useful while some others were not as good as expected mainly because some countermeasures adopted to block the attacks used in the test phase were not as efficient as expected by the theoretical assumptions. In the future more attention can be spent over the security tests and countermeasures where tests give poor outcomes. Moreover, it can be interesting to test the new attacks proposed in real P2P networks to test their true efficacy over real environments. Another interesting test could be run over the resource sharing mechanism, compromising it and showing how it can be weaken or misused and how it can be enforced to make it more reliable. Finally, we tested only one of the protocols proposed in the theoretical part of this thesis but it could have been interesting to test the proposed attacks also on other protocols, e.g. on BitTorrent a widely used P2P Network.

Thanks to…

… my mom Silvia and my dad Michele. I could not think to my life without you two. Since when I was a little child, you always protected me under your wings, teaching me how to live and how to become a man. I owe you one... a big one!

… my sister Cristina, always on my side and always with a smile when a needed it. You will be my little princess forever!

… my girlfriend Katia. It is very difficult to find only one thing to thank you. When I was down, you were beside me to lift me up. When I was happy you were there to joy with me. When I was mad at something, you were there to remind me that the life goes on. You were patient in many occasions, understanding my limits and always helpful when problems arose. Thank you, from the bottom of my heart. I love you little monkey!

… grandma Germana, grandma Renata, uncle Simone, uncle Gabriele, aunt Sara and my little cousin Alice. You were always beside me along all my course of study. I hope I made you proud of me.

… grandpa Renato. You left a big hole in my heart. I will never forgive myself to not telling you 'I love you' every time I could have done it. But surely you know. Thank you grandpa, thank you so much.

… my parents. Anyone of you supported me from the beginning of my life. Each one of you is truly important to me, even if sometime I do not show this.

… my Scout group, the Mestre 6th. Since when I was a child in second grade scouting helped and instructed me to live like a good person and to respect the others, no matter who the others were. Scouting will always be an important part of my life.

… my Scout staff: Tiziana, Mattia, Mauro and Letizia. You all accompanied me along this last year of suffering, happiness, problems and weeps. Sorry to be such a bothering boss and I hope to be good enough to have left you a good sign of my passage.

… my former Scout staff: Marta, Andrea, Marianna and Caterina. A wide part of my last period in my course of study coincided with the time passed together. Thank you for all the support and comprehension you gave me during that period and during this last year. For all the remaining things: It is not a problem of mine…

… my teacher Flaminia Luccio. Again you were the right person to choose to end a path and to present a valuable thesis. Sorry to being such a bad English writer, but I promise to improve from now on!

… all those people who said to me to stop, to leave, that I was the worst or that I would never ever be good enough for something. You just made me stronger to reach my goals.

… all those people I did not thank personally but that surely knows they have an important place in my heart.

.. God, for everything you gave to me or took from me, for all the mistakes I made and that I will make tomorrow. Everything helped me to grow.

Grazie a…

… mamma Silvia e Papà Michele. Non posso concepire la mia vita senza di voi. Fin da quando ero bambino, mi avete sempre protetto sotto le vostre ali, insegnandomi a vivere e a diventare un uomo. Vi devo un favore... uno grosso!

… mia sorella Cristina, sempre dalla mia parte e sempre con un sorriso quando ne avevo bisogno. Sarai sempre la mia piccola principessa!

… alla mia ragazza Katia. È davvero difficile trovare una cosa singola per cui ringraziarti. Quando ero a terra tu eri al mio fianco pronta a tirarmi su. Quando ero felice tu eri lì per gioire assieme a me. Quando mi arrabbiavo per qualcosa tu eri lì per ricordarmi che la vita va avanti. Sei stata paziente in diverse occasioni, comprendendo i miei limiti e sempre disponibile quando i problemi saltavano fuori. Grazie, dal profondo del mio cuore. Ti amo scimmietta!

… nonna Germana, nonna Renata, zio Simone, zio Gabriele, zia Sara e la mia cuginetta Alice. Siete stati al mio fianco durante tutto il mio percorso di studi. Spero di avervi reso fieri di me.

… nonno Renato. Hai lasciato un grande vuoto nel mio cuore. Non mi perdonerò mai il fatto di non averti detto 'Ti voglio bene' ogni volta che ne ho avuta l'occasione. Ma penso che tu lo sappia. Grazie nonno, grazie davvero.

… ai miei parenti. Ognuno di voi mi ha sempre supportato da quando sono nato. Siete tutti importanti per me, anche se non ve lo dimostro.

… Il mio gruppo Scout Mestre 6. Da quando ero in seconda elementare lo scoutismo mi ha insegnato a vivere come una persona giusta e a rispettare gli altri, chiunque fossero questi altri. Lo scoutismo sarà sempre parte della mia vita.

… il mio Staff: Tiziana, Mattia, Mauro e Letizia. Mi avete accompagnato lungo quest'ultimo anno di sofferenze, felicità, problematiche e lacrime. Scusate se sono stato un Capo Reparto così problematico, ma spero di essere stato all'altezza e di avervi lasciato un segno del mio passaggio.

… al mio vecchio Staff: Marta, Andrea, Marianna e Caterina. Con voi ho condiviso un'ampia parte del mio percorso di studi. Grazie per la comprensione che avete avuto durante quel periodo e durante quest'ultimo anno. Per tutto il resto: Non è un problema mio..

… alla mia insegnante Flaminia Luccio. Ancora una volta si è rivelata la persona giusta per concludere un percorso e per presentare una buona tesi. Mi spiace di essere uno scrittore di inglese scadente, ma prometto che da oggi migliorerò!

… tutte le persone che nel corso della mia vita mi hanno detto di fermarmi, mollare, che ero pessimo o che non ero all'altezza delle mie pretese. Mi avete solo reso più forte per raggiungere i miei obiettivi.

… tutte quelle persone che non ho ringraziato personalmente ma che sicuramente sanno di avere un posto importante nel mio cuore.

… Dio, per tutto ciò che mi hai dato e che mi hai tolto, per tutti gli errori che ho commesso e che commetterò domani. Ogni cosa mi ha aiutato a crescere.

Bibliography

BOOKS and PAPERS

[1] Anderson R., Biham E., *Tiger — A Fast New Hash Function*, Proceedings of the 3rd International Fast Software Encryption, Cambridge, 1996

[2] Bisoffi E., Cortella M., *MSPastry protocol implementation in Peersim*, Project for the exam of Distributed Sy-stem, University of Trento, May 2007

[3] Castro M., Druschel P, Hu Y. C., Rowstron A., *Topology-aware routing in structured peer-to-peer overlay networks*, Future Directions in Distributed Computing, pp. 103-107, September 2002

[4] Castro M., Druschel P., Ganesh A., Rowstron A., Wallach D. S., *Secure routing for structured peer-to-peer overlay networks*, OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, Volume 36, pp. 299-314, Winter 2002

[5] Cerri D., Ghioni A., Paraboschi S, Tiraboschi S., *ID Mapping Attacks in P2P Networks*, IEEE GLOBECOM 2005, Global Telecommunications Conference, Volume 3, 28 Nov. - 2 Dec. 2005

[6] Chakareski J., Chan G., Wang M., Wei B., *P2P Cloud Systems*, in Peer-to-Peer Networking and Applications, Volume 12083, Springer, May 2013

[7] Chuiwei L., Honghua W., Zhiyuan L., *A Security Routing Algorithm of P2P Network Based on Multiple Encryption and Positive Detection*, Research Journal of Applied Sciences, Engineering and Technologies, pp. 2392-2398, March 2013

[8] Dabek F., Brunskill E., Kaashoek F. M., Karger D., *Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service*, Proceeding of the 8th IEEE Workshop on Hot Topics in Operating Systems, 2001

[9] Damiani E., De Capitani di Vimercati S., Paraboschi S., Samarati P., Tironi A., Zaniboni L., *Spam Attacks: P2P to the Rescue*, Proceedings of the 13th International World Wide Web conference on alternate track, WWW 2004, New York, USA, May 2004

[10] Dinger J., Hartenstein H., *Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges and a Proposal for Self-Registration*, Proceedings of the first International Conference on Availability, Reliability and Security, 2006

[11] Dong X., Chellappan S., Krishnamoorthy M., *RChord: an enhanced Chord system resilient to routing attacks*, International Conference on Computer Networks and Mobile Computing, pp. 253-260, October 2003

[12] Doucer J. R., *The Sybil Attack*, IPTPS '01, in Proceeding in the 1st International Workshop on Peer-to-Peer Systems, pp. 251-260, Springer-Verlag, London 2002

[13] Forestiero A., Mastroianni C., *Description of the Self-Chord P2P Application*, in Proceedings of the 12th Workshop on Objects and Agents, Rende (CS), Italy, July 4-6 2011

[14] Fujii T., Ren Y., Hori Y., Sakurai K., *Security Analysis for P2P Routing Protocols*, International Conference on Availability, Reliability and Security, pp. 899-904, Fukuoka, 16-19 March 2009

[15] Jyothi B. S., Sharanipragada J., *SyMon: Defending Large Structured P2P Systems Against Sybil Attack*, IEEE 9th International Conference on Peer-to-Peer Computing, pp. 21-30, 9-11 Sep. 2009

[16] Kohli P., Ganugula U., *DDoS Attacks using P2P Networks*, International Institute of Information Technology, 25th April 2007

[17] Kunzmann G., *Recursive or iterative Routing? Hybrid!*, KiVS Kurzbeitrage und Workshop, volume 61 of LNI, pp. 189-192, 2005

[18] Larson P. A., *Dynamic Hash Tables*, in Communications of the ACM, volume 34, pp. 446-457, April 1988

[19] Levine N. B., Shields C., Margolin B. N., *A Survey of Solutions to the Sybil Attack*, Technical Report, University of Massachussets Ahmerst, October 2006

[20] Liang J., Naoumov N., Ross K. W., *The Index Poisoning Attack in P2P File Sharing Systems*, in Proceedings of the 25th IEEE International Conference on Computer Communications, April 2006

[21] Lifshits Y., *Reputation Systems I: Hits, PageRank, SALSA, eBay, EigenTrust, Vkontakte*, Caltech CMI Seminar, 4th March 2008

[22] Liu Y., Liu X., Wang C., Xiao L., *Defending P2Ps from Overlay Flooding-based DDoS*, ICPP 2007, International Conference on Parallel Processing, Xi'an, 10-14 Sep. 2007

[23] Liu Y., Xiao L., Ni M. L., Liu Y., *Overlay Topology Matching in P2P Systems*, in Grid and Cooperative Computing, Lecture notes in Computer Science, Volume 3032, pp. 300-307, 2004

[24] Lua E. K., Crowcroft J., Pias M., Sharma R., Lin S., *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*, IEEE Communication Survey and Tutorial, March 2004

[25] Maccari L., Rosi M., Fantacci R., Chisci L., Aiello L. M., Milanesio M., *Avoiding Eclipse attacks on Kad/Kademlia: an identity based approach*, ICC '09 IEEE International Conference on Communications, Dresden, 14-18 June 2009

[26] Manzanares-Lopez P., Munoz-Gea J. P., Malgosa-Sanahuja J., Sanchez-Aarnoutse J. C., *Anonymity in P2P Systems*, Handbook of Peer-to-Peer Networking, Springer 2010, pp. 785-812

[27] Maymounkov P., Mazières D., *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*, IPTPS '01 in the First International Workshop on Peer-to-Peer Systems, pp. 53-65, Springer-Verlag, London 2002

[28] Montresor A., Jelasity M., *PeerSim: A Scalable P2P Simulator*, P2P '09 IEEE 9th International Conference on Peer-to-Peer Computing, pp. 99-100, Seattle (WA), 9-11 Sep. 2009

[29] Naoumov N., Ross K., *Exploiting P2P Systems for DDoS Attacks*, Proceedings of the 1st International Conference on Scalable Information Systems, Infoscale 2006, Hong Kong, May 30-June 1, 2006

[30] Needels K., Kwon M., *Secure Routing in Peer-to-Peer Distributed Hash Tables*, Proceedings of the 2009 ACM symposium on Applied Computing, pp. 54-58, New York, USA, 2009

[31] Pesce F., *Protocollo Kademlia*, Course on Networks, Ca Foscari University of Venice, a.a. 2008/2009

[32] Pita I., *A formal specification of the Kademlia distributed hash table*, Proceedings of the 10th Spanish Workshop on Programming Languages, PROLE 2010, pp. 223-234, Ibergarceta Publicaciones, 2010

[33] Piyawongwisal P., Xia P., *Sybil Attack and Defense in P2P Networks*, Course on Advanced Computer Networks, Fall 2011

[34] Pretre B., *Attacks on Peer-to-Peer Networks*, Semester Thesis, Swiss Federal Institute of Technology Zurich, Autumn 2005

[35] Ricci L., *Slides from the P2P Systems Course*, University of Pisa, a. y. 2012

[36] Rowaihy H., Enck W., McDaniel P., La Porta T., *Limiting Sybil Attack in Structured P2P Networks*, INFOCOM 2007 26th International Conference on Computer Communications, pp. 2596-2600, 6-12 May 2007

[37] Rowstron A., Druschel P., *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany: pp. 329–350, Nov 2001.

[38] Singh A., Ngan T., Druschel P., Wallach D.S., *Eclipse Attacks on Overlay Networks: Threats and Defenses*, Proceedings of the 25th IEEE International Conference on Computer Communications, pp. 1-12, Barcelona, Spain, April 2006

[39] Steiner M., En Najiary T., Biersack E. W., *Exploiting KAD: Possible Uses and Misuses*, ACM SIGCOMM Computer Communication Review, Volume 37, pp. 65-70, October 2007

[40] Stoica I., Morris R., Karger D., Kaashoek F. M., Balakrishnan H., *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, SIGCOMM '01 in Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149-160, New York, USA, October 2001

[41] Stutzbach D., Rejaie R., *Understanding churn in node-to-node networks*, In IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, Rio de Janeiro, 25-27 Oct. 2006

[42] Vestola M., *Security Issues in Structured P2P Overlay Networks*, TKK Technical Reports in Computer Science and Engineering, Aalto University, 5 May 2010.

[43] Villanueva R., Villamil M., *SecureRoutingDHT: A Protocol for Reliable Routing in P2P DHT-based Systems*, ICIW 2012, The Seventh International Conference on Internet and Web Applications and Services, pp. 260-267, Sep. 2012

[44] Vu Q. H., Lupu M., Ooi B.C., *Routing in Peer-to-Peer Networks*, in Peer-to-Peer Computing, pp. 39-80 **,** Springer 2010

[45] Wang P., Tyra J., Chan-Tin E., Malchow T., Kune D. F., *Attacking the Kad Network*, Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, Instanbul, Turkey, 22-25 Sep. 2008

[46] Wang W., Xiong Y., Zhang Q., *Ripple-Stream: Safeguarding P2P Streaming Against DoS Attacks*, International Conference on Multimedia and Expo, July 2006

[47] Wang X., Yu H., *How to break MD5 and other hash functions*, EUROCRYPT'05 Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques, pp. 19-35, Springer-Verlag, Heidelberg 2005

[48] Xiong L., Liu L., *PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities*, IEEE Transactions on Knowledge and Data Engineering, Volume 16, July 2004

[49] Yue X., Qiu X., Ji Y., Zhang C., *P2P Attack Taxonomy and relationship Analysis*, in Proceedings of the 11th international conference on Advanced Communication Technology, Volume 2, pp. 1207-1210, 2009

[50] Zhao B. Y., Jubiatowicz J., Joseph A. D., *Tapestry: An Infrastructure for Fault Tolerant Wide-area Location and Routing*, EECS Department, University of California, Berkeley, April 2011

WEBSITES

[51] *A brief(ish) history of* P2P,
http://iml.jou.ufl.edu/projects/fall02/moody/history.html

[52] *A Survey of P2P Network Security Issues*,
http://grothoff.org/christian/teaching/2013/2194/li_security_survey.html#encrypt

[53] *Bearshare Official Website*,
http://it.bearshare.com/

[54] *BitTorrent specifications: Bencode coding*,
https://wiki.theory.org/BitTorrentSpecification#bencoding

[55] *Come mascherare indirizzo IP*,
http://aranzulla.tecnologia.virgilio.it/come-mascherare-indirizzo-ip-32250.html

[56] *Cosa sono le comunicazioni Peer-to-Peer (P2P) ?*,
https://support.skype.com/it/faq/FA10983/cosa-sono-le-comunicazioni-p2p-peer-to-peer

[57] *Current version of the Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), March 2012,* http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[58] *Definition for Bug*,
http://www.notrace.it/glossario/Bug/

[59] *Definition for Hash Function*,
http://searchsqlserver.techtarget.com/definition/hashing

[60] *Definition of Crawler*,
http://searchsoa.techtarget.com/definition/crawler

[61] *Definition of Encryption algorithm*,
http://www.businessdictionary.com/definition/encryption-algorithm.html

[62] *Definition of Malware*,
http://www.techterms.com/definition/malware

[63] *Definition of Physical Network*,
http://encyclopedia2.thefreedictionary.com/physical+network

[64] *Definition of Port Number*,
http://searchnetworking.techtarget.com/definition/port-number

[65] *Definition of Random Walk*,
http://www.thefreedictionary.com/random+walk

[66] *Definition of Round-trip time,*
http://searchnetworking.techtarget.com/definition/round-trip-time

[67] *Definition of Spoofing*,
http://www.techterms.com/definition/spoofing

[68] *Distributed Hash Tables*,
http://www.infoanarchy.org/en/Distributed_hash_table

[69] *Emerging applications of P2P technology*,
http://p2peducation.pbworks.com/w/page/8897427/FrontPage

[70] *eMule Project*,
http://www.emule-project.net/home/perl/general.cgi?l=18

[71] *Encryption method for KAD network*,
http://forum.emule-project.net/index.php?showtopic=153325

[72] *Example of Merkel Tree*,
http://cnx.org/content/m29405/latest/Picture%2012.png

[73] *Exploiting the Security Weaknesses of the Gnutella Protocol*,
http://alumni.cs.ucr.edu/~csyiazti/courses/cs260-2/project/html/

[74] *Gnutella*,
http://whatis.techtarget.com/definition/Gnutella

[75] *Grid5000*,
https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

[76] *Hash Tables*,
http://www.pcmag.com/encyclopedia/term/44129/hash-table

[77] *How legal is P2P file sharing?*,
http://features.en.softonic.com/how-legal-is-p2p-file-sharing

[78] *How the Old Napster worked*,
http://www.howstuffworks.com/napster.htm

[79] *Introduction to Peer-to-Peer* Networks,
http://compnetworking.about.com/od/basicnetworkingfaqs/a/peer-to-peer.htm

[80] *Is file sharing legal or illegal?*,
http://www.computerhope.com/issues/ch001042.htm

[81] *Java SE Downloads*,
http://www.oracle.com/technetwork/java/javase/downloads/index.html

[82] *Kazaa*,
http://www.kazaa.com/

[83] *Limewire Official Website*,
http://www.limewire.com/

[84] *Man in the middle attack (fire brigade attack)*,
http://searchsecurity.techtarget.com/definition/man-in-the-middle-attack

[85] *Method of providing digital signatures*,
http://www.google.com/patents/US4309569

[86] *Napa Wine*,
http://napa-wine.eu

[87] *Napster Specification*,
http://opennap.sourceforge.net/napster.txt

[88] *Napster Website*,
http://it.napster.com/start

[89] *Network Address Translation (NAT)*,
http://searchenterprisewan.techtarget.com/definition/Network-Address-Translation

[90] *P2P Glossary*,
http://www.p2pna.com/glossary.html

[91] *P2P Networks*,
http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p1.html

[92] *P2P Routing*,
http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p9.html

[93] *Pastry: a substrate for peer-to-peer applications,*
http://www.freepastry.org/

[94] *Peersim Project*,
http://peersim.sourceforge.net/

[95] *Planet Lab*,
http://www.planet-lab.org/

[96] *Press release: OTT messaging traffic will be twice the volume of P2P SMS traffic by end-2013,*
http://blogs.informatandm.com/12861/news-release-ott-messaging-traffic-will-be-twice-the-volume-of-p2p-sms-traffic-by-end-2013/

[97] *Recent Improvements in the Efficient Use of Merkle Trees: Additional Options for the Long Term*, http://www.rsa.com/rsalabs/node.asp?id=2003

[98] *Routing Definition*, http://www.linfo.org/routing.html

[99] *Scalability definition*,
http://www.dizionarioinformatico.com/cgi-lib/diz.cgi?frame&key=scalabilita

[100] *Scalability definition*, http://www.sitexpress.it/infogloss.asp?currentpage=2863

[101] *Security of Computer Systems*, http://secgroup.ext.dsi.unive.it/teaching/security-course/

[102] *Specifications of the Gnutella Protocol,*
http://rfc-gnutella.sourceforge.net/developer/index.html

[103] *Surge in encrypted torrents blindsides record biz*,
http://www.theregister.co.uk/2007/11/08/BitTorrent_encryption_explosion/

[104] *Ten BitTorrent Clients*,
www.tomsguide.com/us/pictures-story/450-BitTorrent-client-downloads-applications-managers.html

[105] *The Bencode Coding System*,
http://bencode.codeplex.com/

[106] *The Legality of P2P File Sharing Software*,
http://ezinearticles.com/?The-Legality-Of-P2P-File-Sharing-Software&id=654992

[107] *The MD5 digest algorithm specification*,
http://www.kleinschmidt.com/edi/md5.htm

[108] *The MD5 Message-Digest Algorithm*,
http://tools.ietf.org/html/rfc1321

[109] *The Napster Revolution*,
http://wrt-intertext.syr.edu/ix/hart.html

[110] *The R Project for Statistical Computing*,
http://www.r-project.org/

[111] *Top 20 Peer-to-Peer (P2P) & Torrent Software Clients*,
http://www.prohackingtricks.com/2011/06/top-20-best-peer-to-peer-p2p-file.html

[112] *Usage of P2P Networks*,
http://p2peducation.pbworks.com/w/page/8897427/FrontPage

[113] *Usenet Deluxe*,
http://usenet-deluxe.com/it/

[114] *Usenet: in pensione dopo 30'anni di onorato servizio?*,
https://www.linuxsecurity.it/2010/05/24/internet/usenet-pensione-dopo-30-anni-di-onorato-servizio/

[115] *What is Latency*,
http://searchcio-midmarket.techtarget.com/definition/latency

[116] *What is the Bernoulli Trial?*
http://www.thefreedictionary.com/Bernoulli+trial

[117] *What is TTH (Tiger Tree Hashing)?*,
http://www.dslreports.com/faq/9677