



**Eduardo Oliveira
Estanqueiro Rocha**

**Desenvolvimento de um módulo para identificação
de aplicações Internet e de uma interface para a
plataforma DTMS-P2P**

**Development of a module for identification of
Internet applications and of an interface for the
DTMS-P2P platform**



**Eduardo Oliveira
Estanqueiro Rocha**

**Desenvolvimento de um módulo para identificação
de aplicações Internet e de uma interface para a
plataforma DTMS-P2P**

**Development of a module for identification of
Internet applications and of an interface for the
DTMS-P2P platform**

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor Paulo Salvador, Professor Auxiliar convidado e do Doutor António Nogueira, Professor Auxiliar, ambos do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais e irmão por todo o seu incansável e incondicional apoio e por me guiarem sempre na direcção correcta.

o júri

presidente

Prof. Dr. José Luís Guimarães Oliveira
Professor Associado da Universidade de Aveiro

Prof. Dr. Joel José Puga Coelho Rodrigues
Professor Auxiliar da Universidade da Beira Interior

Prof. Dr. Paulo Jorge Salvador Serra Ferreira
Professor Auxiliar Convidado da Universidade de Aveiro

Prof. Dr. António Manuel Duarte Nogueira
Professor Auxiliar da Universidade de Aveiro

agradecimentos

I'd like to begin by expressing my most sincere gratitude to Doutor Rui Valadas and to my supervisors, Doutor Paulo Salvador and Doutor António Nogueira for the great opportunity they gave me. Also for their immense and incredible support, assistance and for their inspiring minds and suggestions, which have always challenged me to cross new barriers and improve myself.

With great joy and pride, I would like to thank the Institute of Telecommunications – Aveiro University Polo, for their great support. Also, to all the great friends I made in this Institute.

I would also like to express my gratitude to all my friends, who I have met in Portugal and in other countries, for all their support, joy, understanding, adventures and great moments. A very particular bow of gratitude goes to my parents and brother, without whom and without their support and encouragement I would not have completed any of the challenges that I have proposed to myself. It is such a huge debt that it will never be able to compensate it. A special gratitude expression goes to Pedro Braumann, Miguel Pinheiro, Hélder Veiga, Maria Coelho, Joana Margarida, Ana Rita. Finally, to the most recent surprise in my life and for bringing so many good things to it: Conni Rinn.

This work was part of the project POSC/EIA/60061/2004 “Internet Traffic Measurements, Modelling and Statistical Analysis”, funded by Fundação para a Ciência e Tecnologia, Portugal.

palavras-chave

Análise de portos, análise estatística, análise de payload, plataforma distribuída, interface gráfico

resumo

Nos últimos anos tem-se registrado um enorme crescimento no número e variedade de aplicações IP. De entre estes numerosos protocolos, há alguns cujas características é importante estudar para conhecer o seu comportamento na rede. Por isso, conseguir efectuar uma exacta correspondência entre tráfego e aplicações reveste-se de grande importância num enorme número de tarefas relacionadas com a gestão de redes e de medições. Estas podem incluir engenharia de tráfego, diferenciação de serviços, monitorização de desempenho e segurança. Várias metodologias têm sido usadas e testadas. A metodologia baseada na análise dos portos utilizados tem-se tornado progressivamente ineficaz pois muitas destas novas aplicações usam portos que não são standard ou são utilizados por outros protocolos. Consequentemente, têm sido utilizados novos métodos para identificar estas aplicações, consistindo nomeadamente na análise das características estatísticas ou na análise do campo de dados dos pacotes. A primeira aproximação apresenta, no entanto, algumas limitações em fornecer a exacta identificação dos diferentes tipos de tráfego IP. Portanto, uma análise mais precisa exige a inspecção do payload dos pacotes. Esta dissertação propõe um módulo de software baseado nesta técnica. Este módulo pode funcionar de forma autónoma ou ser inserido numa plataforma de monitorização de tráfego com uma arquitectura peer-to-peer. Tirando partido da arquitectura distribuída da plataforma de monitorização, o módulo de identificação de tráfego poderá ainda melhorar o seu desempenho. A segunda parte desta dissertação propõe a implementação de uma Interface de Programação de Aplicações (API) para estabelecer a comunicação com a plataforma de monitorização de tráfego. Pretende-se que diferentes módulos consigam, deste modo, executar os diversos comandos na plataforma recorrendo à API para estabelecer a comunicação. Esta dissertação termina com a proposta de um interface gráfico para a mencionada plataforma como um meio de teste da API implementada. Deste modo, criou-se por um interface intuitivo que permite a execução das várias medições possíveis recorrendo à API para comunicar com a plataforma de medição. Também se pretende substituir o uso da linha de comandos, permitindo um uso mais simplificado dos vários comandos que o sistema de monitorização permite. O interface também fornece mensagens de erro para indicar ao utilizador como executar os comandos correctamente. O interface e a API foram desenvolvido na linguagem Java de modo a permitir uma maior portabilidade para outras plataformas computacionais.

keywords

Port-based analysis, statistical analysis, payload analysis, distributed measurement tool, graphical interface

abstract

In the last years we have witnessed a major increase in the number and variety of IP applications. There are some applications whose characteristics are important to study in order to gain a complete knowledge about their behavior in the network. Therefore, an accurate mapping of traffic to applications is of a noticeable importance in a wide range of network management and measurement tasks. These can include traffic engineering, service differentiation, performance/failure monitoring and security. Several approaches have been used. Port-based identification approaches have become inaccurate as many of these emerging applications use non-standard or ephemeral ports or use ports associated to other applications. Thus, new methodologies have been used to identify these applications: analysis based on the traffic statistical properties and analysis based on packet payload inspection. The first approach also presents several severe limitations in providing an exact identification of the different types of traffic. Therefore a more exact identification demands the examination of the user's payload. This thesis proposes an identification software module based on the payload analysis approach to complete traffic classification. This module will be inserted in a monitoring network system with a peer-to-peer architecture (although it can also be used autonomously) and will take advantage of this distributed architecture.

The second part of this thesis provides the implementation of an Application Programming Interface (API) to establish the communication with the traffic monitoring platform. It is intended to allow different modules to execute the various commands in the platform through the use of the API for the establishment of the communication.

This dissertation concludes with the proposal of a graphical interface to the peer-to-peer monitoring system as a means for testing the implemented API. Therefore, an intuitive interface was created which allows the execution of the various commands based on the API for the establishment of the communication with the platform. This interface is also intended to replace command line interfaces, allowing for a more intuitive, simpler, faster and more straightforward deployment of all facilities provided by the monitoring system. It also provides feedback messages that will show how to execute these commands in a correct way. The interface and the API are developed in the Java language to provide more portability to other computational platforms.

LIST OF FIGURES	III
LIST OF TABLES	V
1 INTRODUCTION	1
1.1 Identification of Internet applications	1
1.2 Application Programming Interface and Graphical Interface of the DTMS- P2P platform	4
1.3 Structure of the dissertation	4
2 STATE OF THE ART	7
2.1 Identification of Internet Applications	7
2.2 Graphical Interfaces.....	10
3 THE DISTRIBUTED TRAFFIC MONITORING SYSTEM	13
3.1 System Elements.....	13
3.2 Measurement Commands	15
3.3 Types of measured data	15
3.4 Data Retrieval.....	16
4 MODULE FOR IDENTIFICATION OF INTERNET APPLICATIONS 17	
4.1 Tool Implementation	17
4.1.1 Methodology Overview	18
4.1.2 Parsing the rules	23
4.1.3 Configuring the capture	24
4.1.4 Capturing and storing the packet.....	25
4.1.5 Classifying the flows	28
4.2 Results.....	31
4.3 Conclusions.....	35
5 API IMPLEMENTATION	37

5.1	Introduction.....	37
5.2	API Functionalities.....	38
5.2.1	Change of the status of the client	38
5.2.2	Visualization and changing of the client’s settings	39
5.2.3	Command execution functionality	39
5.2.4	Search and retrieval of results file	41
5.2.5	Listing of files of the client or of a node.....	43
5.3	Conclusions.....	43
6	GRAPHICAL INTERFACE OF THE DTMS-P2P TOOL	45
6.1	The rules of user interface design	45
6.2	Graphical Interface	46
6.2.1	Interface presentation.....	49
6.3	Conclusion	68
7	FINAL REMARKS	69
	APPENDIX I – API METHODS.....	71
	APPENDIX 2 – INTERFACE METHODS.....	83
	BIBLIOGRAPHY.....	95

List of figures

Figure 1 - Hierarchical relationship between the system elements	14
Figure 2 - File of signatures and related parameters.....	24
Figure 3 - Capture parameters	25
Figure 4 - Flow diagram of the capture process	28
Figure 5 - Flow diagram of the classification procedure	30
Figure 6 - Flow diagram of the classification process	31
Figure 7 - Hierarchy of the API and the calling application	37
Figure 8 - Use Cases diagram.....	38
Figure 9 - Flow diagram of the method value_changed().....	42
Figure 10 – Communication Hierarchy	47
Figure 11 - Main Frame of the interface	49
Figure 12 - Block diagram of the interface’s tabs	50
Figure 13 - Block diagram of the Menu bar.....	51
Figure 14 - Message box indicating that the client has been disconnected from the network	52
Figure 15 - Window showing the various client parameters.....	53
Figure 16 - Selection of the node to execute a command.....	54
Figure 17 - List of restrictions associated to the <i>ping</i> command.....	55
Figure 18 - Message indicating that the node does not support the command	55
Figure 19 - Message indicating the conclusion of the download of the results file	57
Figure 20 - Message illustrating the usage of the command.....	57
Figure 21 - Selection of the type of search to perform	58
Figure 22 - Table presenting the search results	59

Figure 23 - Message indicating that the client has already downloaded the file.....	59
Figure 24 - List of files the client owns.	59
Figure 25 - List of files of a node	61
Figure 26 - Frame showing the network representation	62
Figure 27 - Flow diagram of the process of the XML file	64
Figure 28 - Flow diagram of the procedure for the representation of the network	66
Figure 29 - Representation of a network with various elements connected	67
Figure 30 - Representation of a second network with fewer elements connected	67

List of Tables

Table 1 - Characteristic signatures of the different protocols	23
Table 2 - Dimension of the captured traces.....	31
Table 3 - Classification results obtained	32
Table 4 - Statistical information about correctly classified flows	33
Table 5 - Statistical information about misclassified or unclassified flows.....	33
Table 6 - Number of packets needed to classify flows	35

1 Introduction

The emergence of new protocols raised the need for an exact study of the characteristics of these new kinds of traffic. New methodologies were created to identify these applications, since the previously used techniques (namely port-based analysis, which was the most used one) became no longer accurate: statistical analysis of the properties of the generated flows and inspection of the packets payloads. This thesis presents a module for identification of Internet traffic based on packet payload inspection, since we believe this technique presents some advantages over statistical analysis. Section 1.1 will present an introduction to all the above mentioned methodologies, along with an explanation of their advantages and disadvantages.

The identification module will be integrated in a measurement network with a peer-to-peer architecture (called DTMS-P2P platform: Distributed Traffic Monitoring System with a Peer-to-Peer architecture) that is being developed at our research group [Salvador2005], [Veiga2007], [IT2007].

The second part of the dissertation consisted on the development of an Application Programming Interface (API) for the DTMS-P2P platform that is intended to allow multiple modules to interact with the mentioned platform. As a means for testing this implementation, a graphical interface was created which uses the mentioned API to communicate with the monitoring system. This interface was also proposed for replacing the command line interface. It is also intended to represent an intuitive and simple interface that can perform all the different measurement tasks the platform is able to execute in a faster and more intuitive way. The interface will also display general messages in order to advise the user on how to execute each chosen command in the most appropriate way.

1.1 Identification of Internet applications

Over the last few years we have witnessed a major increase in the number and variety of Internet applications. From a set of few and known protocols, we have evolved to a very large number of unknown applications and unidentified traffic. Therefore, an exact analysis and identification of Internet traffic is essential to acquire a precise knowledge of these emerging applications and is also vital to several network related activities, such as security and Quality of Service provision. Traffic studies can be also

very useful to Internet service providers that can use the obtained results to supply better service levels to their costumers and propose new tariffing plans. Besides, some applications are bandwidth-expensive and can lead to congestion problems that will result in unsatisfied clients. Therefore, ISPs and enterprises must have the possibility to block or provide less bandwidth to a certain type of traffic. Knowing which applications are generating traffic and occupying bandwidth and other network resources is of inestimable interest for network administrators that can use such information to plan network resources. From a social point of view, this study can also identify new emerging applications, mainly peer-to-peer and multimedia streaming applications, and communities of users.

All the above mentioned tasks require the ability to perform exact traffic classification. However, there are some obstacles to overcome: some packet headers don't include enough information to enable an exact classification and some applications use arbitrary ports and encryption. Several approaches have been proposed to deal with these difficulties, like for example port-based analysis, statistical and payload analysis. Each one of these techniques has its own advantages and disadvantages.

On an early-stage, classification was based on the ports to which packets were sent to or received from. It is known that some applications use only reserved and well-known ports to communicate: for example, HTTP uses port 80 and DNS uses port 53. Having this knowledge in mind, the technique was based on the examination of packet headers, specially the communicating ports. The next step consisted on associating the traffic of a determined port to a certain application. However, this process can lead to traffic misclassification and has proved itself to be untrustworthy because nowadays many emerging applications, such as peer-to-peer protocols, voice or video transmission over IP networks, use ephemeral ports. Besides, applications may try to disguise themselves by using ports that are usually associated to other applications in order to bypass proxies or firewalls. Thus, nowadays we cannot say that a specific port is associated only with traffic generated by a certain application.

A second technique was used to overcome the barriers imposed by port based classification: the study of the statistical properties of each traffic flow. This technique is based on the fact that different applications generate different traffic patterns. For example, a HTTP browsing generally does not generate as much traffic as a FTP file transfer.

Moreover, applications may also be distinguished based on the traffic direction. As an example, FTP data traffic is only in one direction, while instant messaging applications generate traffic in both directions. As this technique is more accurate, it has also its own disadvantages: it is able to identify the type of application that generated the traffic but not the exact application/client. Classification can also be erroneous due to applications that possess similar statistical properties.

The last approach is payload analysis. This technique consists on the analysis of the packet's payload and is based on the fact that many applications use particular signatures in their packets. These signatures distinguish each protocol from the others. Thus, analysing the packet payload and finding these characteristic strings can lead to a very precise identification. This method, like all previously mentioned methods, has its associated disadvantages: access to user's payload can be very difficult due to privacy and legal issues and some protocols use traffic encryption, making payload analysis useless. An additional barrier is the lack of reliable and available protocol specifications for all those non-standardized and still evolving protocols. Besides, there are several client implementations for the same protocol and some of them do not follow the specifications stated in the officially available documents.

This thesis will present a module for Internet traffic identification using payload inspection. Our methodology involved the investigation of available and reliable documentation about the different protocols in order to identify their particular characteristics and behaviour and the examination of several packet traces in order to confirm the obtained information or discover new relevant information. The main requirements for this approach to be efficient are:

- The used signatures must be accurate and lead to a low misclassification rate;
- Low overhead, in order to allow for a quick search in real-time captures;
- Allow identification in the first packets.

The proposed module was presented in the MCSIS Conference 2007 [Rocha2007].

1.2 Application Programming Interface and Graphical Interface of the DTMS-P2P platform

The second part of this thesis proposes an Application Programming Interface (API) for the DTMS-P2P platform and a graphical interface to the mentioned peer-to-peer measurement network.

The API was implemented to provide a means for communication with the DTMS-P2P platform. Therefore, modules which may need to interact with the mentioned platform can use the mentioned API to achieve it.

The graphical interface was implemented as a means for testing the API as it will send the necessary messages to the platform in order to accomplish a task a user ordered to the interface. The graphical interface was also triggered by an urgent need to create a simplified and intuitive interaction framework between the user and the network. Therefore, we believe this interface constitutes a valuable resource and greatly improves the capability of the DTMS-P2P network.

Several functionalities are envisaged for the interface. It must show the user the DTMS-P2P network elements that are connected to the network. The interface must also allow the execution of the several monitoring actions in any of the system probes and the immediate retrieval of the measurement data files. A second functionality of this interface is to allow the search of measurement data files that are stored in the system in a distributed way, present them to the user in a suitable way manner and, again, allow the download of the selected file(s). The interface must also display the files the client has already downloaded. Moreover, changing the client settings and the operations of connecting and disconnecting from the network should also be enabled and configured through this interface.

As a part of the interface, a graphical representation of the network is also presented. The different monitoring network elements are represented along with the connections between them.

1.3 Structure of the dissertation

This dissertation is organized as follows: Chapter 2 presents the state of art on traffic identification tools and graphical user interfaces, which are the two main modules that have been developed in this Master thesis; Chapter 3 presents some concepts related to

the DTMS-P2P platform; Chapter 4 shows the details of the implementation of the identification module and presents the main results obtained; Chapter 5 introduces the implemented API and its several methods; Chapter 6 presents the graphical interface of the DTMS-P2P network, showing its different possibilities of interaction with the user and, finally, Chapter 7 presents the most relevant conclusions of the developed work.

2 State of the Art

This section presents the state of the art regarding traffic identification tools and graphical user interfaces, which were the two main objectives for this thesis. Basically, we will list, for each one of these topics, the most recent studies that were made and the alternatives that could have been used for the implementation of the modules.

2.1 Identification of Internet Applications

A lot of studies have been made in the area of Internet traffic identification, proposing new methodologies and evaluating their comparative accuracy. In this section, we enumerate the most significant studies and, at the same time, have tried to understand the advantages, drawbacks and limitations of each identification approach.

As mentioned above, port-based approach is no longer a secure and reliable identification method because well-known ports are not associated to a specific application anymore and modern applications use random ports. A study conducted by Madhukar A. and Williamson C. [Madhukar2006] tried to confirm that this technique is no longer a reliable one, by comparing it with other identification methods. Their work used datasets from the University of Calgary campus network. Based on the port analysis methodology, the unknown traffic percentage was 40-65% of the total traffic. This study also showed that unknown traffic was more evident at night periods, which might suggest that this traffic belongs to P2P applications. On a similar study, Sen et al [Sen2004] refer that the default port of the Kazaa protocol accounted for only 30% of the total traffic while the remaining traffic was sent on ephemeral ports. They also proved the above mentioned reasons for this trend. Another study conducted by Dewes C. et al. [Dewes2003], that analysed Internet chat systems, evidenced that using a port identification methodology it is impossible to distinguish HTTP traffic from traffic created by chat applications running on top of the same protocol. Therefore, a new methodology must be used.

The statistical method provides better results, although it has also raised some new important questions. Karagiannis T. et al. [Karagiannis2004a] developed a technique to identify P2P flows based on the connection patterns. Although P2P applications may use random ports or payload encryption, their traffic patterns will not change, and this constitutes the main advantage of this technique. The classification method presented by

Karagiannis T. et al. was based on two steps. The first one consisted on the identification of source-destination IP pairs which simultaneously used TCP and UDP protocols and on the determination of their associated ports. If the used ports were not well-known ports, then the flows were considered as P2P. The second step was based on the structural patterns of the transport-layer between nodes. Generally, for P2P traffic the number of ports used by a host corresponds to the number of connected IP hosts. The results achieved were very accurate and also provided the identification of unknown P2P protocols. Although this method is able to identify P2P traffic, its main disadvantage relies on its incapacity to identify a particular P2P protocol, which is a significant drawback. Madhukar A. and Williamson C. [Madhukar2006] have also conducted a study using this technique. Unlike the work by Karagiannis T. et al., their dataset did not contain any UDP traffic and the TCP traffic only contained the TCP SYN, FIN and RST headers to provide connection-level information patterns. Their methodology started by removing traffic of known non-P2P applications. After this removal step, all traffic that used known P2P ports was considered as P2P flows. Then, the number of distinct IP addresses communicating was calculated; if it corresponded to the number of ports for each {IP, port} pair, then the {IP, port} pair was classified as P2P. Their results were promising, however, as they claim, no proof of the correct classification was made as they did not have any packet payload. New limitations appeared in this work: port masquerading would not be detected as the approach used a list of standard ports for filtering purposes. This heuristic is also ineffective in the case of one IP host that is communicating with another IP host using only one port. Many applications use this connection pattern. On one of their studies, Karagiannis T. et al. [Karagiannis2005] presented a different approach to achieve traffic classification. Their study was based on identifying traffic patterns of host behaviour at the transport layer. They analyzed these characteristics at three levels: the social, the functional and the application level. At the first level, the behaviour of a host was captured by studying its interactions with other hosts. At the functional level, they studied the behaviour of a host based on its role in the network, which consisted of analysing if it acts as a provider or a consumer. At the application level, hosts were examined based on the transport layer interactions on specific ports. Therefore, hosts were associated with applications. With this methodology, they suggested that observing the activity of a host provides more information and can evidence the type of applications the host is running.

The results were very precise in classifying the majority of the captured traffic. However, their work also suffered from the above explained limitations.

Using the statistical properties, it is also possible to distinguish traffic through the use of clustering techniques. Erman J. et al. [Erman2006] use two unsupervised algorithms, K-Means and DBSCAN to perform classification. These algorithms use unlabelled data and group data into clusters based on similarity of behavior. The results indicate that these algorithms are a useful technique for classification. Despite the ability of grouping traffic, this method has to rely on other techniques to label the clusters.

Statistical study of traffic patterns can also include the use of Machine Learned classifiers. Zuev D. and Moore A. [Zuev2005] used the supervised Naive Bayes technique as a traffic discriminator. The results were generally good, although they were not as accurate as desirable in some cases. McGregor A. et al [McGregor2004] used machine learning techniques to create clusters for traffic classification. The results achieved provided accurate clusters to classify traffic. On the other hand, Bernaille L. et al. [Bernaille] used unsupervised clustering, as it relies on unlabeled data samples. They believe that unsupervised clustering is more appropriate as it does not rely on pre-defined classes. This is an advantage, since a single application can have multiple behaviours that should be separated. The results were also very accurate, although some new limitations appeared: traffic with the same statistical behaviour is classified as belonging to the same application, which is not always true; besides, traffic with unknown behaviour is not classified. Using unsupervised clustering methods, Zander S. et al. [Zander2005] proposed an approach to identify an optimal set of flow attributes. The results were also accurate.

A similar study conducted by Haffner P. et al [Haffner2005] used three linear classifiers: Naïve Bayes models, Maxent and AdaBoost. All these algorithms were used due to their learning process scalability and their different and efficient implementations. These classifiers were used to construct signatures that can be used for online classification. The three classifiers were able to achieve low error rates, with almost 99% of correctly classified flows.

Regarding the QoS monitoring and intrusion detection areas many tools were introduced. Cisco's NBAR (Network-Based Application Recognition) [Cisco2007] provides application recognition by using port recognition and packet header information to distinguish traffic. IDSs (Intrusion Detection Systems) are equipped with application

recognition modules that operate on a signature recognition basis through the use of payload analysis.

Based on the same methodology, Karagiannis T. et al [Karagiannis2004b] have recently developed a study to identify P2P traffic. Their analysis was only based on examining the user's payload and the results were very precise. On the first mentioned work, Karagiannis T. et al also used payload analysis as a comparative technique to their statistical approach and the results obtained proved that this technique achieves better results although showing also some disadvantages. The accuracy of the results was measured by the occurrence of false positives and false negatives. False positives relate to traffic misclassified as P2P, while false negatives refer to P2P traffic that the identifier failed to classify. Their results had less than 5% of false positives and false negatives, proving this to be the most correct way of identifying traffic. On a similar study, Haffner et al. [Haffner2005] developed a technique that is able to automatically determine applications signatures and the obtained results showed an error rate less than 1%. Sen S. et al [Sen2004] used on their work application signatures to identify traffic. The obtained results were also impressive and proved that this technique is the most promising one to use. Another study that can be mentioned was the study by Dewes C. [Dewes2003], which consisted on the identification of Internet chat protocols, such as IRC and Messenger. The technique used was payload analysis and it missed less than 8.3% of all existing chat connections. A. Moore and K. Papagiannaki [Moore2005] have also used this technique to identify traffic and the results obtained were also very accurate.

2.2 Graphical Interfaces

Graphical interfaces can be implemented in several programming languages. In the following lines we will be presented some of these languages.

GTK+ is a multi-platform toolkit used for the creation of graphical user interfaces. It offers a set of widgets which are suitable for complete application suites. It is based on three libraries:

- Glib: it is the low-level core library that forms the basis of GTK+ and GNOME. It provides data structure handling for C, portability wrappers, and interfaces for several runtime functionalities.

- Pango is a library for layout and rendering of text, with an emphasis on internationalization.
- The ATK library provides a set of interfaces for accessibility. Through it an application or toolkit can be used with such tools as screen readers, magnifiers, and alternative input devices.

GTK+ has been designed to support a range of languages, not only C/C++. Using GTK+ from languages such as Perl and Python provides an effective method of rapid application development.

Glade is a user interface-building program. It is used to rapidly prototype GTK+ and GNOME applications. It allows an application author to dynamically add, remove, and modify widgets and their layout. The interfaces designed are stored in XML format which allows an easy integration with external tools.

Interfaces can also be implemented using wxWidgets, which is a C++ framework providing GUI facilities on several platforms. The advantage of its platform-independent class library cannot be overstated, since GUI application development is very time-consuming, and sustained popularity of particular GUIs cannot be guaranteed. An interface can become obsolete if it addresses the wrong platform or audience. wxWidgets helps to insulate the programmer from these changes. Although wxWidgets may not be suitable for every application, it provides access to most of the functionalities a GUI program normally requires and also to network programming, PostScript output, and HTML rendering. It provides a far cleaner and easier programming interface than the native APIs.

Using Java programming language, the Abstract Window Toolkit (AWT) can be chosen to create an interface. It is the Java original platform-independent graphics toolkit and is also part of the Java Foundation Classes (JFC), which is the standard API for providing a GUI. The AWT provides the connection between the developed application and the native GUI. It offers several layout managers and the interface to input devices. As its components depend on the native counterparts, AWT components are called *heavyweight components*. This dependence brings platform specific limitations.

The Standard Widget Toolkit (SWT) is a toolkit maintained by the Eclipse Foundation. SWT's implementation has more in common with the *heavyweight components* of AWT. This confers benefits such as more fidelity with the underlying native windowing toolkit but it causes an increased exposure to the native platform in the

programming model. SWT is relatively simpler than Swing, which is an alternative to the mentioned toolkits. This has led some people to state that SWT lacks functionality when compared to Swing. As its components are also *heavyweight*, SWT suffers from the same portability limitation of AWT.

Using Java, Swing appears as another alternative. Swing widgets provide more sophisticated GUI components than the Abstract Window Toolkit. It supports pluggable look and feel, which means that any supported look and feel on any platform can be shown. Swing is platform independent in both implementation and expression (Java). It allows the custom implementation of framework interfaces through which users can override the default implementations. Swing is a component-based framework and its objects asynchronously fire events and respond to a well known set of commands specific to the component. Swing can also respond at runtime to fundamental changes in its settings. However, its components rely on AWT containers and are often described as *lightweight* because they do not require allocation of native resources in the operating system's windowing toolkit. Much of the Swing API is a complementary extension of the AWT rather than a direct replacement. In fact, every Swing *lightweight* interface exists in an AWT *heavyweight* component.

As the DTMS-P2P tool was implemented in Java, this language was chosen for the accomplishment of the interface because some of the resources and classes of the DTMS-P2P platform need to be used for a correct interaction between the interface and the network. For all its advantages, Swing was chosen for this implementation.

3 The Distributed Traffic Monitoring System

The Distributed Traffic Monitoring System with a Peer-to-Peer Architecture (DTMS-P2P) is a versatile, scalable and easily manageable traffic monitoring system based on a P2P hierarchical architecture [Salvador2005]. This system can be used to perform both active and passive measurements. Given that the monitoring elements may have different computational resources (e.g. processing capabilities, storage space or network connections) and the availability of those resources may vary drastically over time, the DTMS-P2P tool was implemented with a totally distributed hierarchical architecture similar to Gnutella 0.6. The adoption of a P2P architecture allows high tolerance to failures and distributed storage of measured data. This architecture is also advantageous for traffic monitoring in wide area network environments. Moreover, access and querying of measured data can be performed using traditional P2P file sharing schemes.

3.1 System Elements

The system consists of two main entities: the probe and the client. The probe performs the measurements and stores the results. The client is the interface between the monitoring system and the user. It is used to configure the system (e.g. configure the mode of the probes, request the list of probes in a measurement group, etc), configure the measurements and retrieve the measured data. Probes can run multiple software modules and are responsible for their integration in the P2P platform.

To improve the system scalability, probes are organized in groups and each group is responsible for monitoring a particular network area. Within a group, one or more probes, called super-probes, are responsible for controlling other probes and communicating with other groups. The super-probe element has the same definition of the Ultra-peer element of the Gnutella network. The super-probes are probes with enough available resources (CPU usage, free memory, storage capacity, etc) that can be used to control other probes. A monitoring element can alternate between both modes of operation in order to adjust to different network conditions and resources availability. These elements can run tests measurements. Figure 1 represents the hierarchical relationship between the system elements.

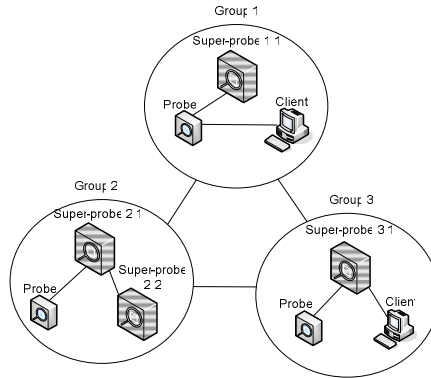


Figure 1 - Hierarchical relationship between the system elements

Groups are identified through the use of a unique ID called Group ID. This value is assigned to each node before it is initiated.

All network elements (probe, super-probe or client) must keep a list of known nodes (probes and super-probes) in the network. This list is used in the connection setup, when a node is trying to connect to the network. Before starting an element, the network administrator must provide it with this list of node addresses. For example, this list can be supplied through a file (node cache file).

The system should guarantee connection to any node connected to the network. The probes keep only one connection active, the connection to a super-probe of its measurement group. All super-probes should be interconnected. A super-probe acts as a proxy to the monitored network for the probes that are connected to it. This configuration guarantees scalability of the network by reducing the number of network nodes that are involved in message handling and routing, as well as reducing the actual traffic among them.

A client can connect to any super-probe of any group to obtain information of the entire monitored network. This information can be used to configure measurements at any probe and to retrieve their results. Thus, access to the monitoring system is completely distributed. Besides, a client can connect to a probe to obtain the address of a super-probe to which it should connect. The addition of a new probe to the monitoring system should be transparent: the probe connects to a super-probe and is automatically integrated on the network.

The DTMS-P2P system should support any monitoring system. When running a node, the network administrator must specify which monitoring systems the node will

support and with which restrictions. Different monitoring systems may have different restrictions regarding their running mode or ports.

Nodes must store the results of all scheduled measurements in a file (heavy data file). Thus, for each configured measurement session there is a heavy data file with the measurement results. These files are stored at the node that created them and are possibly replicated at other nodes (super-probes included). The replication improves the system reliability, since data can be retrieved even if the node that made the measurements becomes inactive or inaccessible. The search and retrieval methodologies for measured data that is stored at the nodes are similar to the ones used in P2P file sharing applications.

3.2 Measurement Commands

The DTMS-P2P system is used to configure and execute, at any element of its network, any monitoring command. These modules must be installed at the remote node and will be used to perform the measurements.

In this network, the client is the interface to configure the measurement modules installed at the nodes. Through it, a user can choose the module he wants to use and which measurement should be done. Also, the client must illustrate to the user how to configure the command. To achieve this, the client requests to the remote node the restrictions of the command.

3.3 Types of measured data

In the network there are two types of measured data files: light data and heavy data files. The first one stores the system parameters (nodes addresses) and statistics (round trip time - RTT) related to each group. This file is generated periodically by each super-probe and is broadcasted to all super-probes of the network. The file comprises the RTT statistics between a super-probe and all the other elements of its measurement group and all the other measurement groups connected to it. As the file only contains the RTT statistics, it provides a coarse view of the network. Any client connected to a measurement group can request the file to its super-probe.

The heavy data file stores the results of all measurements and can include packet or flow information, and various statistics related to the command. Therefore, for each configured command there will be a heavy data file with its results. The file is stored at the

node which generated it and can be replicated at other nodes of the same measurement group or other measurement groups. This replication improves the system's scalability as the heavy data file can be downloaded even if the node which generated it becomes inactive or inaccessible. A download can also be more efficient if it is simultaneously made from multiple sources that store the file.

3.4 Data Retrieval

A user running a client element is able to retrieve the results of previously configured test measurements. A user is also able to get the results of previously performed test sessions that are shared between any elements of the DTMS-P2P network. To gain access to these results the user must perform a search in the network to discover where they are stored.

To perform a search a client must first send a request to its super-probe. This super-probe will then broadcast the received message to all super-probes connected to it, in the case of a global search. Any super-probe receiving this request must forward the message to all nodes they are connected to. In the case of a local search, which is only performed on the node's measurement group, the super-probe sends the request to all connected nodes. In both cases, the nodes sharing files which satisfy the search criteria must answer to the request. After receiving the responses to the search command, the client presents the results to the user and then he can choose which files to download.

4 Module for identification of Internet Applications

This chapter presents and explains the implemented module for capturing and identifying traffic using the payload analysis method. This technique was adopted as we believe it is the most accurate methodology. The developed tool attempts to identify traffic only through payload analysis, does not look at the used ports and also does not pay any attention to the statistical properties of the captured flows. With this methodology, we can reach very accurate results.

The identification tool relies on a database of characteristic signatures that will permit the identification of the various protocols. It also allows users to write their own rules for traffic detection, which makes it adaptable to new emerging protocols.

The proposed methodology has some advantages over the others, such as:

- It can identify applications that use ephemeral ports or try to disguise themselves through the use of reserved ports;
- It allows the identification of flows with similar characteristics;
- It permits the identification of the client that generates the traffic;
- It identifies traffic based only on the first packets of the flow.

An important advantage of this tool is its scalability, as there is no limit for the traffic identification capability as new rules can be easily added, thus adapting to new protocols. The rules are written in a simple format, with a set of parameters that are vital for the payload inspection: the rules enable inspection within a chosen range of the payload and also allow for the search of characteristic strings in hexadecimal format.

The tool is intended to identify all kinds of traffic, provided that the corresponding rules are available. It was not designed to identify a restricted set of protocols, but all applications a user may wish to identify. The developed module can also be incorporated at any probe of a totally distributed monitoring platform that is being developed at our research centre.

4.1 Tool Implementation

The implementation of the classifier involved many phases. The first is the rules' processing which are passed to the program in a file. The second phase is the session

configuration, which consists in processing some parameters selected by the user (these parameters will be described later). The next phase comprises the initialization of all structures that are needed to capture packets and the initialization of the capture itself. The next stage is the classification of flows, which is based on the pre-established rules. This sub-section will provide an overview of the methodology and the subsequent explanation of all the phases.

4.1.1 Methodology Overview

The proposed tool captures traffic and groups it into flows using the five-tuple: Source IP Address, Source Port, Destination IP Address, Destination Port and Higher-layer Protocol. Flows are saved into a hash table for later analysis. This hash table will contain several informations about the flows:

- number of packets;
- minimum, average and maximum packet size;
- minimum, average and maximum packet inter-arrival time;
- source and destination IP addresses;
- source and destination ports.

The inter-arrival time is defined as the period of time elapsed between the sending of two consecutive packets in the same direction. However, these values will not be used for flow classification.

The classification process is implemented during the capture. The identification of the underlying protocol and application is achieved through the use of a database of distinctive strings that are passed through a list of rules. This list is created when the program begins. These are the characteristic strings related to a protocol, which are usually carried in the beginning of the payload and distinguishes it from all the others. As an example, most Gnutella packets carry in their payload the string “Gnutella”, differentiating undoubtedly this protocol from all others. This fact compelled us to investigate a set of protocols and applications in order to determine each particular digital signature. A series of applications were chosen, followed by a study of their characteristic traffic. Payload inspection was done and some signatures were determined. In the following paragraphs some of the studied protocols will be enumerated.

- **MSN Messenger**

The MSN Messenger network is an instant messaging network created by Microsoft and is one of the most used messaging services [MSN2007]. A client must first connect to the server to get access to the network. A series of packets are sent to perform registration. After gaining access, the client is able to communicate with other connected clients. After capturing and studying the traffic generated by this protocol, some of the identified strings were: “VER”, “CON”, “NLN”, “BYE”, “XFR”, “FLN”, “USR”, “JOI”, “CAL”, “MSG” and “PNG”.

- **Yahoo Messenger**

This protocol is used by the Yahoo Messenger instant messaging clients. The messages of this protocol always begin with the protocol name, “YMSG”. However, if the application is placed behind a firewall or a proxy server, HTTP routes are used and, consequently, we will have HTTP requests and responses instead of messages with a string in the payload. These known requests are used for uploading messages from the client and for downloading all messages which have been stored in the server. In this case, the client stays connected until it fails to send a request for a certain period of time. The identification of this protocol relies on these operational features.

- **ICQ/OSCAR**

This protocol is used in the instant messaging program with the same name [OSCAR2007]. ICQ uses the FLAP protocol to facilitate datagram-oriented communication between clients and servers. The identification of this protocol can be done when a client sends an “HTTP GET” message to the ICQ's proxy. This message carries the “login.icq.com” string in the payload, allowing its detection. Besides, flows are identified through the FLAP ID byte, which takes the value 0x2a and is always placed at the beginning of the payload.

- **RFB**

The RFB (Remote Frame Buffer) Protocol is used for remote access to graphical interfaces [Richardson2006]. Connections in this protocol begin with a handshaking between both elements in an attempt to agree in the protocol version to use. It starts with the server sending the highest version it supports, to which the client replies with the version that should be used. The packets exchanged in this phase have, in their payload, the string “RFB xxx.yyy\n”, where xxx and yyy are the major and minor version numbers.

- **SQL**

The SQL (Structured Query Language) Protocol is used for database management systems communication that takes place between a server and a client. The messages used for identification are “SELECT DATABASES” and “SHOW DATABASES”.

- **SIP**

SIP (Session Initiation Protocol) is an application-layer signalling protocol for creating, modifying and terminating multimedia sessions over an IP network [SIP2007]. SIP packets carry, in their payload, some strings that can be used for identification purposes: “OPTIONS”, “REGISTER” and “SIP”.

- **H.323**

This is a standard protocol for multimedia communications that was designed to support real-time transfer of audio and video data over packet networks, like IP [IEC2007] [Javvin2007]. The standard involves several different protocols covering specific aspects of Internet telephony: H.225 RAS is intended to provide registration and authentication services between clients and gatekeepers (entities that are responsible for managing a group of terminals and gateways); H.225 call control is used for establishing and configuring connections between endpoints and creates a reliable TCP channel for multimedia sessions; H.245 provides end-to-end control messages between endpoints. Both H.225 and H.245 have, in their payload, a field called Protocol Identifier that will be used for identification.

- **HTTP**

Hypertext Transfer Protocol (HTTP) is the network protocol used to transfer information on the World Wide Web (WWW). It is based on a client-server architecture and follows a request-response model. The “HTTP/1.1 200 OK” message was the only one used for identification because, as we will see later, many applications use HTTP messages. So, our methodology first analyzes all HTTP-like messages in order to verify if they belong to any application different from HTTP; if this is not the case, the flow is classified as web browsing.

- **NetBIOS**

NetBIOS protocol allows applications running on different computers to communicate with each other. Detection of this protocol is based on the NameQuery field

of the UDP packets. This field is located on the third byte of the payload and takes always one of two possible values: 0x0110 and 0x110a.

- **eDonkey**

The eDonkey network is populated with servers and clients. Clients connect to one server to get network services and this connection operates as long as the client is in the system. Servers do not communicate with each other and perform general indexing services. A client uses one TCP connection to a server in order to get information from it regarding desired files and other connected clients. As servers do not store files and files are broken into chunks, clients can use several connections to other clients to upload and download. This means that a client can download different pieces of the same file from different clients [Kulbak2005]. After analyzing eDonkey packets, we have discovered that every packet includes a characteristic string equal to 0xe3 immediately after the TCP header.

- **Gnutella**

The Gnutella network follows a P2P decentralized model, where every client can be simultaneously client and server [Gnutella2007]. These are called *servents*, or *gnodes*, and can perform both client and server tasks: they provide interfaces through which clients can issue queries and view search results and also accept queries from other *servents*. As a result of its distributed nature, the network will not stop working if one or more *servents* go offline. A session begins when a *servent* connects to another by sending the following message to advise its presence and requesting connection:

```
GNUTELLA CONNECT/<protocol version string>\n\n
```

The receiver of this message then replies with the next message accepting the connection and returning a list of currently active *servents*:

```
GNUTELLA OK\n\n
```

Once a *servent* is connected to the network, it communicates with all other connected *servents*, sending the first message all over the network. The *servents* reply to this message sending a Gnutella packet about them. When a file is found and chosen for download, a separate HTTP session is established between the client and the host of the resource. The *servent* that wishes to download the file sends a HTTP packet:

```
GET /get/<File Index>/<File Name>
/HTTP/1.0 \r \n
Connection: Keep-Alive\r\n
Range: byte=0-\r\n
User-Agent: <Name>\r\n
\r\n
```

A note should be made: if the *UserAgent* field is captured, an identification of the client is also possible.

Our methodology for identifying of the messages of this protocol was based on the fact that the TCP payload should begin with the string “GNUTELLA”. When receiving HTTP messages, the *UserAgent* field should also have the same string or the name of the used client.

- **Direct Connect**

In the network created by this protocol there are hubs, clients and a *HubListServer*. Hubs are central servers to which clients connect, thus, implying a centralized network. Hubs also facilitate communication between clients and give information about them while responding to file searching queries. All hubs are registered on the *HubListServer*, which then acts as a name service. Clients discover hubs by asking the *HubListServer*. Clients also store files and respond to search queries for those files. When a file is requested for download, communication is established directly between the involved clients in a true P2P fashion. In our study, we have noticed that the TCP commands always have the format:

```
$command_type field1 field2 ...|
```

The command type field can be one of the following: “Logedin”, “Key”, “MyNick”, “Lock”, “Direction”, “FileLength”, “HubName”, “Send”, “Get”, “Canceled”, “Validate”, “GetPass”, “MyPass”, “Hello”, “MyINFO”, “GetINFO”, “GetNick”, “Nick”, “OpList”, “MultiConnect”, “Connect”, “Rev”, “Kick”, “SR”, “Search”, “OPForce”, “ForceMove”, “GetListLen”, “ListLen” and “MaxedOut”.

- **Bit Torrent**

The network created by this protocol comprises clients and a central server that coordinates actions from all clients and does not have any knowledge about the files’ contents [BitTorrent2007]. Servers do not search for files; clients browse the Web searching for a torrent file that contains the metadata about the file. The philosophy of Bit

Torrent also relies on breaking files into chunks and distributing them among users. Users who download a torrent file are also uploading it to the remaining clients [Bit T., 2007]. The signature used by this protocol is:

```
<0x13><BitTorrent protocol>
```

Table 1 resumes all the gathered characteristic signatures that were enumerated above.

Protocol	Signature	Transport protocol
MSN Messenger	“VER”, “CON”, “NLN”, “BYE”, “XFR”, “FLN”, “USR”, “JOI”, “CAL”, “MSG”, “PNG”	TCP
Yahoo Messenger	“YMSG”	TCP
ICQ	0x2a	TCP
RFB	“RFB xxx.yyy\n”	TCP
SQL	“SELECT DATABASES”, “SHOW DATABASES”	TCP
SIP	“OPTIONS”, “REGISTER”, “SIP”	TCP/UDP
HTTP	“HTTP/1.1 Get”	TCP
NetBIOS	0x0110, 0x110a	UDP
eDonkey	0xe3	TCP
Gnutella	“Gnutella”	TCP
Direct Connect	“\$Logedin”, “”Key”, “\$MyNick”, “\$Lock”, “\$Direction”, “\$FileLength”, “\$HubName”, “\$Send”, “\$Get”, “\$Canceled”, “\$Validate”, “\$GetPass”, “\$MyPass”, “\$Hello”, “\$MyINFO”, “\$GetINFO”	TCP
Bit Torrent	“0x13BitTorrent protocol”	TCP

Table 1 - Characteristic signatures of the different protocols

4.1.2 Parsing the rules

As mentioned above, a database of strings is supplied to the application. This consists of a file with all signatures that must be identified and a series of related parameters that will determine the way the packet’s payload should be observed. The file and some of the rules are shown in Figure 2.

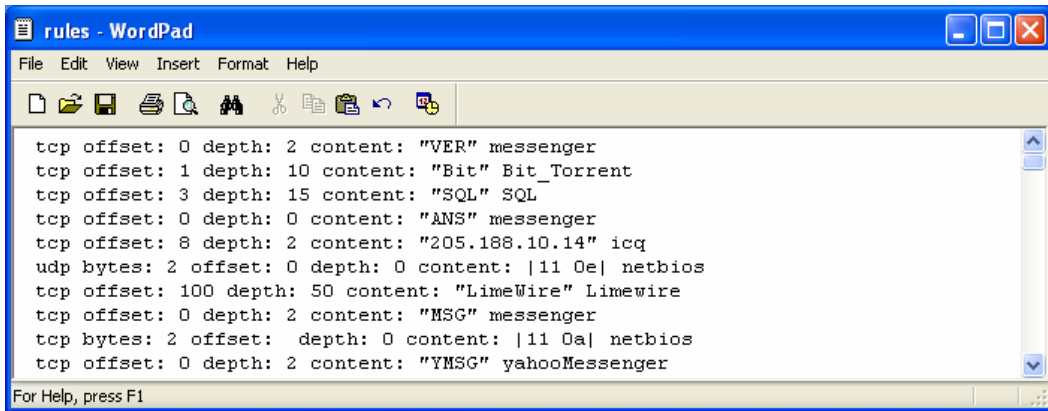


Figure 2 - File of signatures and related parameters

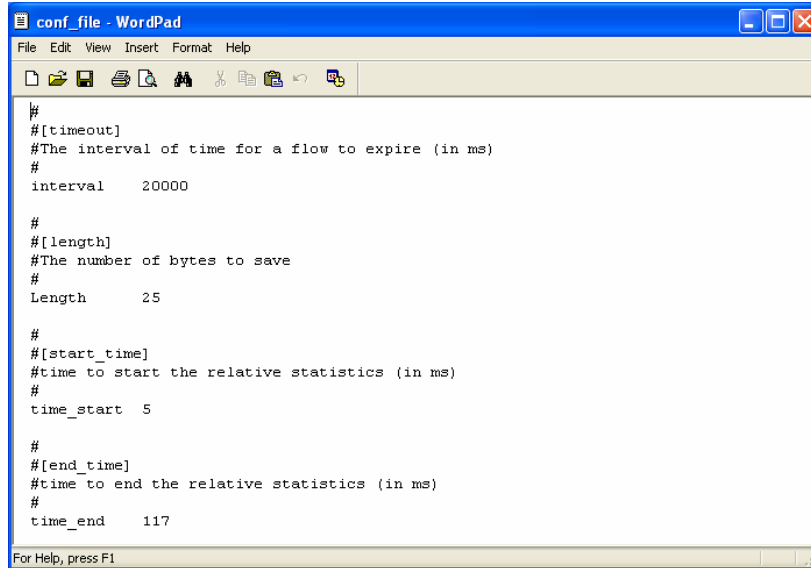
The parameters are:

- Protocol: the transport layer protocol used in the IP communication (TCP or UDP). Search is only performed in the packets that match the selected protocol.
- Offset: from where to start the search in the payload. This value is written in bytes.
- Depth: defines until which byte the search will be performed. This parameter, together with the offset, allows the examination of the payload within a chosen range of bytes. This will lead to a quicker and more accurate examination of the payload.
- Content: the characteristic string to search for. This can be written in ASCII format or in hexadecimal, since some signatures are only in this format.
- The last parameter is the message to print when identification is achieved.

A structure is created to store all the information regarding the rule and all rules are parsed together to form a linked list. This list will be used each time a new packet arrives and a continuous search is made in its payload according to the parameters that are stored on each element of the list.

4.1.3 Configuring the capture

The user is urged to configure certain parameters of the capture. This configuration is made through a file. Figure 3 shows the capture parameters.



```
conf_file - WordPad
File Edit View Insert Format Help
#
#[timeout]
#The interval of time for a flow to expire (in ms)
#
interval 20000
#
#[length]
#The number of bytes to save
#
Length 25
#
#[start_time]
#time to start the relative statistics (in ms)
#
time_start 5
#
#[end_time]
#time to end the relative statistics (in ms)
#
time_end 117
For Help, press F1
```

Figure 3 - Capture parameters

These parameters include:

- The interval of time for a flow expiration,
- The number of payload bytes that must be captured,
- The time instants for starting and ending the collection of relative statistics for each flow, in milliseconds. These statistics are related to the chosen time interval.

4.1.4 Capturing and storing the packet

As previously mentioned, each time a new packet arrives its payload is examined. Besides, packets are grouped into flows and this is done by using a hash table. Elements are saved in this table using a key that identifies them and allows for the search of a particular flow in the list. This table stores elements of the *flow* structure type. Each element saves the following information:

- Statistics concerning the incoming and the outgoing packets. This information contains the number of packets sent, the maximum, minimum and average packet size, the maximum, minimum and average inter-arrival time and the time the first and the last packet arrived;
- Information concerning the five-tuple (Source IP Address, Source Port, Destination IP Address, Destination Port, Higher-layer Protocol);

- A string containing the name of the file where the captured payload was saved;
- Flags indicating if the FIN flag of the TCP connection flow has already been seen, if the flow can be erased and if the flow has been classified;
- A pointer to the list of rules;
- Integers indicating the number of packets with payload which have been captured and the number of packets analysed.

A second linked list of elements of the type structure *flow* is constructed and elements are inserted each time a new flow arrives, which occurs when no match in the hash table is found. The reason for maintaining this second list is to have an exclusive list the program will examine and consequently will insert or erase elements according to the results of the classification process. It should also be noted that the hash table stores all the information concerning the flows statistics, information that will be presented at the end of the capture. Therefore, no element can be removed.

The linked list is constructed as follows: the first arrived flow will be located at the head of the list and each time a new flow arrives it is placed on the next position. When the first element is classified, this flow is erased from the second list to avoid being processed again, and a new head of the list is found in the next element. This process finishes when all the flows are classified. In the case of having flows which were not classified, the application finishes the process when all flows expire.

The capture process starts by performing a search in the hash table each time a packet is captured. If no match is found in this list, meaning that a packet from a new flow has arrived, a new element is created and inserted in both lists mentioned above. The variable in the structure that stores the starting time of the flow registers the capturing instant time of this packet and the remaining elements are initialized. The file where each captured payload will be saved to is created with the following name:

```
SourceIPAddress SourcePort<->DestinationIPAddress DestinationPort
```

Besides, the structures where the statistics of incoming and outgoing packets will be saved are created and initialized.

For the case of a packet that belongs to a flow that has been already instantiated, the corresponding statistics are actualized with the payload size and inter-arrival time values.

The following step consists of saving the payload to the corresponding file. This only occurs if the maximum number of packets with payload has not been captured yet. This is done since only a limited number of packets needs to be saved to achieve classification. Moreover, saving all packets with payload from a flow would be very expensive in terms of time and resources and, as we will see later, most of the saved data will not be read due to the fact that classification is achieved using only the first packets. Also, only a limited number of bytes, chosen by the user, are saved to the file. At this stage, we perform an examination of the payload before deciding how many bytes will be saved. If the packet corresponds to an HTTP Get message, then 200 bytes of the payload are saved. This amount of bytes was chosen since many P2P protocols request downloads of files using this message. Therefore, this packet can be misclassified as an HTTP flow when it is associated to a P2P flow. In this case, a bigger number of bytes must be saved in order to identify the client that made the request. This will allow a differentiation between the indicated protocols. If the packet does not correspond to the stated message, the number of bytes indicated by the user is saved. This procedure continues until the maximum number of packets with payload has been captured, the flow has been classified or the flow has expired. This occurs after the period of time defined in the configuration process passes. When capturing packets, the time elapsed since the starting instant of the related flow is determined and if this value is within the values chosen by the user the related statistics are updated. The state diagram shown in Figure 4 illustrates the capture process.

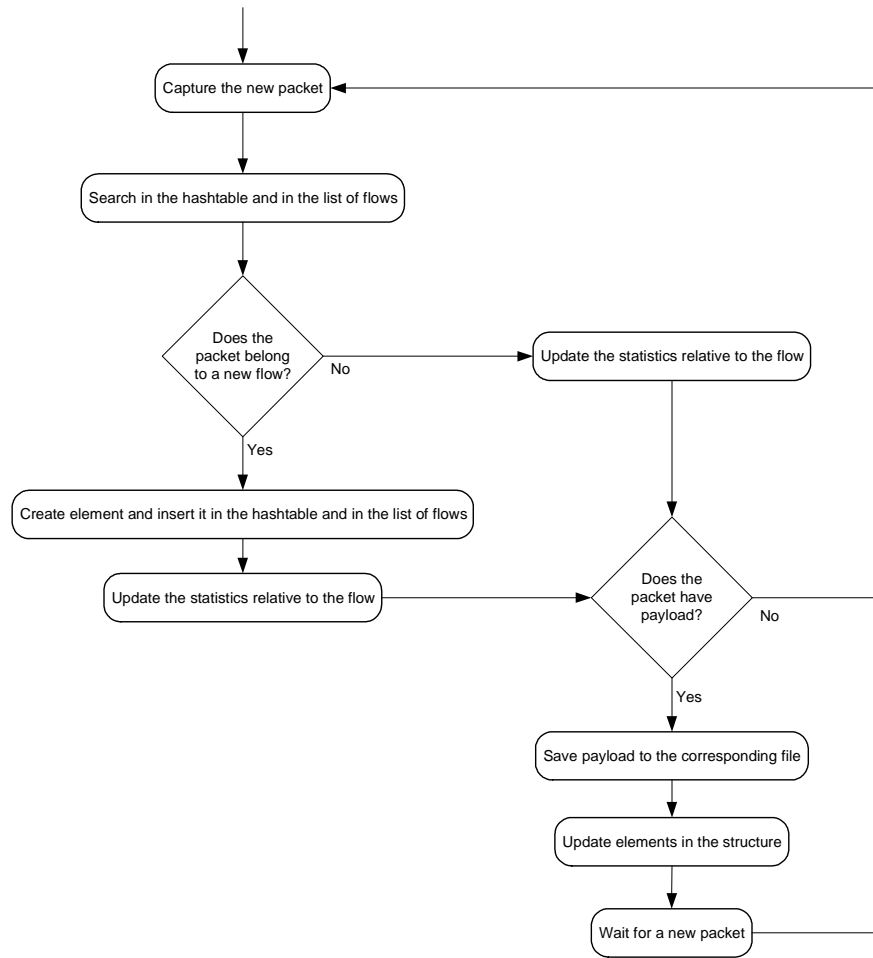


Figure 4 - Flow diagram of the capture process

4.1.5 Classifying the flows

After processing all rules and configuring the session, the capture of packets begins. At the same time, a thread is launched to classify the captured flows. The address of the head of the list of flows is passed to this thread. Then, the thread waits until enough packets with payload have been captured and starts the classification process.

The process initiates by verifying if the analysed flow has already been classified, if the FIN flag of the TCP flows has been captured and if all captured packets with payload have been read. In this case, the flow is erased from the linked list and the new head is put on the next element or simply a connection between the preceding element and the next is performed. Otherwise, the flow will be analyzed. This analysis starts by checking if there is any packet from the flow that has not been examined yet. In the structure that stores

information related to flows, there are fields that indicate how many packets have been captured and how many have been processed. If this is the case, the corresponding payload will be evaluated. The integer that contains the number of packets read from the flow is updated to avoid a second processing of this string. Then, the file with the name of the flow, contained in the structure, is opened and the string of the related packet is read. Subsequently, a new function is invoked and will perform a comparison between the payload that is read from the file and the strings that are passed in the rules. This comparison respects the limits imposed in each rule concerning the range of payload to inspect. As soon as a correspondence between the payload and one of the rules happens, the flow is considered to be classified. A pointer (belonging to the structure) that will contain the message to be printed is updated with the message field present in the rules and all flags, needed to indicate that the flow is classified, are actualized. The message indicating the type of flow is saved to the file as well as all related statistics data. The relative statistics are also printed to the file: these statistics are calculated based on the time interval the user has chosen. The flow is removed from the list and the program moves to the next element. If the end of the list is reached, the application moves back to the beginning and processes the whole list again. In this way, we are continuously inspecting for expired flows and decreasing the size of the list. This allows for a quicker classification. The flow diagram shown in Figure 5 summarizes these steps.

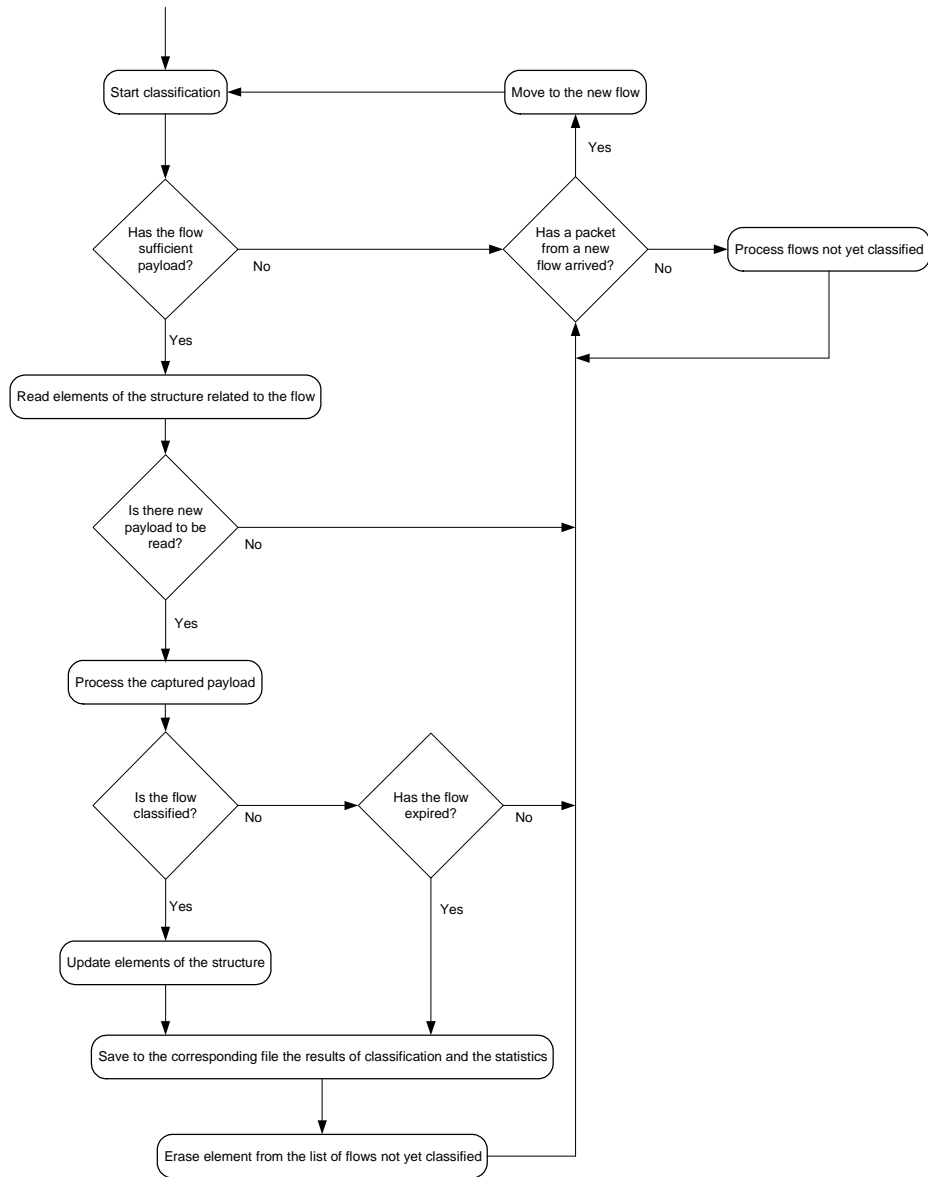


Figure 5 - Flow diagram of the classification procedure

As shown in the previous diagram, the list of flows is continuously processed. Each time a packet is classified or processed, the program waits for the arrival of a new flow. Until this occurs and sufficient payload has been captured, the program passes through the stated list and if a flow is considered as expired it is erased from the list and the process repeats itself.

Figure 6 shows the classification process.

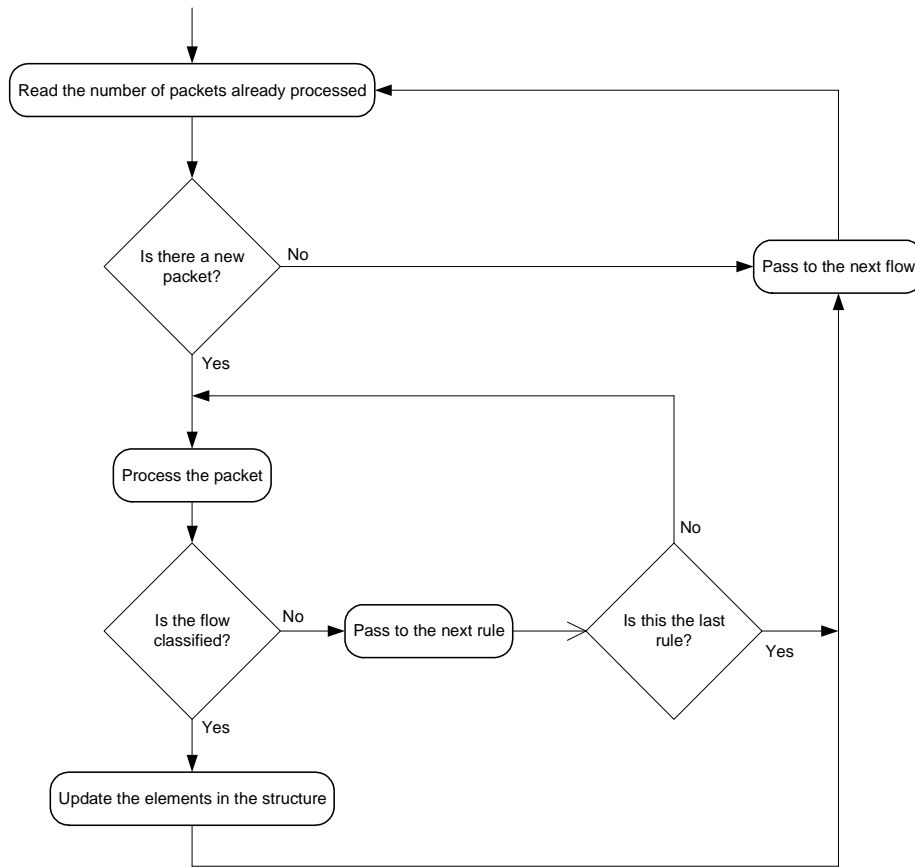


Figure 6 - Flow diagram of the classification process

4.2 Results

To test the developed application, we have generated traffic belonging to the applications identified above. Each application was run individually in order to obtain completely known and identified traces, each one related to a specific application. Traces were captured using Ethereal and, for each packet, we have stored the full header and the first 180 bytes of payload. Table 2 shows the dimension of the captured traces.

Type of application	Number of captured flows
P2P	512
Web browsing	90
Instant messaging	128
VoIP	35
Other	115

Table 2 - Dimension of the captured traces

After being captured, the traces were analyzed by the application and the classification results were examined. The application was able to distinguish and identify all protocols and, when required, it was also able to identify the client related to the captured flow. Table 3 presents the results achieved.

Protocols	Number of Analyzed Flows	Correctly classified flows	Incorrectly classified flows	% of correctly classified flows
e-Donkey	158	156	2	98.73%
Gnutella	150	146	4	97.4%
Direct Connect	41	40	1	97.5%
Bit Torrent	163	157	6	96.32%
HTTP	55	51	4	92.72%
HTTPS	40	40	0	100%
NetBios	85	81	4	95.3%
Messenger	78	68	10	87.18%
ICQ	40	40	0	100%
Yahoo Messenger	10	10	0	100%
RFB	20	20	0	100%
H.323	25	17	8	68%
SIP	10	10	0	100%
SQL	10	10	0	100%

Table 3 - Classification results obtained

From the above table, it can be concluded that the obtained results are very accurate. For P2P traffic, the average percentage of correctly classified flows was about 97%. For web browsing applications, the average percentage of correctly identified flows reached 92.72%. This also included the identification of all clients. For instant messaging protocols, the average value was of 95.72%. In the case of VoIP applications, the average classification efficiency was equal to 84%. An explanation for the misclassified flows will be provided later in this section. Table 4 shows some statistical values related to the correctly identified flows. As can be seen, the correctly identified flows have high numbers of packets per flow and bytes per packet. For P2P protocols these flows correspond to file transfers and to message exchanges between *peers*, while for instant messaging protocols they are related to conversations between clients. For all other cases, these flows are related to successful connection establishments between the different elements of the protocol and to traffic that was exchanged between them, leading to a high number of bytes in the payload.

Protocols	Average number of packets	Average number of bytes per packet
e-Donkey	5	77
Gnutella	17	96
Direct Connect	478	94
Bit Torrent	16	78
HTTP	18	94
HTTPS	20	114
NetBios	3	174
Messenger	60	87
ICQ	23	105
Yahoo Messenger	11	75
RFB	510	57
H.323	16	107
SIP	5	76
SQL	43	99

Table 4 - Statistical information about correctly classified flows

Table 5 shows some of the statistical properties of the misclassified or unclassified flows. An explanation for this misclassification will also be provided in the next paragraphs.

Protocols	Average number of packets	Average number of bytes per packet
e-Donkey	2	32
Gnutella	6	88
Direct Connect	6	40
Bit Torrent	7	43
H.323	8	320
NetBios	5	720
Messenger	76	68

Table 5 - Statistical information about misclassified or unclassified flows

For the e-Donkey case, it can be observed that the average number of packets is low, as well as the number of bytes per packet. These may indicate that the flows are due to *Acknowledge* or *KeepAlive* messages exchanged between client and server. These messages are usually composed by packets with no payload and the respective flows have few packets. Comparing the values obtained for this protocol in the last two tables, a difference can be easily seen.

For the Gnutella protocol, misclassified or unclassified flows have a similar number of bytes per packet as the classified ones. The reason for the misclassification lies on the

fact that these flows were originated by an *HTTP Get* message that was issued to download a file. These messages didn't carry the *UserAgent* field inside the captured payload. Therefore, these flows were classified as HTTP, but this problem can be easily solved if a higher number of payload bytes are captured. When these flows carried the mentioned field correct identification was achieved, as well as identification of the client that issued the request.

For the Direct Connect protocol, the misclassified packets have no payload at all or a small number of payload bytes, indicating that these can be *Acknowledge* flows. They can be also related to file transfers that were automatically stopped by the client and then restarted.

The lowest efficiency occurred for the H.323 protocol. Identification of its messages was based on searching for a particular field within the payload: the *ProtocolIdentifier* field. Some flows, related to *OpenLogicalChannel* and *LocationRequest* messages, do not carry this identifier which prevents identification.

The misclassified NetBIOS flows were due to an unexpected *NameQuery* flag.

For the Messenger protocol, misclassified flows can be related to file transfers made in binary and without any signature. This agrees with the values shown in the previous table: these packets have an average of 76 packets per flow and an average of 68 bytes per packet.

To illustrate the efficiency of the methodology, Table 6 presents the number of packets that were read in order to perform the identification. As shown, the technique is very effective. For almost all cases, it was able to accomplish protocol identification in the first captured stream. When this situation did not occur, it was able to identify based on the subsequent packets. However, it should be noted that in these cases it is not necessary to examine a high number of packets. This demonstrates the efficiency of payload analysis and that it can achieve very accurate identification while saving computational and memory resources. It also proves that the gathered signatures respected the main requirement of accuracy. The results obtained also prove that the developed tool can be used at high-speed links.

Protocol	Identification in the first packet	Identification in the following packets/ Number of packets read
e-Donkey	100%	
Gnutella	100%	
Direct Connect	100%	
Bit Torrent	100%	
HTTP	50%	38.89% - 2 packets 11.11% - 3 packets 5.55% - 4 packets 5.55% - 5 packets
HTTPS	100%	
NetBIOS	100%	
Messenger	100%	
ICQ	100%	
Yahoo Messenger	100%	
RFB	100%	
H.323	100%	
SIP	66.67%	33.33% - 2 packets
SQL	100%	

Table 6 - Number of packets needed to classify flows

4.3 Conclusions

As shown in the previous section, the developed module achieved very accurate identification results. All chosen protocols were identified with a very high percentage of correctly identified flows. Besides identifying the protocol that generated a given flow, in some cases the module was also able to identify the client. This consists of very interesting information and proves the capability of the module for identifying applications. Moreover, all the flows were identified in the first packets which also proves the efficiency of the used methodology.

5 API Implementation

5.1 Introduction

The second part of this thesis consists on the implementation of an API to establish the communication with the DTMS-P2P platform. In this chapter it will be presented and shown how it establishes the communication between application level modules and the DTMS-P2P platform. These application level modules may consist of graphical interfaces or statistical traffic monitoring tools which may execute several management actions on the platform using the API.

The API consists of a class which, through its different methods, sends messages to the monitoring network and its elements, retrieving also information from them. The role the API performs is illustrated in Figure 7. The concepts related to the mentioned platform were already described in Chapter 3.

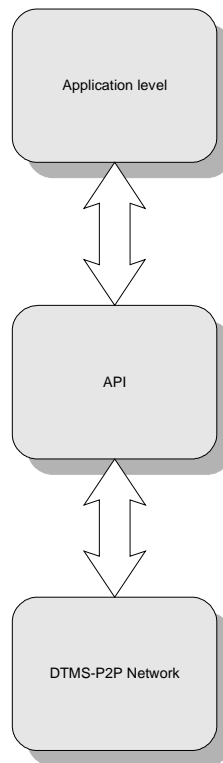


Figure 7 - Hierarchy of the API and the calling application

5.2 API Functionalities

The API must allow the calling applications to perform all the tasks that any client of the DTMS-P2P network performs: these tasks are listed in the diagram of Figure 8. Therefore, the API must, for each task an application may request to it, send the proper message to the network with the necessary parameters for a correct performance of the chosen command. In this way, the API has, for each command, a method that sends the respective message to the network and retrieves its results. Application level modules can use these methods to communicate with the DTMS-P2P network and its nodes. In the following sub-sections, will be listed the several methods of the API according to the functionality they belong to. For each method, the model of use cases can be consulted in Appendix A.1.1.

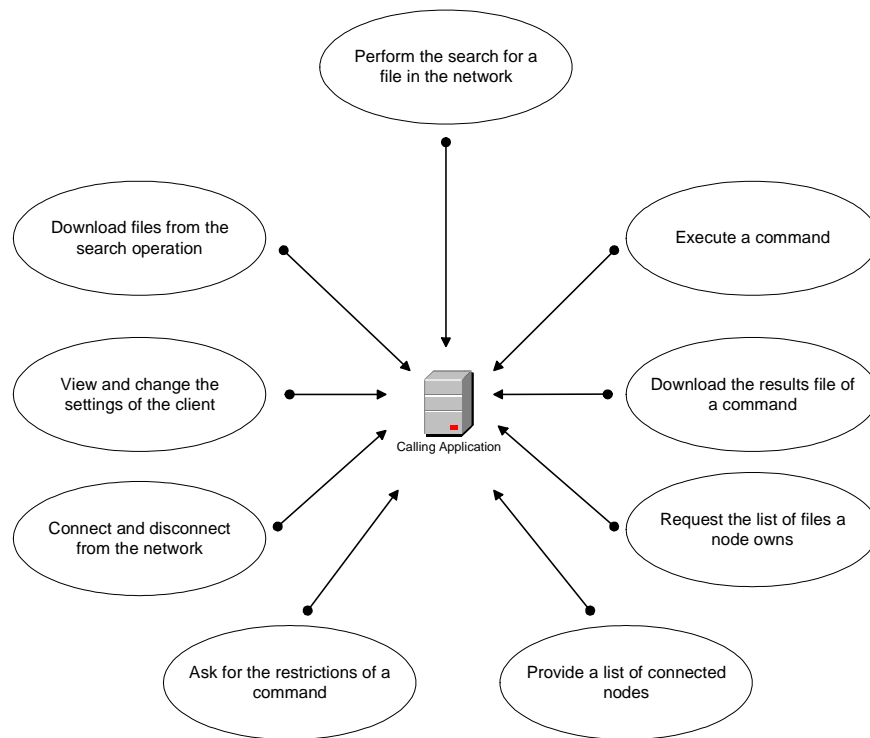


Figure 8 - Use Cases diagram

5.2.1 Change of the status of the client

In the following sub-sections will be explained the two methods which perform the connection and disconnection of the client from the network.

5.2.1.1 Disconnect method

To disconnect the client from the network, the *disconnect()* method can be used. This shuts down the connection of the client to its super-probe and, therefore, from the network [Veiga2007].

5.2.1.2 Connect method

After disconnecting the client from the network, the connection can be performed through the API. The method *connect()* starts the *OutgoingConnectionManager* that connects the client to the network [Veiga2007]. A variable will then indicate that the client is connected to the network.

5.2.2 Visualization and changing of the client's settings

The API must enable the retrieval of the client's settings as well as their modification. Therefore, it holds methods which list the settings and change their values. Moreover, a method was also created to change the settings to their default value. These are listed and explained below.

5.2.2.1 Show_settings method

This method returns some of the client settings in order to allow its presentation to the user by the upper-layers.

5.2.2.2 Listener_settings method

This method applies the new settings to the client which are passed as parameters to it. It subsequently changes each of the settings that were displayed by the *show_settings()* method.

5.2.2.3 Listener_default method

This method applies the default settings to the client. These are obtained from the client.

5.2.3 Command execution functionality

This sub-section lists all the methods of the API which allow the execution of the various steps that are involved in the performance of the several monitoring commands in

the DTMS-P2P platform. These steps are: the request of the restrictions of a command, the execution of the chosen command and the retrieval of the file which saves the achieved results. The methods which enable these operations are explained below.

5.2.3.1 Get_restrictions method

This method is used to retrieve, from a node, the restrictions associated to a command. It starts by verifying that a node has been selected to execute the command. If a node was selected, the message *requestListOfCommandRestrictions* will be sent to it [Veiga2007]. This message requests the selected element to send the restrictions of the command that the user selected. These are rules that must be respected by the user in order to achieve a correct performance of the command and indicate which parameters must and must not be used. After sending this message, the API locks in the resource used by the client to signalize it has received the restrictions. Subsequently, it constructs the messages that will be used by the upper-classes. If a node wasn't selected, the method urges for the need of the selection of a network element.

5.2.3.2 Execute_command method

This method executes the command selected by the user, with all the respective parameters. When calling applications ask the API to execute a command, this method will read the typed command along with the typed parameters. It will then construct the message *processCommandRequest* which will comprise the measurement group in which the command will be performed along with the IP address of the node which will execute it [Veiga2007]. The method also provides messages that indicate the state of the command execution and any error that may have occurred during its execution. In this manner, the user is taught how to correctly execute a command and is always aware of its state of execution.

5.2.3.3 Download_results_file method

In the case of a correct execution of the command, the name of the file where the results were saved is presented to the upper-layer modules and the possibility of downloading it is also provided. To complete this task, the method *download_results_file()* was implemented. It starts by searching for the node which has the chosen file and subsequently sends a message to the node indicating the intention of downloading it.

Afterwards, an instance of the *MultiSourceDownloader* is created, with all the necessary parameters and the download of the file is executed [Veiga2007]. When the download is finished, a variable of the API is updated indicating that the download is finished.

5.2.4 Search and retrieval of results file

As previously mentioned, the API must enable the search and retrieval of measurement files. Therefore the API holds two methods which allow the performance of the mentioned tasks: one for retrieving the list of files which satisfy a search criterion while the second performs the download of these files.

5.2.4.1 Show_search_results method

As previously mentioned, one of the operations the application layer modules can request is the search of a file in the network using a search criterion. To perform this task, the method *show_search_table()* was created. This function receives as parameters a string with the criterion to be used in the search and the measurement group where the search will be performed. Subsequently, it verifies if the introduced criterion is valid. Then, it selects the measurement group in which the search will be performed, according to the value received as a parameter, and sends a *resultsSearch* message to the nodes of the selected group with the inserted parameter [Veiga2007]. In response to this message, the nodes which own files matching the selected criterion send *QueryHit* messages to the client indicating that they own those files. When the client receives these messages, this method processes the name of the indicated files in order to present some characteristics, which include:

- The measurement group of the client;
- The IP address of the client;
- The measurement group of the node;
- The IP address of the node;
- The date of execution of the command;
- The command itself.

Subsequently, all these characteristics are inserted in a data array which can be used by the upper-layer modules.

5.2.4.2 Value_changed method

This method enables the download of files resulting from the search operation. It receives the name of the selected file and subsequently starts its download. Then, an instance of the *MultiSourceDownloader* class is created which before starting the download, tests if it has already been downloaded [Veiga2007]. If this is the case, the variables of this method are updated and indicate this situation. Otherwise, when the download is complete, the variables are updated so that they indicate the completion of the download. The operation of this method is shown in Figure 9.

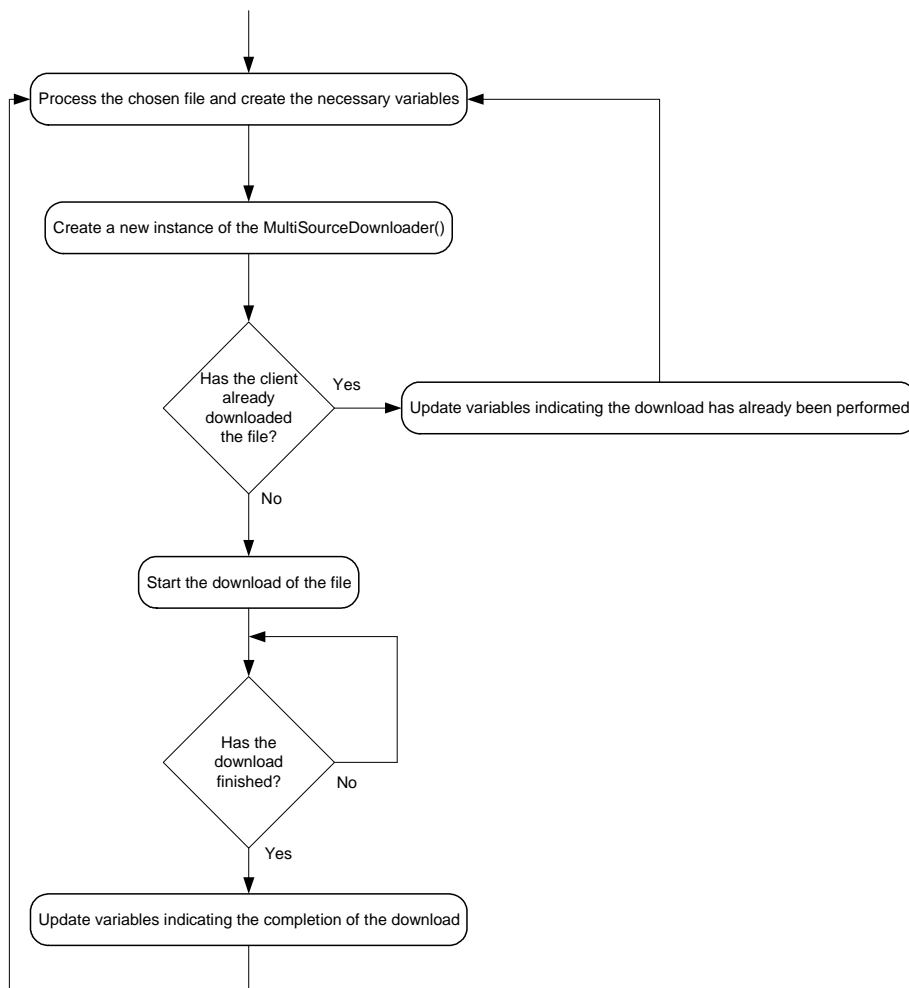


Figure 9 - Flow diagram of the method `value_changed()`

5.2.5 Listing of files of the client or of a node

In the following will be listed the methods which allow the user to visualize the files a client and a node of the DTMS-P2P platform hold.

5.2.5.1 Show_files method

A functionality the API must support is the presentation of the files the client owns. The method *show_files()* was implemented to provide this information to the upper-layer modules and it uses a variable of the client that contains the list of all files [Veiga2007]. The method creates two data arrays, one for each type of file (heavy-data and light-data files). These arrays contain various informations about the files, as mentioned in Section 5.2.4.1

Each file is placed in the corresponding array and both can be used by the upper-layer modules which can make use of the information provided to present the files.

5.2.5.2 Get_file_list method

The API may retrieve the list of files owns. This method receives the element selected and consequently downloads its list of files which is present in a file named *IPAdressOfTheNode_FileList.xml*. After the download, the method parses the XML file and adds all the file's names to a data array. This array can be subsequently used by the calling applications.

5.3 Conclusions

This section presented the API, a module that is in charge of the communication between modules needing to interact with the DTMS-P2P platform and the platform itself. The different functionalities of this API were introduced and their functions were explained. The API consists of several methods that can be used by modules needing to communicate with the network and retrieve information from it. It sends the required messages to the nodes of the DTMS-P2P platform and retrieves their responses. Subsequently, these are sent to the modules which use the API for communicating with mentioned platform. In the following chapter will be explained a graphical interface which was implemented in order to test the API.

6 GRAPHICAL INTERFACE OF THE DTMS-P2P TOOL

This chapter explains the implementation of the graphical interface implemented for testing the API. This interface was implemented to test the API explained in the previous section and allows the user to accomplish the several tasks that can be performed through the DTMS-P2P client. It uses the previously explained API for communicating with the DTMS-P2P platform. All concepts that are necessary to understand the interaction between the interface and the mentioned platform were explained in chapters 3 and 5.

The following section enunciates some principles that should be taken into account when implementing an interface. The remaining sections explain its implementation and its various functionalities.

6.1 The rules of user interface design

In the past, computer programs were designed having in mind that the user had to adapt somehow to the system. This approach is not appropriate nowadays, where the system has to adapt itself to the user [Mandel1997].

Users should have a successful interaction with systems in order to gain confidence on themselves. Well-designed interfaces should guide users to learn and enjoy what they are doing. They can also challenge the user to explore the interface behind their normal usage.

To achieve this level of interaction, several studies have been made and several principles have been discussed and agreed. The three main principles are:

- Place users in control of the interface;
- Reduce user's memory load;
- Make the user interface consistent.

The first rule indicates that the user should be provided with the ability of controlling the interface and performing all the operations he intends to in a simple manner. The interface should also:

- Allow users to change focus;
- Display descriptive messages;
- Provide reversible paths and exits.

The second rule explains that the interface should avoid users from having to remember information while interacting with it. Therefore the interface should:

- Provide visual clues;
- Provide shortcuts;
- Use real-world metaphors;
- Provide visual clarity.

The third rule illustrates that consistency is a key aspect of interfaces. The interface must teach users how to perform a command so that it can be applied to other situations. It also means that users should see information in the same logical and visual way.

6.2 Graphical Interface

As mentioned above, the main goals of the user graphical interface are to test the API explained in Chapter 5 and to present the measurement tool in an easily usable way to the user.

As previously mentioned, the interface will use the API for the communication with the DTMS-P2P network in order to allow the user to perform the several tasks allowed by the client of the monitoring platform. This is illustrated in Figure 10.

A fourth module was also implemented to create a graphical representation of the network. This module communicates with the application layer that instructs it to present the network representation and communicates with the API to get the necessary information from the network. The implementation of this module will be explained in sub-section 6.2.1.6.

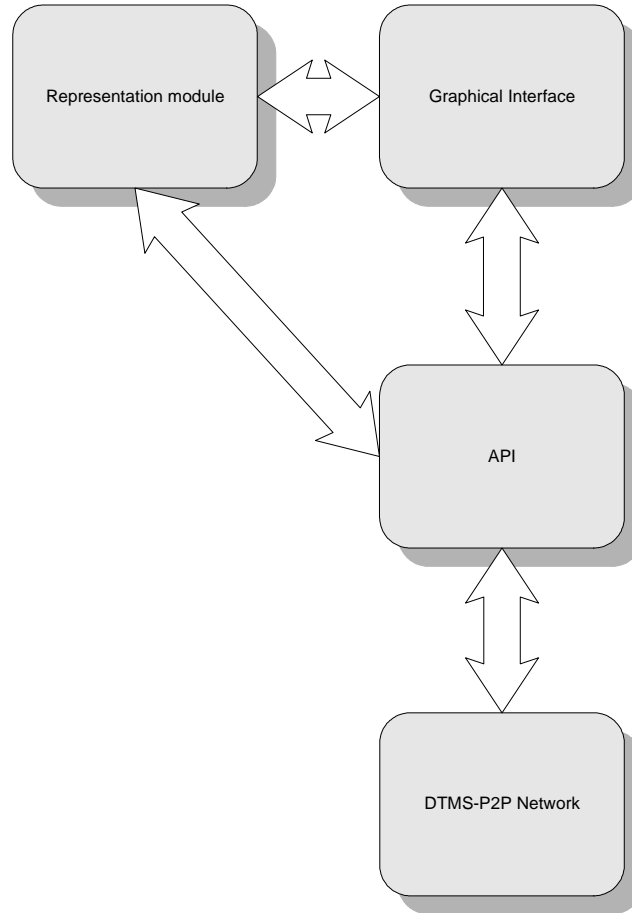


Figure 10 – Communication Hierarchy

Prior to the presentation of the interface, the client must connect itself to a super-probe in the network. This is done using the XML file *ClientCache* that contains a list of super-probes to which it must connect. Also as a probe can promote itself to a super-probe, all their addresses must be provided to assure that the client gains connection to the network. Consequently, the file contains all connected probes since there is no guarantee that the super-probes exist. The client will attempt to connect to the super-probes in the order that they appear in the *ClientCache* file. If no attempt is successful, it will try to connect to the probes.

As previously explained, when the client is connected it will download the *LightData* file in order to know the existing network nodes that will permit its representation.

The interface was developed using Swing components. For displaying or receiving parameters from the user, the most suitable components were used. For the insertion of

parameters for the various commands, text fields were chosen. For presenting various elements to the user and enable the choice of a particular element, combo-boxes were used. These components can be inserted into panels that can contain more components, such as buttons, tabs and many more [Eckel2002], [Zukowsky2005], [Walrath2004]. Each panel uses a layout manager responsible for positioning its several components regardless of the screen size and platform. Layout managers discover the space a component needs by calling the component's *getMinimumSize()*, *getPreferredSize()* and *getMaximumSize()* methods. These report the minimum, preferred and maximum sizes a component requires to be properly displayed. Therefore, each component must know the space requirements it needs. The layout manager will then use the component's space requirements to resize the components and arrange them on the panel. A layout manager was attached to each created panel where components were placed.

Each of the Swing components has the property to report all events that may happen and also report each kind of event separately. Therefore, event listeners were attached to the several components of the interface in order to allow it to know which operation the user has executed and act accordingly. In the next sections the implementation of the interface and the various operations a user can perform will be presented and explained.

The main frame of the interface, illustrated in Figure 11, presents all the previously mentioned components necessary to accomplish the various operations. The main frame consists of three tabs, each one responsible for a task, and a menu that presents to the user additional functionalities of the interface. To create the main frame, an object of the class *JFrame* was used. Using some methods of this class, the close operation, the content panes and the size parameters are set. The method to set the frame visible is also set. A frame can contain several components such as panels, tabs, etc. The frame also allows the exhibition of menus. The block diagram of the constructor of the class that creates and places all the components is shown in appendix A.2.1.

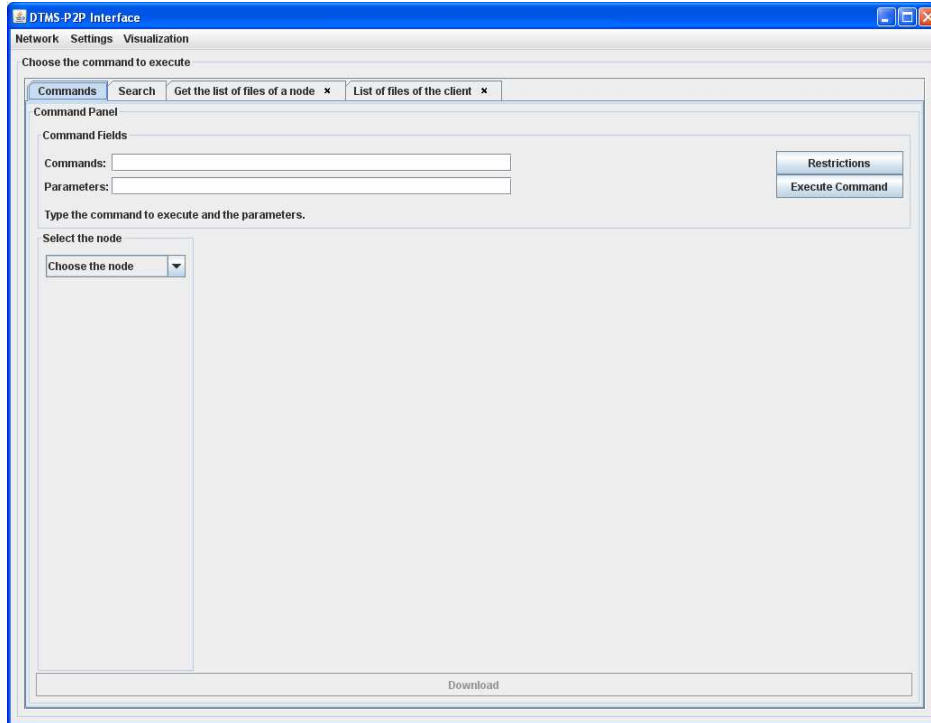


Figure 11 - Main Frame of the interface

6.2.1 Interface presentation

In this section will be explained the interaction between the user and the interface. The communication between the interface and the API will also be mentioned. This section provides sub-sections with the explanation of each of the interface functionalities, the way it allows the execution of the various tasks and the way it indicates to the user how to perform them. Figure 12 shows the block diagram of the implemented interface. The following sub-sections will list and explain the various functionalities the interface implements and how to correctly execute them.

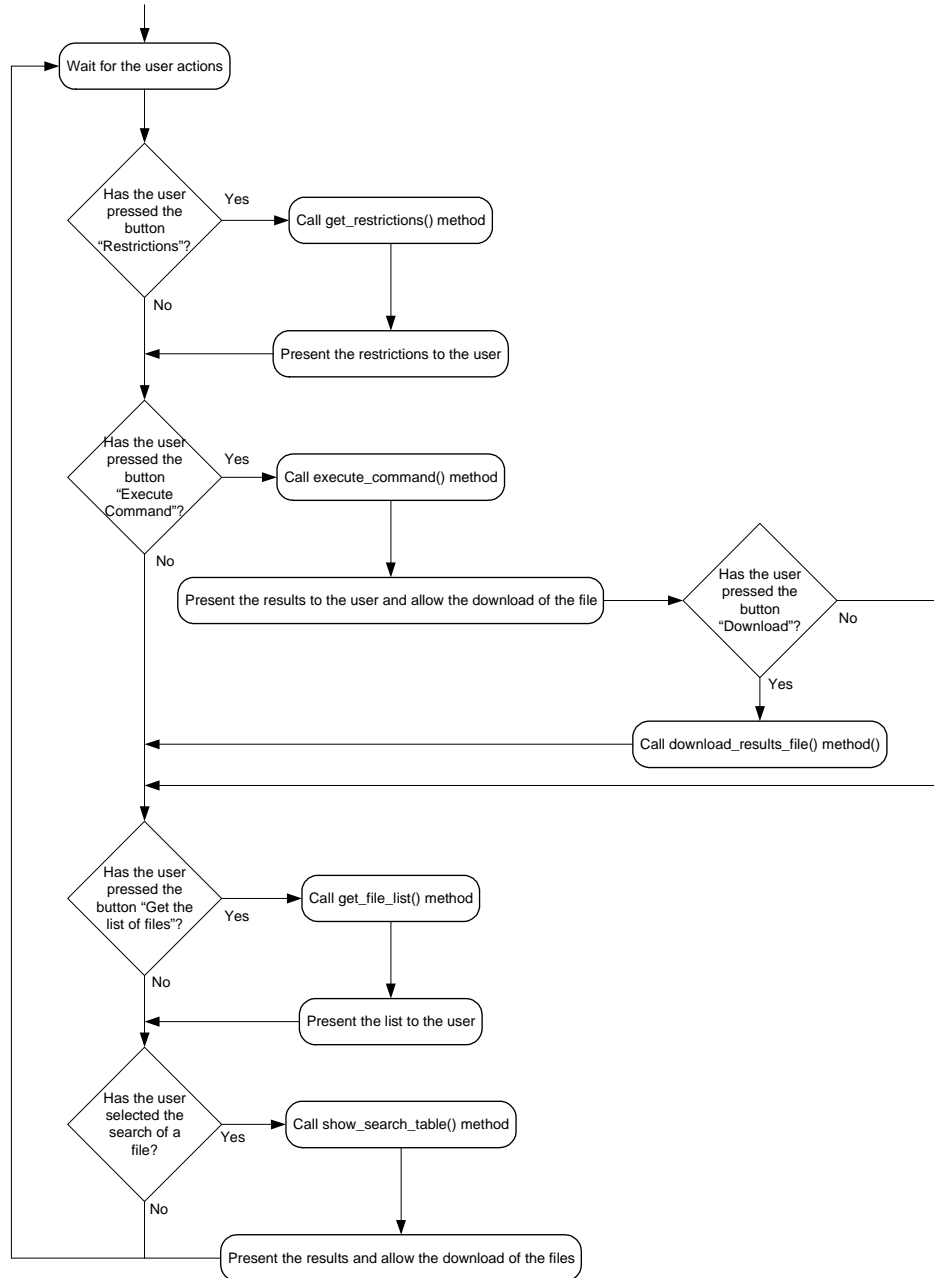


Figure 12 - Block diagram of the interface's tabs

Figure 13 shows the block diagram of the interface menu bar. The bar includes various items that allow a more organized and perceptive presentation of the information. The menu bar has also several event listeners connected and each item invokes the appropriated method from the API.

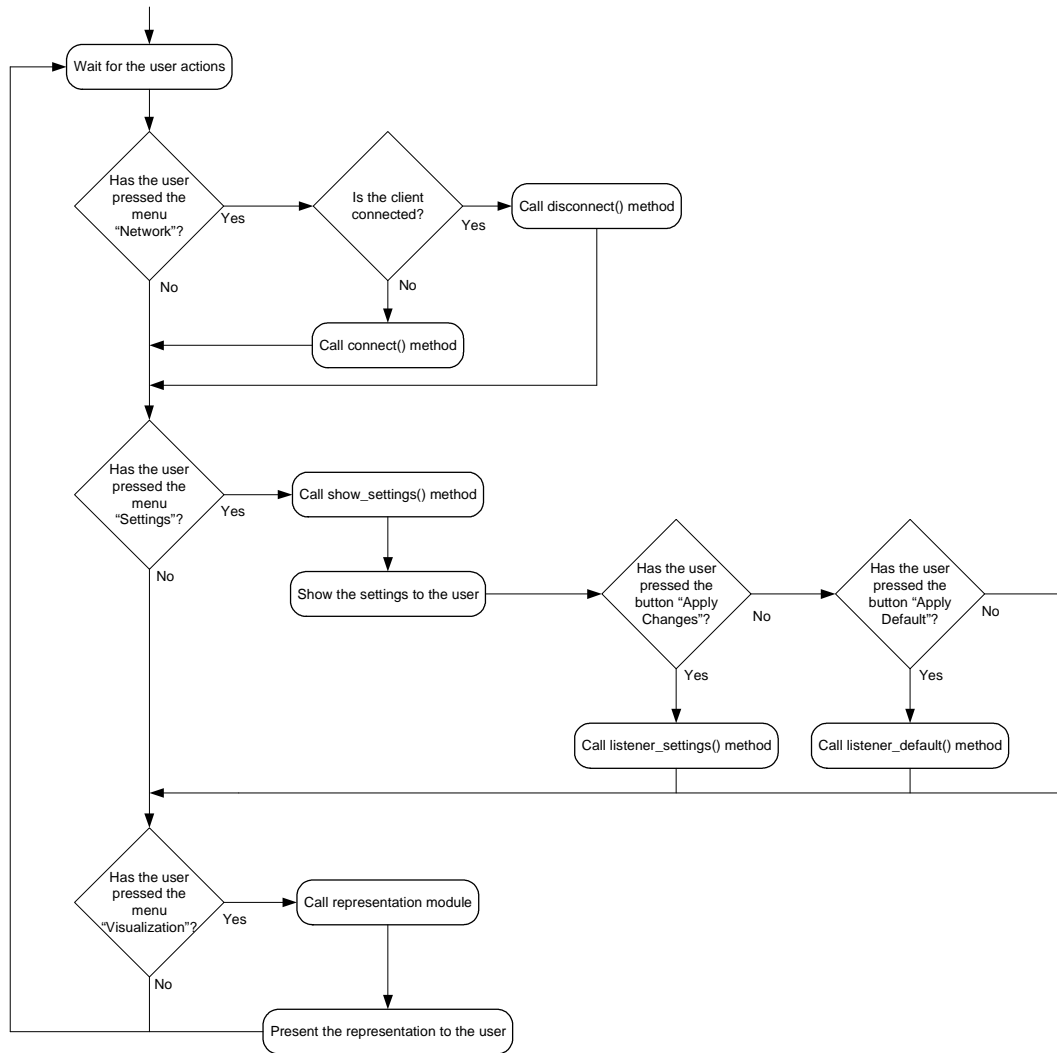


Figure 13 - Block diagram of the Menu bar

6.2.1.1 Change of the status of the client connection

As previously explained, the API has methods that perform the connection and disconnection of the client from the network. These are the *disconnect()* and *connect()* methods, which were explained in sub-sections 5.2.1.1 and 5.2.1.2.

To complete the connection or disconnection from the network, a menu is presented to the user with the title “Network”. If the client is connected this menu presents the possibility of disconnecting and if the client is disconnected the option of connecting is shown. The item the user must press is in the following menu: Menu → Network → Disconnect/Connect.

If the user chooses to disconnect, a message box appears after the client completes the task indicating that the client is no longer active, as shown in Figure 14. Subsequently, the menu presents the possibility of connection. If the user chooses to connect, a window message appears indicating that the operation will be executed.

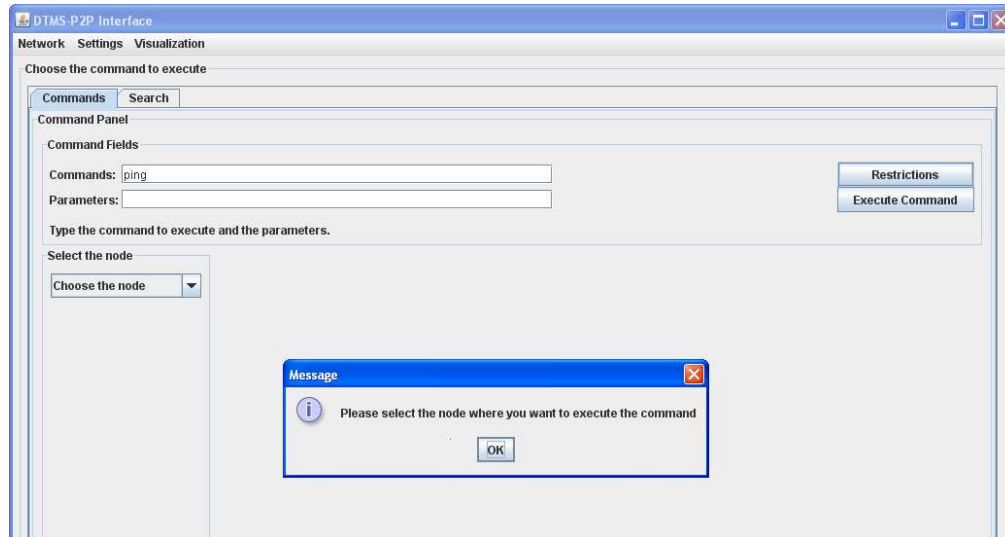


Figure 14 - Message box indicating that the client has been disconnected from the network

6.2.1.2 Change of the client settings

The view and change of the client settings is enabled through the use of the menu “Settings” that allows the user to change the client parameters. When the user presses the menu, a new window appears showing the current client parameters. This window consists of several text fields, one for each client’s setting, and a button indicating that the new settings can be applied to the client. A second button is presented to indicate that the user pretends to apply the default settings to the client. This is shown in Figure 15. The method that shows the client’s information is the method *show_settings()* mentioned in sub-section 5.2.2.1.

After changing one or various parameters, the user can indicate that these parameters can be applied to the client by pressing the previously mentioned button. The event listener will then invoke the method *listener_settings()* explained in Section 5.2.2.2 and when this operation is concluded a message will appear pointing out this fact.

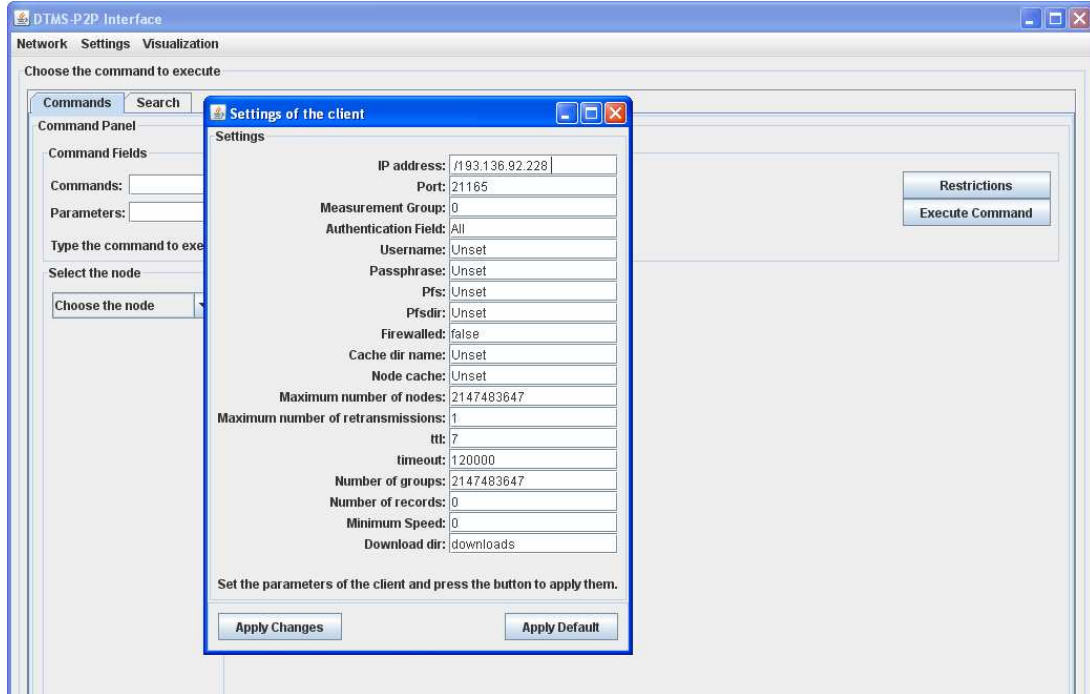


Figure 15 - Window showing the various client parameters

If the user wishes to apply the default settings to the client, the button “Apply Default” can be pressed. The method *listener_default()*, explained in section 5.2.2.3, is then invoked and these settings are applied to the client and a message appears indicating the appliance of the settings.

If the user places the mouse pointer in a text field, a tooltip text will appear indicating the function of the setting and its default value.

6.2.1.3 Command execution functionality

The performance of a command can be achieved through the use of a proper tab dedicated only to this functionality. The method from the API that performs the command inserted by the user is the *execute_command()*, depicted in sub-section 5.2.3.2.

The tab contains numerous components that allow the user to complete a command and interact with the network. This tab is divided in two panels. The first panel contains two text fields that allow the user to execute the command he wishes: the first allows the user to choose the command while the second lets the user insert the parameters necessary for the command to be executed as intended. Two buttons were also inserted in this panel:

one for asking the restrictions associated to a command and the other to execute it. The second panel contains a combo-box with the list of all elements connected to the network and allows the user to choose the node where the measurement is to be executed. Figure 16 shows how a node can be selected.

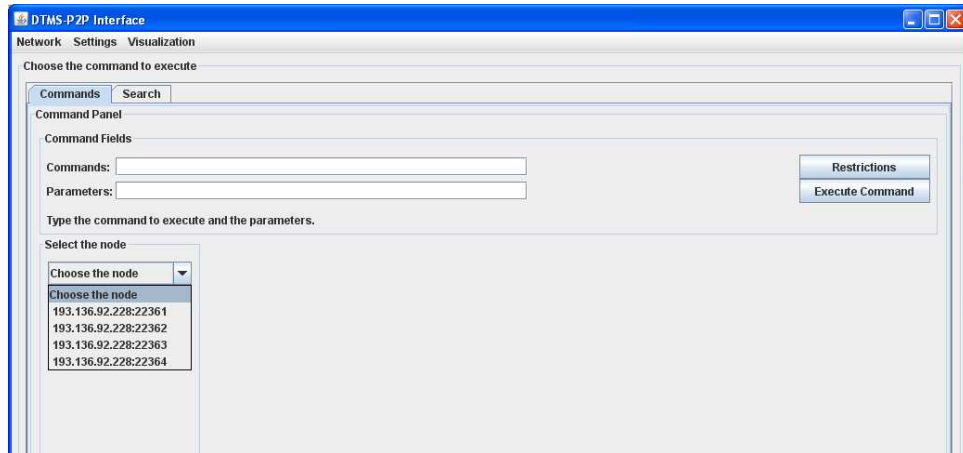


Figure 16 - Selection of the node to execute a command

After the user selects the node, all the elements that are necessary to establish the connection to the element are created and the connection is performed. However, if the user types a command without previously choosing the element, an error message should be presented in order to alert to the fact that the user must first choose a node to execute the command. The message will present the following instruction: “Please select the node where you want to execute the command”. If the client is disconnected from the network, the interface asks the user to perform the connection.

After the user selects a node and inserts the command to execute, and if the button “Restrictions” is pressed, the corresponding event listener calls the method *get_restrictions()*, referred in sub-section 5.2.3.1, that in turn sends a *requestListOfCommandRestrictions* message to the indicated element. When the interface obtains the response of the API, the received restrictions will be presented in the “Commands” tab in the form of a new table that is also inserted inside a tab. The restrictions are sent to the *get_restrictions()* function using the following format: `<mustUse></mustUse><doNotUse></doNotUse> .`

The parameters that the user must insert for a correct execution of the command are listed between the tags `<mustUse>` and `</mustUse>`. The parameters the user must not

insert for the execution of the task are shown inside the tags <doNotUse> and </doNotUse>. The interface then processes the received restrictions and presents them in a table with two columns that indicate the mentioned rules.

For each command the user selects, there will be a tab with the associated restrictions and related messages concerning the command execution. Figure 17 presents the interface with a tab that is related to the *ping* command and its restrictions.

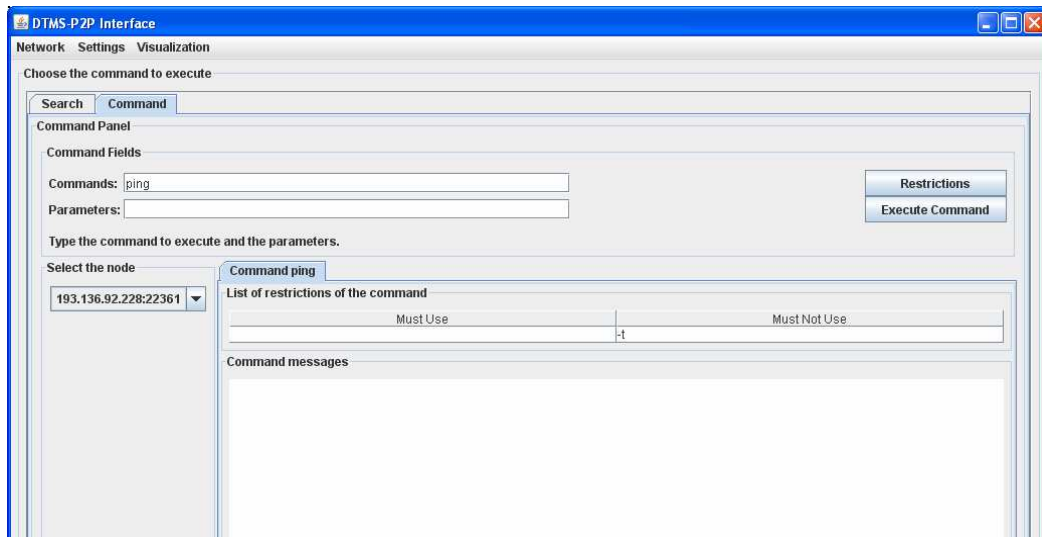


Figure 17 - List of restrictions associated to the *ping* command

If the node does not support the chosen command, a message is printed in the tab indicating this fact to the user. This is shown in Figure 18.



Figure 18 - Message indicating that the node does not support the command

However if the user does not want to see the restrictions, the button “Execute Command” can be pressed to execute the command with the parameters that are set in the text field labelled “Parameters”. Although, if the user pressed the first button, he can execute the command by inserting the parameters in the appropriated text field and press the second button. The interface will then call the *execute_command()* method explained in sub-section 5.2.3.2 that will construct a message with the inserted preferences and send it to the corresponding node. Afterwards, when the interface receives the response from the API, it will indicate that the command has been executed. While the command is not completed, the interface indicates that the node is executing the command.

If the command was successfully executed, the interface will present a message to the user with the name of the file where the achieved results were saved. The name of the file follows the following format:

```
MeasurementGroupOfTheClient_IPAddressOfTheClient_MeasurementGroupOfTheNode_
    e_ IPAddressOfTheNode_DateOfExecution_Command.res
```

The message is presented inside the tab that is related to the command. Also, in the “Commands” panel a new button is shown in order to give the user the possibility of downloading the file.

If the user presses the download button, indicating that he intends to download the file, a message stating that the file will be downloaded is written in the tab. The event listener that is attached to the button calls the method *download_results_file()*, depicted in sub-section 5.2.3.3, that sends a message to the node where the file is located, indicating the intention of downloading it. When the download is finished, a new message is also written in the same tab. This situation is shown in Figure 19. The download of the results file is always related to the command that is represented in the tab and was selected by the user. In this way, if the user chooses to download the results file of a command which has not been executed yet, an error message appears.

The user can insert a new command and a new tab will appear indicating the restrictions and all the above explained messages. In the case the user does not enter any parameter and the command needs at least one to be executed, the interface displays an error message together with the usage of the command. This case is shown in Figure 20. If the user does not respect the restrictions imposed by the command or the inserted parameters are incorrect, a message is displayed indicating that the node refused to execute the command.

6.2.1.4 Search Functionality

The search for a file that is stored at some network node is allowed by the interface. A tab named “Search” shows the user all the necessary fields to execute this task. The search can be performed locally, only in the measurement group where the client is connected, or globally, in the entire network [Veiga2007]. Therefore, this choice must be presented to the user and this is achieved through the use of a combo-box that lists all the measurement groups connected in the network and a last option of global search. Figure 21 illustrates this.

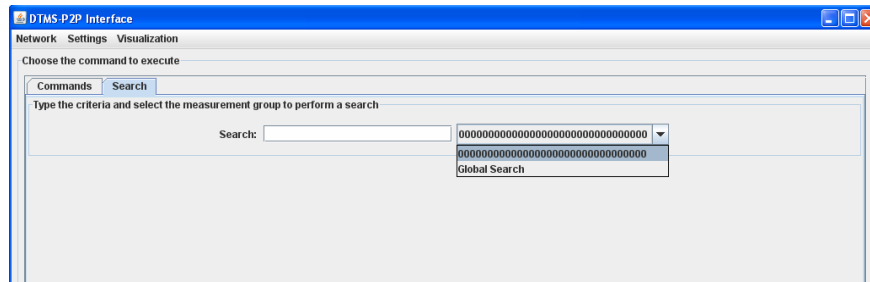


Figure 21 - Selection of the type of search to perform

After choosing the type of search that will be executed, the criterion must be typed in the text field presented in the tab. After inserting it, the user presses the Enter key a first search is executed with a small list of files matching the criterion. If the user wants to see with more results, the same key must be pressed again and a more extensive search is realized. The results are exhibited in this tab in a table format. This table includes six columns that indicate several parameters related to the command, such as: the measurement group of the client that executed the command, the IP address of the client, the measurement group of the node, the IP address of the node, the date of the command execution and the name of the command. The last column indicates if the client has downloaded the file. Each table is enclosed within a tab. As an example, Figure 22 presents a frame showing the results of a search with the *ping* criteria. The function that performs the search is the *show_search_results()*, presented in sub-section 5.2.4.1.

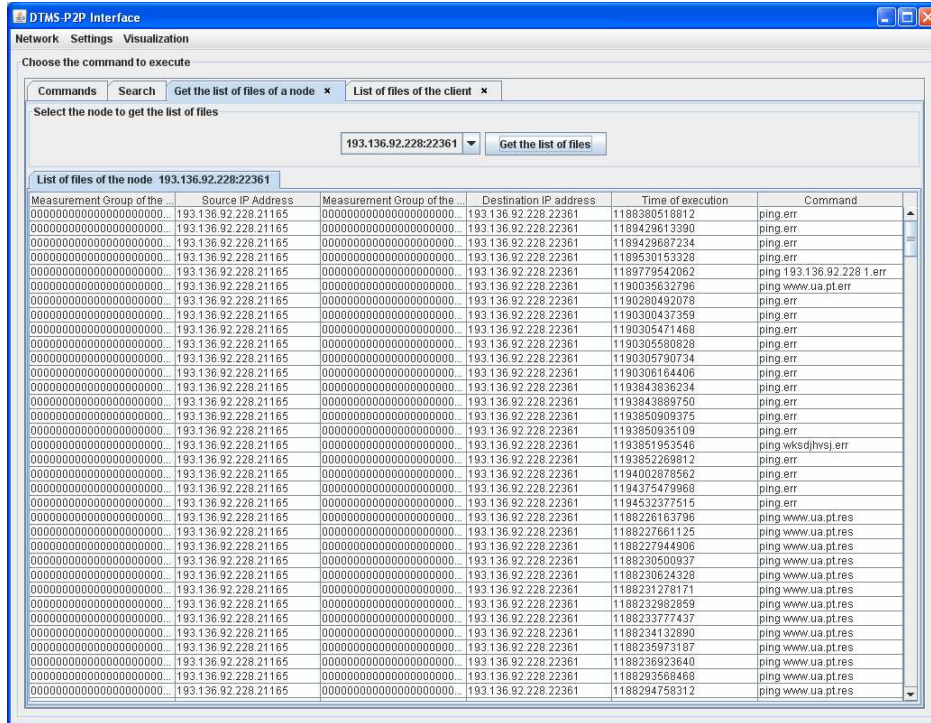


Figure 25 - List of files of a node

6.2.1.7 Visualization of the Network

To visualize the representation of the network, the user can press the menu “Visualization” that will display an item. This item will make the interface present the mentioned illustration. It is located in the following menu: Menu → Visualization → Visualize Network.

If the user presses the referred menu item, a frame with the network representation will be exhibited. This frame shows the different components and the respective connections. For each type of element a different image is displayed in order to allow the distinction between the different types of nodes. Each super-probe’s IP address is written just above its icon. Figure 26 shows a representation of a network with some connected nodes. The next section will explain how to accomplish this representation.

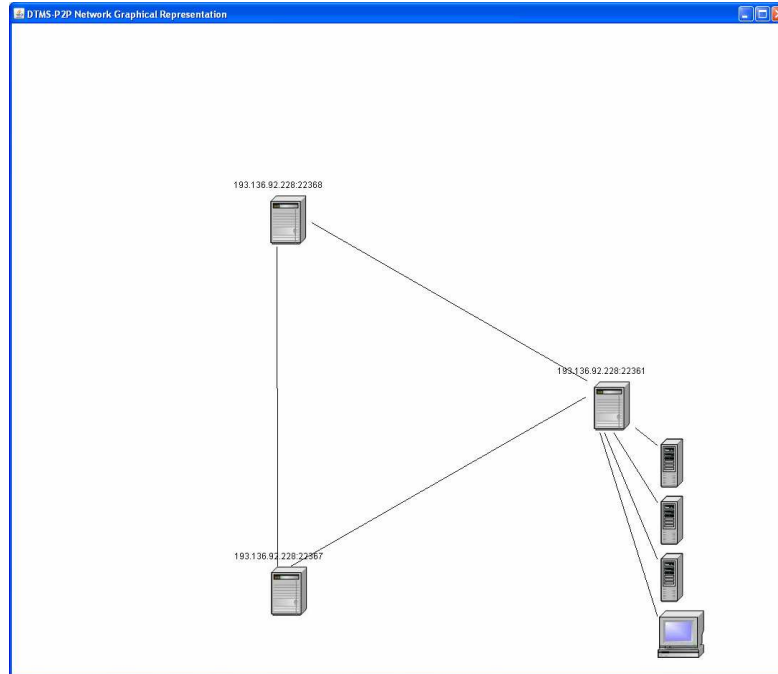


Figure 26 - Frame showing the network representation

6.2.1.8 Graphical representation of the DTMS-P2P network

To graphically represent the DTMS-P2P network, the client must first download the *LightData* file. This file, written in XML format, contains the list of all the active super-probes, probes and clients. It also evidences to which super-probe a node is connected to. The XML format allows for a more precise insertion of contents and retrieval of significant search results.

The *SAXParser_class* is used to process the downloaded file and begins by searching elements in the file which begin with the string “SuperProbe”. When this string is found, a new element is created and all the associated information is saved into the respective class. An element of the class *Attributes* is used to search for information relative to the element, such as the IP address and the group number. These are called the attributes of the root element and provide all the information relative to a super-probe. It also creates the necessary structures to save the mentioned information.

All nodes connected to a super-probe are saved in the XML file as child elements and their attributes as sub-children. ”. This allows for a hierarchical structure of information. The function subsequently searches for the string “DTMS_P2P_Element” and when it is found in the file, the corresponding characteristics are saved to the corresponding class.

A super-probe's information is saved in an element of the *SProbe* class. This class contains the following:

- A string saving the IP address;
- A string saving the group ID;
- Integers saving the position where the super-probe will be represented in the screen;
- A list of elements of the class *node* containing all the elements connected to it.

Information concerning to a node is saved on an element of the *Node* class. This class contains the next fields:

- A string saving the IP address;
- An integer saving the mode of the element: probe or client;
- A string containing the group ID;
- An integer saving the *RTT* parameter

Two lists were created to store the elements of each class. Each list has an iterator and elements can be added as information is found.

When an element beginning with the string “SuperProbe” is found, the instantiated object is added to the list of super-probes. If an element starting with “DTMS_P2P_Element” is found, the function fills the GroupID field of the corresponding node with the value of the similar field of the super-probe to which it is connected. Subsequently, the loading of the IP address of the probe or client is performed and this attribute is saved as a sub-child of the child element *node*. The object will then be added to the list of nodes of the related super-probe. The fields *mode* (that indicates if the node is a probe or a client) and *RTT* (round-trip-time) are also filled. When this process ends, all the necessary information about all the elements of the existing network is located on the created lists.

The diagram in Figure 27 illustrates this implementation.

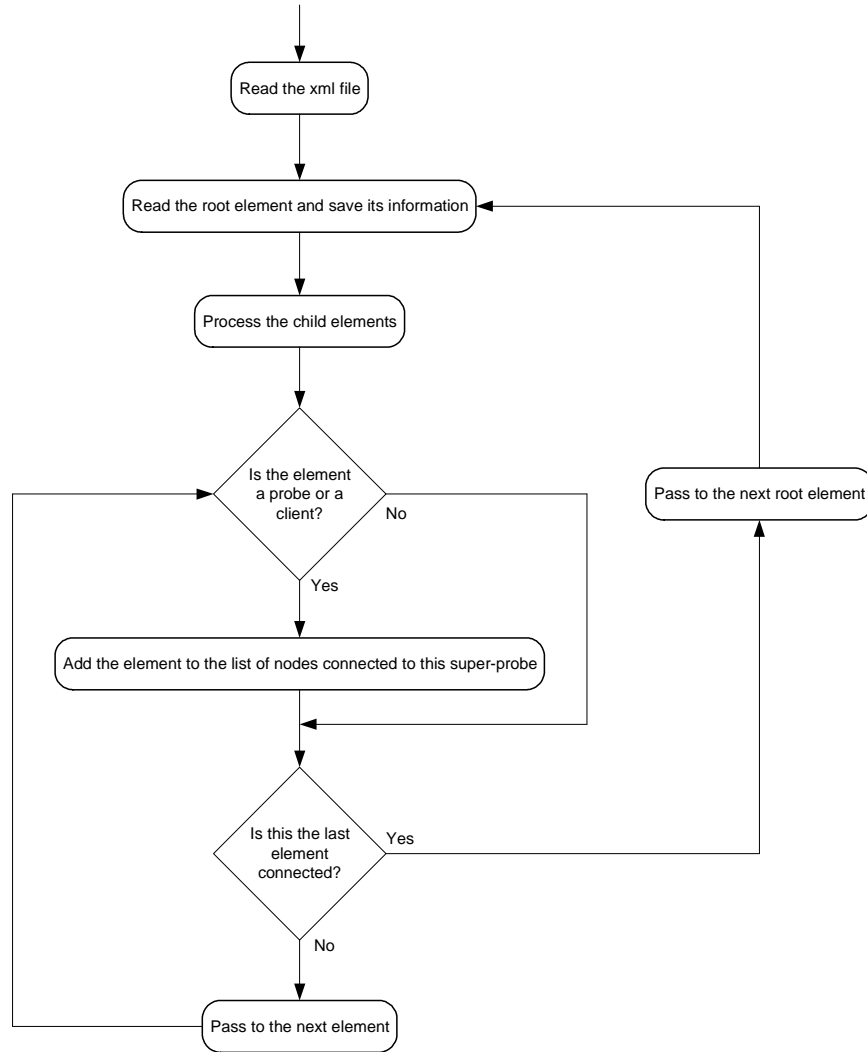


Figure 27 - Flow diagram of the process of the XML file

After having all the needed information, the graphical representation can be made. The methodology used was to display the super-probes in a circle, with the distance between them depending on how many elements must be presented. Around each one of these super-probes, the connected probes and clients, if any, are represented. For drawing this representation, the class *ImageApplet* was created. The class receives, as parameters, the lists that were constructed during the parsing of the XML file and uses their information. In the following lines the implementation of the graphical representation of the network will be explained.

A frame was used to display the representation of the network. A frame is a top-level window with a title and a border. The size of the frame includes all the area assigned

to the border. The dimensions of the border can be obtained using the *getInsets* method. The frame, which is an instance of the class *JFrame*, has decorations elements such as a border, a title and buttons for closing the window.

The function that draws the network iterates through the lists of super-probes and probes/clients (these are saved in the same list) and displays them according to their connections. The function begins by calculating the angle of separation between each super-probe. This depends on their number and an appropriate value is found in this manner. Then the function loads a variable with an image which will stand for the super-probes and draws the first one in the XML file. Afterwards, the function iterates through the child elements of the super-probe and chooses the image to load according to their mode of operation. If the element is a super-probe, the list of connected super-probes is run through to discover if it has already been represented. In this case, a simple connection between these two elements is performed. To find this element, the function must iterate through the list of existing super-probes and find the element with the same IP address. When it is found and if its coordinates are not zero, indicating that the element is already represented, a line connecting these two coordinates is represented. Otherwise, no action is performed and the second super-probe will be represented when the corresponding element in the list is found. If the child element of the super-probe is a client or a probe, the corresponding image is loaded and represented. The element will be placed near the corresponding super-probe. Its position will depend on the zone of the image where the super-probe is represented. A line representing the connection between the super-probe and its child element is also shaded. The flow diagram represented in Figure 28 shows this process.

The entire plane of representation is divided into four regions. The region where the super-probe is represented will determine where the connected nodes will be illustrated. With this, it is intended to provide a simple means to avoid different probes and clients to be superposed. Probes and clients will be shown below the super-probe, if it is placed in an angle higher than 180° , or above if the value of the angle is lesser, i.e., they will always be placed on the outer side of the circle created by the super-probes. Besides, their position relative to the super-probe depends on the number of elements which are connected to it, i.e., the node will be farther from the root element according on how many probes/clients

are already illustrated and on their distance to the super-probe. This last value is the *RTT* (round-trip time).

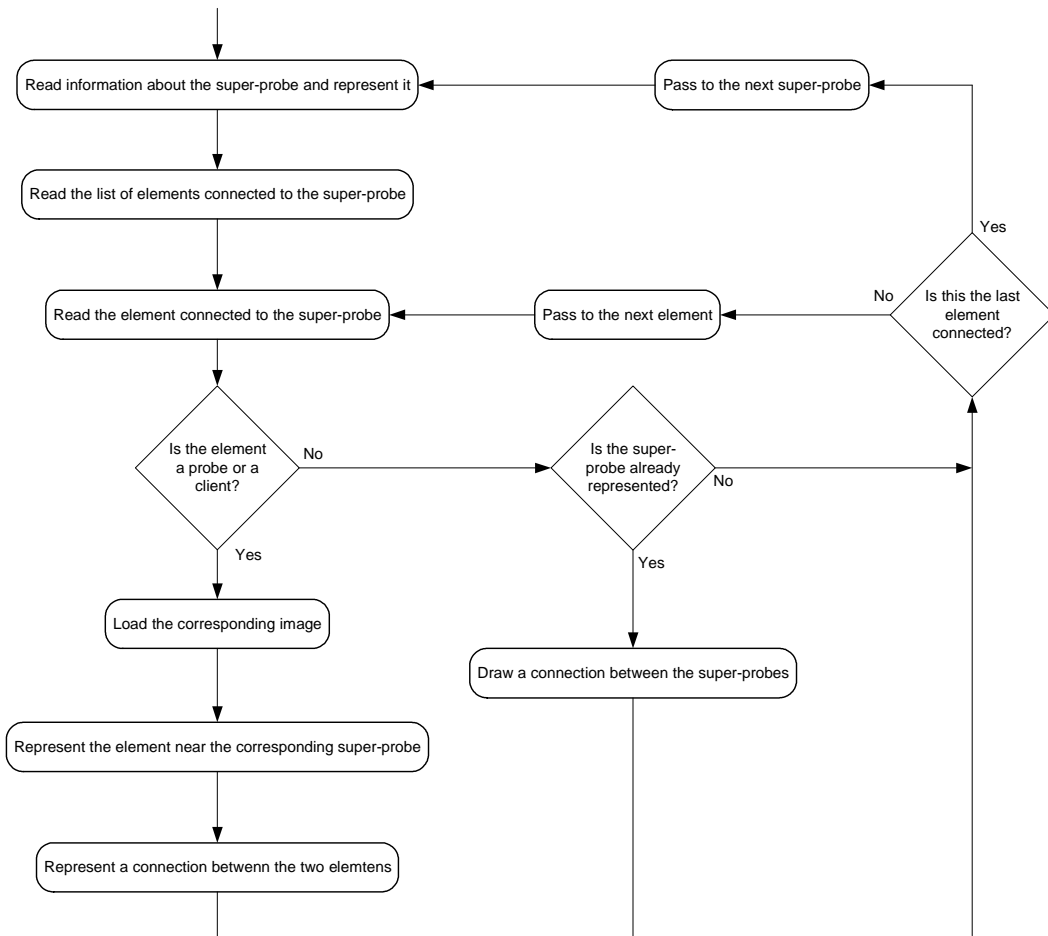


Figure 28 - Flow diagram of the procedure for the representation of the network

After all these procedures and all images represented, the super-probes must be repainted. This is due to the fact that the lines connecting these components will overlap the images that have been placed. Therefore, a refresh of the image must be done. This is achieved by simply iterating through the list of super-probes, consulting their coordinates and repositioning the correspondent image in the place that is indicated by the coordinates.

After concluding the explained steps, the representation is ready. The following images will show some illustrations of networks with different number of super-probes and probes/clients. Figure 29 shows a representation of a network with eight super-probes with several clients and probes. Figure 30 illustrates a network with three super-probes and their related nodes.

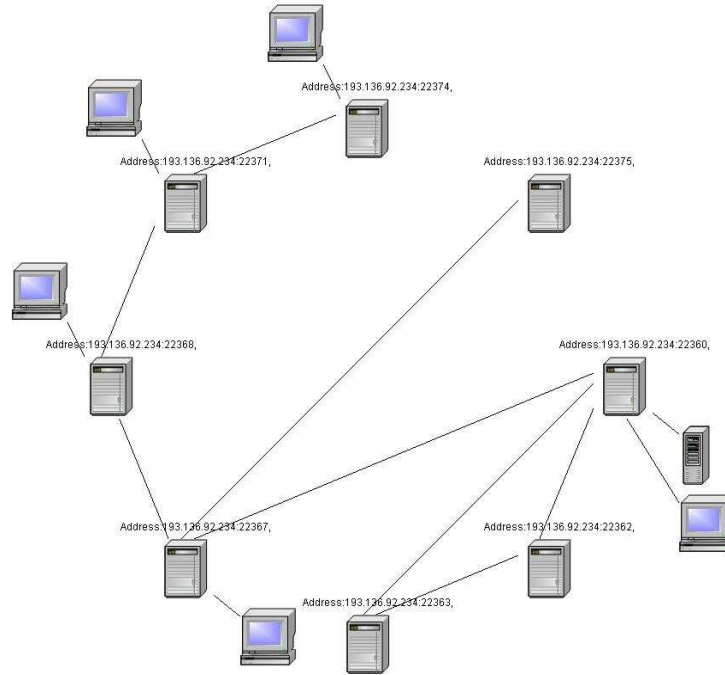


Figure 29 - Representation of a network with various elements connected

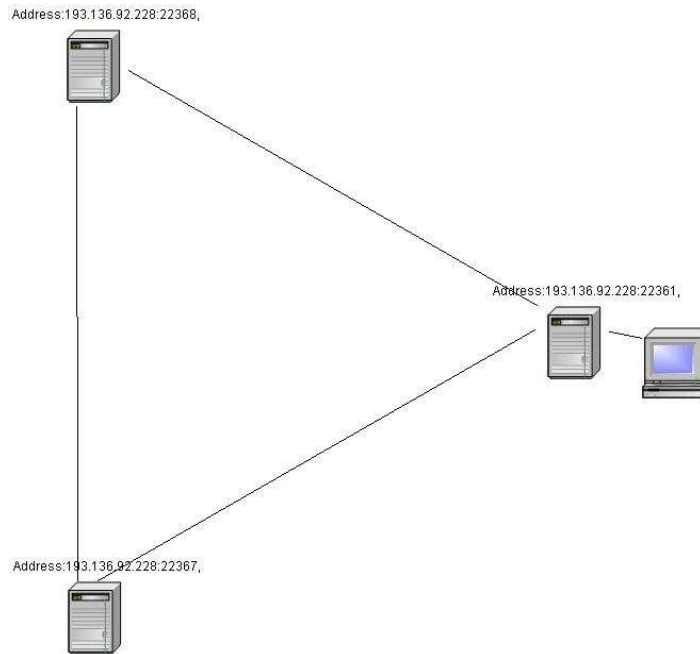


Figure 30 - Representation of a second network with fewer elements connected

6.3 Conclusion

The previous sections have shown all operations an interface can perform and the results that were obtained with the user's interaction. In this chapter was also demonstrated that the API explained in Chapter 5 can implement the communication between modules in upper-layers and the DTMS-P2P platform.

The interface allows a user to complete all the tasks a client of the network is supposed to conclude and provides intuitive responses to the user so that he can have an insight about what the network is executing. It also warns the user about incorrect tasks he requests the network to do and shows how these can be correctly executed. The interface also displays messages indicating the conclusion of the tasks that have been ordered by the user.

The frames displayed by the interface are intuitive, with all necessary fields properly identified, and provide visual clues to guide the user in the execution of a given command.

The results obtained from the execution of the different commands are presented in an efficient manner and many operations can be carried out over them.

The advantage of using this interface is that users that don't have an extensive knowledge on how the network and clients interact can obtain the desired results based on fast and intuitive steps and without the need of having a deep insight into how they operate together.

7 FINAL REMARKS

This dissertation proposed a module for identification of Internet applications based on the analysis of the packets payload. The results presented proved that this methodology constitutes a very accurate one, with very efficient results. The module was able to identify all the protocols it was proposed to and that identification was achieved after analyzing very few packets of the corresponding flows.

The second part of the thesis presented an API implemented in order to establish the communication of the peer-to-peer measurement platform with other modules. In the corresponding chapter was shown how the API establishes the communication and the several messages it sends to the platform and how it retrieves the obtained results. The various methods it uses were explained and it was also shown how these can be used by calling applications.

In the Chapter 6 was introduced a graphical interface for the DTMS-P2P platform. The aim of this implementation was to test the performance of the API and also to provide a simple means of interaction between the user and the client of the DTMS-P2P platform. This interface communicates with the DTMS-P2P network through the API whose methods perform the different tasks a user may demand from the interface. The API returns the results of such commands and, in this manner, the interaction between user and network is achieved.

In the mentioned chapter, the interaction between the API and the module presenting the interface was proved to be a successful one. As the user orders tasks in the interface, it invokes the corresponding method from the API, retrieving information and results from it. This testing proved that the API is able to interact successfully both with the monitoring platform and, in this case, with the graphical interface.

As a conclusion, the implemented interface also successfully replaces the command line interface as it provides various visual clues to show the user how to execute the command he intends to and also shows messages indicating the correct or incorrect performance of the commands.

Appendix I – API Methods

A.1.1 - Method *connect()*

Name	connect()
Purpose	Connect the client to the network
Requisites	The client is disconnected
Syntax	boolean connect()
Input Parameters	
Output Parameters	Boolean variable indicating that the client is connected
Event Sequence	API
	<ol style="list-style-type: none">1. API starts the <code>OutgoingConnectionManager()</code>2. API updates the Boolean variable indicating that the client is connected

A.1.2 - Method disconnect()

Name	disconnect()
Purpose	Disconnect the client from the network
Requisites	The client is connected
Syntax	boolean disconnect()
Input Parameters	
Output Parameters	boolean variable indicating that the client is disconnected
Event Sequence	API
	<ol style="list-style-type: none">1. API shuts down the connection of the client to the super-probe2. API updates the Boolean variable indicating that the client is disconnected

A.1.3 - Method *show_settings()*

Name	show_settings()
Purpose	Present the current settings of the client
Requisites	
Syntax	String[] show_settings()
Input Parameters	
Output Parameters	A string array with the parameters of the client
Event Sequence	API
	<ol style="list-style-type: none">1. API reads the settings of the client2. API processes the settings and places them in a data array

A.1.4 - Method *listener_settings()*

Name	<i>listener_settings()</i>
Purpose	Apply the new settings to the client
Requisites	
Syntax	boolean <i>listener_settings</i> (String[] settings)
Input Parameters	String[] settings – array containing the new settings of the client
Output Parameters	A boolean variable indicating the appliance of the new settings
Event Sequence	API
	<ol style="list-style-type: none">1. API reads the settings of the client2. API applies the new settings to the client3. API writes the new settings to the <i>conf_client</i> file4. API updates the Boolean variable indicating the appliance of the new settings

A.1.5 - Method *listener_default()*

Name	listener_default()
Purpose	Apply the default settings to the client
Requisites	
Syntax	boolean listener_default()
Input Parameters	
Output Parameters	A boolean variable indicating the appliance of the default settings
Event Sequence	API
	<ol style="list-style-type: none">1. API reads the default settings of the client.2. API applies the default settings to the client.3. API writes the default settings to the <i>conf_client</i> file.4. API updates the Boolean variable indicating the appliance of the default settings.

A.1.6 - Method *get_restrictions()*

Name	get_restrictions
Purpose	Ask to the node which will execute the command the restrictions.
Requisites	User has entered a command and pressed the button “Restrictions”.
Syntax	boolean get_restrictions (String[] array, String command)
Input Parameters	String[] array – array[0]: IP address of the node array[1]: the measurement group of the node String command: the command to be performed
Output Parameters	A boolean variable indicating that the API has received the response from the node
Event Sequence	API
	<ol style="list-style-type: none"> 1. API tests if a node was selected to execute the command. 3. API sends a <i>ListOfCommandRestrictionsRequest</i> message to the node. 3. API waits for the response of the node and updates variable indicating it has received the restrictions.

A.1.8 - Method `download_results_file()`

Name	Download_results_file()
Purpose	Download the file with the results
Requisites	The command has been executed
Syntax	boolean download_results_file(String file)
Input Parameters	String file – a string containing the name of the file to download
Output Parameters	A boolean variable indicating if the download has finished
Event Sequence	<p data-bbox="630 646 678 678">API</p> <ol data-bbox="630 699 1377 1146" style="list-style-type: none"><li data-bbox="630 699 1377 783">1. API tests if a file was selected for download and processes its name.<li data-bbox="630 804 1377 888">3. API creates an instance of the <i>MultiSourceDownloader</i> to perform the download of the file<li data-bbox="630 909 1377 993">4. API runs the <i>MultiSourceDownloader</i> to initiate the download of the file<li data-bbox="630 1014 1377 1146">5. API updates the boolean variable indicating the completion of the download or throws an exception if the client has already downloaded the file

A.1.9 - Method *show_search_results()*

Name	show_search_results()
Purpose	Present the list of files which satisfy a search criterion
Requisites	
Syntax	boolean show_search_results (String criterion, String group)
Input Parameters	String criterion – the criterion to perform the search String group – the measurement group where to perform the search for files
Output Parameters	A boolean variable indicating it the completed the processing of the search results.
Event Sequence	API
	<ol style="list-style-type: none"> 1. API sends a <i>resultsSearch</i> message to the nodes of the selected measurement group 2. The API processes the <i>QueryHit</i> messages sent by the nodes 3. The API processes the names of the files in the <i>QueryHit</i> messages and places them in a data array according to the parameters. 4. API updates variable indicating it has completed the processing of the search results.

A.1.10 - Method *value_changed*

Name	value_changed()
Purpose	Download the file the user selected in the calling application
Requisites	
Syntax	boolean value_changed (String file)
Input Parameters	String file – the name of the file to download
Output Parameters	A boolean variable indicating the completion of the download. Throws an exception if the client has already downloaded the file.
Event Sequence	API
	<ol style="list-style-type: none">1. API processes the name of the file to download.2. API creates an instance of the <i>MultiSourceDownloader</i> which will perform the download of the file.3. API starts the <i>MultiSourceDownloader</i> to initiate the download of the file.4. API updates the boolean variable to indicate the completion of the download of the file or throws an exception if the client has already downloaded the file.

A.1.11 - Method *show_files()*

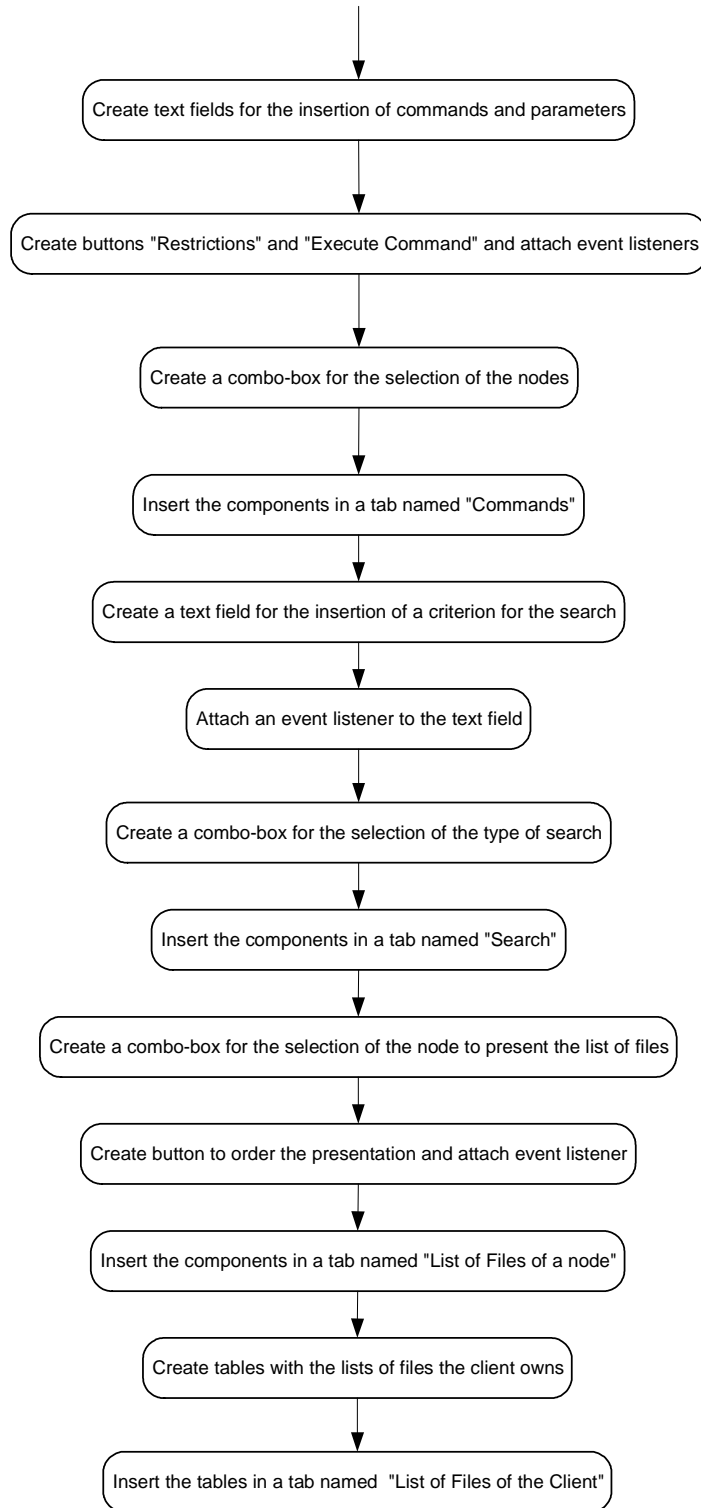
Name	show_files()
Purpose	Present the list of files the client owns
Requisites	
Syntax	boolean show_files()
Input Parameters	
Output Parameters	A boolean variable indicating the completion of the process of the list of files
Event Sequence	API
	<ol style="list-style-type: none">1. API requests the list of files the client owns2. API processes the list of files and places each file on the corresponding data array.3. API updates variable indicating it has completed the process.

A.1.12 - Method `get_file_list()`

Name	<code>get_file_list()</code>
Purpose	Present the list of files a node owns.
Requisites	
Syntax	<code>boolean get_file_list(String IP_address)</code>
Input Parameters	String: the IP address of the node.
Output Parameters	A boolean variable indicating it has processed the list of files of a node.
Event Sequence	API
	<ol style="list-style-type: none">1. API processes the IP address of the node.2. API downloads the file with list of files the node owns.3. API parses the file.4. API places the name of the files in the data array.5. API updates boolean variable.

Appendix 2 – Interface Methods

A.2.1 - *Constructor*



A.2.2 - Method *notify_disconnect()*

Name	notify_disconnect()				
Purpose	Notify the API to disconnect the client from the network.				
Requisites	The client is connected.				
Event Sequence	<table border="1"><thead><tr><th>User</th><th>Interface</th></tr></thead><tbody><tr><td>1. User presses the menu item "Disconnect".</td><td>2. Interface calls the method <i>disconnect()</i> from the API. 3. Interface presents a message to the user indicating that the client is disconnected.</td></tr></tbody></table>	User	Interface	1. User presses the menu item "Disconnect".	2. Interface calls the method <i>disconnect()</i> from the API. 3. Interface presents a message to the user indicating that the client is disconnected.
User	Interface				
1. User presses the menu item "Disconnect".	2. Interface calls the method <i>disconnect()</i> from the API. 3. Interface presents a message to the user indicating that the client is disconnected.				

A.2.3 - Method *notify_connect()*

Name	notify_connect()		
Purpose	Notify the API to connect the client to the network.		
Requisites	The client is disconnected.		
Event Sequence	<table border="0"><tr><td data-bbox="581 480 1015 533">User</td><td data-bbox="1015 480 1383 533">Interface</td></tr></table>	User	Interface
	User	Interface	
<table border="0"><tr><td data-bbox="581 533 1015 949">1. User presses the menu item “Connect”.</td><td data-bbox="1015 533 1383 949">2. Interface calls the method <i>connect()</i> from the API. 3. Interface presents a message to the user indicating that the client is connected.</td></tr></table>	1. User presses the menu item “Connect”.	2. Interface calls the method <i>connect()</i> from the API. 3. Interface presents a message to the user indicating that the client is connected.	
1. User presses the menu item “Connect”.	2. Interface calls the method <i>connect()</i> from the API. 3. Interface presents a message to the user indicating that the client is connected.		

A.2.4 - Method *get_settings()*

Name	get_settings()				
Purpose	Notify the API to get the settings of the client.				
Requisites	The client is connected.				
Event Sequence	<table><thead><tr><th>User</th><th>Interface</th></tr></thead><tbody><tr><td>1. User presses the menu item "Show Settings".</td><td>2. Interface calls the method <i>show_settings()</i> from the API. 3. Interface presents the settings to the user.</td></tr></tbody></table>	User	Interface	1. User presses the menu item "Show Settings".	2. Interface calls the method <i>show_settings()</i> from the API. 3. Interface presents the settings to the user.
User	Interface				
1. User presses the menu item "Show Settings".	2. Interface calls the method <i>show_settings()</i> from the API. 3. Interface presents the settings to the user.				

A.2.5 - Method *apply_settings()*

Name	apply_settings()				
Purpose	Notify the API to apply the new settings to the client.				
Requisites	The client is connected.				
Event Sequence	<table border="1"><thead><tr><th>User</th><th>Interface</th></tr></thead><tbody><tr><td>1. User presses the button “Apply Settings”.</td><td>2. Interface calls the method <i>listener_settings()</i> from the API. 3. Interface presents a message to the user indicating that the settings have been applied.</td></tr></tbody></table>	User	Interface	1. User presses the button “Apply Settings”.	2. Interface calls the method <i>listener_settings()</i> from the API. 3. Interface presents a message to the user indicating that the settings have been applied.
User	Interface				
1. User presses the button “Apply Settings”.	2. Interface calls the method <i>listener_settings()</i> from the API. 3. Interface presents a message to the user indicating that the settings have been applied.				

A.2.6 - Method *apply_default()*

Name	apply_default()				
Purpose	Notify the API to apply the default settings to the client.				
Requisites	The client is connected				
Event Sequence	<table border="1"><thead><tr><th>User</th><th>Interface</th></tr></thead><tbody><tr><td>1. User presses the button “Apply Default”.</td><td>2. Interface calls the method <i>listener_default()</i> from the API. 3. Interface presents a message to the user indicating that the default settings have been applied.</td></tr></tbody></table>	User	Interface	1. User presses the button “Apply Default”.	2. Interface calls the method <i>listener_default()</i> from the API. 3. Interface presents a message to the user indicating that the default settings have been applied.
User	Interface				
1. User presses the button “Apply Default”.	2. Interface calls the method <i>listener_default()</i> from the API. 3. Interface presents a message to the user indicating that the default settings have been applied.				

A.2.7 - Method *listener_restrictions()*

Name	<i>listener_restrictions()</i>				
Purpose	Notify the API to show the restrictions associated to a command.				
Requisites	User has selected a node to execute the command and inserted a command				
Event Sequence	<table border="0"><tr><td data-bbox="589 531 1019 575">User</td><td data-bbox="1019 531 1375 575">Interface</td></tr><tr><td colspan="2" data-bbox="589 575 1375 1050"><ol style="list-style-type: none">1. User selects a node2. User enters a command to view its restrictions3. User presses the button "Restrictions"4. Interface calls the method <i>get_restrictions()</i>5. Interface presents the restrictions</td></tr></table>	User	Interface	<ol style="list-style-type: none">1. User selects a node2. User enters a command to view its restrictions3. User presses the button "Restrictions"4. Interface calls the method <i>get_restrictions()</i>5. Interface presents the restrictions	
User	Interface				
<ol style="list-style-type: none">1. User selects a node2. User enters a command to view its restrictions3. User presses the button "Restrictions"4. Interface calls the method <i>get_restrictions()</i>5. Interface presents the restrictions					

A.2.9 - Method *listener_button()*

Name	listener_button()				
Purpose	Notify the API to download the results file.				
Requisites	The command has been executed.				
Event Sequence	<table border="1"><thead><tr><th data-bbox="587 489 1008 535">User</th><th data-bbox="1008 489 1383 535">Interface</th></tr></thead><tbody><tr><td data-bbox="587 535 1008 1054">1. User presses the button to download the file.</td><td data-bbox="1008 535 1383 1054">2. Interface tests if the user has executed the command. 3. Interface calls the method <i>download_results_file()</i>. 4. Interface presents a message to the user indicating the end of the download.</td></tr></tbody></table>	User	Interface	1. User presses the button to download the file.	2. Interface tests if the user has executed the command. 3. Interface calls the method <i>download_results_file()</i> . 4. Interface presents a message to the user indicating the end of the download.
User	Interface				
1. User presses the button to download the file.	2. Interface tests if the user has executed the command. 3. Interface calls the method <i>download_results_file()</i> . 4. Interface presents a message to the user indicating the end of the download.				

A.2.10 - Method *actionPerformed()*

Name	actionPerformed()
Purpose	Notify the API to perform the search of a file.
Requisites	User entered a search criteria and the type of search to be performed.
Event Sequence	User Interface
	1. User enters a search criterion and selects the type of search to perform. 2. Interface calls the method <i>show_search_results()</i> . 3. Interface presents the results to the user.

A.2.11 - Method *notify_download_file()*

Name	notify_download()
Purpose	Notify the API to perform the download of a file resulting from a search operation.
Requisites	User performed the search of files.
Event Sequence	User Interface
	1. User selects a file for download from the table containing the search results. 2. Interface calls the method <i>value_changed()</i>. 3. Interface presents a message to the user.

A.2.12 - Method *listener_button_list()*

Name	<i>listener_button_list()</i>										
Purpose	Notify the API to perform the download of the list of files of a node.										
Requisites	The user has selected a node to retrieve the list of files.										
Event Sequence	<table border="1"><thead><tr><th data-bbox="587 533 1008 575">User</th><th data-bbox="1008 533 1377 575">Interface</th></tr></thead><tbody><tr><td data-bbox="587 575 1008 684">1. User selects a node to get its list of files.</td><td data-bbox="1008 575 1377 684"></td></tr><tr><td data-bbox="587 684 1008 793">2. User presses the button “Get the list of files”.</td><td data-bbox="1008 684 1377 793"></td></tr><tr><td data-bbox="587 793 1008 882"></td><td data-bbox="1008 793 1377 882">3. Interface calls the method <i>get_file_list()</i>.</td></tr><tr><td data-bbox="587 882 1008 993"></td><td data-bbox="1008 882 1377 993">4. Interface presents the list of files.</td></tr></tbody></table>	User	Interface	1. User selects a node to get its list of files.		2. User presses the button “Get the list of files”.			3. Interface calls the method <i>get_file_list()</i> .		4. Interface presents the list of files.
User	Interface										
1. User selects a node to get its list of files.											
2. User presses the button “Get the list of files”.											
	3. Interface calls the method <i>get_file_list()</i> .										
	4. Interface presents the list of files.										

Bibliography

Books:

[Eckel2002] B. Eckel, *Thinking in Java*, 3rd ed., Prentice-Hall, 2002.

[Mandel1997] T. Mandel. *The Elements of User Interface Design*, John Wiley & Sons, 1997.

[Walrath2004] K. Walrath, M. Campione, A. Huml, and S. Zakhour. *JFC Swing Tutorial: A Guide to Constructing GUIs*, 2nd ed, Addison Wesley, 2004.

[Zukowski2005] J. Zukowski. *The Definitive Guide to Java Swing*, 3rd ed., Appress, 2005.

Articles and Documents:

[Bernaille] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, K. Slamatian, “Traffic Classification On The Fly”, In *ACM SIGCOMM Computer Communication Review*, Vol. 36, N° 2, pp. 23-26, 2006.

[BitTorrent2007] *Bit Torrent*, <http://www.bittorrent.com>, 2007.

[Cisco2007] Cisco NBAR. <http://www.cisco.codwiarpublic/732~ecldqos/nbar/>.

[Dewes2003] C. Dewes, A. Wichmann, and A. Feldmann. “An analysis of Internet chat systems”. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, Oct 2003.

[Erman2006] J. Erman, M. Arlitt, and A. Mahanti. “Traffic Classification Using Clustering Algorithms”, In *Proceedings of ACM SIGCOMM Mininet Workshop*, Pisa, Italy, September 2006.

- [Gnutella2007] Gnutella2. <http://www.gnutella2.com>
- [Haffner2005] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, “ACAS: Automated Construction of Application Signatures”, In *SIGCOMM’ 05 MineNet Workshop*, Philadelphia, USA, August 22-26, 2005.
- [IEC2007] IEC, 2007. *International Engineering Consortium*, <http://www.iec.org/online/tutorials/h323>
- [IT2007] Institute of Telecommunications – Networks and Multimedia. http://www.it.pt/area_p_3.asp, 2007.
- [Javvin2007] Javvin. *H.323: ITU-T VOIP Protocols Overview*, <http://www.javvin.com/protocolH323.html>, 2007.
- [Karagiannis2004a] T.Karagiannis, A.Broido, N.Brownlee, kc claffy, and M.Faloutsos. “Transport Layer Identification of P2P Traffic”, In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC 2004)*, pp. 121-134, 2004.
- [Karagiannis2004b] T. Karagiannis, A.Broido, N.Brownlee, kc claffy, and M.Faloutsos. “Is P2P dying or just hiding?”, *Proceedings of the IEEE Globecom 2004 - Global Internet and Next Generation Networks*, 2004.
- [Karagiannis2005] T. Karagiannis, D. Papagiannaki, and M. Faloutsos. “BLINC: Multilevel Traffic Classification in the Dark”, Technical report, 2005. <http://www.cs.ucr.edu/~tkarag/papers/BLINC TR.pdf>, 2005.
- [Kulbak2005] Y. Kulbak and D. Bickson. *The eMule Protocol Specification*, University of Jerusalem, Israel, 2005.
- [McGregor2004] A. McGregor, M. Hall, P. Lorier, and Brunskill J. “Flow Clustering Using machine Learning Techniques”, In *Passive & Active Measurement Workshop*, 2004, France, April, 2004.

- [Madhukar2006] A. Madhukar and C. Williamson. “A Longitudinal Study of P2P Traffic Classification”, In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pp. 179 – 188, 2006.
- [Moore2005] Moore A. W. and Papagiannaki K. “Toward the accurate identification of network applications”, In *Passive & Active Measurement Workshop*, Boston, USA, March 2005
- [MSN2007] MSN. *MSN Messenger Protocol*, <http://www.hypothetic.org/docs/msn/general/overview.php>, 2007
- [OSCAR2007] OSCAR. *OSCAR (ICQ v7/v8/v9) Protocol Documentation*, <http://iserverd.khstu.ru/oscar>, 2007.
- [Richardson2006] T. Richardson. *The RFB Protocol*, <http://www.realvnc.com/docs/rfbproto.pdf>, 2006
- [Rocha2007] E. Rocha, H. Veiga, R. Valadas, P. Salvador, and A. Nogueira. “Module for identifying Internet Applications and its integration in a peer-to-peer measurement tool”, In *MCCSIS 2007*, Lisbon, Portugal, 2007
- [Salvador2005] P. Salvador and R. Valadas. “A Network Monitoring System with a Peer-to-Peer Architecture”, In *Proceedings of the Third International Workshop on Internet Performance, Simulation, Monitoring and Measurements*, Warsaw, Poland, pp. 115-122, 2005.
- [Sen2004] S. Sen, O. Spatscheck, and D. Wang. “Accurate, Scalable In-Network Identification of P2P Traffic using Application Signatures”, In *Proceedings of the 13th International World Wide Web Conference*, NY, USA, pp. 512-521, 2004.

[SIP2007] SIP C. “SIP Center”,
<http://www.sipcenter.com/sip.nsf/html/What+Is+SIP+Introduction>, 2007.

[Veiga2007] H. Veiga. “Distributed Traffic Measurement System with a Peer-to-Peer Architecture”, 2007.

[Zander2005] S. Zander, T. Nguyen, and G. Armitage. “Automated Traffic Classification and Application Identification using Machine Learning”. In *LCN’05*, Sydney, Australia, Nov 15-17, 2005.

[Zuev2005] D. Zuev and A. Moore. “Traffic Classification using a statistical approach”, In *Passive & Active Measurement Workshop*, Boston, USA, March 2005.