CENG 491

DETAILED DESIGN REPORT

THE CODEFATHER INC. PRESENTS

Auto-Identification/Classification of Common IP Protocols

Project Members

Abdülkadir VARDAR

Erol SERBEST

Hüseyin SÖZER

Irfan GÖREN

Instructor: Meltem YÖNDEM TURHAN Project Asistant: Cağatay ÇALLI

Company: SIEMENS

Table of Contents

The CodeFather inc.

The CodeFather inc.

1. Introduction

1.1 Purpose of this Document

This document is written to show our work for final designs of our project.

1.2 Overview

We give more specific and design related information about the project and state objects and goals of our project. We gave information about our project team structure and process model. Preliminary schedule are reported.

2. General Description

2.1 Project Background

Network analysis is the process of capturing network traffic and examines it. Closely to determine what is happening on the network. A network analyzer decodes the data packets of common protocols and displays the network traffic in readable format for users. Network analysis is also known by several other names: traffic analysis, protocol analysis, sniffing, and packet analysis. A network analyzer is a combination of hardware and software. Although there are differences in each product, a network analyzer is composed of five basic parts, however first three parts; hardware, capture driver and buffer are irrelevant with our project. The last two parts are described below:

Real-time analysis: This feature analyzes the data as it is captured. For capturing network packages, we will use WinPcap. WinPcap will enable our software to capture network packages bypassing the protocol stack. Since WinPcap is the

industry-standard tool for link-layer network access in Windows environments, it is compatible with our requirements.
Additionally, WinPcap is completely compatible with libpcap. This means that we can use it to port our (if any) UNIX or Linux tools supplied to Windows. This also means that our Windows application will be easily portable to UNIX if we want.

Decode: This part displays the contents of the network traffic with descriptions so that it is human-readable. Each protocol has its own specific decode, so a good network analyzer must support a lot of protocols.

2.2 Project Definition

For most of the widely used IP based protocols such as FTP, POP3, SMTP, NNTP, HTTP, IRC, ICQ, and YMSG; well-known TCP (or UDP) ports are used. Every protocol uses some specific port number (80 for HTTP, 110 for POP3 for instance). During data traveling along the network, protocol information is obtained by using the port number associated with a specific protocol or application. However, there are some programs like Skype or Windows Messenger, which do not use peculiar port number mainly because of security. Furthermore, there are many applications available for users to choose which port number to use. This means that while identifying the protocol, port number cannot be trusted. To identify which protocol is used, our program uses new methods.

In addition to deciding the protocol without using the port number, our program records the summary of important data transmitted with that protocol. However, our program does not need to listen to the network directly (but we can); instead, we are supplied pcap files that include a lot of network packages. Each package has its own protocol. After given a pcap file our program works like in that way: Firstly, automatically identify which application layer protocol is actually involved for a given flow of IP traffic passing through any TCP or UDP port. And then, after we are supplied the merging of the data, our program extracts some useful data transferred with the protocol. After extracting the data from packages, necessary data is saved in the database.

2.3 Project Goals and Scope

Captured network packets, that shall be identified is saved in the PCAP file format. This pcap file is the input of program. Identification process should also determine when the identified protocol is no longer available in the flow through the identified port. Since program handles real-time captures and the size of that captured packages will be too large, it must have high performance and low latency, to meet the user demands. The list of protocols that should be identified is below:

SIP

POP3

Appendix B includes the simple description of these protocols. Required output for detected protocols which are at the fifth Application layer according to the five-layer TCP/IP model:

POP3: Download mail messages

SMTP: Simple text files

SIP: Voice files in Microsoft ASF format

3. General Constraints

Members related:

The CodeFather Inc. is established by 4 senior computer engineering students of Middle East Technical University's Computer Engineering Department. This project is being produced for senior project course of the said department. The situation presents some constraints:

- The project process began in October 2007 and will end in June 2008. This makes about 9 months of development of which 5 months left.
- The development team is bound by the senior project course schedule, which imposes deadlines on reports, phases and etc.

- The members of the development team have significant course and academic work not related to this project, severely limiting their effective project development time.
This project is related to the network knowledge which is not covered by the courses yet.

Implementation related:

Auto sensing and extracting data are two concepts that require extensive programmer expertise. Even worse, this is a development step that is repeated many times and in many different ways depending on issues such as time constraints and programmer experience. Users and maintainers of such systems do not possess the necessary programming background to be effective maintainers as such. This results in a boundary that often keeps users from exploring new ideas and directions.

Since this project is thought as a module of a real time system, speed is a very significant aspect. Also considering the density of the network traffic, efficiency is very important.

4. Process

4.1 Team structure

Our project topic is very specific and non-modular problem. Since this topic is difficult to implement, we have to have long team life. Also project development process requires new ideas, so we decided our team structure to be Democratic Decentralized (DD). We have no permanent leader. We communicate within each other horizontally.

4.2 Process model

Since we progress in the project step by step analysis, initial design, detailed design, prototype preparation, implementation, testing and maintenance, we

considered that linear sequential model (Waterfall Model) is the most appropriate process model for our project. In linear sequential model, the phases are followed in a manner that when one phase is finished the next step starts. When all the requirements are specified and understood, the design step starts and according to the requirements the system is designed and after the design process the implementation of all components of the system are accomplished and the life cycle of the process moves to the testing and the faults of the earlier phases are removed here.

5. Requirements

5.1 System requirements

We have chosen Windows as the operating system for our project because of its wide range use around the world and healthy background.

5.2 Hardware requirements

Since our project should have enough capability of communicating with real time systems in the most efficient way, hardware system should be very fast, provide available memory and processor. Minimum system requirement for the development process of our project is:

1800 MHz processor

512 MB ram

5 GB of available space

5.3. Non Functional Requirements

5.3.1 Security

Our project is a part of the project that is developed by siemens. So security issues are completely belongs to siemens. But we will be considering security aspects while designing and coding in order not to allow system crashes.

5.3.2 Performance

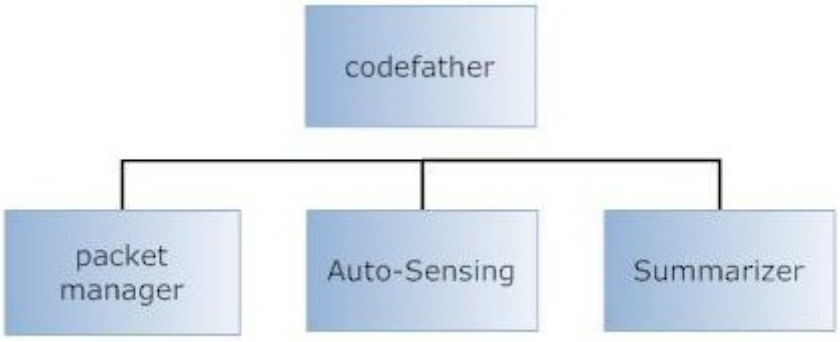
Since our project have to be applicable to real time systems, performance is very important for the health of the system. We give a great importance to the performance and efficiency and chose our development environments and tools according to these aspects.

5.3.3 Reliability

Our program will work properly as long as the protocols of the packages in pcap files are supported.

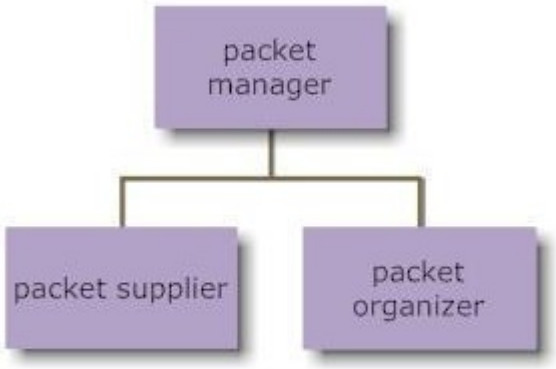
PROJECT MODULES

The codefather project has three main modules. These modules are Packet Manager, Auto-sensing and Summarizer.



1) Packet Manager

Since there are two main functions that Packet Manager does, we divide Packet Manager into two modules. First one is Packet Supplier Module that captures packets from a network device or gets packets from a pcap file directly. Options declared by the user, another is Packet Ordering module. Let us describe these modules.



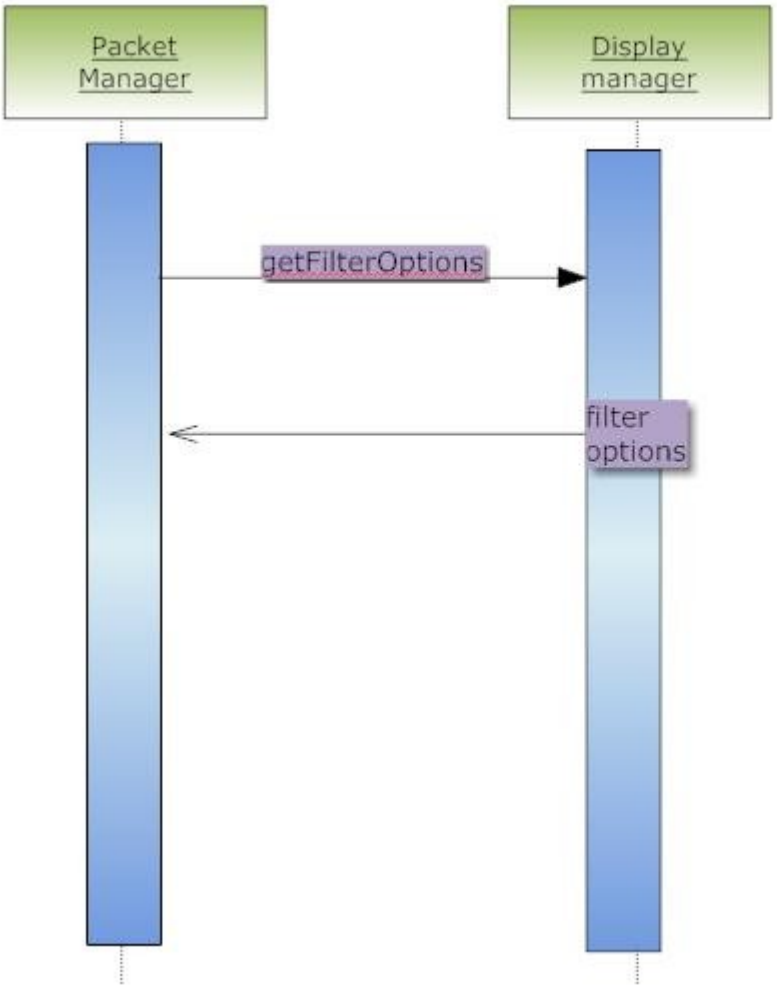
1.1) Packet Supplier

Network packages supplied to the codefather project from this module. A network package can be captured by listening a port or pre-captured packets from pcap files. After obtaining these packets, these packets must be filtered. The word “filtering” we used here basically means ip filtering which is a mechanism that decides which types of IP datagram’s will be processed normally and which will be discarded.
(http://www.faqs.org/docs/linux_network/index.html)

Since IP filtering is a network layer facility, we do not understand anything about the
application using the network connection. We only know about the connections themselves like datagram source, destination and protocol type. So we have decided
Our filtering criteria as:
1) Datagram source address (where it came from)
2) Datagram destination address (where it is going to)

Here we do not filter protocol type according to the layer 4 of OSI Reference model, because we only interested in TCP/IP protocol suite. Since system input that come from either real time sniffing or already captured in a pcap file that will be already filtered
according to the protocol, we will have only datagrams with TCP/IP protocol suite. These filtering criteria are determined by the user. As figured out below the input

Manager will get filtering options from Display Manager and using this filter options it
will decide whether a packet is allowed or not. By filtering user can allow all packets,
only packets from a certain ip number, packets going to a certain ip number or both.
Once the system starts to get input, filtration is done dynamically. Filtered undesired
packets are simply ignored and the rest are sent to be handled by
Packet Organizer module.



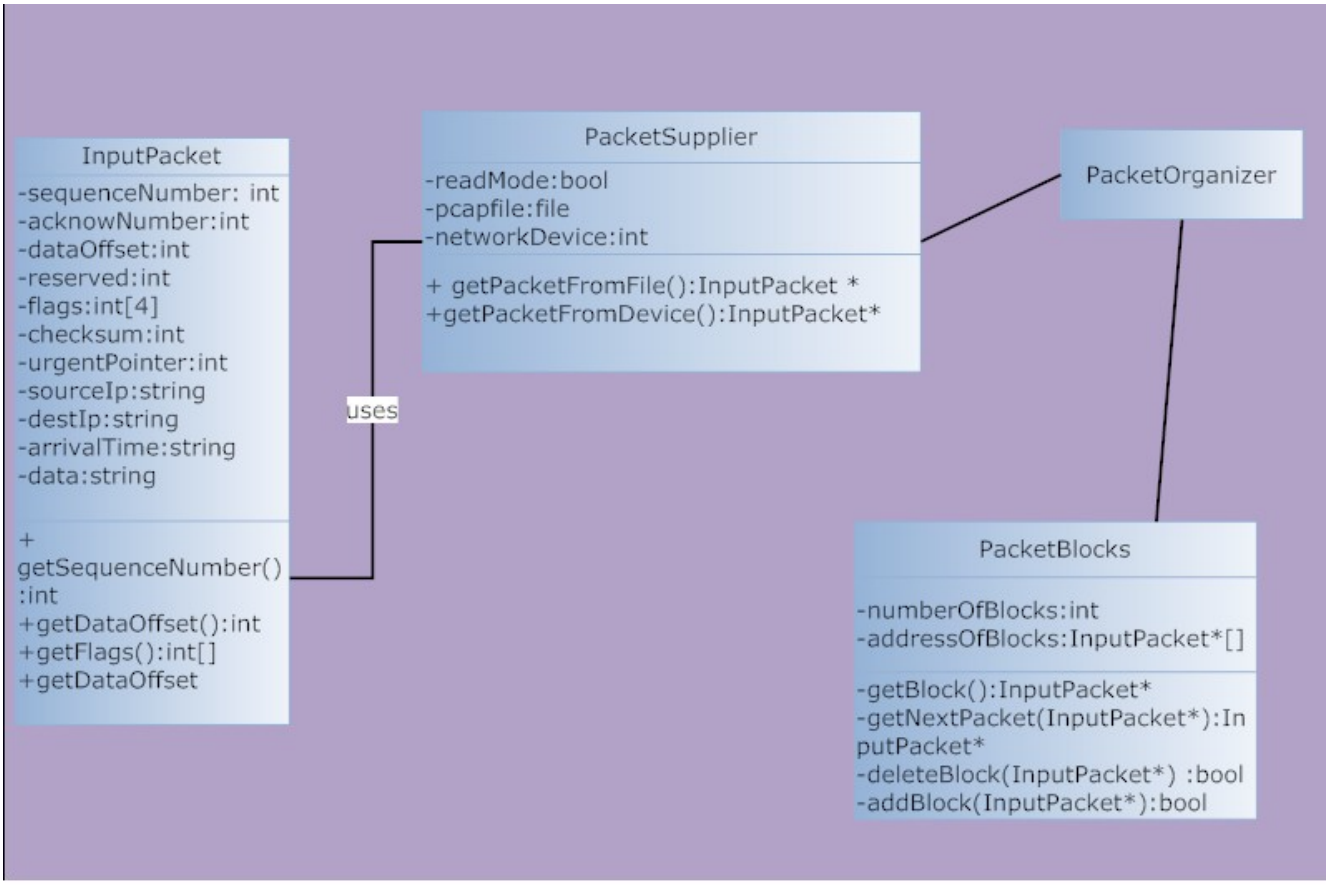
1.2) Packet Organizer

According to the IP protocol specifications, the IP protocol which is at layer 3 of OSI Reference model provides the delivery of packets but it does not necessarily maintain
the order of packets. Moreover, we do not know other facts about the reordering, such as how many positions was the packet displaced and how often it happened. At
layer 4 TCP protocol can tolerate packet displacement by 1 or 2 positions and be able to sort these packets. But if displacement is much more then this TCP will continue
receiving unordered packets. TCP will also suffer in a packet loss situation. (COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY (2005) -
Shall we worry about Packet Reordering? by Michal Przybylski, Bartosz Belter, Artur Binczewski)

After packages supplied they must be reordered, but if there is any lost in packages, then reordering is become a challenge. This module is supplied to us from ***Siemens*** Company.

Packet Manager Class Diagram

The Codefather inc.



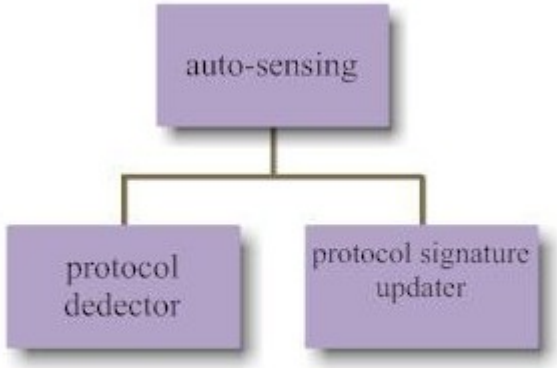
InputPacket class is a generic packet class. It contains a lot of TCP values. These values are used for filtering and ordering.

PacketSupplier class is responsible for reading inputs from either a pcap file or listening to a port. After a packet read, then it is sent to ***PacketOrganizer*** class. Also filtering is done by this class.

PacketOrganizer class is responsible for reordering the packets. If there is not any loss in packages then this class orders the packets properly. This class is provided from SIEMENS. After ordering packets, related packets constitute packetBlock.

PacketBlocks is like a buffer. After filtering and reordering related packets, for example, packets that construct a mail, we must store them, because packet reading is continuous. That is why we need this class.

2)Auto-Sensing



Protocol Signatures

Since we do not have protocol and port knowledge, we do not know where the data we looked for is. We have to deal with a huge raw data. In order to deal with this unknown data load a methodical and systematic identification process must be followed. One of the most common processes is signature analysis and this is the main work load of our Auto Sensing mechanism. Very similar to the idea of using fingerprints to identify the individuals in criminals, protocol signatures are used to identify applications and protocols. In general meaning, signatures are pattern keys which are chosen for uniquely identifying an associated protocol but we must keep in mind that these signatures can change in time

Methods of Signature Analysis

There are several possible methods of analysis used to identify protocols. Some of these are analysis by port, by string match, by numerical properties, by behavior

and heuristics. Analysis by port is excluded from our project. Rests are explained briefly.

Analysis by String Match

Analysis by string match is the search process for a sequence of string or numeric values within the contents of the packet. Moreover string search may be done for several strings distributed within a packet or several packets.
There are many string match algorithms and best known are Boyer-Moore Algorithm, Naïve string search algorithm, Knuth-Morris-Pratt algorithm and Rabin-Karp algorithm. We used a more efficient version of Boyer Moore string match algorithm which is considered as one of the most efficient string matching algorithm.

Only analysis by string match may not be enough for determining application layer protocol since many applications declare their names within the protocol itself. Kazaa is one example of this situation. In User-Agent field Kazaa declares its name with a HTTP GET request. During transportation of data if user agent information is missing, string match analysis method will probably conclude the result of HTTP protocol, not Kazaa. So we have to apply several methods to ensure proper result.

Analysis by Numerical Properties

Analysis by numerical properties is the investigation of arithmetic and numerical characteristics of a protocol within a packet or several packets. Some examples of properties analyzed are payload length, the number of packets sent in response to a specific transaction and the numerical offset of some fixed string (or byte) value within a packet. For example while some applications establishing a TCP connection, a certain amount of bytes of data is transported between server and client.

Analysis by Behavior and Heuristics

Behavioral analysis can be thought as the way a protocol acts. Heuristic analysis can be thought as statistical parameters of examined packet transactions. Behavioral and heuristic analyses are combined to provide improved estimation capability. For instance, let us consider HTTP and P2P protocols. If the packet length histogram can be observed it becomes obvious that while pure HTTP packets tend

to concentrate around a few hundred bytes in length, P2P control layer information tends to use shorter packet lengths. In this way it may be possible to conclude whether a connection carries HTTP packets or P2P packets.

The first thing that Auto Sensing mechanism does is checking BUFFER (orderedPackets) for the packets not analyzed yet with the protocol id -1. Then it forks child processes for each protocol. Each process searches the relevant protocol signature in the packet data. There are signatures of the protocols kept in file(s). Processes will get the signatures from this file(s) and will try to match these with the packet data using pattern recognition algorithms.

7.4 PATTERN RECOGNIZER

One of the main parts of our program is pattern recognizer module. As understood from the name, pattern recognizer is responsible for identifying protocol type when IP packets are not so clear to determine which protocol it is.

In normal flow of the program all IP packets are going to input manager and then to auto sensing module. Auto sensing module senses protocol type and this information is passed to the summarizer module. Auto sensing module is determining the protocol type of the packet without using the port information. Although some applications/protocols use always same port, which enables us to detect protocol type very easily, nowadays lots of applications are giving the chance of selecting which port to use for communicating to the end-user. Since end-user can change the port used by application, if detection of protocol type is based on the port information, detection will be failed.

Protocols have commonly changing attributes. For that reason auto sensing module alone can be obsolescent when a little change have made in the structure of the protocol. So we must have a mechanism that can classify protocols even if some changes have occurred in the structure.

In some cases auto sensing module can not determine the type of the protocol. When such a case occurs input will be send to the pattern recognizer module. After that point pattern recognizer takes the control and determines the type of the protocol and gives this information to the summarizer module.

Pattern recognizer constitutes of three parts; Filtering, feature analyzing and classification.

Filtering:

This part of the pattern recognizer is responsible for getting rid of the unnecessary data from the input. The more successful filtering the more efficient will be the program.

Our program will analyze real time traffic and not only on big servers but also on personal computers sometimes network traffic can be several MB/sec. So we must use both CPU and memory resources efficiently. If this is not the case our program can not be used on systems which have huge network traffics since it wastes all of the system resources. So even if we have a running distribution of the program if it can not operate efficiently we will be failed. And we are aware of how it is important so that we will arrange our testing process according to it. We will be testing not only at alpha and beta versions but also in the development process using partial testing. As a consequence we can see which part of program is the bottleneck for the performance and fix it before proceeding. However if testing is done at the end of the coding process both finding which part of the code is the bottleneck and finding how can we make it better will be impossible. Even if we can state the problem since integration with other modules have done, data structures are determined, database tables are designed, making it efficient with another structure at that part of code can damage other parts of code which may cause a whole module to be corrupted.

Efficiency is the key point for our program and pattern recognizer module is the key module for efficiency and filtering part of this module enhance efficiency very much.

Filtering part takes information which we will use and discards other parts of the data. For example we are interested in TCP payload, other parts of an IP packet can be removed in filtering process.

Feature Analysis:

-Parameter extraction: As an output of we will get defined parameters in the filtered data.

-Feature extraction: This method will be used for the purpose of dimension reduction. Dimension reduction can be beneficial not only for reasons of computational efficiency but also because it can improve the accuracy of the analysis. As an input this method have P dimensional parameter vector and it modifies this parameter vector to an R dimensional new parameter vector where R is less than P. This step is not necessary if parameter extractor is already has reduced the dimension of the data vector so that pattern classifier has matched the data that is if output vector's dimensionality of parameter extraction step is equal or lower.

Classification:

Classification is responsible for determining for an object to which group it matches. Since pattern matching and pattern recognition are different then each other. Pattern matching only can group the objects where defining class and objects are matching exactly. However pattern recognizing techniques can group although there is not an one-to-one correspondence between defining class and objects. For pattern recognition only similar data patterns are sufficient.

There are several classification methods which can be considered as . We can use;

- Common property concept

  The common properties of the all protocols are saved on our specially designed common property database. And packets are analyzed according to the common properties extracted from them. Then defined common properties and new coming common properties data patterns are scanned. Similar patterns have grouped under same class.

- Support vector machines

  Support vector machines map input vectors to a higher dimensional space where a maximal separating hyper plane is constructed.

- Neural Approach

  Neural approach uses artificial neural networks to classify data patterns.

Actually, what pattern recognition module does is a machine learning activity. We analyzed machine learning methods and we decided to use supervised learning because it is suitable for data classification via statistical information of given input

set. There are a lot of implementations of supervised learning. Among them Bayesian network is capable of computing the probabilities of the presence of various conditions in a beforehand given input set. Even if this input set is subjective Bayesian network can classify them. And by using dynamically changing data it can update Bayes's conditioning which it relies on.

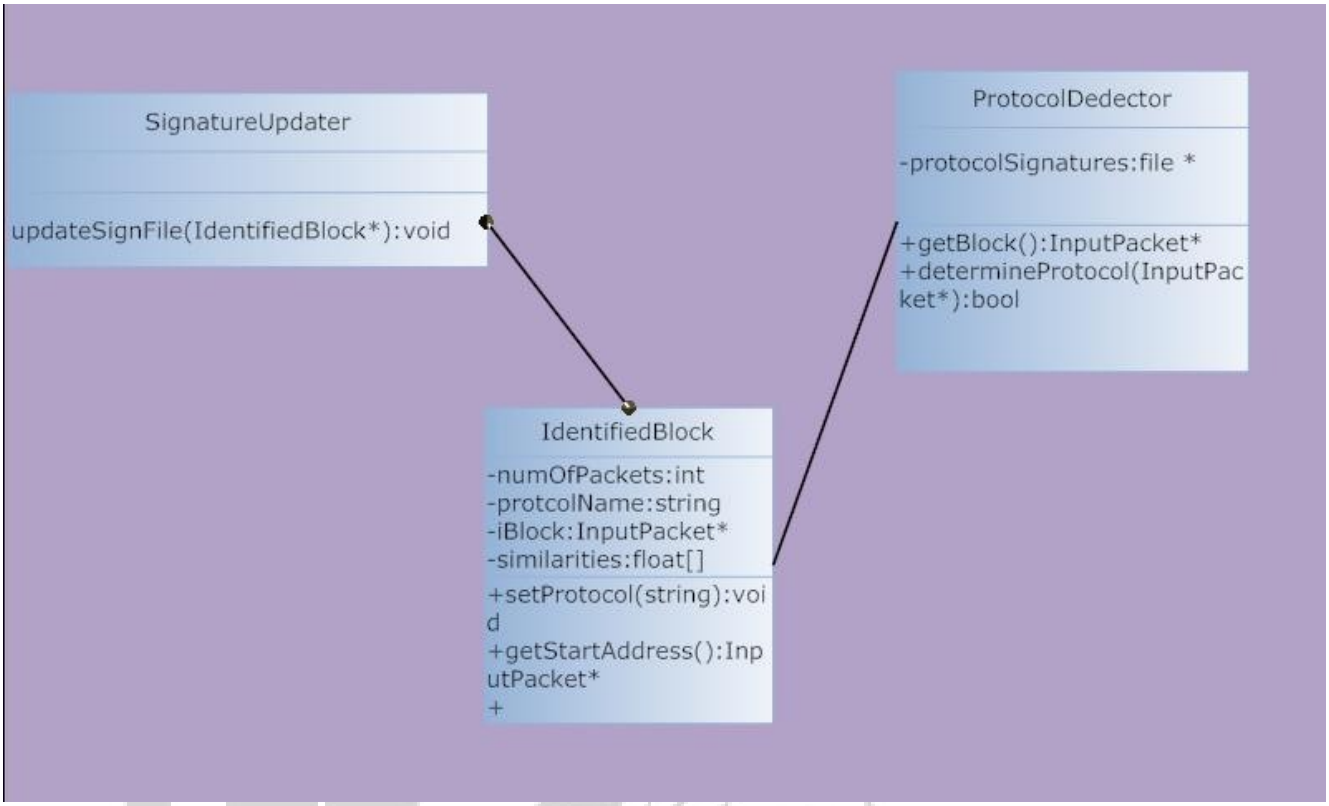
An approach of Bayesian network is hidden markov model. This model will be implemented using Viterbi algorithm. We have chosen these because with an unknown parameter it can deduce its type by using observable parameters.

Our observable parameters are the keywords of the protocols. This observable parameter set at first only constituted of the available protocol information. But since this is an machine learning this set will be enlarged in time. Some unknown parameters can grouped into the observable parameters if they can reach some ratio.

Let’s look at the technical implementation of the algorithm. We will have some states and we can assume we have a special kind of finite state machine. These states can be keywords and some protocol specific entity. And we have observations which can be protocol names . And since we are working on a statistical and not exact environment we will not have start condition or transition condition. We will have start probabilities and transition probabilities. And we will give these probabilities by using state statistics.

And we will have emission probabilities. These probabilities give us the percent of the protocol matching. For example in a given state a packets protocol probability can be as POP3: 35% SMTP: 40% and FTP: 25%. If we think hidden markov model as a finite state machine, we will change from one state to another by using these probabilities and derive a result. Then using this result we will update our observable pattern set to be used by time.

Auto sensing class diagram

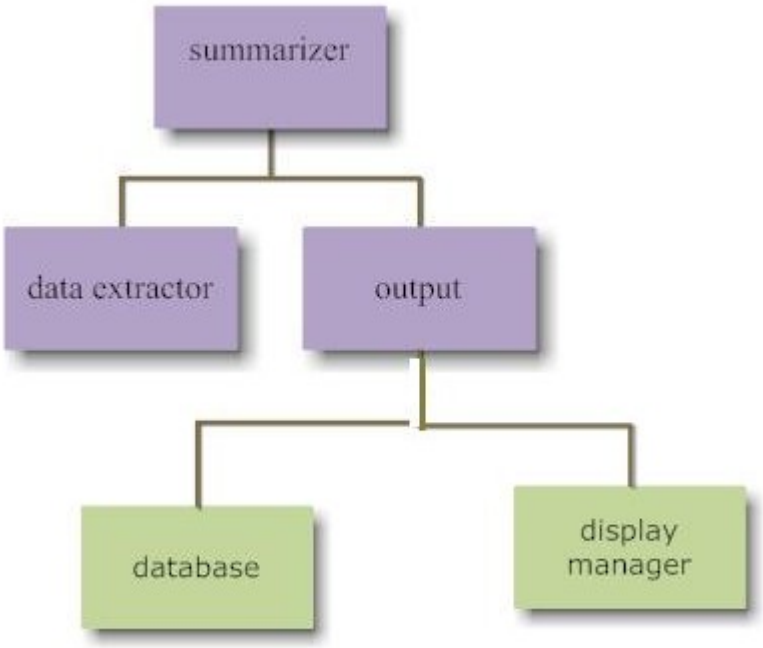


ProtocolDedector class gets a packet block(group of related packets) from ***PacketBlocks*** class. This is the most important class of the project, because using pattern recognition techniques, the protocol of packets are determined here. Also we keep the signature of protocols in a file. After protocol was identified this class produces an output: ***IdentifiedBlock***

IdentifiedBlock class is needed in order to keep the packet blocks whose auto-sensing process finished. protocolName attribute is either a protocol or unknown. This class is also the input of summarizer.

Signature updater is updating the signature file. Therefore this file is dynamically chanced. In order to determine the chances the output of the protocolDedector(identifiedBlock) must be used.

3)Summarizer



We will design summarizer as a module that consists of two sub modules: data extractor and output.

3.1) Data Extractor

It takes a packet block or an individual packet whose protocols is identified or unidentified by auto-sensing, as input and extracts necessary information (data) from those packets. These packets are either a text based protocol such as SMTP or an instant messaging protocol like YMSG (yahoo messenger) or unknown protocols. The packets, whose protocols are not identified, are sent to the output module directly. If the protocols of the packets are known, then extraction starts. After extracting necessary information, this module sent the extracted data and packet information's like source IP, protocol, size, etc to output module. To understand how this module works let us concentrate on an example. Suppose that we receive an e-mail that consists of 20 different packages. Auto-sensing module determines the protocol of this mail then data extractor gives that mail(packages) from auto-sensing. Suppose the protocol of this mail is SMTP. Next we eliminate first a few

packets, which are related with connection, and extract each remaining packet separately and sent these packet blocks to output module.

3.2) Output

Output module has also two sub modules: database and display manager.

3.2.1) Database

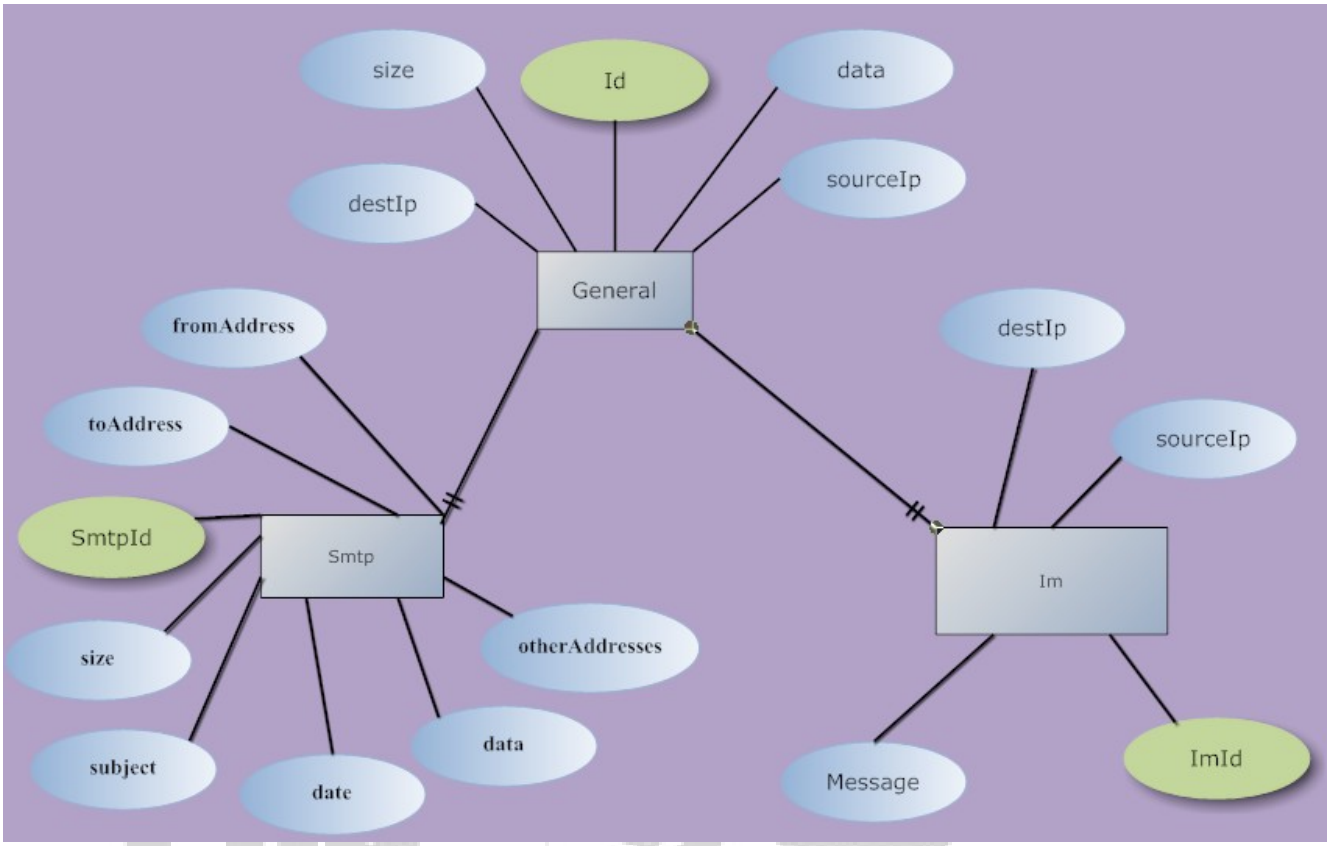
After the data's are extracted, they are come to this module. The extracted data saved in database. All database operations are held by this module. Since we deal with huge inputs a database is a must for our project. In database we hold three things:

- e-mails (that comes through SMTP protocol)
- instant messaging entries (that comes through one of the IM protocols)
- unknown packets

E-mails are saved in .eml format, the format of other files are any file format. We also save the packages whose protocol has not been determined.

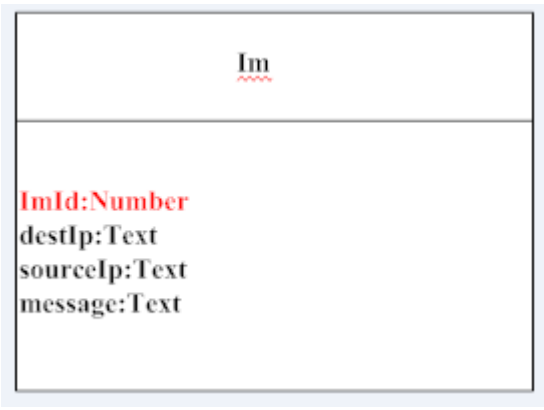
Entity Relation diagram for database

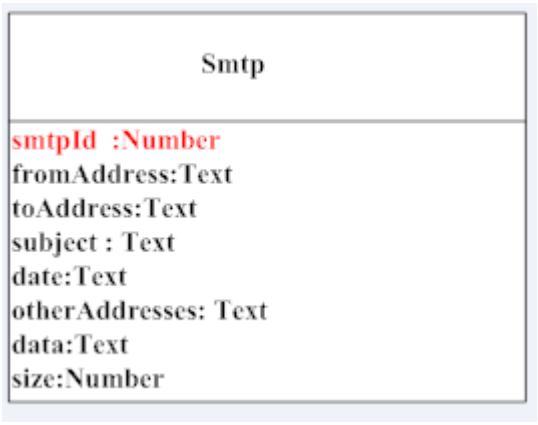
The Codefather inc.



Entities

General
id :Number data :Text size :Number destIp :Text sourceIp :Text





Database Tables and construction

We have three tables for the database. These tables are SMTP, Im and general.(general is used for unknown protocols). Since the smtp and Im are derived from general we choose general's id as primary key. Below the tables are shown:

Database Table for general file:

Name	Kind	Description
id	Number	
data	Text	
size	Number	
destIp	Text	
sourceIp	Text	

In this table id represent the id number of each entry, it is also key value. Data represent the data in the file, since general file format is used for unidentified packets, data is representing 'row data'. Size means 'size of the file'. destIp and sourceIp represent the ip addresses.

Creating general Table:

```
CREATE TABLE `codefather`.`general` (
`id` int(10) unsigned NOT NULL auto_increment,
`date` varchar(16) NOT NULL,
`size` int(10) NOT NULL,
`sourceIp` varchar(20) NOT NULL,
`destIp` varchar(20) NOT NULL,
PRIMARY (`id`) ) ;
```

Database Table for Im file:

Name	Kind	Description
ImId	Number	
destIp	Text	
sourceIp	Text	
message	Text	

ImId is represents the Im file id it is also primary key for this files. destIp and sourceIp shows the IP numbers and message shows the instant message content(entry).

Creating Im Table:

```
CREATE TABLE `codefather`.
`Im` ( `ImId` int(10) unsigned NOT NULL auto_increment,
`sourceIp` varchar(20) NOT NULL,
`destIp` varchar(20) NOT NULL,
`message`  varchar(500) NOT NULL,
PRIMARY KEY (`ImId`) );
```

Database Table for Smtp file:

Name	Kind	Description
smtpId	Number	
fromAddress	Text	
toAddress	Text	
subject	Text	
date	Text	
otherAddresses	Text	
data	Text	
size	Number	

SmtpId is the primary key for mail packages. We also save destination address (from address) and source address (toAddress). If the number of destination address is more than one, all of them saved too. Subject represents the subject of e-mail and date represents the date. The message body is saved in data and total file length is saved in size.

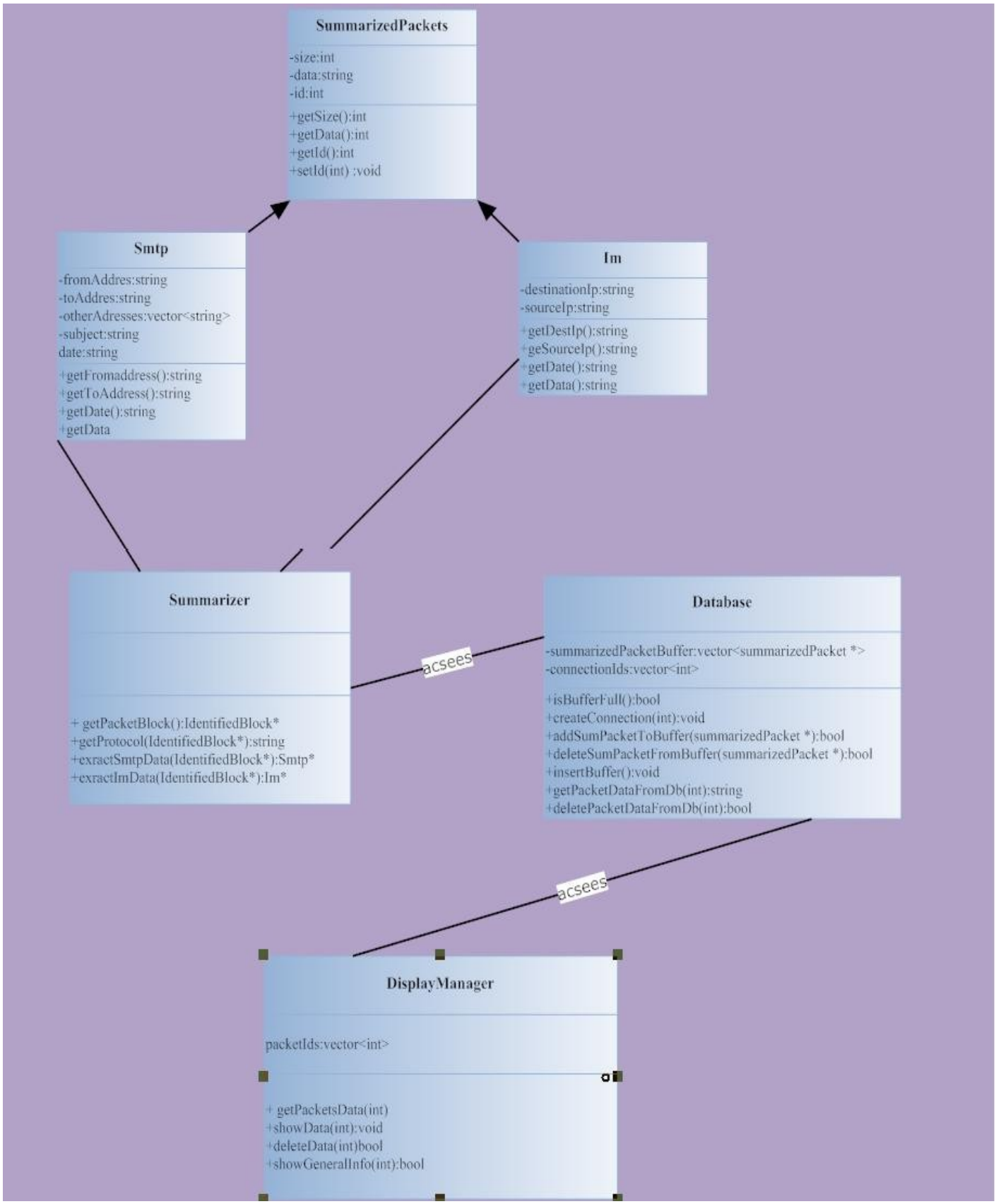
Creating Smtp Table:

```
CREATE TABLE `codefather`
`Smtp` ( `SmtpId` int(10) unsigned NOT NULL auto_increment,
`fromAddress` varchar(50) NOT NULL,
`toAddress` varchar(50) NOT NULL,
`subject` varchar(50),
`data`  varchar(5000) NOT NULL,
`otherAddresses` varchar(500),
`size` int(10) NOT NULL,
PRIMARY KEY (`SmtpId`) );
```

3.2.2) Display Manager

This module is communicated with user via graphical user interface. According to user commends this module shows the extracted data of packets. In order to do that a connection to database must be created. This module also shows the information about packets (protocol, size, etc.).

Summarizer class diagram



SummarizedPackets: generic class for summarized packets. A summarized packet is either a SMTP packet or one of the IM (instant messaging) protocol's packet. This class contains common attributes for these packages.

Smtp: this class is responsible for storing data of smtp packages. If we capture a mail's packets, this mail's important information's are store here. This class is inherited from ***SummarizePackets*** class.

Im like Smtp class is inherited from ***SummarizePackets*** class. Im class is responsible for instant messaging packages.

Summarizer class operates on IdentifiedBlock class's objects. If auto-sensing determine the protocol then summarizer extract data of packets. (Summarizer extracts only SMTP and one instant messaging protocol, probably YMSG). After extracted the data and information of packets, these must be directed to the database class. If the protocol is not determined then this packets are directly sent to database class. In the future these packets may become identifiable because we update protocol signature file regularly.

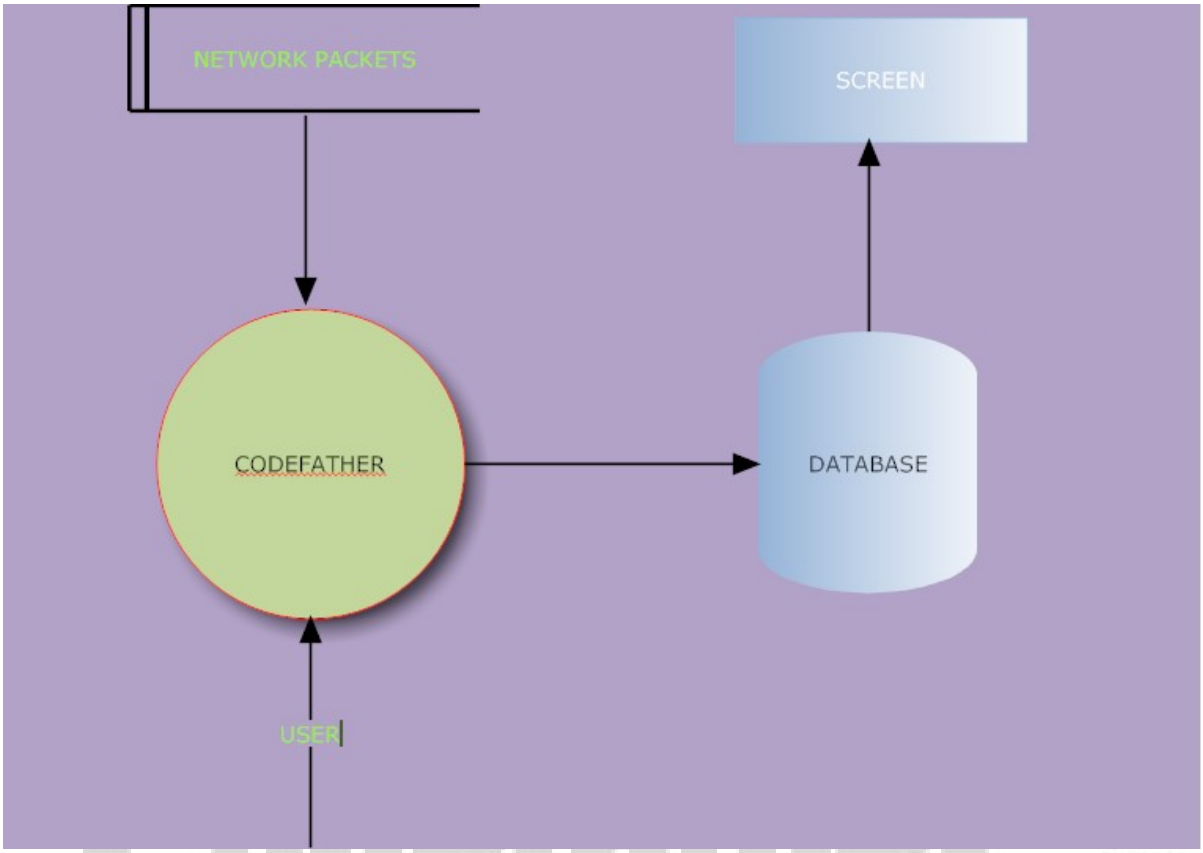
Database class is responsible for adding, deleting, getting records from database. A record is a mail, or an instant message or unknown packets. There is a buffer inside database class. When this buffer is full we insert the new records to database. This operation is important because of efficiency.

Display Manager is responsible for user interactions, it also interact with database class. What will show to user from GUI is determined by this class.

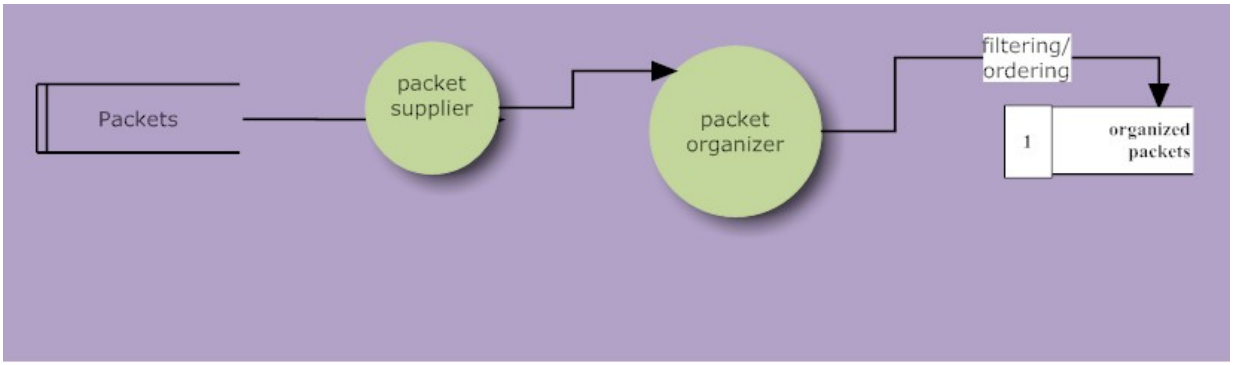
DATA FLOW DIAGRAMS

1)Level 0 DFD’s

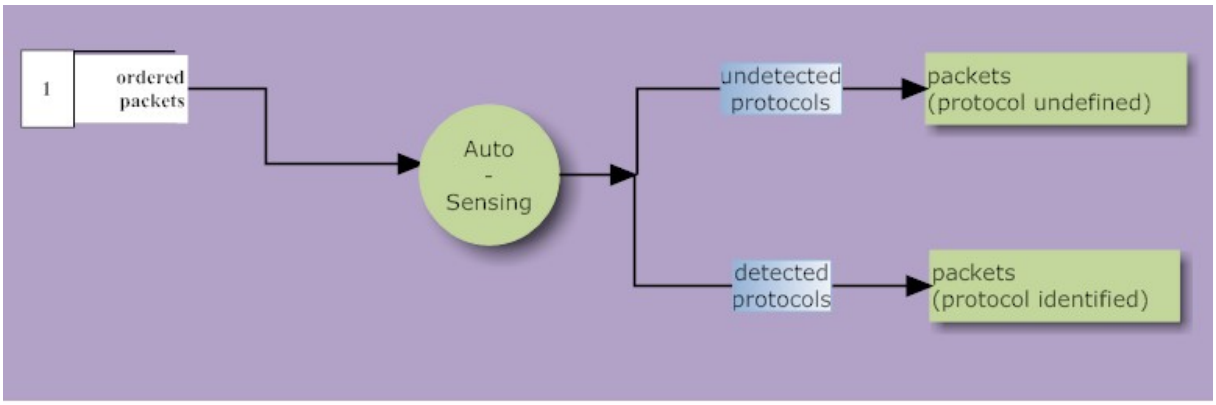
a)general level 0 dfd



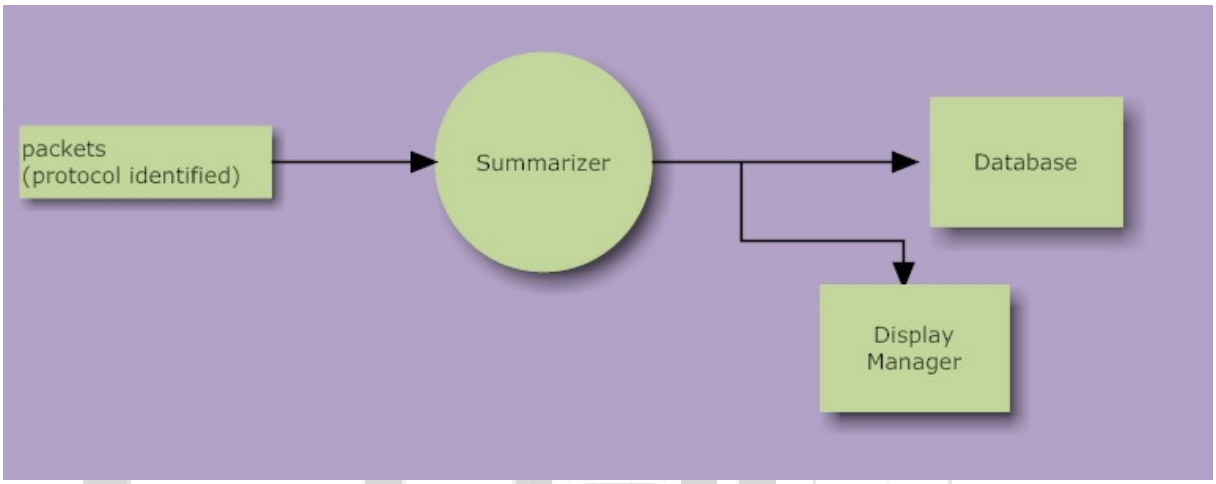
b)packet manager



b)auto-sensing



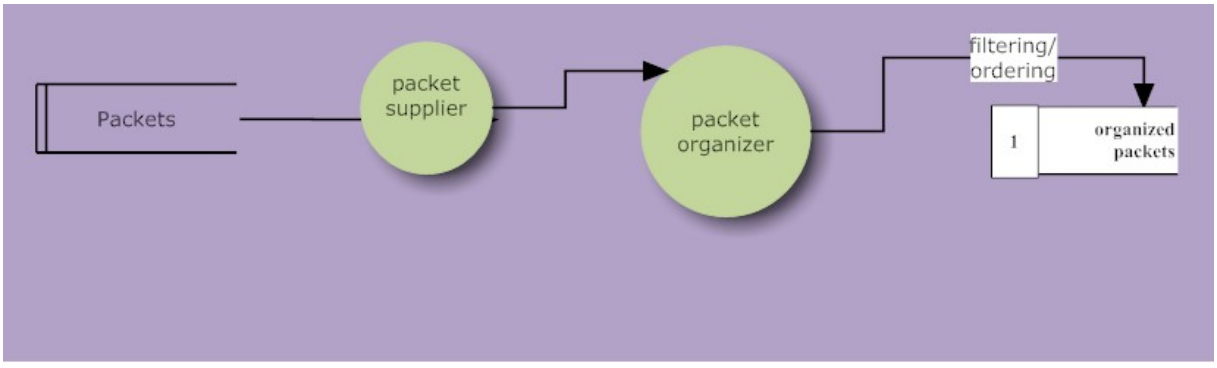
c)Summarizer



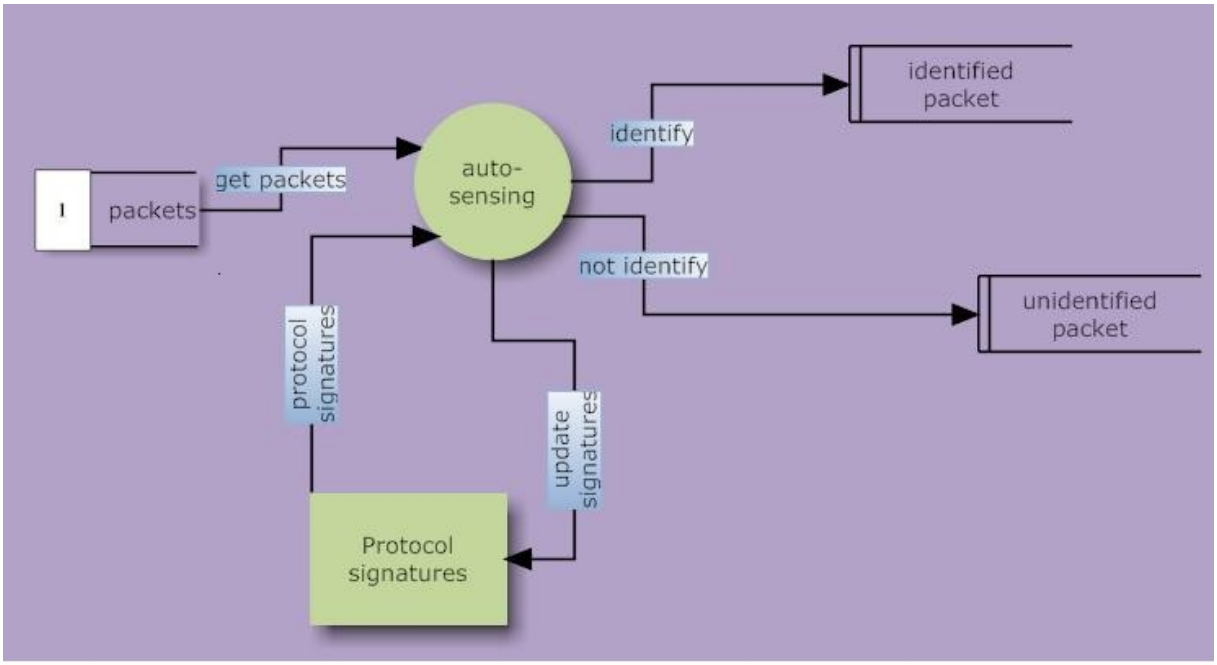
CodeFather inc.

2)Level 1 DFD's

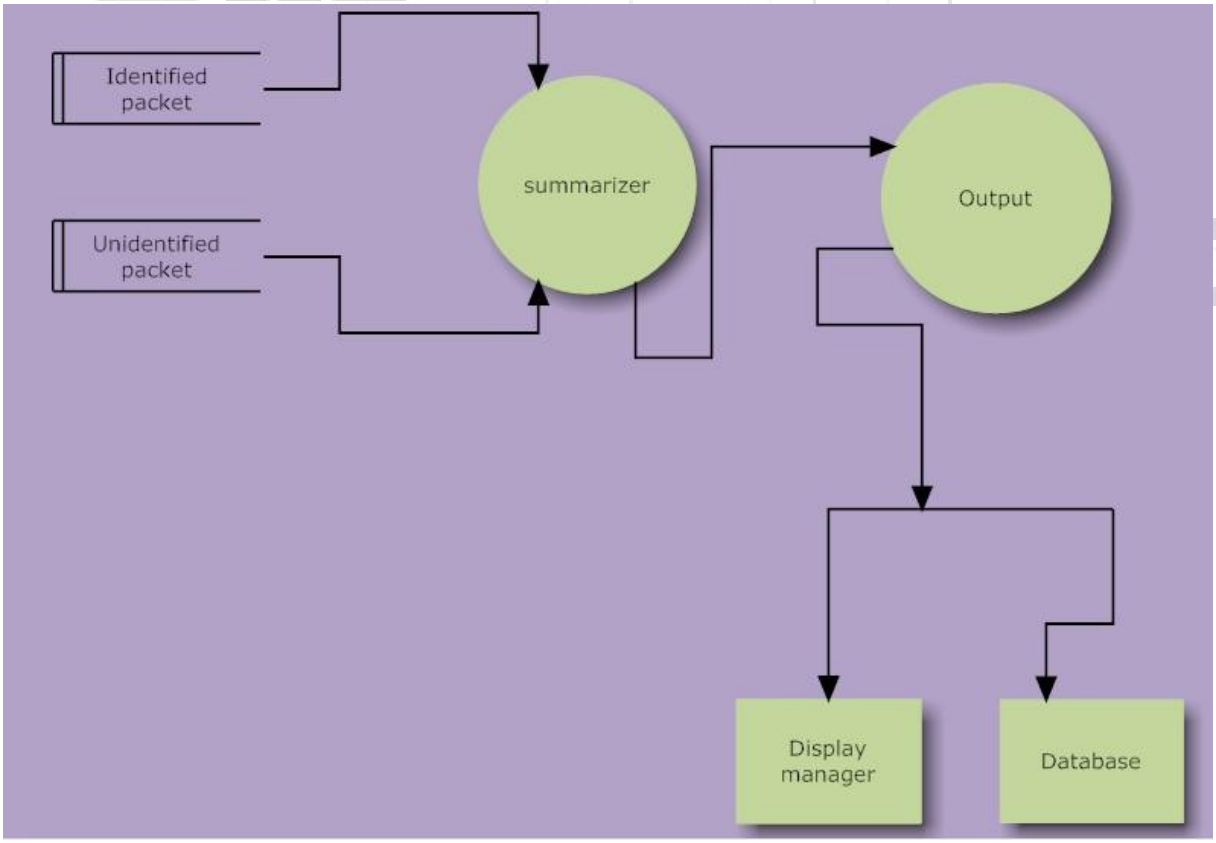
a)packet manager



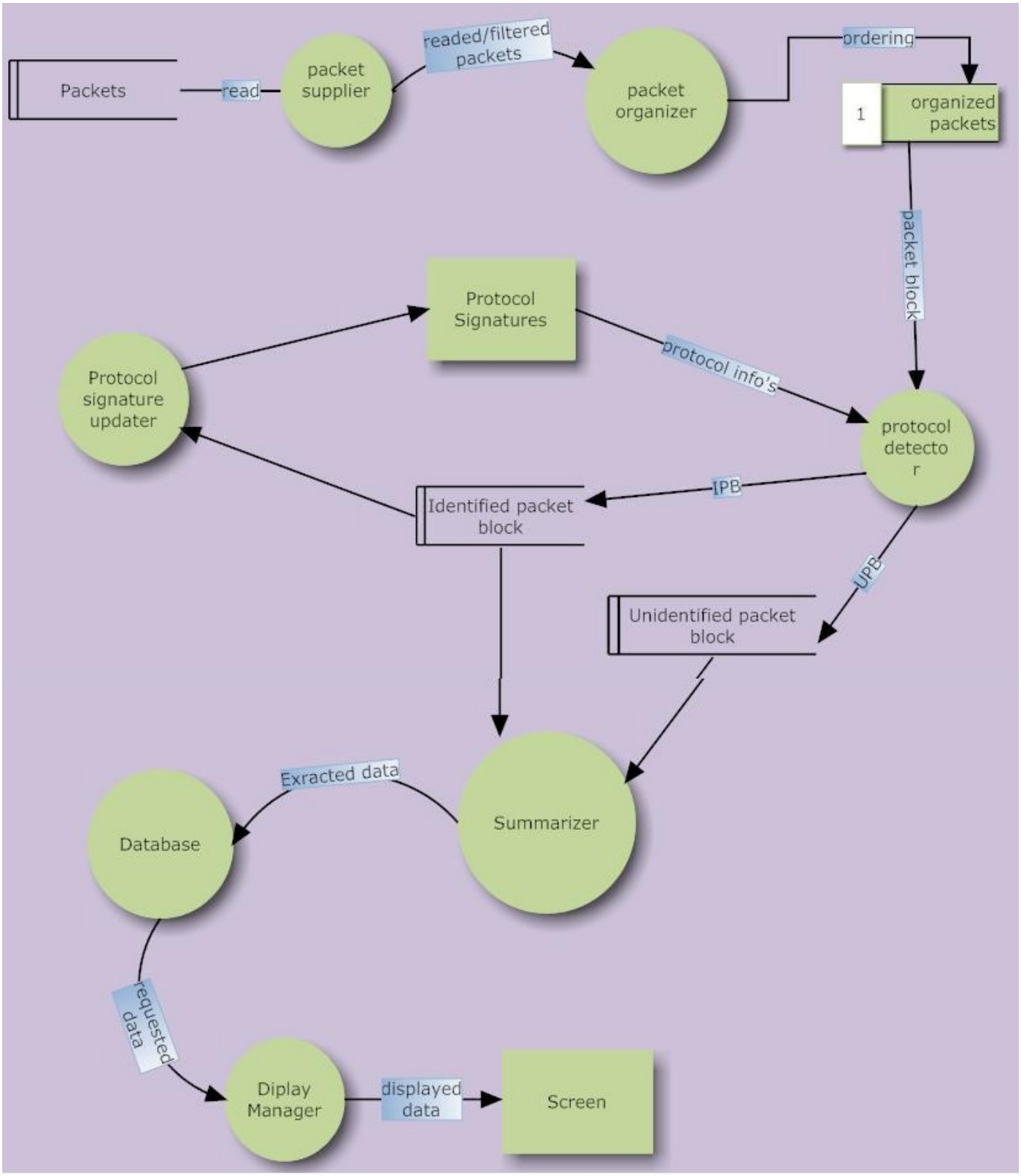
b)auto sensing



c)summarizer



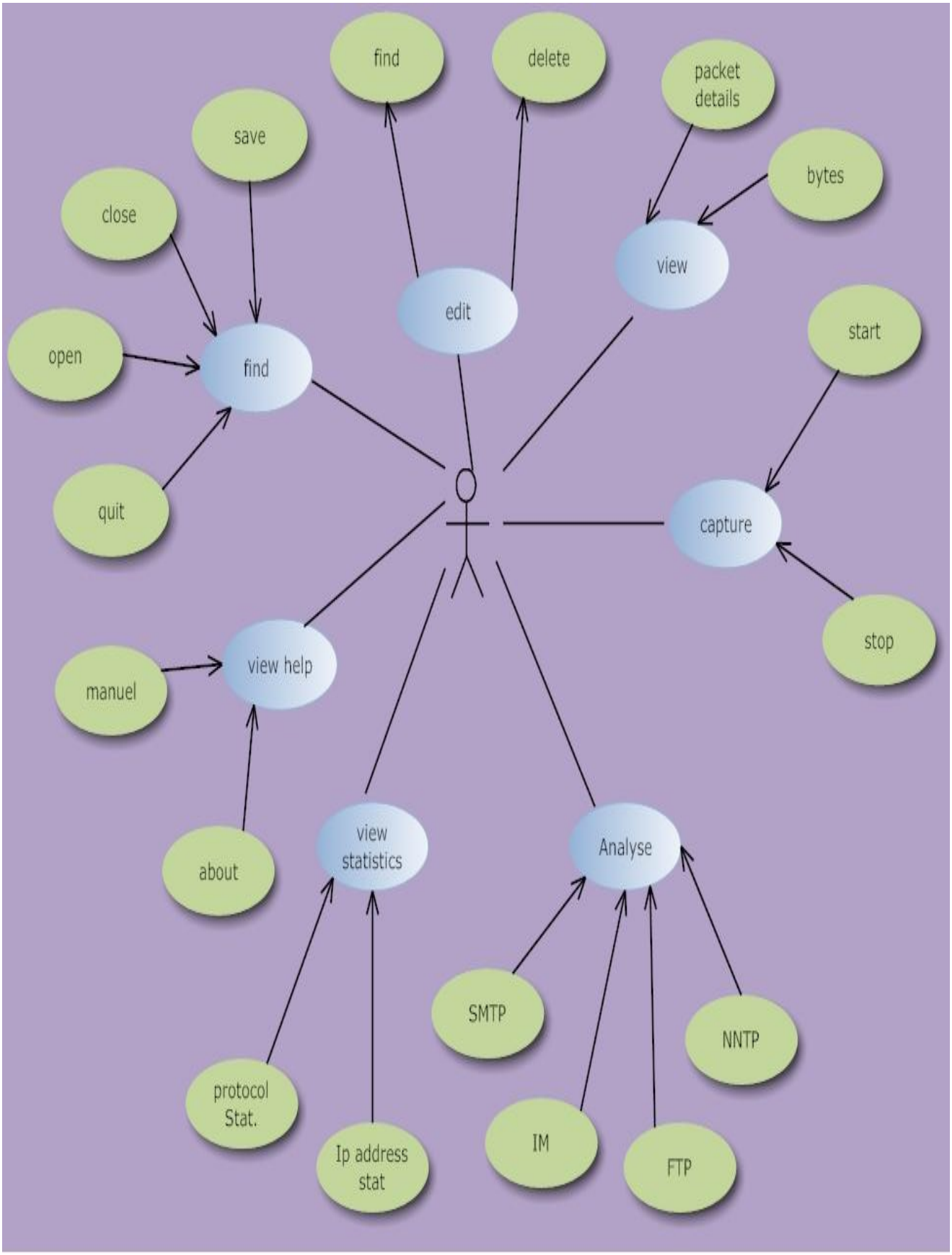
3)Level 2 DFD of project



DATA DICTIONARY

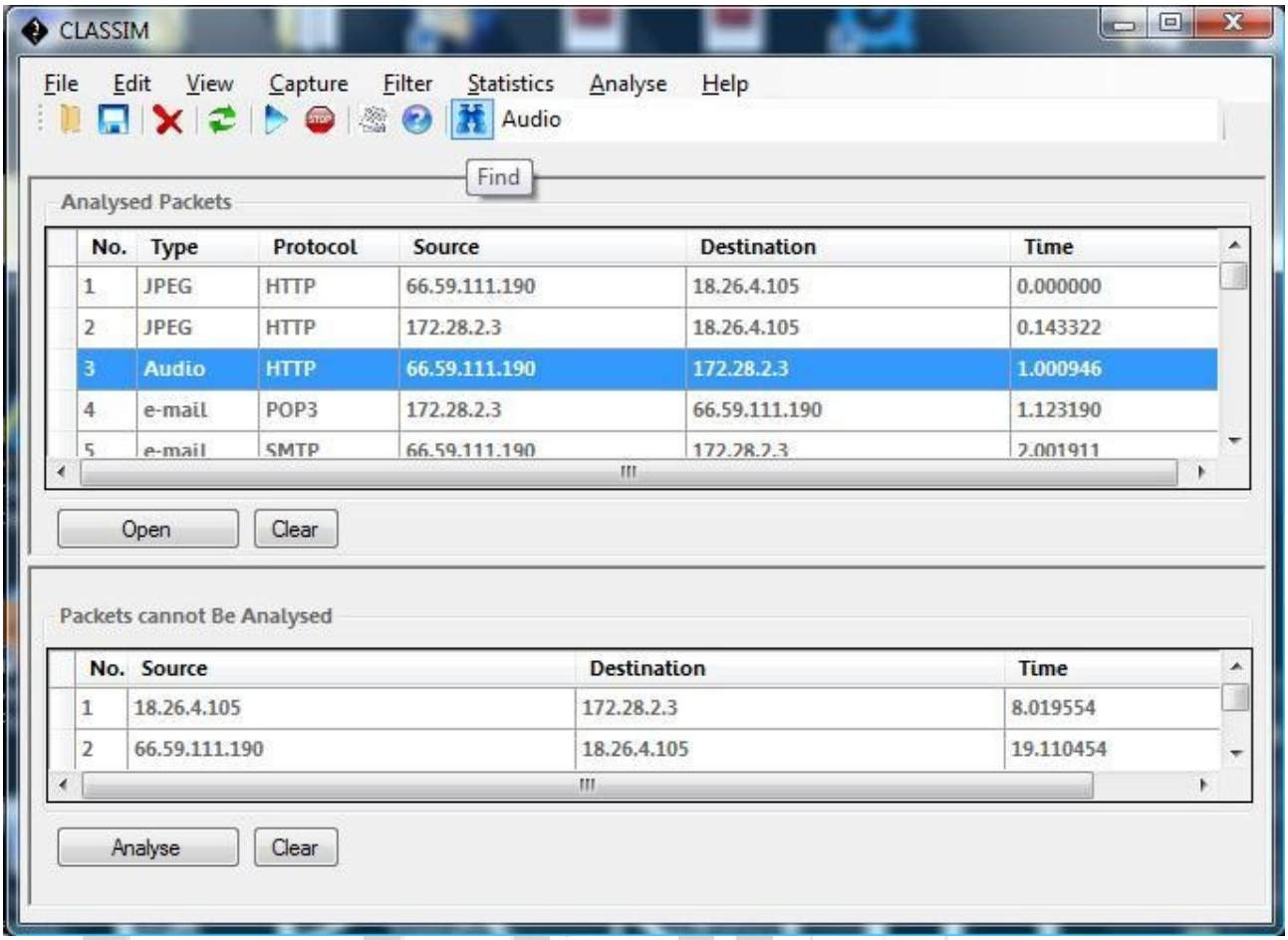
NAME	DESCRIPTION
read	Obtaining network packets from a pcap file or real time captured. This packets are supplied project by packet supplier
Read/filtered packets	Packets supplier get the packets and filtered them, so unnecessary packets are eliminating from the project here.
Ordering	Packet reorganizer module reorders the read packets since their normal coming order is not the order that we want. After packets are ordered they are stored in the organized packets. In the organized packets the related packets are stored like a block of packets.
Packet Block	Protocol detector gives a packet block from organized packets. By using pattern recognition techniques determine protocol.
Protocol info's	Protocol detector read the protocol signatures in order to determine new coming packets protocol. These signatures read from a file.
IPB(identified packet block)	If the protocol of a packet block was determined by protocol detector an IPB created. This packet is given by summarizer.
UPB(unidentified packet block)	If the protocol of a packet block was not determined by protocol detector an UPB created. This packet also given by summarizer.
update	Updating the protocol signatures by determining new signatures from a identified packet block
Extracted data	Summarizer extract data from packets whose protocol was determined. Then this summary sent to the database. Database class insert them to database.
Requested data	The user may want to see the summary or packets information's on the screen. This summary and information transacted from database and display manager gives these things.
Displayed data	After the data transacted from the database, display manager shows the packet information and summary to the screen.(by using gui)

USE CASE DIAGRAM



7.5GRAPHICAL USER INTERFACE (GUI) and DISPLAY MANAGER

The user interface of our program will mainly look as below:



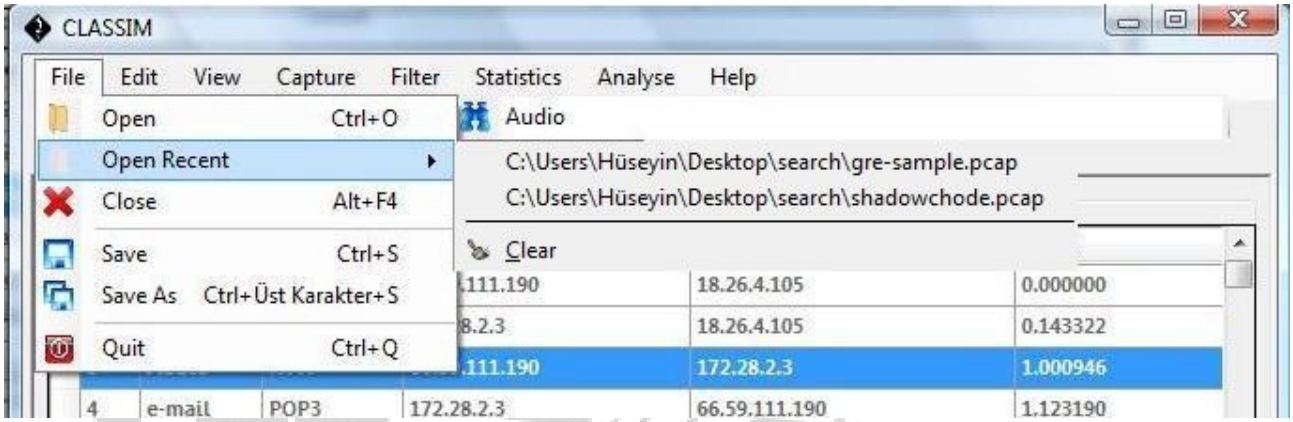
This is a sample screenshot. This first coming screen is composed of three parts; namely menu bar, tool bar and two panels. User can start and stop the process, see and filter the analyzed and non-analyzed packets in the panels; make search through panels; for analyzed packets, see packet details and open the packets (for example, if the packet is a type of audio; actually, no need for making own media player for the program; hence, open with windows media player for instance), analyze the non-analyzed packets for selected protocols, save the packets, show and hide the panels.

In the window, there exist two panels which contain analyzed files and non-analyzed packets, separated by a splitter.

7.5.1. MENU BAR

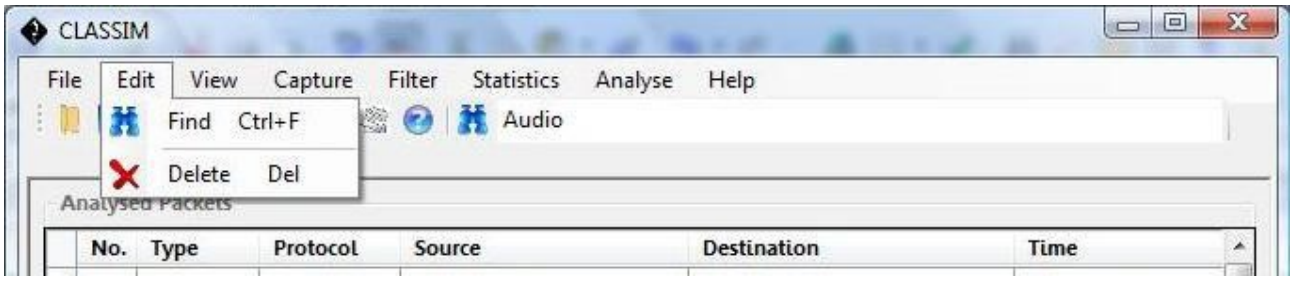
Menu bar is composed of eight items; namely File, Edit, View, Capture, Filter, Statistics, Analyze and Help. Let us have a look at each more in detail.

7.5.1.1. FILE MENU



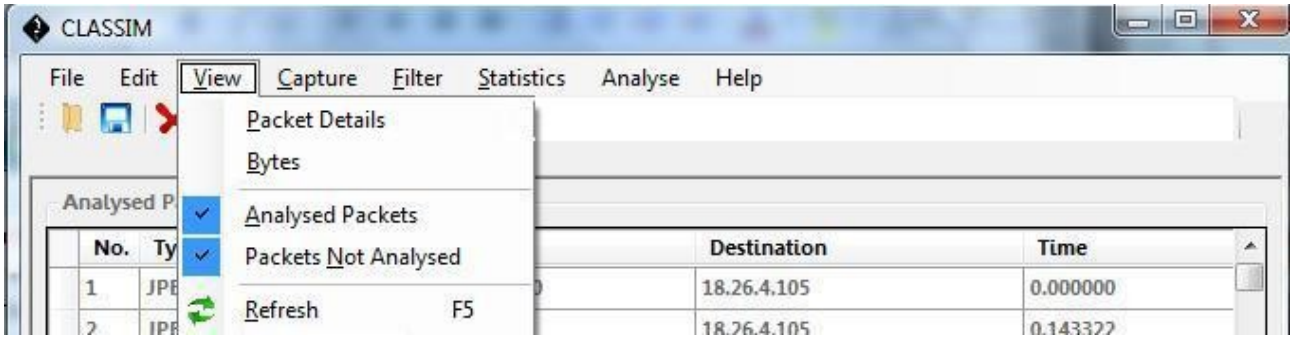
By this menu, user can open an existing pcap file and open a recent file to analyze it, save the packets. Clicking on 'Close' closes the panels and results, whereas clicking on 'Quit' not only closes the panels but exit the program entirely.

7.5.1.2. EDIT MENU

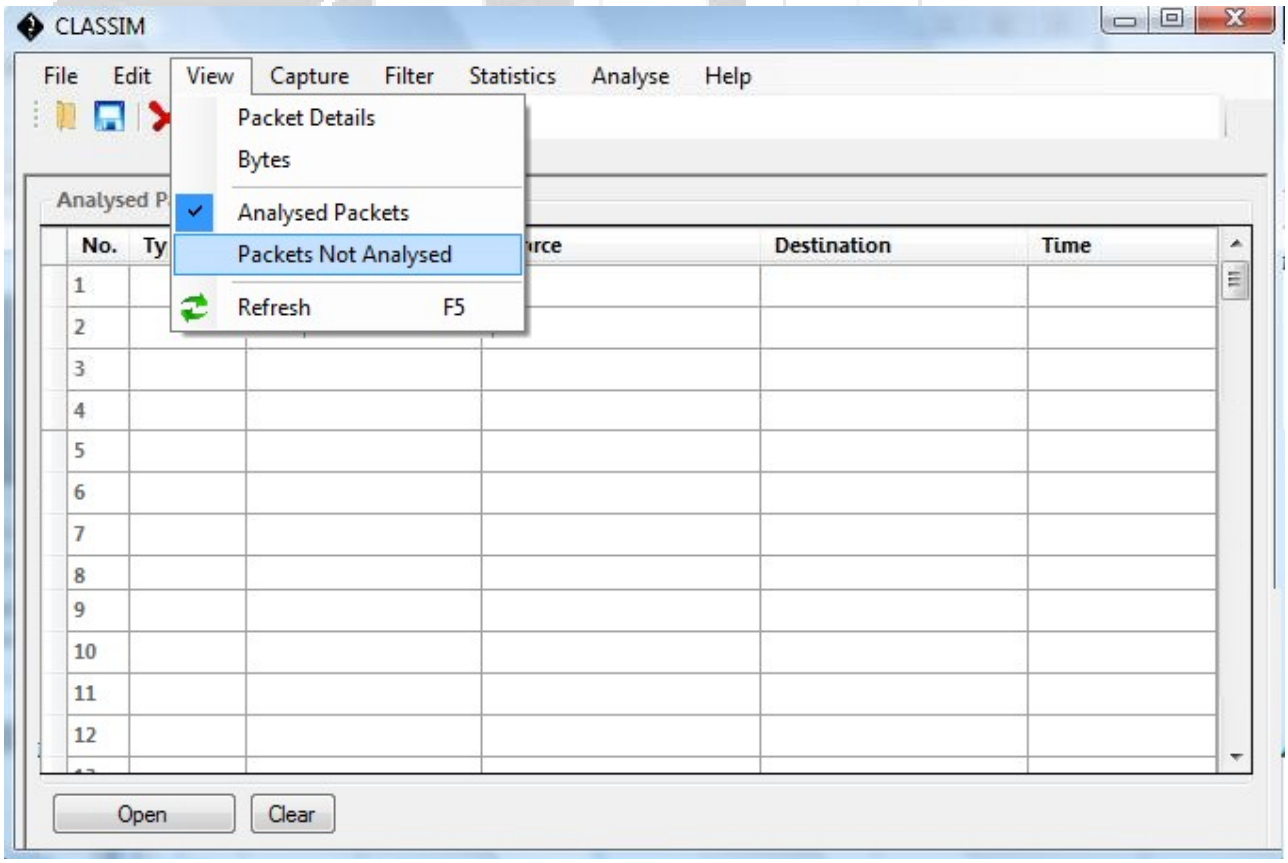


Edit menu is used for finding out a keyword through the panels, and deleting a line (i.e. a file or a packet) from panels.

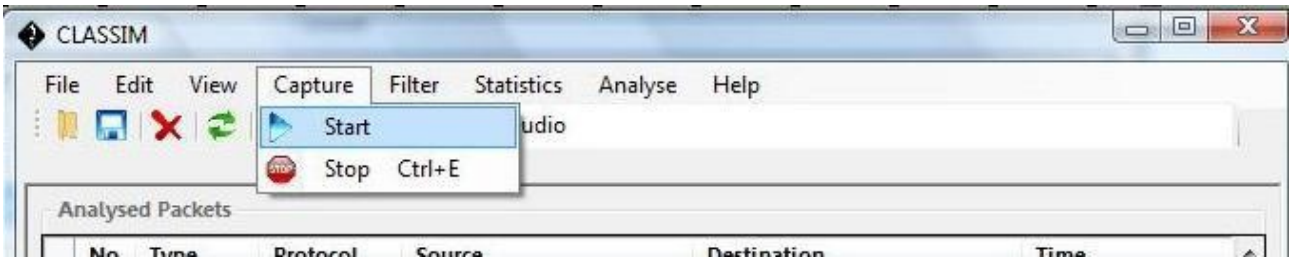
7.5.1.3. VIEW MENU



User can see packet details (headers, payloads, bytes) of the files or packets (i.e. if analyzed, files; if not, packets) if required. Moreover, there exist two panels separated by a split. If the user desires to see more lines of analyzed files (or non-analyzed packets), the other panel can be ticked off in order only one panel to take place in the entire window. Below is a sample screenshot for this situation:

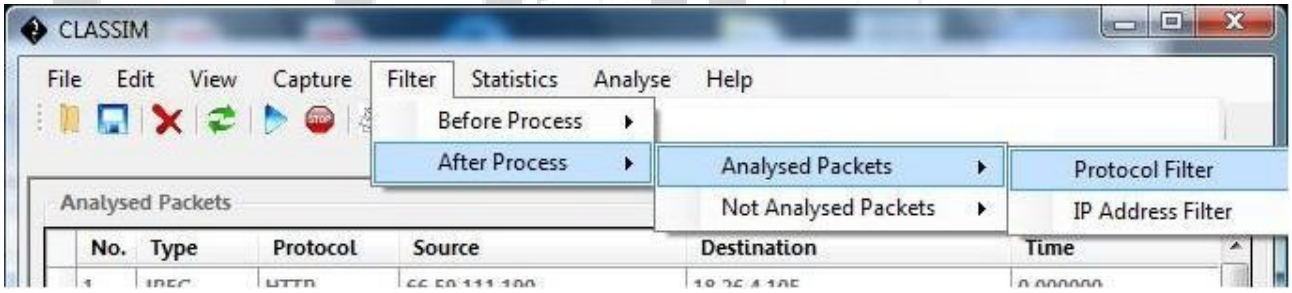


7.5.1.4. CAPTURE MENU



From the capture menu, by clicking on 'Start', real-time capturing process is initiated, and the process is stopped by clicking on 'Stop'.

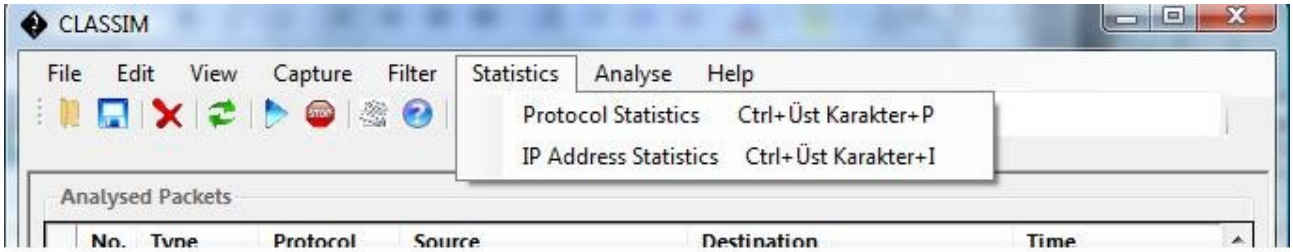
7.5.1.5. FILTER MENU



There are two kinds of filtering. One is filtering before capturing. Filtered items are captured for this case. Other one is filtering the captured files (if not analyzed, packets). If the user makes filtering through non-analyzed packets, s/he should follow up:

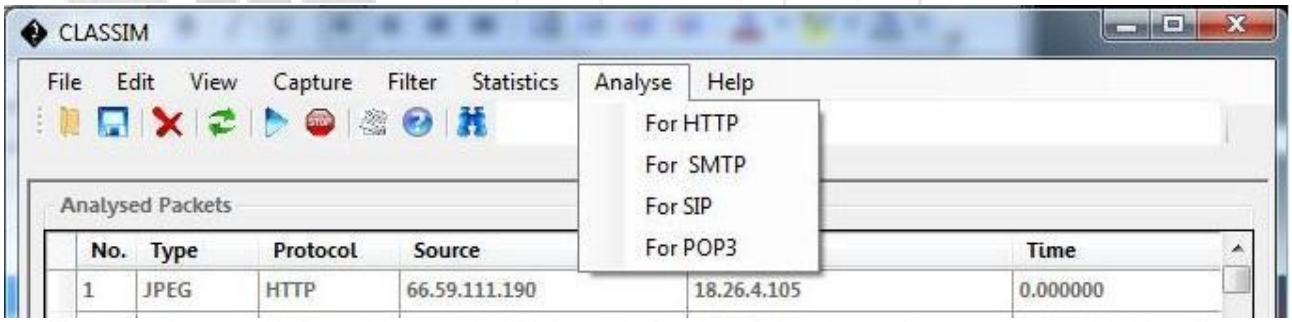
Filter -> After Process -> Non Analyzed Packets -> Protocol Filter. After following up these steps, there appears a dialog box, and enter what to filter in textbox.

7.5.**1.6. STATISTICS MENU**



User can see the comparison of protocols and IP addresses density of the network traffic of which s/he is analyzing from the 'Statistics' menu. User can show the numbers and percentages of the protocols and IP addresses of the network traffic.

7.5.**1.7. ANALYZE MENU**



There may also be non-analyzed packets inevitably. The user can analyze and decode them for four of layer 7 protocols; namely, SMTP, SIP, and POP3 using 'Analyze' menu bar.

7.5.**1.8. HELP MENU**

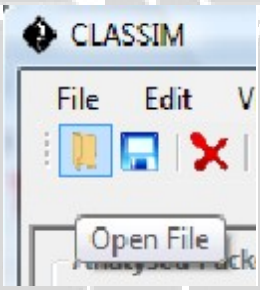
Inside the 'Help' menu, user can view manual of this program, and there also exists 'About'.

7.5.**2. TOOL BAR**

Current Tool Bar functionalities are related with frequently used Menu Bar items. They are;

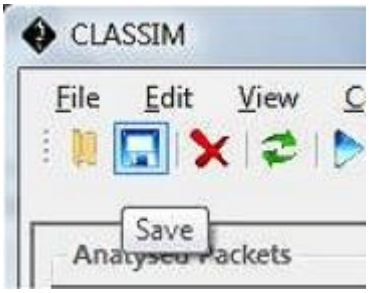
7.5.**2.1. Open**

Opens an existing pcap file:



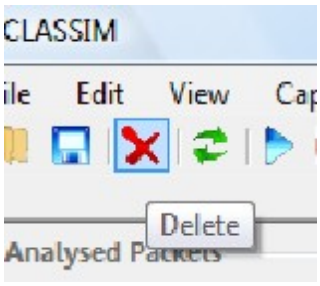
7.5.**2.2. Save**

Saves the packets:



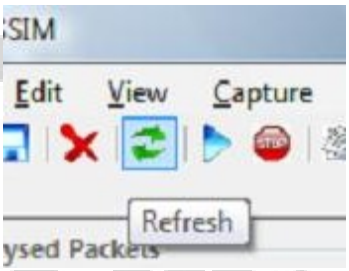
7.5.**2.3. Delete**

Deletes files and packets:



7.5.**2.4. Refresh**

Refresh:

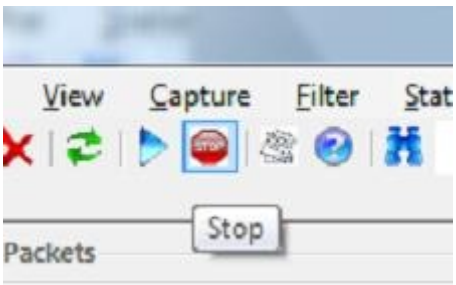


7.5.**2.5. Start!**

Starts capturing:

7.5.2.6. Stop

Stops the process:



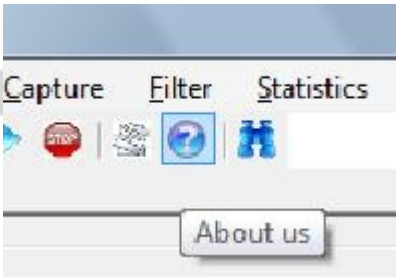
FreeFather inc.

7.5.**2.7. Manuel**

Views the manual:

7.5.**2.8. About us**

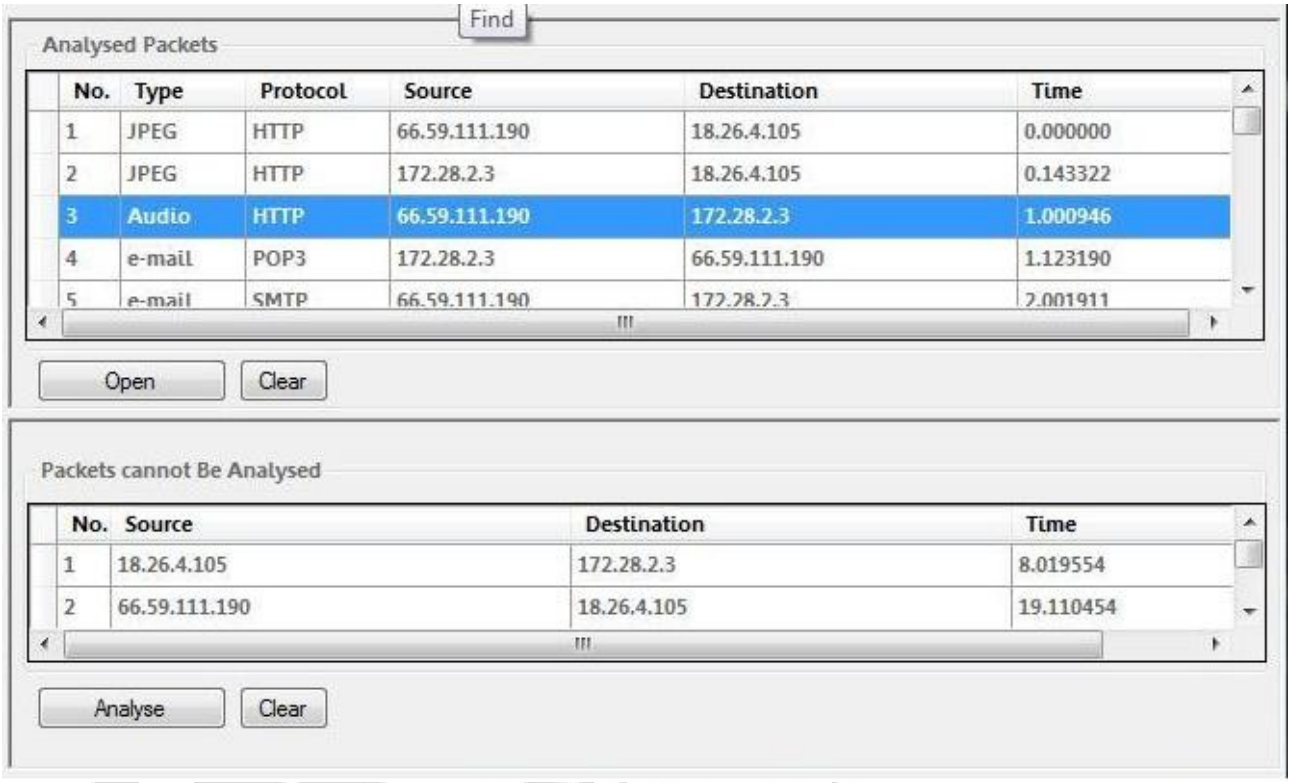
Shows 'About us':



7.5.2.9. Find

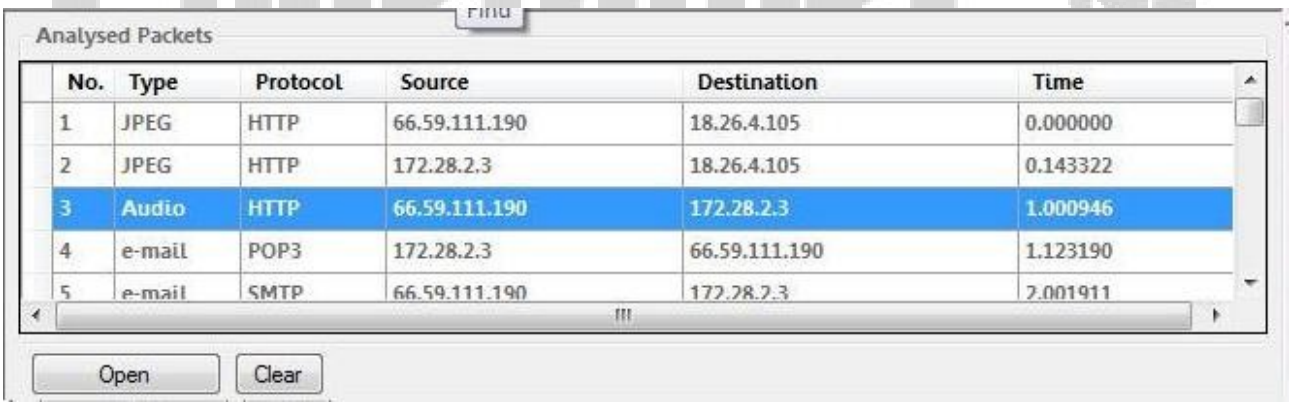
Makes search:

7.5.3. PANELS



As seen above, there are two panels in the window; one for analyzed packets, and other for non-analyzed.

7.5.3.1. Analyzed Packets

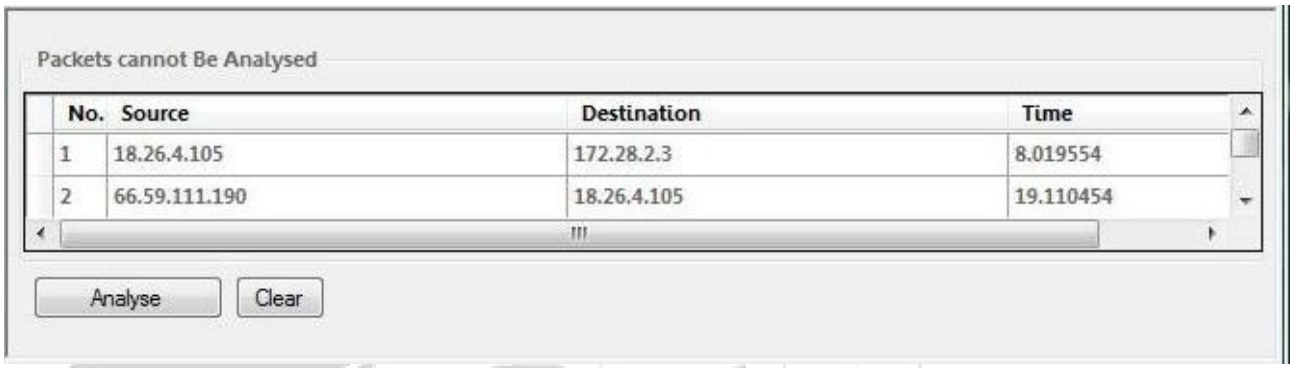


This section is for analyzed packets as stated above. Each line denotes decoded data in the format of JPEG, audio or whatever it is. In each column, there exists one file's type, in which layer 7 protocol it is sent, sender IP, receiver IP and time. Actually, we have not decided how to keep and show the time for certain. In

the final design, we may prefer to keep one file's total capture time (i.e. time difference) for analyzed files.

There exists two buttons at the bottom; open and clear. 'Open' opens the selected line, i.e. selected file with the right default application. 'Clear' clears the current elements of the panel. Moreover, when user right-clicks on a file; there are three options: 'Open', 'Delete' and 'See Packet Details'.

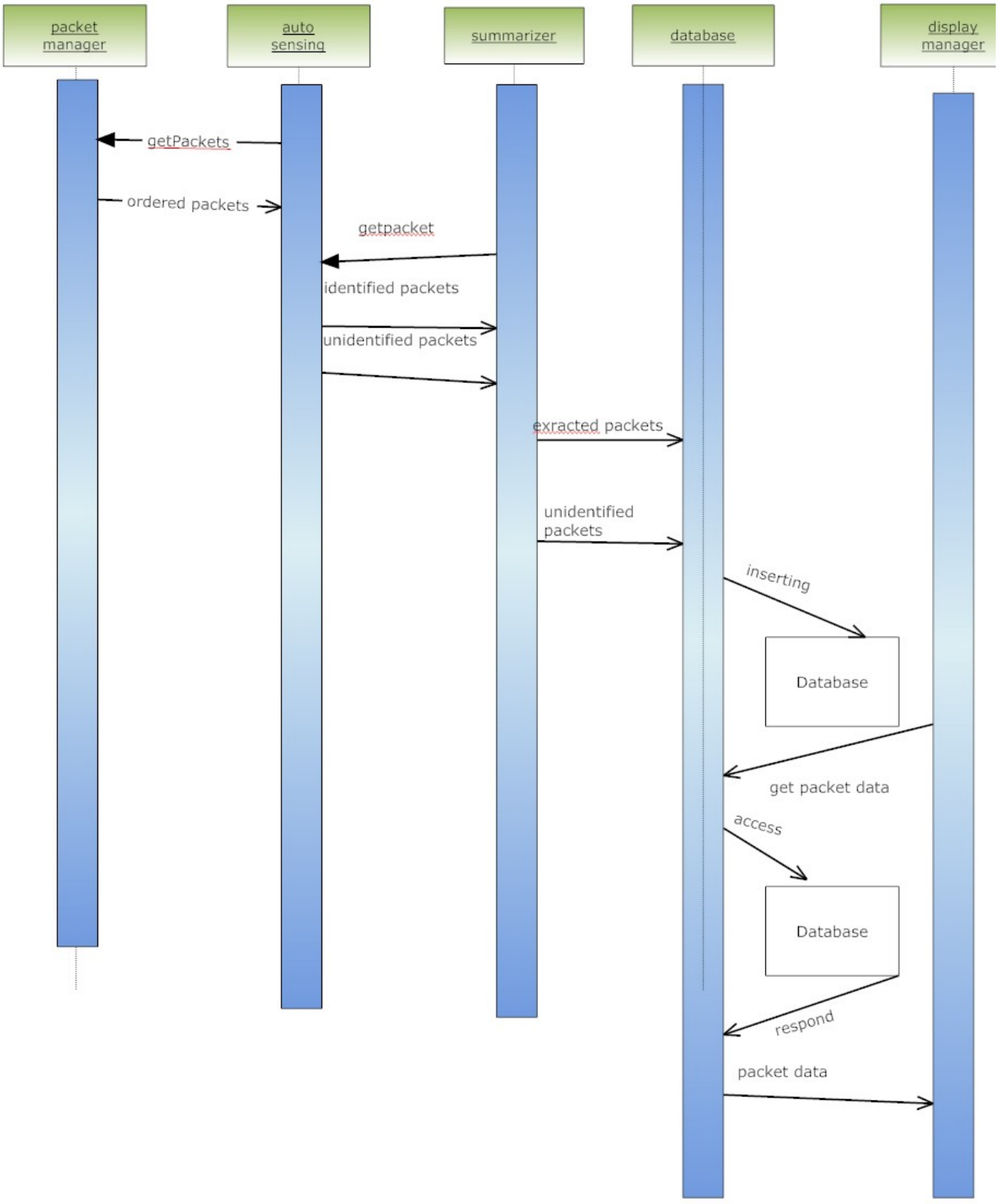
7.5.3.2. The Packets Not Analyzed



In this panel, there exist the unsuccessful results. This time, lines denote not completely analyzed files, but packets whose protocols could not be analyzed. There can only be shown source and destination IP addresses, and time in the format that, starting to capture time is set to zero and the non-analyzed packets' time difference from starting is kept for each packet.

The functionalities of the buttons at the bottom are similar to the other panel. When the user clicks on the 'Analyze' button, there appears a dialog box, and user enters for which protocols to analyze. 'Clear' cleans the current elements of the panel.

APPENDIX A: Sequential Diagram


packet manager
auto sensing
summarizer
database
display manager
getPackets
ordered packets
getpacket
identified packets
unidentified packets
exracted packets
unidentified packets
inserting
Database
get packet data
access
Database
respond
packet data

References:

-D.M.J. Tax and R.P.W. Duin. Data domain description by support vectors. In M. Verleysen,
editor, *Proceedings ESANN*, pages 251 – 256, Brussels, 1999. D Facto.

-V. Vapnik and A. Lerner. Pattern recognition using generalized portraits. *Avtomatika i*
Telemekhanika, 24:774 – 780, 1963.

-B. Sch¨olkopf, A. Smola, and K.-R. M¨uller. Kernel principal component analysis. In B. Sch¨olkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support*
Vector Learning. MIT Press, Cambridge, MA, 1999. 327 – 352.

http://axiom.anu.edu.au/~williams/papers/P126.pdf
http://wikipedia.org/
http://www.monkey.org/~dugsong/dsniff/
http://www.kismetwireless.net/
http://www.tcpdump.org/
http://www.wireshark.org/
http://www.wildpackets.com/
www.microsoft.com/
Wireshark&Ethereal Network Analyzer Toolkit by SYNGRESS

The GodFather inc.