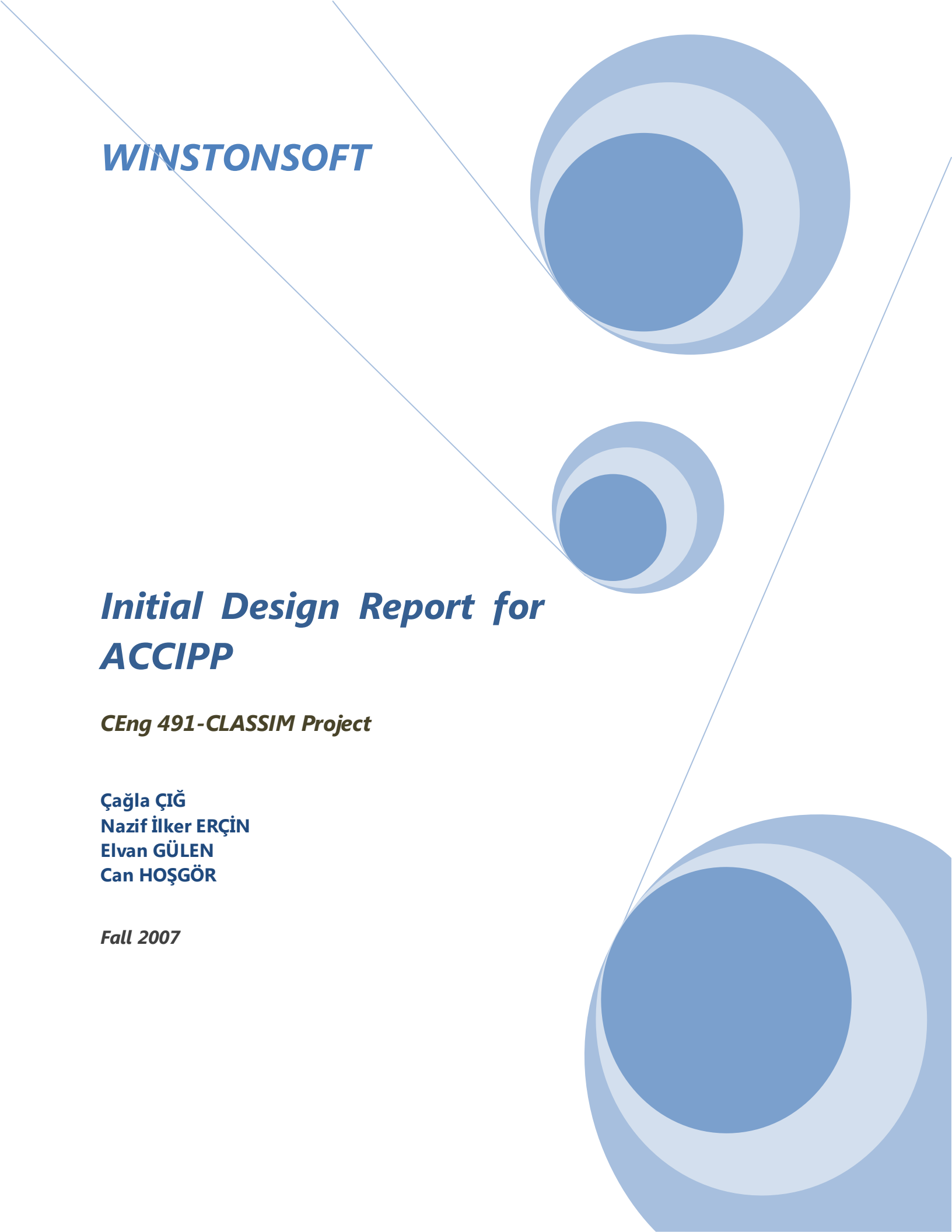


WINSTONSOFT

Initial Design Report for ACCIPP

CEng 491-CLASSIM Project

Çağla ÇIĞ
Nazif İlker ERÇİN
Elvan GÜLEN
Can HOŞGÖR

Fall 2007

Table of Contents

1. Introduction

In general terms, a packet sniffer (also known as a network analyzer or protocol analyzer) is a software or hardware that can monitor, state statistical information about and log traffic passing over a digital network or part of a network. As data is being transferred over the network in real time, the task of the sniffer is to capture ideally every packet and analyze its content in accordance with the appropriate RFC document or other specification. Most network analyzers allow port-specific tracking, i.e. they label protocol connections only by looking at port numbers. However, the need to overcome the limitations of traditional port-based protocol analysis arises since in today's networks an increasing ratio of the traffic (totaling roughly 5.6 million connections [1]) resist correct classification using TCP/UDP port numbers. The reason for that increase is the rising desire to evade security monitoring and policy enforcement.

1.1 Problem Definition

Relying on well-known port numbers such as 80 for HTTP may not always be possible since applications may use arbitrary ports. The main reasons for that choice of usage are benign reasons and malicious intent. Benign reasons result from lack of user privileges, obfuscation, multiple versions; adversarial applications such as Skype bypassing firewalls. On the other hand, malicious intent results from the desire to evade from security monitoring like IRC bot-nets using ports other than the ones they are assigned to (666x/TCP). The necessity to distinguish these arises from the prevalence of the problem and has the consequence of the need for a need approach for dynamic analysis using auto sensing mechanism that performs port independent network analysis.

The auto-identification/classification of common IP protocols software to be developed for Siemens is a new system. It will be used as an application for capturing packets over the network and identifying most of the widely used IP protocols such as HTTP, NNTP, POP3,

IMAP, SMTP, WTP, SIP, FTP etc. The project is designed to run on both Windows and common flavors of UNIX thus it is platform independent. ACCIPP should be equipped with a user-friendly with an intuitive, easy-to-operate GUI that will provide quick and comfortable operation.

1.2 Project Goals

The project is aimed to satisfy the following goals:

- Identify the protocols such as SMTP, NNTP, POP3, IMAP etc.
- Capture some popular file formats like *avi*, *wmv*, *jpg* etc. from the detected protocols.
- Log instant messenger conversations.
- Give output in an appropriate format.
- Monitor and supervise network traffic for performance and security and bandwidth usage.
- Gather and report network statistics and help troubleshoot network problems.
- Generate and view reports in tables and charts on network usage.
- Filter suspect content such as spam, and denial of service attacks from network traffic.
- Spy on other network users and collect sensitive information.
- Debug client-server communications.
- Show relevant information like IP, protocol, host or server name etc.
- Determine when the identified protocol is no longer available in the flow through the identified port.
- High performance and low-latency (Real-Time) detection capability.
- Recognize incomplete protocol sessions.

1.3 What Has Been Done So Far

- Obtained knowledge about the project:

  By the help of the meetings with Siemens and feedback from the project assistant, the project scope and goals were clarified.

- RFC documentations research has been done:

  Widely used mail protocols which are SMTP, POP3, IMAP, NNTP were fully examined. All specific arguments, keys, statuses, restrictions and commands were learned. Besides, a general idea was gained about how the server hosts start the protocol service and how the server and the client respond to the commands until the connection lost. Moreover, all the project members connected to mail services through the related protocols and tested how the protocol works.

  HTTP protocol was started to be researched but since it is a comprehensive protocol, a small part of it has been analyzed.

  These studied information about the protocol specifications are useful for the Auto-sensing mechanism of the project. Since Auto-Sensing Mechanism constitutes the main part of the project, this research is extremely important for the future of ACCIPP.

- Network sniffer research has been done:

  To observe how the Pcap files are handled by other sniffer programs, the project was attached importance to network sniffer research. Through the guidance of assistant and the representative of Classim, Wireshark Network Protocol Analyzer was set up and capturing TCP protocols was managed to investigate the content of the packets. Also SmartSniff was examined so that network traffic was observed the with another program.

- Programs with similar features were analyzed:

  TCPxtract and EtherPeg were analyzed. TCPxtract extracts files from network traffic based on file signatures. This tool may help for extracting .jpeg, .doc, .avi or etc from the packets. EtherPeg is a program that shows all the JPEG pictures going through the network traffic.

- Some project related papers and materials were read:

  Until now, a huge amount of research has been conducted on the following topics:

  I. Clustering Classification,
  II. No Port Network Protocols Detection,
  III. Feature Extraction for Integrated Pattern Recognition Systems,
  IV. Network-Based Application Recognition and Distributed Network-Based Application --Recognition
  V. Port Independent Protocol Identification.

  During that research the following materials have been examined:

  I. http://documents.wolfram.com/applications/neuralnetworks/NeuralNetworkTheory/2.1.3.html
  II. http://www.ucl.ac.uk/oncology/MicroCore/HTML_resource/Clus_and_Class_popup.htm.
  III. "Clustering and Classification Methods for Gene Expression Data Analysis", Garrett-Mayer E., Parmigiani G., 2004.
  IV. No Port Network Protocols Detection Presentation by Sevgi Yaşar
  V. Feature Extraction for Integrated Pattern Recognition Systems by X. Wang and K. K. Paliwal
  VI. Network-Based Application Recognition and Distributed Network-Based Application Recognition by CISCO

VII. Dreger H., Feldmann A., et.al, "Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection"

VIII. http://cs.northwestern.edu/~ychen/classes/cs450-s07/lectures/pia.ppt

IX. http://www.icir.org/robin/papers/usenix06.pdf

- Prototype related operations were researched and studied:

  The prototype must include capturing and reading packets, then filtering them according to some parameters and reordering them. Other than this, a simple Auto-sensing Mechanism should be implemented. For these requests, necessary research was conducted.

- Necessary libraries were researched:

  libpcap was analyzed. This library is useful for capturing packets from the network traffic and reading them.

- MSN protocol was examined using reverse engineering:

  MSN protocol is not a proprietary protocol, so that the research couldn't be done on the RFC documents. However, from different resources MSN Messenger Protocol was examined and still is. Most of the status and error commands were studied.

- Worked on prototype design and implementation:

  First of all, a prototype GUI was designed. It contains nearly all important operations that the user can have the control of the program. Also, an initial prototype of the project was designed and started to be developed. For capturing network packets, a Pcap based tool namely WinPcap was used. This tool is compatible with the unix versions of libpcap, so that the project can have platform independency. With the

help of WinPcap tool, capturing module was implemented, and a primitive protocol recognizer for SMTP was designed.

1.4 Future Work

The design phase of the project has almost reached its maturity. Project requirements and scope were defined clearly, so most of the data structures, classes and data interactions within the program have already been determined. In the future, these structure schemas are going to be studied and improved constantly. The existing class hierarchy will be preserved with possible improvements, as well as new classes and data structures might be designed in the future as more and more concepts become understood.

The exact structure and functionality of the Auto-Sensing module needs conducting some further research. The project assistant recommended some academic papers and online materials about pattern recognition algorithms (especially the Support Vector Machines and Hidden Markov Model). These documents are going to be deeply studied so that each member will have an understanding of related subjects since the modules that require pattern recognition are going to be designed and implemented by all team members.

Currently all mail-related protocols that the program is to support have been studied. Recognizers of these protocols are almost finished and incorporated into the prototype. In the future, more protocols namely HTTP, XMPP(Jabber), YMSG, MSN and SIP are going to be studied, and recognizers about these protocols are going to be implemented.

A preliminary user interface for the program was designed. First of all, this user interface module will be added to the prototype. After feedback about functionality and appearance of this user interface is received from project supervisors and testers; the user interface will

go under some modifications so that the user interface module used in the prototype will gradually evolve into the user interface module that will be used in the final product.

2. Design Constraints

Project constraints can be grouped like the following:

2.1 Time Constraints

Since senior project design is a two semester course, the project will have to be finished by the end of May 2008. All design, implementation and testing must strictly meet this deadline and complete in this 7 month period. Besides, there is going to be a prototype demo that will be released by January 2008. There is roughly one and an half months to finish the ACCIPP design and prepare the demo.

2.2 Language Constraints

For performance reasons, the language for the project is decided to be C++. Platform independency and code portability is an implementation constraint, thus all C++ code for this project will conform to ISO C++ standard. Development environment will be Visual C++ for Windows port, and a suitable GCC based environment for Unix/Linux ports. Qt library is planned to be used for the development of OS independent user interface system.

2.3 Data Constraints

A fair amount of primary storage space is required to hold various data structures used for analyzing data flow over the network. If the user chooses to save some data for later analysis, or the data cannot be processed in real time, the need for secondary storage space arises.

2.4 Performance Constraints

ACCIPP will be exposed to high network traffic while dealing with real-time incoming packets. Under these circumstances, the number of packets arriving per unit time will be quite large and average processing time given to a packet should be kept minimal. Since ACCIPP intends to recognize a large number of protocols, the user should select only a subset of these in order to avoid starvation/packet drops. In a typical case where the number of packets per second is around 100 and presuming that the user might be running other applications, a maximum of 7-8 ms can be spent on each packet. Under such heavy load, ACCIPP should rely on predefined rule-based recognition engine rather than the relatively slow learning/training method.

2.5 User Interface Constraints

ACCIPP is not a user interface oriented application. Main work of the project is system programming. However, the user interface still is important for being understood and being used easily by the user. So the interface must be kept simple and easy to use. Names of *menu*s and other *gui* elements will be easy to understand and straightforward. Accessibility features must be taken into consideration for handicapped users.

3. Project Requirements

Understanding the needs of the project, the project requirements should be specified. During the determination of requirements analysis, the steps taken are as follows :

3.1 Functional Requirements

Below, the functional requirements for ACCIPP are explained briefly.

3.1.1 Capturing Packets

The input will be captured from a network device or taken as already existing Pcap files.

3.1.2 Preprocessing

The captured packets may need reordering and/or defragmentation. The preprocessing mechanism handles these operations.

3.1.3 Filtering

The user of the system may not want to receive irrelevant data that s/he is not working on. Thus the filtering mechanism is employed to filter the packets which are of concern. Filters can be defined by several identities of connections such as IP addresses or protocol data.

3.1.4 Auto-Sensing

The system is expected to identify the packets without using port information. Auto-Sensing mechanism takes action in this identification process using some *Artificial Intelligence* algorithms.

3.1.5 Processing Identified Connections

The proper output for the analyzed protocol of the connection are sent to output mechanism.

3.1.6 Output Mechanism

The data that is received from the system, will be displayed as reports or user interface summaries. If asked, more detailed information about the connection can be given as output.

3.2 Non-Functional Requirements

In this section various non-functional requirements such as usability, portability, reliability and documentation will be mentioned.

3.2.1 Usability

The program has to be easily adaptable for novice users, and powerful enough for experienced users. User interface elements such as menu items and command buttons have to be as clear and self-explanatory as possible. They should provide tooltips where applicable. The resulting graphs should allow the user to obtain rapidly an overall grasp of the material presented.

3.2.2 Portability

The software package is designed to be a cross platform product, therefore it should not rely on machine and/or OS dependant functionality such as byte ordering and non-standardized system calls. Consequently the program will be able to compile on different computer systems without being altered.

3.2.3 Reliability

The software package is planned to be used in large and corporate networks, thus it is a critical requirement that the software functions consistently under such circumstances.

3.2.4 Documentation

User documentation includes *online help* and user manual for the product. A hardcopy of the *user's manual* will also be provided with the software package.

3.3 Software Requirements

In this section, the external software packages ACCIPP depends on will be presented.

3.3.1 Operating System

ACCIPP shall function on Windows versions starting from Windows 2000, and major Linux distributions like Debian, RedHat etc.

3.3.2 External Packages

ACCIPP requires the presence of an external libpcap compatible packet sniffer and an adequate network adapter in cases where real-time processing is deemed necessary.
In addition to that, Qt library must be installed in order to have user interface functionality.

3.4 Hardware Requirements

In this section, hardware requirements for the software project are presented.

3.4.1 Minimum Hardware

In order to have basic functionality, a system with 256 MB Memory, Pentium III class CPU, 10 MB Hard disk space is required.

3.4.2 Recommended Hardware

To be able to make full use of the auto-sensing facility and store statistical information in the database backend, a system with at least 1 GB Memory, 2.5 Ghz Pentium IV class or higher CPU, 4 GB Hard disk space is required.

4. User Interface Design

The user interface lets the user see the connections on the system. When the program is first opened, the connection list is empty. After that, user selects a Pcap file to process offline a network device to process real-time. Then the user selects Start Capture from the File menu and the program begins its work. As soon as new packets arrive, the program populates the connection list. Until the protocol is fully recognized, the appropriate row in the connection list is updated with the resolved protocol match values. During the process, when the match percentage becomes greater than a predefined threshold value then the protocol name and match percentage fields are filled with appropriate values. The program continues processing until the connection is closed, thus this value may change several times during the process. In case the program is unable to match the connection data with any of the protocol patterns available, it shows Unknown as the protocol name. For each connection the list pane shows the matched protocol name, match percentage value, IP address of the local computer, local port number, IP address of the destination computer, and remote port number, start and end times of the connection. For connections that are not closed yet, the end time field is empty. Below is a sample screenshot of user interface mentioned above:

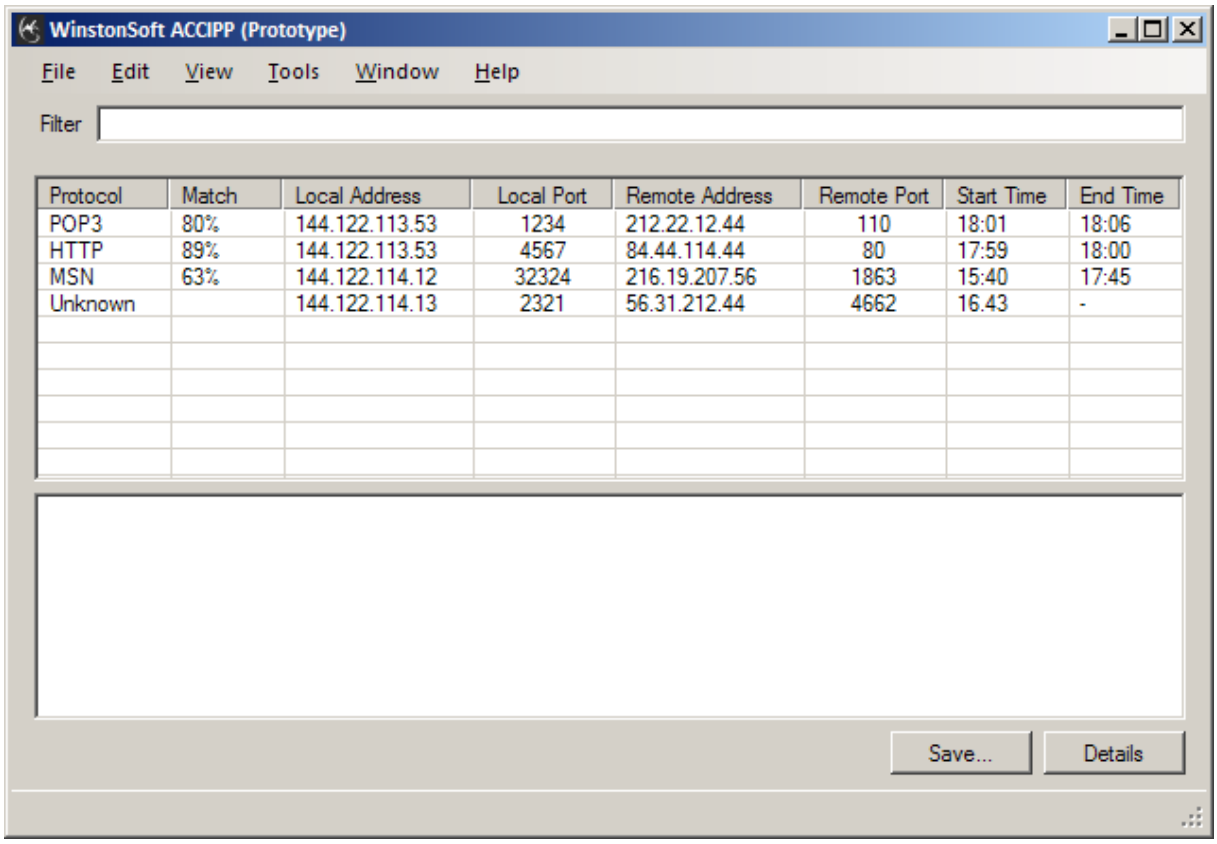


Figure : Main Window of ACCIPP Prototype UI

Since a connection may match more than one protocol pattern, only the most matching protocol is shown on the connection list. However, the user can click on a row to see its match values with other protocols. If the protocol recognition is not complete yet (i.e. End Time field is blank), the Short Summary Pane below the connection list shows only the match percentage values with protocol patterns. Also, Save and Details buttons below the short summary pane are disabled. However, when the protocol recognition is finished, these buttons become enabled and the information of the identified protocol ("The protocol cannot be identified!" or "Identified Connection: Protocol Name") is shown additional to the match percentage values. A related user interface screenshot is shown below:

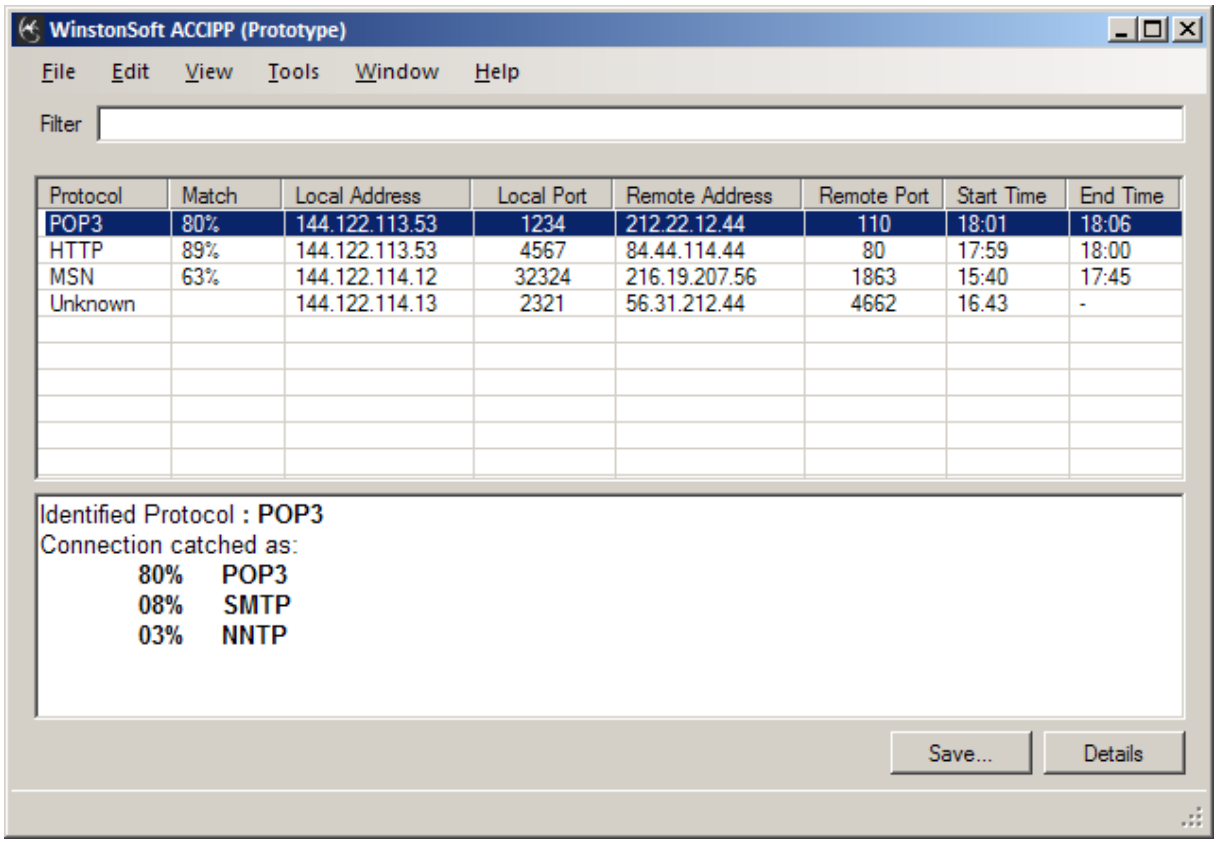


Figure : ACCIPP Prototype UI(Short Summary Pane)

As you can see, the first connection is identified as POP3 and the buttons are enabled. Additionally the final match percentages are shown in the Short Summary Pane. The save button automatically stores the detailed information into the database. If the user wants to see this detailed information about the selected connection, he/she can click the Details button. After that, the Long Summary Window pops up at right hand side of the main window.

Clicking another connection from the connection pane does not affect the Long Summary Window. That means, several Long Summary Windows can be displayed simultaneously.

An example screenshot of the user interface after clicking the details button is shown below:

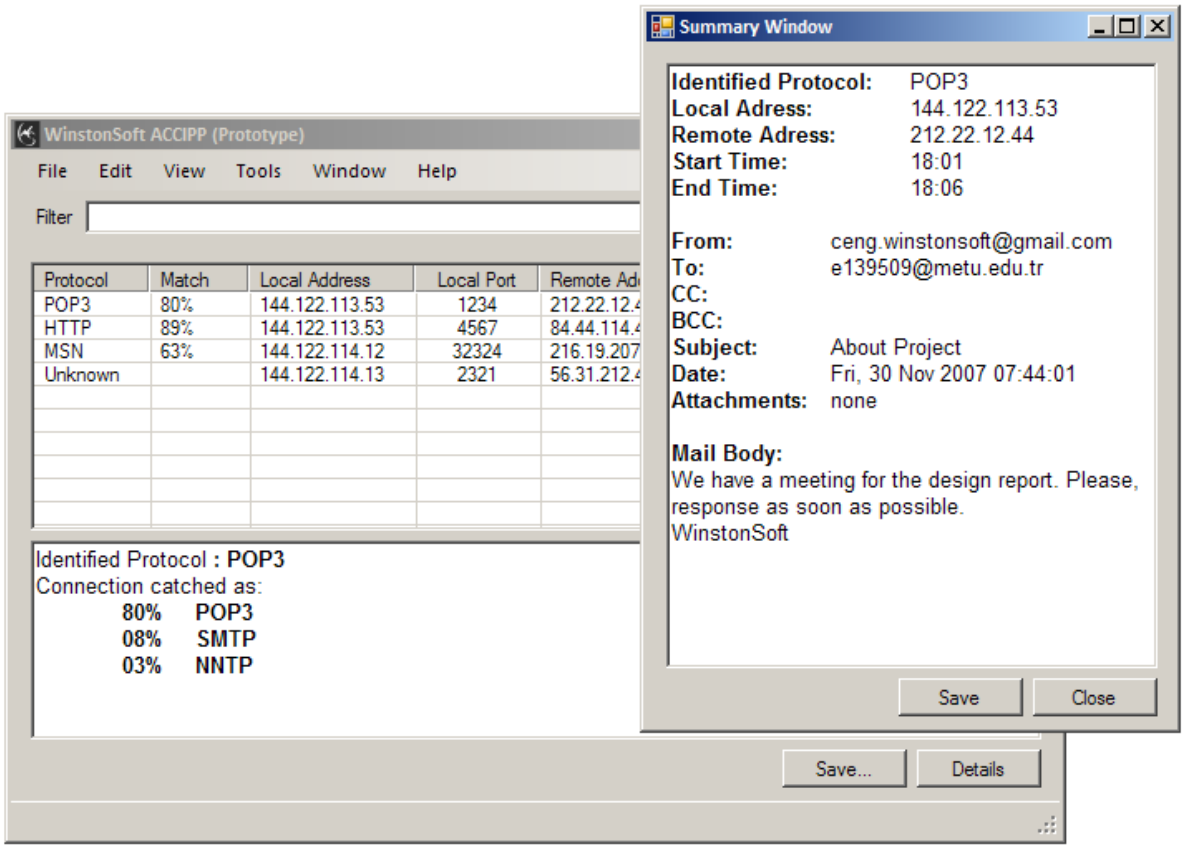


Figure : ACCIPP Prototype UI(Long Summary Popup Window)

The save button on the Summary Window is used for saving the long summary to the database. By using the close button, the Summary Window can be dismissed.

Since the connection pane fills with a huge amount of information over time, the user may want to filter information that is displayed in the connection pane. For example, the user may want to see connections from a specific IP range, or connections that use a specific protocol, or connections that are opened over a specified time interval. The filter field in the main window allows the user to enter a combination of these filters. For example, if the user types “pop3” into the filter, only the connections that have pop3 as the highest match percentage are displayed on the connection pane. The help menu includes documentation for the exact syntax of expressions that can be entered into the filter field, along with other help topics.

Some commands do not have a corresponding button shown on the main window. Instead, these commands are available through the menu bar of the program. The menu commands that are not enabled at the moment will be shown in grayed state. A short informative text is displayed on the status bar when the mouse is hovered over a menu item.

The exact menu commands are subject to change. However, some commands will certainly be available in the final product. These commands are:

- File: Open Pcap File, Open Network Device, Start Capture, Stop Capture, Save as Pcap, Close, Quit.
- Edit: Copy to Clipboard, Clear All, Preferences.
- View: Toolbar, Status Bar, Summary Pane.
- Tools: Database Query, Statistics.
- Window: Tile, Cascade, Arrange Icons.
- Help: Help Contents, About Program.

Detailed information about what these commands do can be found in the use-case diagrams and the use case scenarios.

5. Database Design

ACCIPP comes with a database that the end user may use to get detailed statistical information about both the detected and unknown connections. Also the end user will be able to preview the details of the previously added connections in summary format. To do this, a database is designed where protocols are mapped to related entities as shown below:

Web_Page: HTTP

Instant_Messaging: MSN, YMSG, JABBER, SIP

News: NNTP

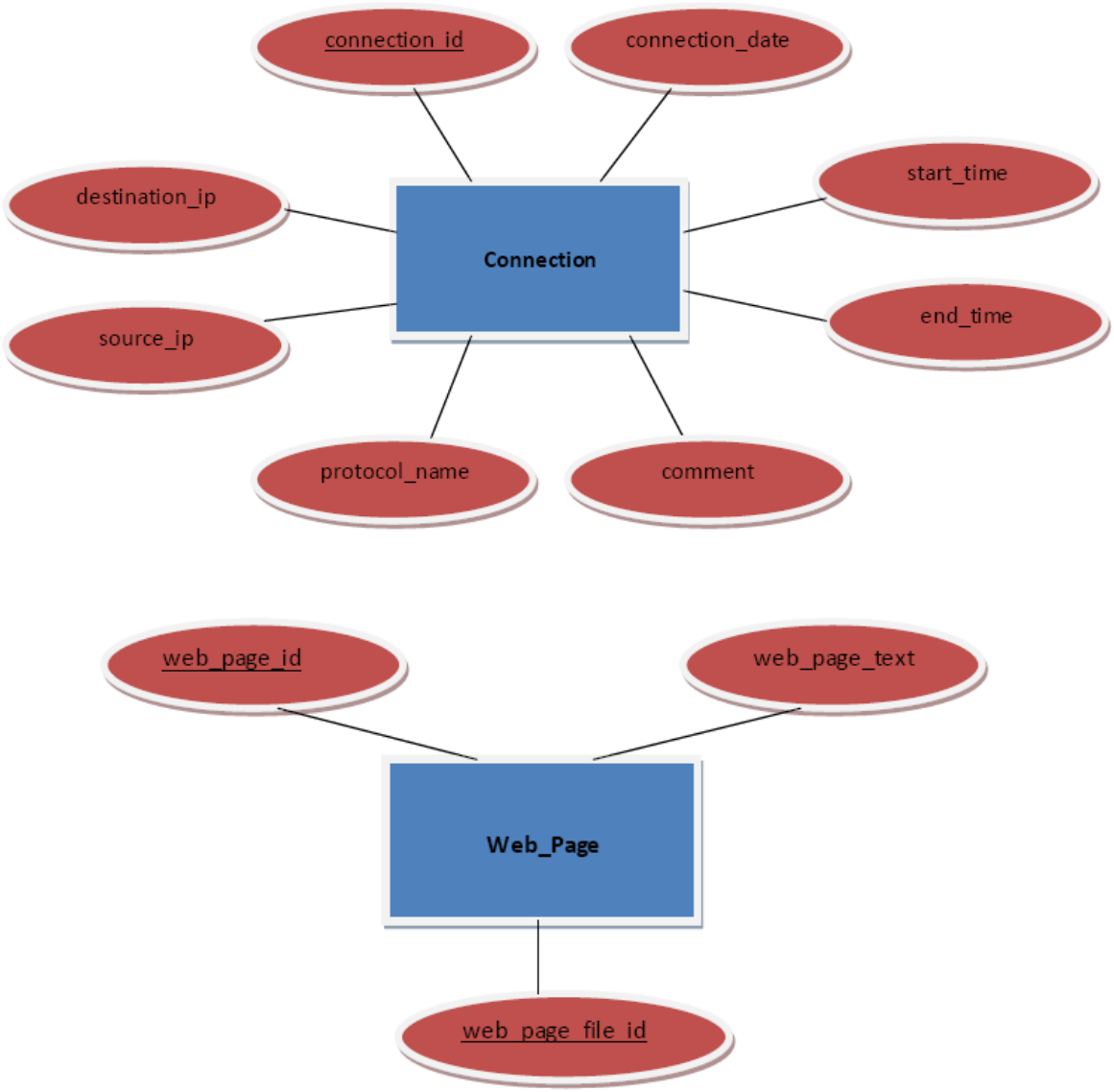
Email: IMAP, POP3, SMTP

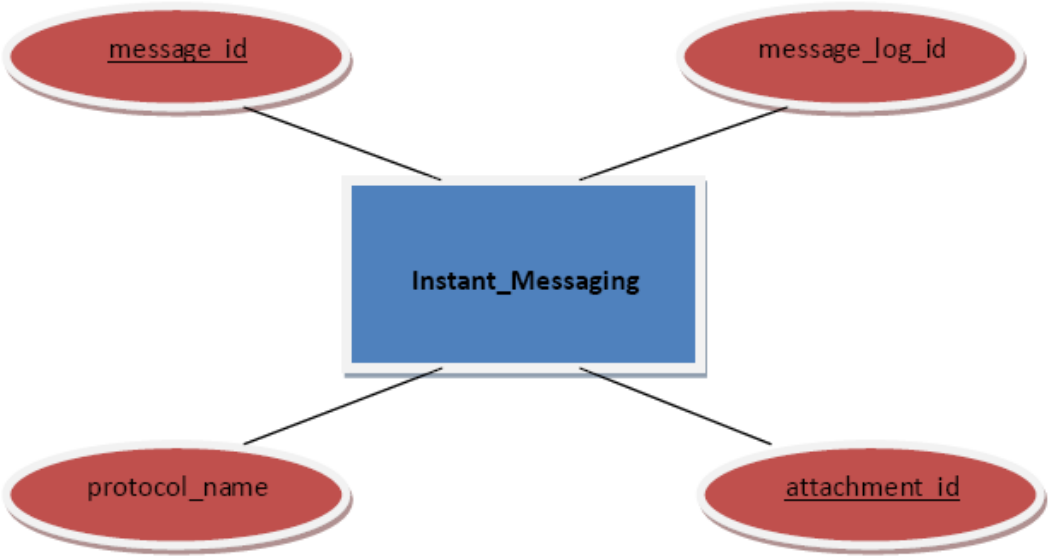
Unknown: all protocols that have not been detected yet.

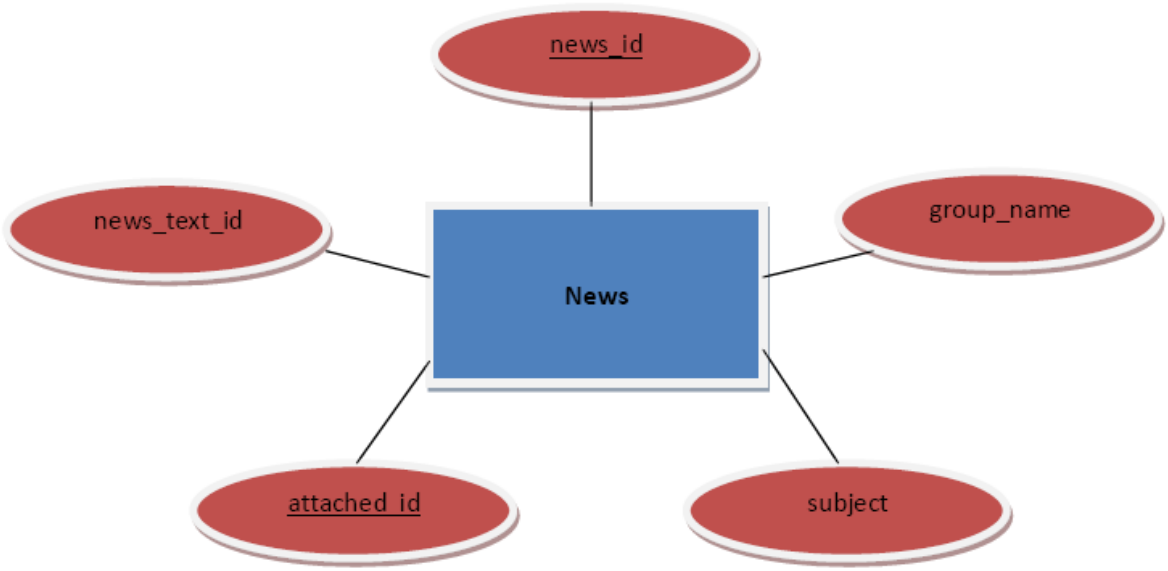
In addition to this, also included in the database are the Connection entity where general properties of all connections are stored and the File entity where the file-related properties are stored.

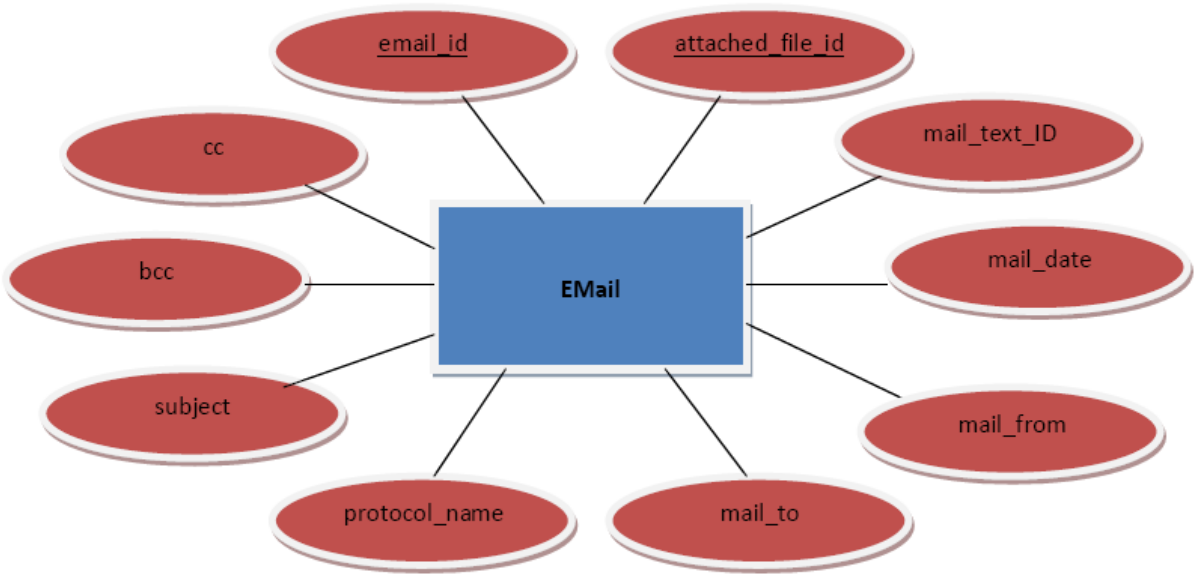
5.1 Entity-Relationship Diagrams

ER Diagrams for ACCIPP Database

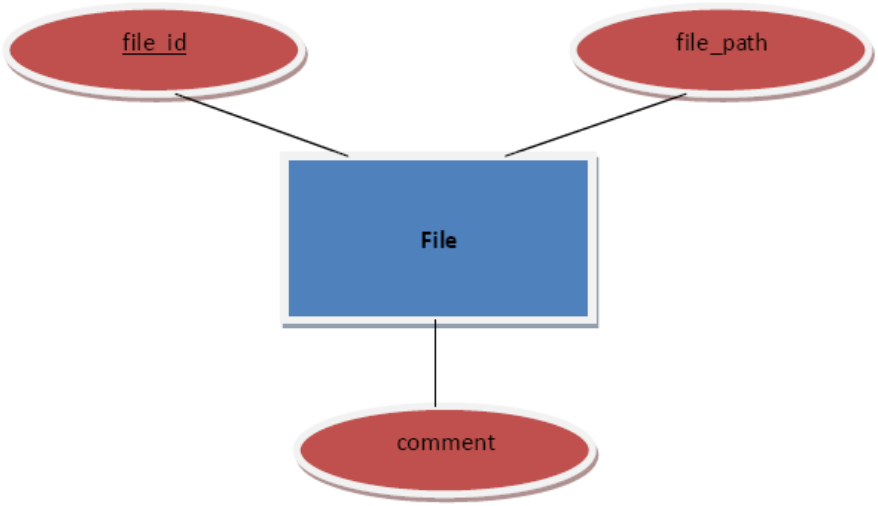



message_id
message_log_id
Instant_Messaging
protocol_name
attachment_id

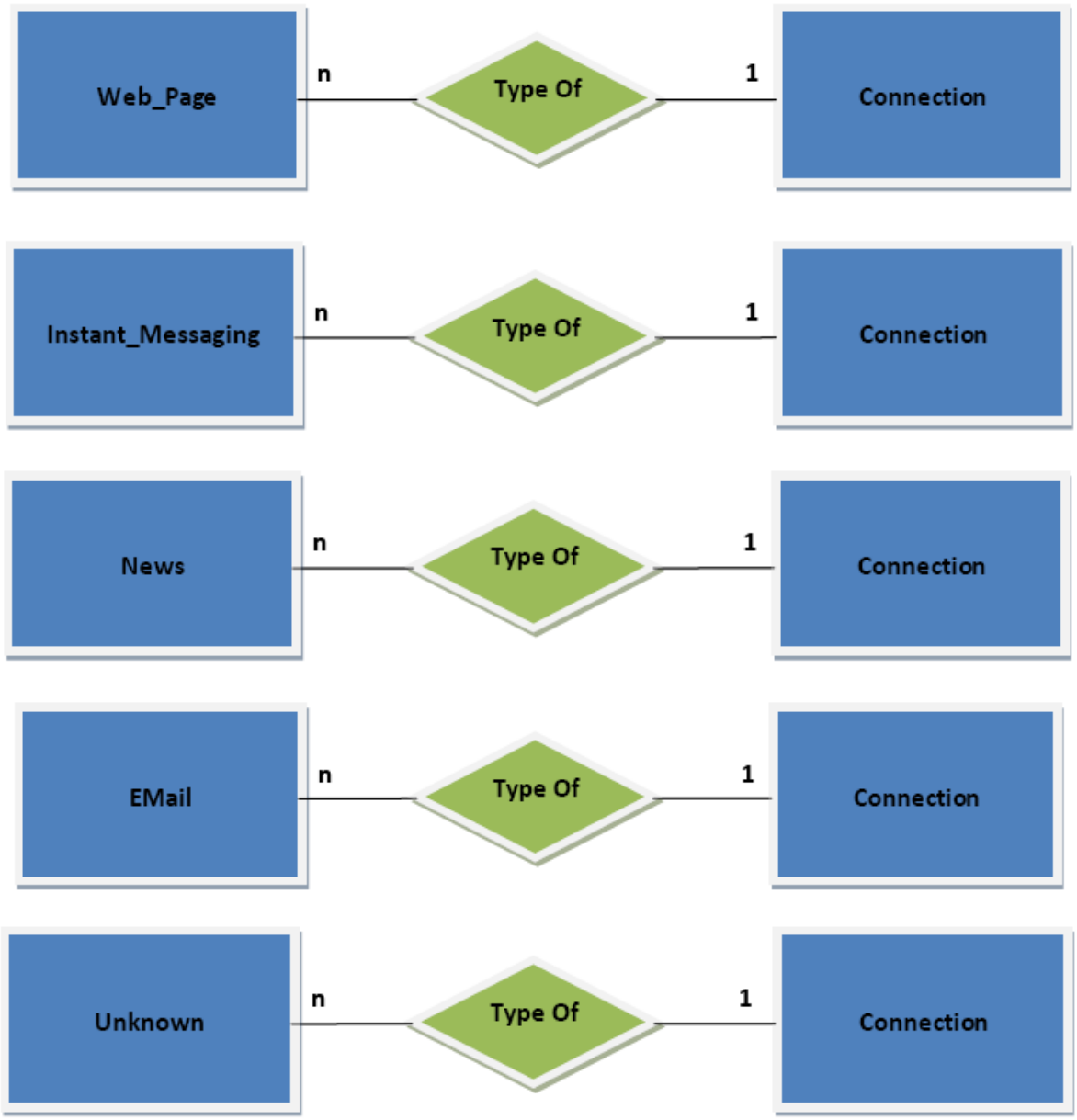

news_id
news_text_id
group_name
News
attached_id
subject

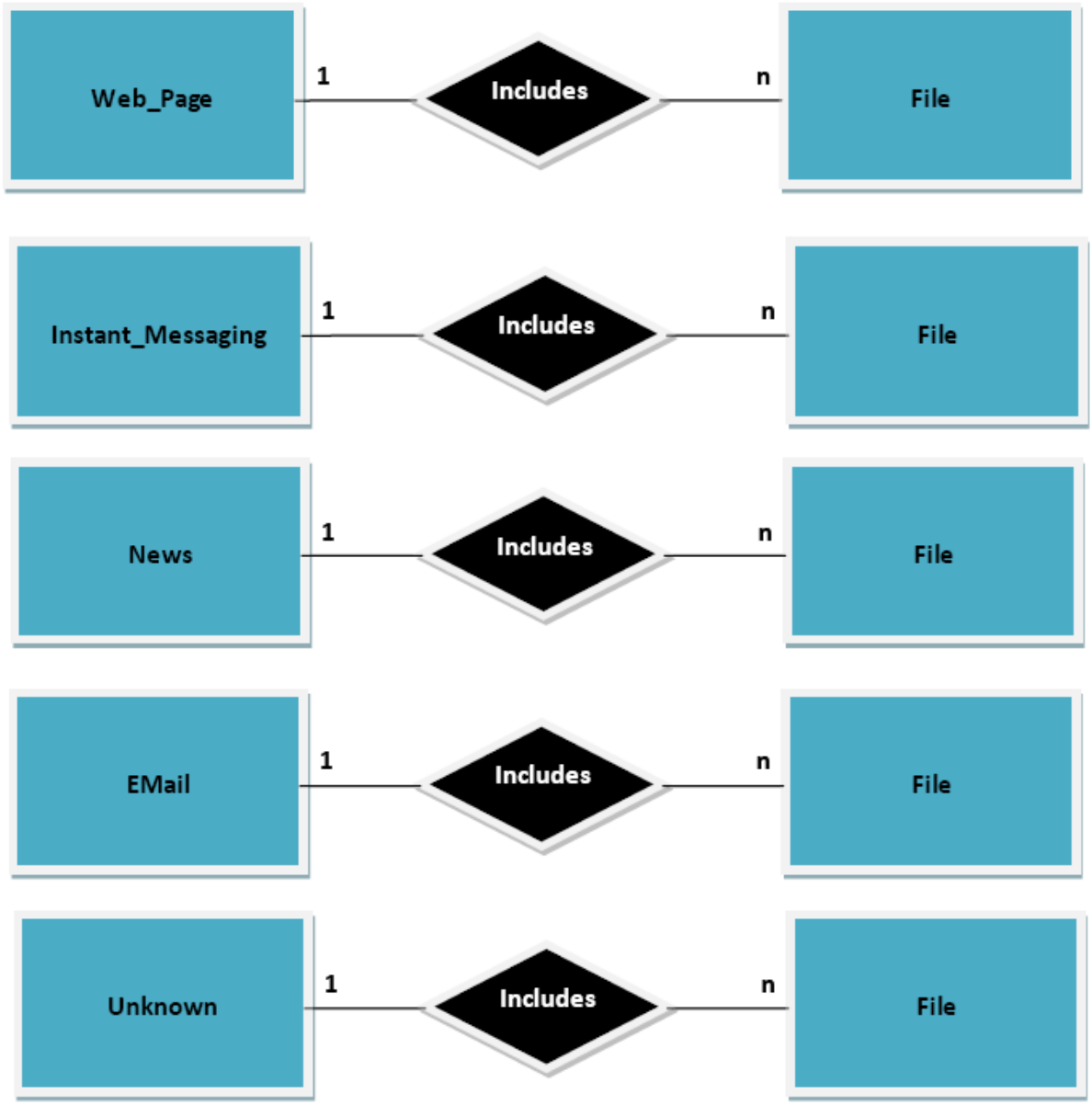

email_id
attached_file_id
cc
mail_text_ID
bcc
EMail
mail_date
subject
mail_from
protocol_name
mail_to


unknown_id
Unknown
unknown_pcap_path

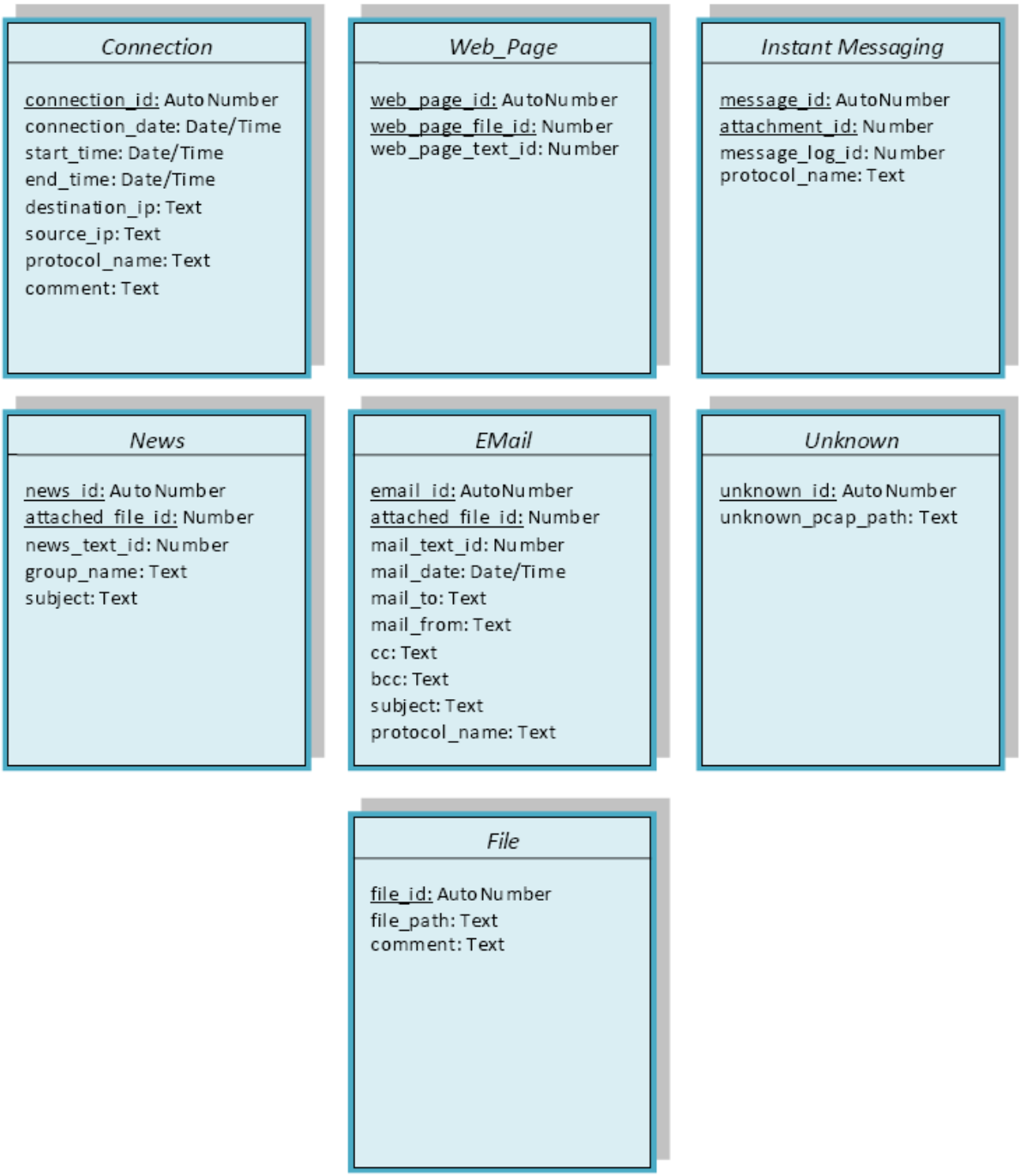

file_id
file_path
File
comment

Relations


Web_Page
n
Type Of
1
Connection
Instant_Messaging
n
Type Of
1
Connection
News
n
Type Of
1
Connection
EMail
n
Type Of
1
Connection
Unknown
n
Type Of
1
Connection


Web_Page
1
Includes
n
File
Instant_Messaging
1
Includes
n
File
News
1
Includes
n
File
EMail
1
Includes
n
File
Unknown
1
Includes
n
File

Entity Sets


Connection
connection_id: AutoNumber
connection_date: Date/Time
start_time: Date/Time
end_time: Date/Time
destination_ip: Text
source_ip: Text
protocol_name: Text
comment: Text
Web_Page
web_page_id: AutoNumber
web_page_file_id: Number
web_page_text_id: Number
Instant Messaging
message_id: AutoNumber
attachment_id: Number
message_log_id: Number
protocol_name: Text
News
news_id: AutoNumber
attached_file_id: Number
news_text_id: Number
group_name: Text
subject: Text
EMail
email_id: AutoNumber
attached_file_id: Number
mail_text_id: Number
mail_date: Date/Time
mail_to: Text
mail_from: Text
cc: Text
bcc: Text
subject: Text
protocol_name: Text
Unknown
unknown_id: AutoNumber
unknown_pcap_path: Text
File
file_id: AutoNumber
file_path: Text
comment: Text

5.2 Data Descriptions

The attributes and the related data types for each table constituting the database are as follows:

The attributes containing a key icon indicates the corresponding attribute or attribute group being the primary key of the related table. In the design, the primary key of the Connection table is set to AutoNumber data type, because as new connections are added to the database, they are assigned a number and this number is equal to a primary key value of one of the five tables each corresponding to a group of similar protocols. In addition to this, the primary key of the File table is also of AutoNumber data type since each row stands for a distinct file. These files are pointed to by the Web_Page, Instant_Messaging, News, EMail and Unknown tables by means of foreign keys to store the files included in these five connection types. The following are the figures formed using MS Access 2007® only to visually illustrate the prototype database. However, the database system of the project is going to be implemented using MySQL (see Section 4.1.4).

Connection

Field Name	Data Type	Description
connection_id	AutoNumber	
connection_date	Date/Time	
start_time	Date/Time	
end_time	Date/Time	
destination_ip	Text	
source_ip	Text	
protocol_name	Text	
comment	Text	

Web_Page

Field Name	Data Type	Description
web_page_id	Number	
web_page_file_id	Number	
web_page_text_id	Number	

Instant_Messaging

Field Name	Data Type	Description
message_id	Number	
attachment_id	Number	
message_log_id	Number	
protocol_name	Text	

News

Field Name	Data Type	Description
news_id	Number	
attached_file_id	Number	
news_text_id	Number	
group_name	Text	
subject	Text	

EMail

Field Name	Data Type	Description
email_id	Number	
attached_file_id	Number	
mail_text_id	Number	
mail_date	Date/Time	
mail_to	Text	
mail_from	Text	
cc	Text	
bcc	Text	
subject	Text	
protocol_name	Text	

Unknown

Field Name	Data Type	Description
unknown_id	Number	
unknown_pcap_path	Text	

File

Field Name	Data Type	Description
file_id	AutoNumber	
file_path	Text	
comment	Text	

5.3 Entity Descriptions

Connection:

Connection entity is formed to store the necessary information to define a connection in a general manner. No matter to which type of protocol it belongs, all types of connections are stored in this table. The description for each attribute of the Connection entity can be found below.

connection_id: This attribute is used to define each connection uniquely. Although connection id of the connection uniquely identifies each connection, system assigns an integer valued identifier to each connection to manage them easily.

connection_date: It is the date that the connection is grasped from the network traffic. The value of the date attribute will be gathered from the system date and stored in Date/Time format in the database.

start_time: It is the time that the program starts to work on the packets of the related connection. The value of this attribute will be taken from the Pcap file header if the program is working offline and from the system clock if the program is working online. The value is stored in Date/Time format in the database.

end_time: It is the time that the program gets the last packet of the related connection either offline or online. If the program is working online, the value of this attribute will be gathered from the Pcap file header and from the system clock if the program is working online. The value is stored in Date/Time format in the database.

destination_ip: This attribute contains the IP number of the destination network device that the packets of the corresponding connection arrived. It is stored in the database in Text format.

source_ip: This attribute contains the IP number of the source network device where the packets of the corresponding connection are sent from. It is stored in the database in Text format.

protocol_name: This attribute stores the name of the protocol that is detected by the program. It is used to define to which protocol class(Web_Page, Instant_Messaging, News, Email) table the related connection will be added to. It is stored in the database in Text format.

comment: This attribute is used for miscellaneous information about the connection. It is in Text format.

Web_Page:

If the detected connection is of type HTTP, then the connection will be added to this table with only three attributes.

web_page_id: This attribute defines each Web_Page connection uniquely, therefore it is the primary key of this entity. It is stored in the database in Number format. This is also a foreign key to the Connections entity through the *connection_id* attribute.

web_page_file_id: This attribute defines the files attached to the corresponding Web_Page. This is also a primary key as there may be more than one file attached to the same Web_Page entry. The contents of this file can be accessed through the *file_id* of the Files entity. This attribute is stored in Number format.

web_page_text_id: This attribute defines the file that contains the textual content of the related Web_Page. The file itself can be accessed through the related *file_id* of the Files entity. This attribute is stored in Number format.

Instant_Messaging:

If the detected connection is of the types MSN, YMSG, JABBER, SIP, then the connection will be added to this table with four attributes.

message_id: This attribute defines each Instant_Messaging entry uniquely, therefore it is a primary key of this entity. It is stored in the database in Number format. This is also a foreign key to the Connections entity through the *connection_id* attribute.

attachment_id: This attribute defines the files sent using the related instant messaging protocol. This is also a primary key as there may be more than one file sent with the current connection. The contents of this file can be accessed through the *file_id* of the Files entity. This attribute is stored in Number format.

message_log_id: This attribute defines the file that contains the textual log of the related Instant_Messaging entry. The file itself can be accessed through the related *file_id* of the Files entity. This attribute is stored in Number format.

protocol_name: Since there are multiple protocols related to this entity (MSN, YMSG, JABBER, SIP), this attribute defines the protocol name that the current connection is using. This attribute is stored in Text format.

News:

If the detected connection is of type NNTP, then the connection will be added to this table with five attributes.

news_id: This attribute defines each News entry uniquely, therefore it is a primary key of this entity. It is stored in the database in Number format. This is also a foreign key to the Connections entity through the *connection_id* attribute.

attached_file_id: This attribute defines the files attached to the corresponding News post. This is also a primary key as there may be more than one file attached to the same News entry. The contents of this file can be accessed through the *file_id* of the Files entity. This attribute is stored in Number format.

news_text_id: This attribute defines the file that contains the textual content of the related News post. The file itself can be accessed through the related *file_id* of the Files entity. This attribute is stored in Number format.

group_name: This attribute is used to define the news group name that the news message is sent to. The value of this attribute is stored in the database in Text format.

subject: This attribute is used to define the subject of the post which briefly describes what the post is about. It is stored in the database in Text format.

EMail:

If the detected connection is of type IMAP, POP3, SMTP, then the connection will be added to this table with ten attributes.

email_id: This attribute defines each EMail entry uniquely, therefore it is a primary key of this entity. It is stored in the database in Number format. This is also a foreign key to the Connections entity through the *connection_id* attribute.

attached_file_id: This attribute defines the files attached to the corresponding EMail. This is also a primary key as there may be more than one file attached to the same Email entry. The contents of this file can be accessed through the *file_id* of the Files entity. This attribute is stored in Number format.

mail_text_id: This attribute defines the file that contains the textual content of the related EMail. The file itself can be accessed through the related *file_id* of the Files entity. This attribute is stored in Number format.

mail_date: This attribute defines the date when the related Email is sent. It is saved in Date/Time format in the database.

mail_to: This attribute defines the e-mail address where the corresponding EMail sent to. It is saved in the database in Text format.

mail_from: This attribute defines the e-mail address where the corresponding EMail received from. It is saved in the database in Text format.

cc: This attribute defines the Carbon Copy receivers of the related EMail entry. It is saved in the database in Text format.

bcc: This attribute defines the Blind Carbon Copy receivers of the related EMail entry. It is saved in the database in Text format.

subject: This attribute is used to define the subject of the EMail message which briefly describes what the mail is about. It is stored in the database in Text format.

protocol_name: Since there are multiple protocols related to this entity (IMAP, POP3, SMTP), this attribute defines the protocol name that the current connection is using. This attribute is stored in Text format.

Unknown:

If the program cannot identify the protocol of a connection, it saves a Pcap file related to that connection in the disk. Any connection that cannot be identified, is added to this table in the format below.

unknown_id: This attribute defines each Unknown connection entry uniquely, therefore it is a primary key of this entity. It is stored in the database in Number format. This is also a foreign key to the Connections entity through the *connection_id* attribute.

unknown_Pcap_path: The program saves a Pcap file of the unidentified connection in the disk for later offline/online operation. The path of this Pcap file is stored in the database under this attribute. It is stored in the database in Text format.

File:

The files that are saved using the program is added to this table using the following format. Instead of embedding the files to the database directly, it is preferred to put the file_paths in the database so that the database does not itself allocate a large amount of disk space. Additionally, in cases such as a lately classification of an unknown connection does not enforce the database to move the related files from one place to another, speeding up the process.

file_id: This attribute uniquely identifies a file that is saved using the program, therefore it is a primary key for this entity. It is stored in the database in an AutoNumber format.

file_path: This attribute shows the path of the related file, which can be used to access the file on the disk. It is stored in the database in Text format.

comment: This attribute includes additional information about the corresponding file such as the codec information, names of the programs that the file can be opened with etc.

5.4 Creating ACCIPP Database

```
CREATE DATABASE `accipp` /*!40100 DEFAULT CHARACTER SET latin1 */;

/*CONNECTION*/
DROP TABLE IF EXISTS `accipp`.`connection`;
CREATE TABLE  `accipp`.`connection` (
  `connection_id` int(10) unsigned NOT NULL auto_increment,
  `connection_date` datetime NOT NULL,
  `start_time` datetime NOT NULL,
  `end_time` datetime NOT NULL,
  `source_ip` varchar(20) NOT NULL,
  `destination_ip` varchar(20) NOT NULL,
  `protocol_name` varchar(10) NOT NULL,
  `comment` varchar(50) NOT NULL,
  PRIMARY KEY  USING BTREE (`connection_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*WEB_PAGE*/
DROP TABLE IF EXISTS `accipp`.`web_page`;
CREATE TABLE  `accipp`.`web_page` (
  `web_page_id` int(10) unsigned NOT NULL auto_increment,
  `web_page_file_id` int(10) unsigned NOT NULL,
  `web_page_text_id` int(10) unsigned NOT NULL,
  PRIMARY KEY  USING BTREE (`web_page_id`,`web_page_file_id`),
  KEY `web_page_file_id` (`web_page_file_id`),
  KEY `web_page_text_id` (`web_page_text_id`),
  CONSTRAINT  `web_page_text_id`  FOREIGN  KEY  (`web_page_text_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT  `web_page_file_id`  FOREIGN  KEY  (`web_page_file_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `web_page_id` FOREIGN KEY (`web_page_id`) REFERENCES
`connection` (`connection_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*INSTANT_MESSAGING*/
DROP TABLE IF EXISTS `accipp`.`instant_messaging`;
CREATE TABLE  `accipp`.`instant_messaging` (
  `message_id` int(10) unsigned NOT NULL auto_increment,
```

```
  `attachment_id` int(10) unsigned NOT NULL,
  `message_log_id` int(10) unsigned NOT NULL,
  `protocol_name` varchar(10) NOT NULL,
  PRIMARY KEY  USING BTREE (`message_id`,`attachment_id`),
  KEY `attachment_id` (`attachment_id`),
  KEY `message_log_id` (`message_log_id`),
  CONSTRAINT  `message_log_id`  FOREIGN  KEY  (`message_log_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT  `attachment_id`  FOREIGN  KEY  (`attachment_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `message_id` FOREIGN KEY (`message_id`) REFERENCES
`connection` (`connection_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*NEWS*/
DROP TABLE IF EXISTS `accipp`.`news`;
CREATE TABLE  `accipp`.`news` (
  `news_id` int(10) unsigned NOT NULL auto_increment,
  `attached_id` int(10) unsigned NOT NULL,
  `news_text_id` int(10) unsigned NOT NULL,
  `subject` varchar(45) NOT NULL,
  `group_name` varchar(45) NOT NULL,
  PRIMARY KEY  USING BTREE (`news_id`,`attached_id`),
  KEY `attached_id` (`attached_id`),
  KEY `news_text_id` (`news_text_id`),
  CONSTRAINT  `news_text_id`  FOREIGN  KEY  (`news_text_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `attached_id` FOREIGN KEY (`attached_id`) REFERENCES
`file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT  `news_id`  FOREIGN  KEY  (`news_id`)  REFERENCES
`connection` (`connection_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*EMAIL*/
DROP TABLE IF EXISTS `accipp`.`email`;
CREATE TABLE  `accipp`.`email` (
  `email_id` int(10) unsigned NOT NULL auto_increment,
  `attached_file_id` int(10) unsigned NOT NULL,
  `mail_text_id` int(10) unsigned NOT NULL,
  `mail_date` datetime NOT NULL,
  `mail_to` varchar(45) NOT NULL,
  `mail_from` varchar(45) NOT NULL,
  `cc` varchar(45) NOT NULL,
  `bcc` varchar(45) NOT NULL,
  `subject` varchar(45) NOT NULL,
  `protocol_name` varchar(45) NOT NULL,
  PRIMARY KEY  USING BTREE (`email_id`,`attached_file_id`),
```

```
  KEY `mail_text_id` (`mail_text_id`),
  KEY `attached_file_id` (`attached_file_id`),
  CONSTRAINT `attached_file_id` FOREIGN KEY (`attached_file_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `email_id` FOREIGN KEY (`email_id`) REFERENCES
`connection` (`connection_id`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `mail_text_id` FOREIGN KEY (`mail_text_id`)
REFERENCES `file` (`file_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

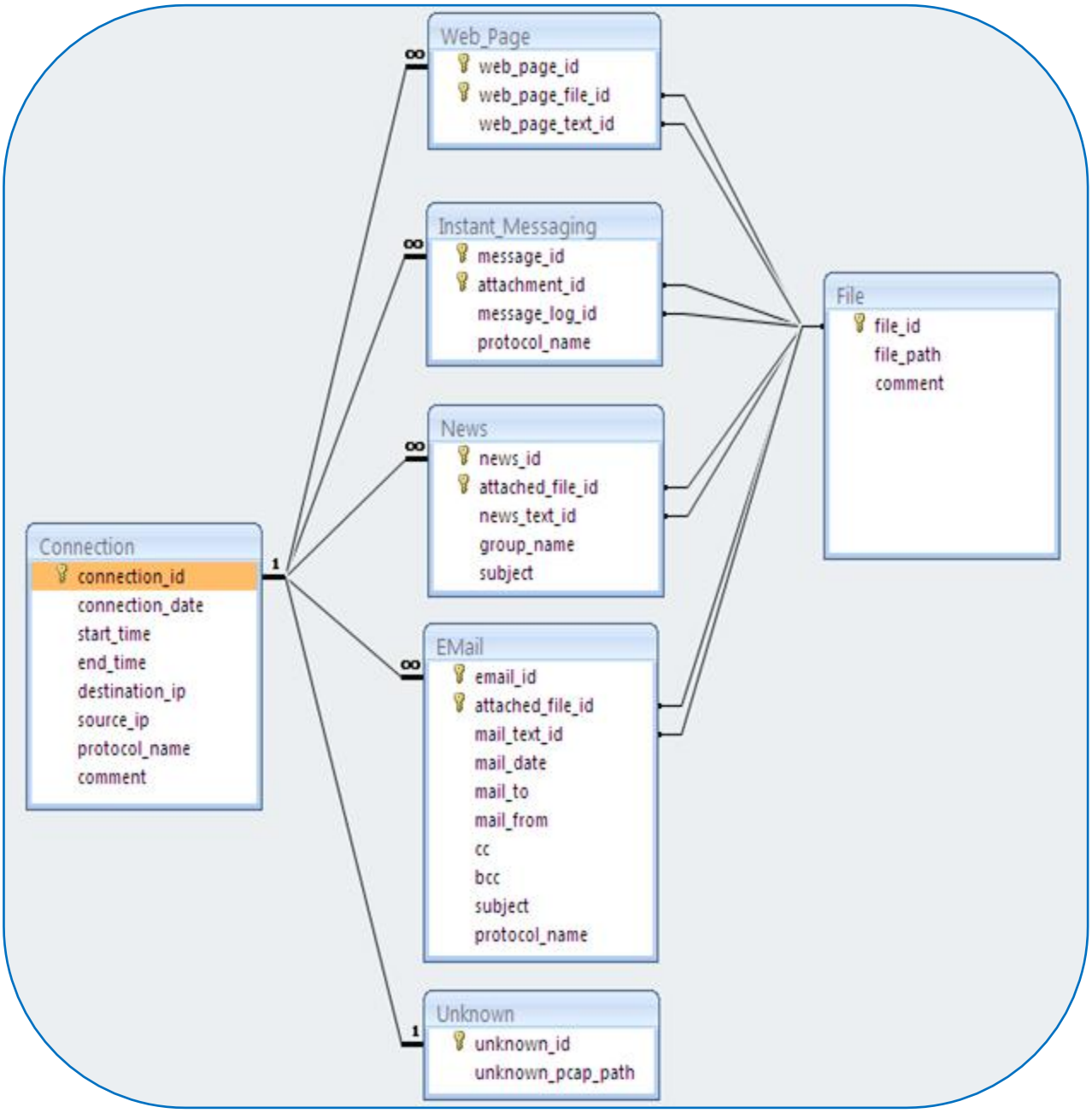
/*UNKNOWN*/

```
DROP TABLE IF EXISTS `accipp`.`unknown`;
CREATE TABLE `accipp`.`unknown` (
  `unknown_id` int(10) unsigned NOT NULL auto_increment,
  `unknown_Pcap_path` varchar(100) NOT NULL,
  PRIMARY KEY (`unknown_id`),
  CONSTRAINT `unknown_id` FOREIGN KEY (`unknown_id`) REFERENCES
`connection` (`connection_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

/*FILE*/

```
DROP TABLE IF EXISTS `accipp`.`file`;
CREATE TABLE `accipp`.`file` (
  `file_id` int(10) unsigned NOT NULL auto_increment,
  `file_path` varchar(50) NOT NULL,
  `comment` varchar(45) NOT NULL,
  PRIMARY KEY USING BTREE (`file_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

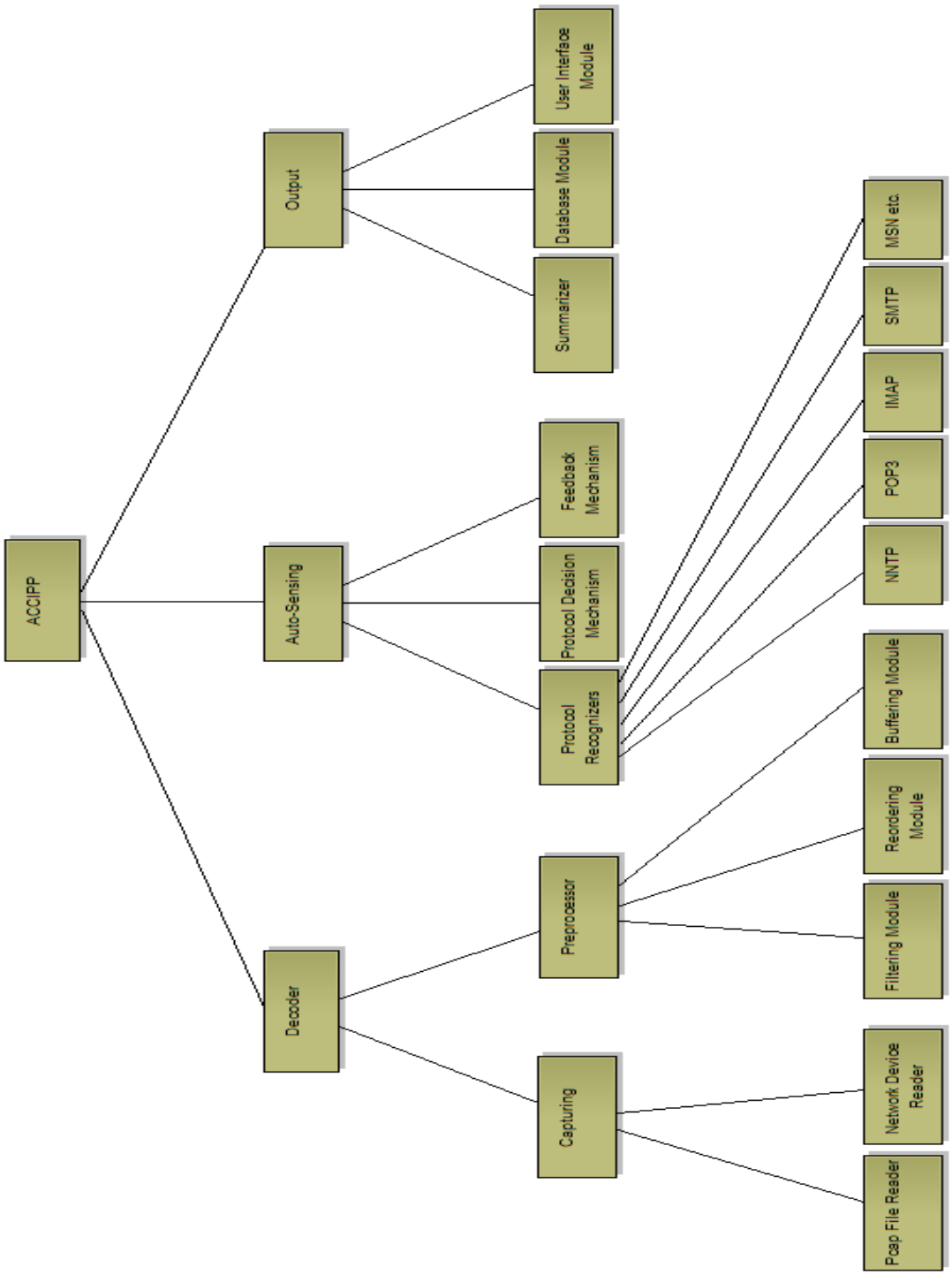
The resultant database schema and the related relations are as follows:



6. Architectural Design

6.1 Structure Chart

The following is the structure chart that represents the module hierarchy of ACCIPP.



6.2 System Modules

The ACCIPP program consists of three main modules that are ***Decoder***, ***Auto-Sensing*** and ***Output*** modules. Through the user interface, the user selects an input source (a Pcap file or a network device) to process, and then gives the Start Capture command. After that, network packets begin to enter our program through the Decoder module, and data flows through the ***Decoder***, ***Auto-Sensing*** and ***Output*** modules respectively. Together, these modules let the user view information about network connections on a system. These modules are going to be elaborated in the following sections.

6.2.1 Decoder Module

Decoder module is the base module of ACCIPP. After the user selects the input device and make the modules work by selecting 'Start Capture', decoder module takes action and reads the packets from the input source by the ***Capturing Module***. Afterwards, the captured packets goes into the ***Preprocessing Module*** where they are prepared for the ***Auto-sensing Module.*** There, they are processed by some specific operations that will be mentioned later on.

This module does not contain any learning mechanism as ***Auto-sensing Module*** does. The data-flow is fairly straightforward. Even if this module seems to be very simple, it is a must for the further modules.

Below the sub-modules of the Decoder Module is described:

6.2.1.1 Capturing Module

This module is responsible of capturing packets from the network. Packets can be obtained from a Pcap File by the ***Pcap File Reader*** or from a network device by the ***Network Device Reader***. If the user chooses capturing packets from a Pcap file, offline process can be achieved (a Pcap File can be processed in the future thanks to ***Pcap File Reader)***. Or if the user chooses capturing packets from a network device, real-time processed can be achieved. Namely, whenever a packet comes through a connection, ***Network Device***

Reader captures the real-time incoming packets and then send them to the ***Preprocessing Module***.

6.2.1.2 Preprocessing Module

This module takes the captured packets and then applies a few operations on these packets. These operations are handled by some sub-modules which are ***Filtering Module***, ***Reordering Module*** and ***Buffering Module***.

Captured packets are first go into the ***Filtering Module***. Here, they are handled according to some filtering parameters. The packets coming from protocols other than TCP or UDP, are eliminated because ACCIPP does not try to recognize protocols which does not come from TCP or UDP. Besides, here some checksum comparison are performed on the captured packets. If packets have invalid checksum values then they are eliminated too. In detail, before sending the packet, the sender calculates the sum of the bytes and then adds this information to the header. When the packet comes, sum of the bytes of the packet is calculated one more time by our module and compare the checksum value in the header with the value it's found. If they are not same, then it means the packet is broken or incomplete. So the module discards the packet because there is no need to process on a broken packet. After the operation of filtering is finished, the filtered packets are transmitted to the next module.

When the filtered packets come into the ***Reordering Module***, they are reordered according to their TCP sequence number. This number can be found in the headers. Since packets may come in an unordered way, ***Reordering Module*** should sort the packets to make them same as the original data stream. For example; if the ***Filtering Module*** does not eliminate the broken packets and if a packet does not include the TCP sequence number by some reason, this module wouldn't be able to reorder these packets, and if the packets cannot be reordered properly, then these packets cannot be processed correctly by the ***Auto-Sensing Module***. After the packets leave this module, they are redirected to the next module.

After the packets are reordered they come into the ***Buffering Module***. In this module, packets are stored in buffers. They can be placed in a buffer one by one or in a buffer of ten packets, or differently. The number of packets in a buffer is determined as necessary. For ex; if a packet cannot be processed in real-time, it should stay in a buffer until it can go into the protocol recognizers. Besides, for line oriented protocols such as POP3, the packets that form a single line should stay in a buffer until the line is complete. After the this Module finishes it work then these preprocessed packets are sent to the ***Auto-Sensing Module***. So generally, there is nothing left to do in the ***Decoder Module.***

6.2.2 Auto-Sensing Module

Auto Sensing module is the part that does the actual protocol recognition. Protocol recognition is done in three steps. The packet sequences coming from the ***Decoder*** module first enter the ***Protocol Recognizers***, and then results coming from the ***Protocol Recognizers*** are collected and directed to the ***Protocol Decision Mechanism***, where the final decision about the protocol type of a connection is made. Finally, the collected data enters the ***Feedback mechanism*** that updates ***Protocol Recognizer*** modules with the newly collected information. The following sections contain detailed descriptions of the sub-modules in the ***Auto-Sensing Module***.

6.2.2.1 Protocol Recognizers

Protocol Recognizers is the common name given to a set of modules that are each responsible for recognizing one specific protocol. These modules all share a common interface and communicate with the top level auto-sensing module through this common interface. However, they have no communication between each other. User can enable/disable a protocol recognizer as he/she wishes, yet this does not affect other recognizers as they are not aware of each other. Consequently, protocol recognizer modules are planned to be implemented as “plug-ins” that users can add/remove depending on their needs.

Although a protocol recognizer is designed to be stand-alone, there might be some exceptions to this schema where a protocol recognizer may depend on another one. For instance; most instant messaging applications use HTTP for file transfers, and this makes the instant messaging recognizer module depend on the HTTP recognizer module.

A protocol recognizer takes a packet sequence as input and runs the pattern matcher (that will probably be a derivative of *Support Vector Machines* and/or Hidden *Markov Model*) against the contents of this packet sequence. The pattern matcher generates a value, called *Match Value*, which indicates the match percentage of packet sequence with the protocol pattern. The protocol recognizer is also responsible for extracting some protocol-specific information, called *Match Data*, from the contents of packet sequence. Finally the protocol recognizer outputs this match value and data pair to the ***Protocol Decision Mechanism***.

6.2.2.2 Protocol Decision Mechanism

The Protocol Decision Mechanism is responsible for collecting *Match Value* and *Match Data* pairs from the ***Protocol Recognizers*** and choosing the protocol that has the highest match with the packet sequence contents. For the time being, The Protocol Decision Mechanism simply picks the protocol with the highest match value that exceeds a pre-defined threshold value and labels the connection as this protocol. This threshold value helps eliminating spurious matches, since it ensures that match values that are too small will not be taken into consideration.

Protocol Recognizer modules are designed to run concurrently (presumably in separate threads/processes) thus, Protocol Decision Mechanism must wait until all ***Protocol Recognizers*** to complete their work until a decision can be made. This makes implementing some sort of signaling mechanism between processes necessary.

After the ***Protocol Recognizer*** collects output from all ***Protocol Recognizers*** and chooses the one with the highest match value it redirects this output to two different modules, namely ***Feedback module*** and the ***Summarizer***.

6.2.2.3 Feedback Mechanism

The Feedback mechanism contributes to the automated learning part of the ***Protocol Recognizer*** modules. After the ***Decision Module*** chooses the protocol with the highest match value, Feedback mechanism updates the protocol pattern in the corresponding ***Protocol Recognizer Module***. If the packet contents cannot be recognized by any of the available protocol recognizers, the ***Decision Mechanism*** labels the connection as "*Unknown*" and this connection bypasses the Feedback Mechanism. This ensures that protocol patterns are not updated with wrong or defective information.

6.2.3 Output Modules

After the packet sequences are processed in ***Auto-Sensing*** module, valuable information along with the matching protocol names is extracted from these packets. Output modules are responsible for producing human-readable output from the information coming from ***Auto-Sensing*** modules. Since the program is able to present it output in various manners, there is a separate sub-module for each output format. Currently there are three output modules, namely ***Summarizer module***, ***Database module***, and the ***User Interface*** module.

6.2.3.1 Summarizer Module

This module takes the data from the ***Auto-Sensing Module*** and produces a summary that is appropriate for the recognized protocol out of this data. For instance; the Summarizer module will generate a report that contains the mail subject, sender and recipient names and mail body for a recognized POP3 connection. This report also includes match percentages that come from other protocol recognizers.

For an unknown connection; if the program can extract any information from the packet contents, the report will include them along with the match percentages.

Summarizer Module is also in relation with the ***Database module***, so that the user can store and retrieve summaries in a central database.

6.2.3.2 Database Module

The Database module is responsible for performing all database operations requested by the user or other components the system. The database module is used for reading, adding, updating and deleting records in the central database.

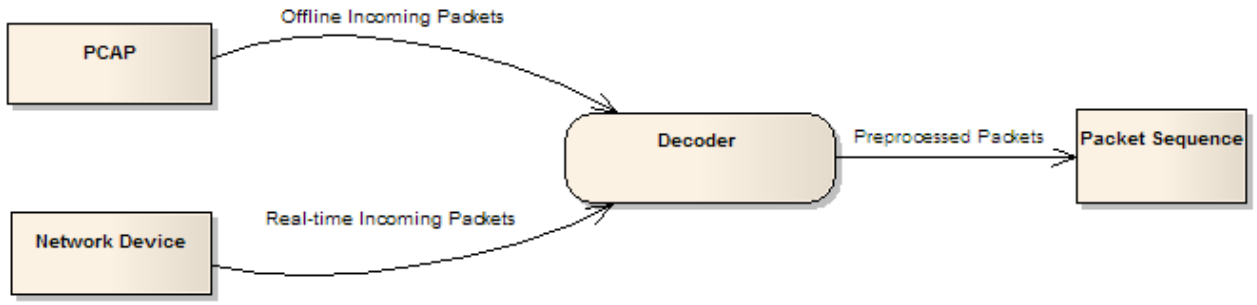
When the user chooses to save a summary (for later retrieval) or wishes to see a past summary, the ***Summarizer module*** accesses the Database through this module. Additionally, the Database module is used to generate statistical information out of database contents. The exact structure of database can be found in Data Design section.

6.2.3.3 User Interface Module

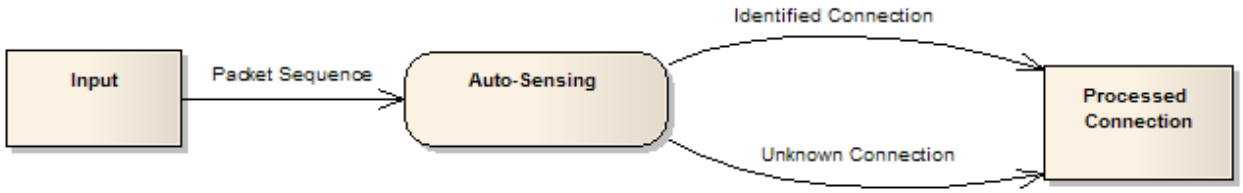
The user interface module is responsible for handling all interactions with the user and the rest of the system. The user uses all functionality within other modules through the user interface. The primary functions of this module are; presenting the user the results of protocol recognition process, visualizing statistical information in charts, and letting the user give commands to the program.

6.3 Data Flow Diagrams

6.3.1 Level 0 DFD of Decoder Module

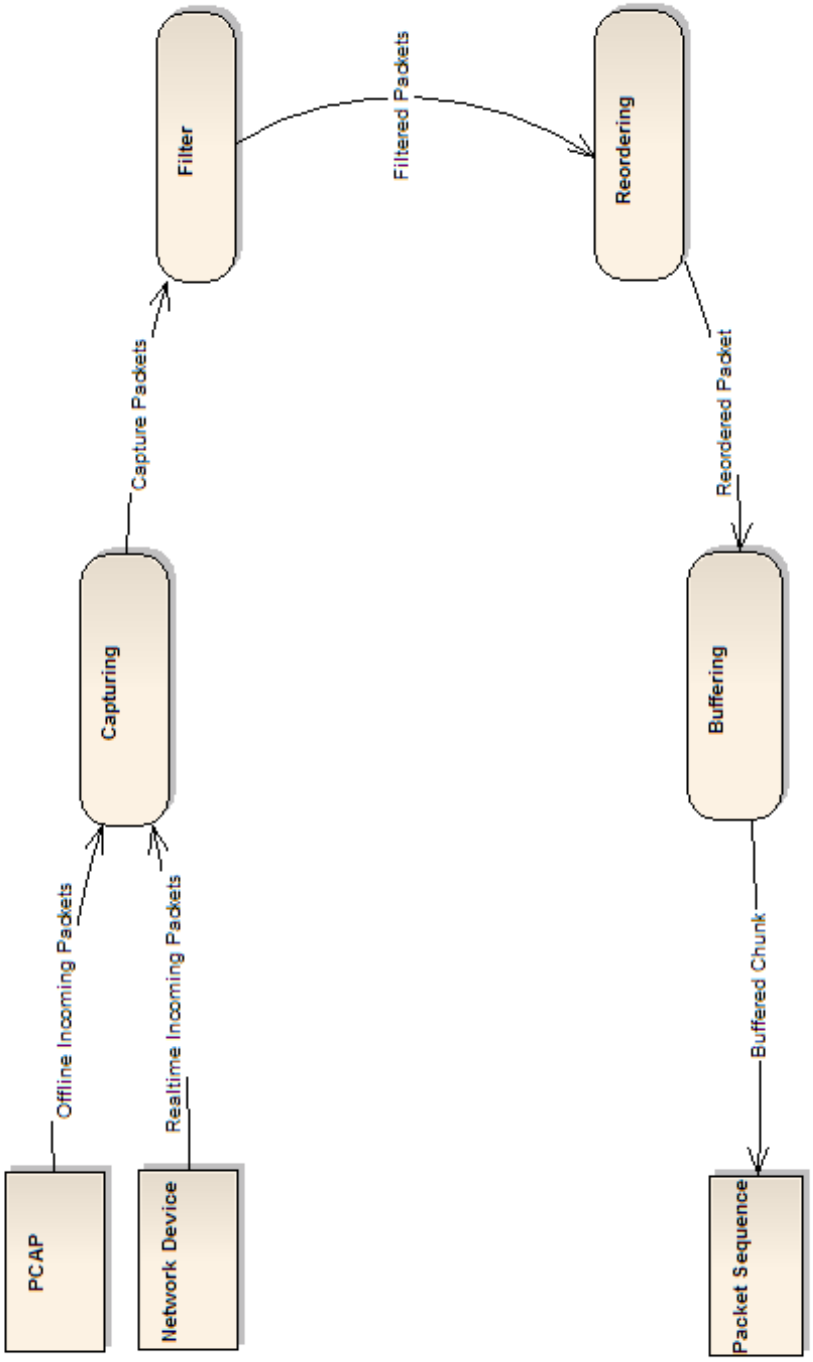


6.3.2 Level 0 DFD of Auto-Sensing Module

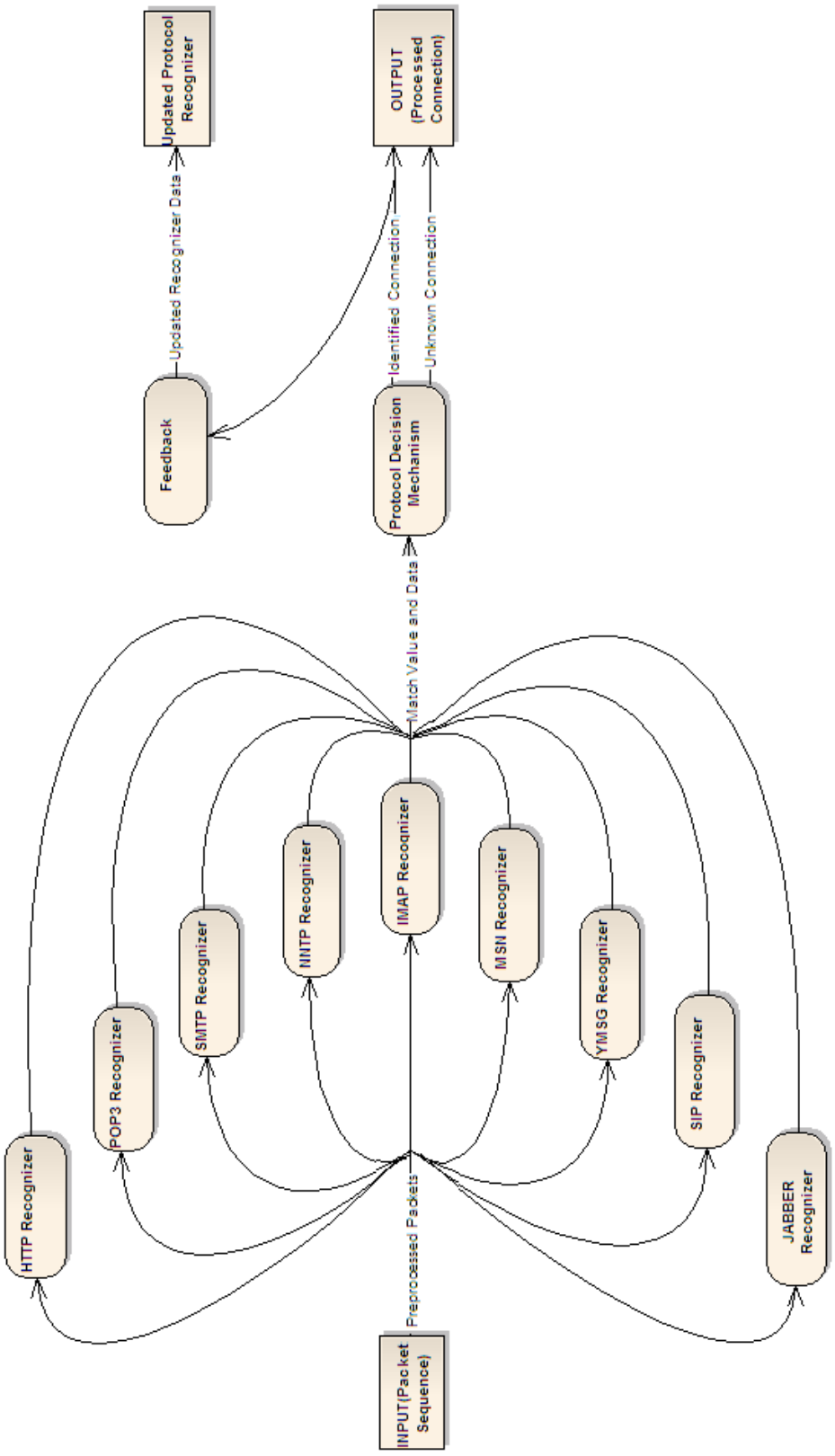


6.3.3 Level 0 DFD of ACCIPP

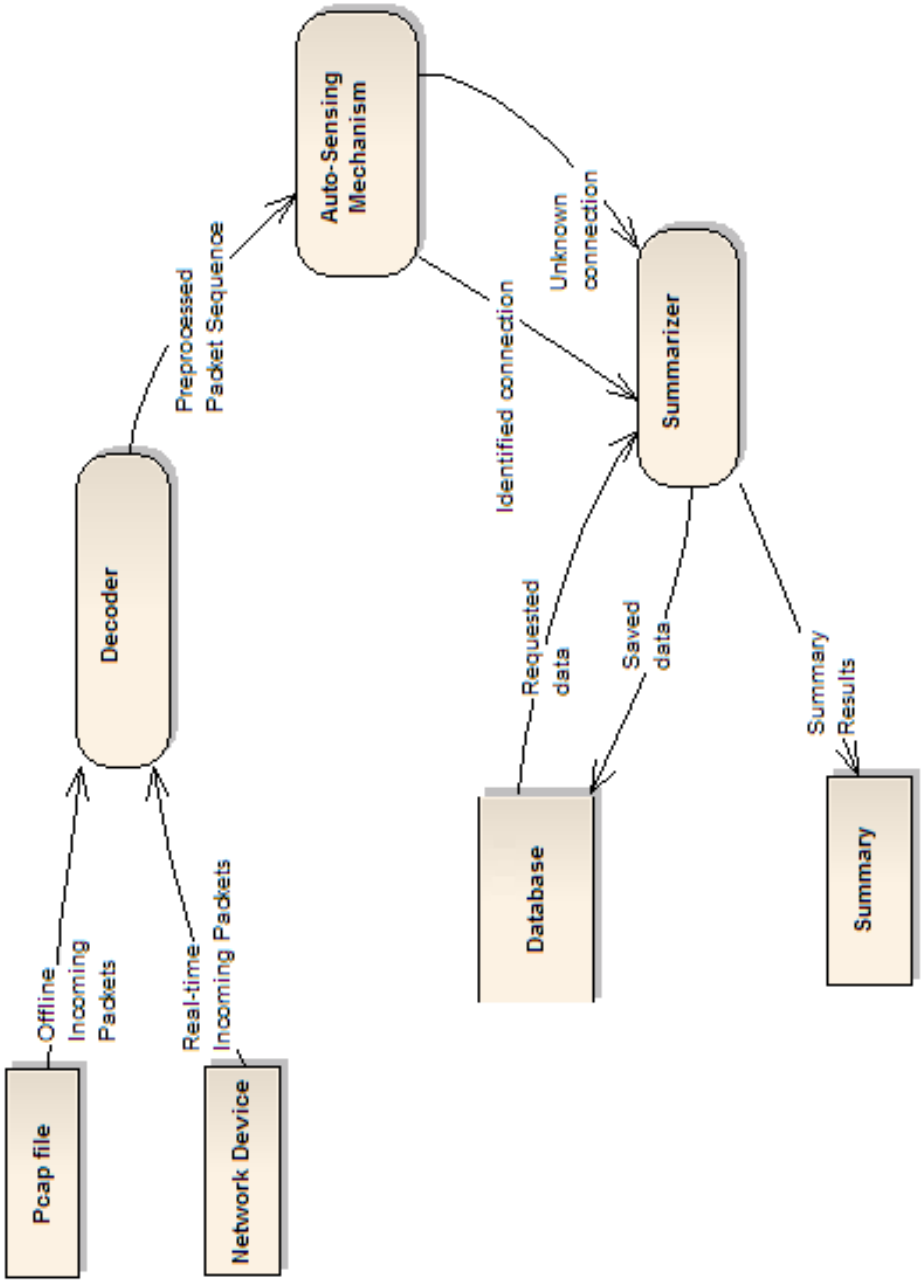
6.3.4 Level 1 DFD of Decoder Module



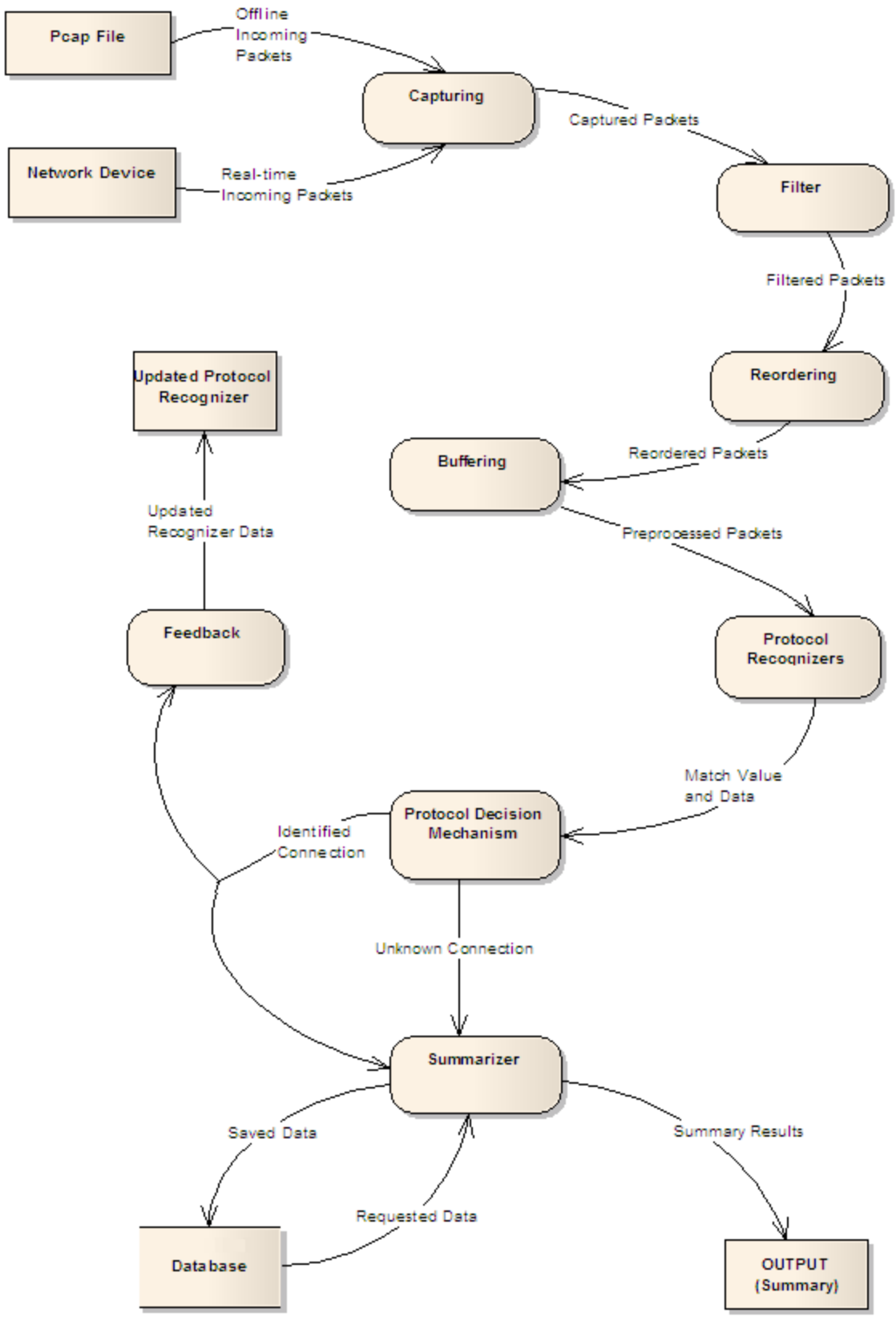
6.3.5 Level 1 DFD of Auto-Sensing Module



6.3.6 Level 1 DFD of ACCIPP



6.3.7 Level 2 DFD of ACCIPP



6.4 Data Dictionary

Name: Offline Incoming Packets

Aliases: Pcap File

Where & How used: Capturing (Input)

Description: These are the packets that come from a Pcap file that was generated beforehand. These packets are processed in offline mode.

Name: Real-time Incoming Packets

Aliases: Network Device Packets

Where & How used: Capturing (Input)

Description: These are the packets that come from the network device instantaneously. These packets are processed in real-time mode.

Name: Captured Packets

Aliases: None

Where & How used: Capturing (Output)
Filter (Input)

Description: After real time incoming packets and offline incoming packets enter the Capturing module, they become Captured packets, which go into the Filter module afterwards.

Name: Filtered Packets

Aliases: None

Where & How used: Filter (Output)
Reordering (Input)

Description: Captured packets that enter the Filter module and they are checked against some filtering parameters. For example ACCIPP only deals with TCP and UDP packets, therefore packets coming from other protocols like ICMP are ignored. The filter module also performs some checksum comparison on the captured packets, and discards packets with an invalid checksum value. Packets that are processed in the Filter module enter the Reordering module as Filtered Packets.

Name: Reordered Packets

Aliases: None

Where & How used: Reordering (Output)

Buffering (Input)

Description: Network packets may not necessarily be transmitted in the order they are meant to be received. So they need to be reordered in order to reassemble the original data stream. After filtered packets arrive at the reordering module, they are sorted according to their TCP sequence number that is stored in packet headers. After packets are processed in the reordering module, they are delivered to the Buffering module.

Name: Preprocessed Packets

Aliases: Buffered Chunk

Where & How used: Buffering (Output)

Protocol Recognizers (HTTP Recognizer, POP3 Recognizer, SMTP Recognizer, NNTP Recognizer, IMAP Recognizer, MSN Recognizer, YMSG Recognizer, SIP Recognizer, JABBER Recognizer) (Input)

Description: After packets are filtered and reordered, they enter the buffering module. Here packets are stored consequently in buffers. If a packet cannot be processed real-time, it needs to be buffered. So the packet stays in this buffer until it gets processed by the protocol recognizers. Preprocessed packets that leave the buffering module are finished with the decoder part of the program and enter protocol recognizers such as HTTP and POP3 concurrently where they will be checked against protocol patterns.

Name: Match Value and Data

Aliases: None

Where & How used: Protocol Recognizers (HTTP Recognizer, POP3 Recognizer, SMTP Recognizer, NNTP Recognizer, IMAP Recognizer, MSN Recognizer, YMSG Recognizer, SIP Recognizer, JABBER Recognizer) (Output)

Protocol Decision Mechanism (Input)

Description: After packets become processed in protocol recognizers, a match percentage value and protocol specific data is produced in each protocol recognizer. This match percentage shows how much the packet contents match with the protocol pattern, and data contains human-readable information such as mail body for POP3 protocol. Then these match values and data enter Protocol Decision Mechanism.

Name: Identified Connection

Aliases: Processed Connection

Where & How used: Protocol Decision Mechanism (Output)

Summarizer (Input)

Feedback (Input)

Description: All match values are gathered by the protocol decision mechanism to decide which protocol the connection resembles most. If the highest match value exceeds a threshold value, then the connection becomes Identified Connection, and this connection information goes into Summarizer to prepare a connection summary. Besides, the connection data enters feedback module so that pattern recognizers are able to update themselves by using this data.

Name: Unknown Connection

Aliases: Processed Connection

Where & How used: Protocol Decision Mechanism (Output)
Summarizer (Input)

Description: All match values are gathered by the protocol decision mechanism to decide which protocol the connection resembles most. If the highest match value does not exceed a threshold value, then the connection becomes Unknown Connection and the resolved connection information goes into Summarizer to prepare a connection summary.

Name: Updated Recognizer Data

Aliases: Updated Protocol Recognizer

Where & How used: Feedback (Output)

Description: For Identified Connections, the connection data enters the feedback mechanism so that the associated protocol recognizer is able to update itself with the new information. The exact method to form the updated recognizer data has not been decided yet.

Name: Saved Data

Aliases: Summary Info

Where & How used: Summarizer (Output)
Database (Input)

Description: The summary prepared in the summarizer is stored into the Database. When the user wishes to save the summary, it is transformed into an appropriate format that is compatible with the database backend. This information can later be retrieved from the database and the summary can be reproduced using the retrieved data.

Name: Requested Data

Aliases: Old Summary Info

Where & How used: Database (Output)
Summarizer (Input)

Description: The user may want to retrieve an old summary from the database. In that situation the data needs to be transferred from the database backend to the summarizer module. This data is called requested data. By using this data, an identical copy of the old summary can be reconstructed.

Name: Summary Results

Aliases: Output

Where & How used: Summarizer (Output)

Description: The data that is formed in an appropriate format in the summarizer module is displayed to the user as summary results. These results also contain protocol specific information coming from the protocol recognizers and match values of the protocols.

7. System Design

Use case, class, sequence and activity diagrams can be found in this section.

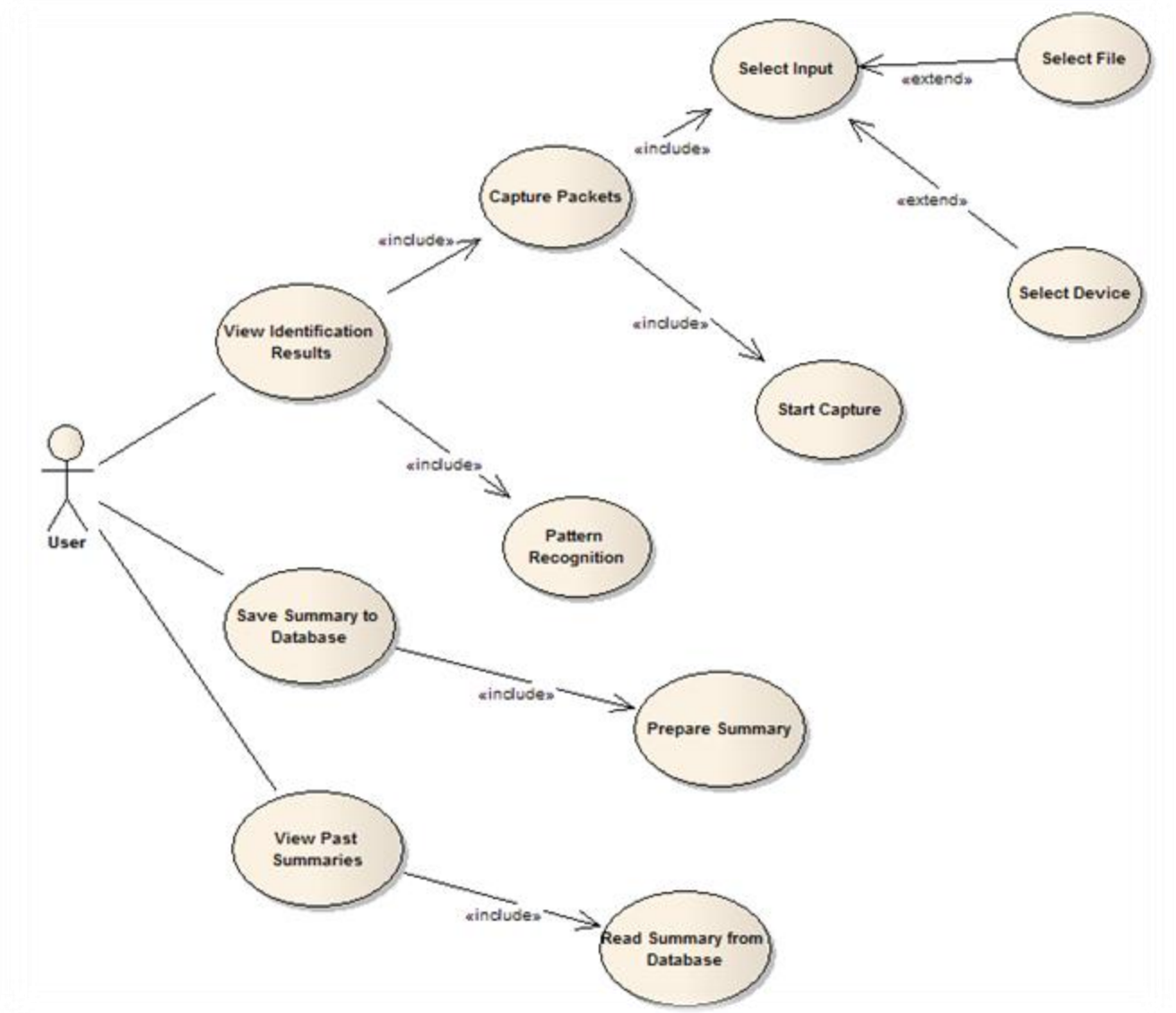
7.1 Use Cases

In this section, use case diagrams and scenarios can be found.

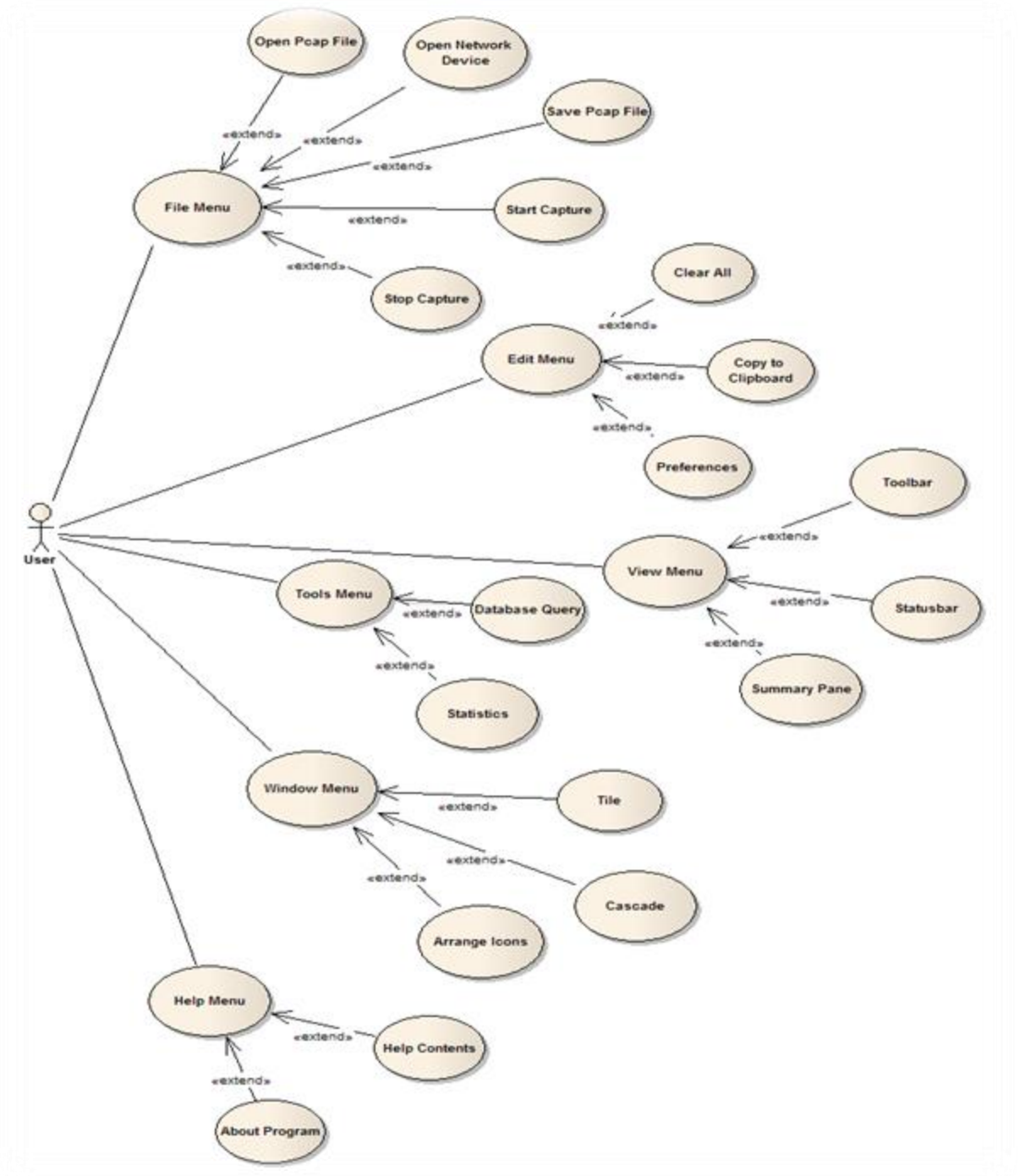
7.1.1 Use Case Diagrams

Menu and end-user use case diagrams are shown below.

7.1.1.1 Menu Use Case Diagram



7.1.1.2 Use Case Diagram for the end-user



7.1.2 Use Case Scenarios

Use case scenarios for the use case diagrams above are situated below.

7.1.2.1 Scenario for Use Case Diagram 1

Open Pcap File: The program can be used in either offline or real-time mode. In offline mode, all the network packets are captured stored on a secondary storage device beforehand. This command allows the user to select a previously generated Pcap file for offline processing. After the user clicks this menu item, a standard open file dialog is displayed where the user can either type the name of a file or browse through the file system and choose it.

Open Network Device: This command is used to enter real-time mode. In real-time mode, packets are not read from a Pcap file but they are captured from a network device. Since they are not yet stored anywhere, they must be processed on-the-fly. After this menu item is clicked, a dialog box containing the list of available network devices is presented to the user. The user selects the network device he/she wants to examine and then closes the dialog.

Save Pcap File: This menu command lets the user to choose a file on his/her disk to store the captured packets in Pcap format. The user may want to do this in two scenarios: The pattern recognition engine might not be completely trained and therefore does not work at full capacity yet. The user chooses to save the packets as a Pcap file, so that he/she can analyze them in the future when the pattern recognition engine performs relatively better. Another case is that the user may wish to examine the contents of captured packets with another Pcap compatible application, for instance WireShark. Therefore, apart from being an intelligent protocol identification application, ACCIPP can also be used as a general purpose packet sniffer.

Start Capture: This command allows the user to begin processing packets. Depending on the input source selected previously, the program begins reading packets from either a

Pcap file or a network device. If the user has not selected an input source yet, this command has no effect. As soon as new packets begin to arrive, they are redirected to the pattern recognition engine and gradually, connections begin to appear on the Connection List.

Stop Capture: This command allows the user to stop processing packets. Even if new packets arrive from the network device or there are further packets available in the Pcap file, they will be discarded. Since those packets do not enter the pattern recognition engine, they won't have any effect on the identifications results or the summary.

Clear All: This command allows the user to clear all the entries in the connection list pane along with all summaries and identification data (such as match percentages and pattern recognizer status) associated with them. When new packets arrive, they will be treated as new connections and might be identified differently since the previous states of the pattern recognizer is no longer available.

Copy to Clipboard: This command allows the user to copy the contents of Short Summary Pane to the clipboard so that the user then may paste and use this information in other applications.

Preferences: This command allows the user to change or view various settings of the program. Such settings may include, but are not limited to: Appearance of user interface elements (fonts, colors etc.), Whether or not the program starts upon system startup, Configuration parameters for pattern recognition engine, and Enabling or Disabling some protocol handlers (presumably for performance reasons).

View Menu: This menu includes commands to toggle visibility of some user interface elements like Toolbar, Status bar and Summary Pane.

Database Query: This command allows the user to enter a query in order to see information that is stored in the database. The database includes valuable information captured from the protocol connections. For example; the database includes all received mail through POP3 sessions. The user may want to query the database for listing the mails

that are received in a specific time interval, or the user may want to see all connection events from a certain IP address, and so on. Database query command provides a link between the user and the data captured through the capture engine.

Statistics: This command allows the user to generate statistical information from the database. This information gives an overall grasp about the protocol connections to the user. For example, the user may want to see which protocol is used most from a specific IP address, how much bandwidth is used by protocols etc. The Statistics command is in close relation with the Database Query mentioned above. This gives the user the chance to form some highly customized statistical data from the database.

Window Menu: This menu includes commands for changing the positions and sizes of the sub-windows. For example, when more than one Summary Window is visible, the user may want to tile these windows in order to see all of them at once.

Help Menu: This menu allows the user to access program documentation that helps the user get used to the program.

7.1.2.2 Scenario for Use Case Diagram 2

View Identification Results:

The user can view details of connections such as source and destination IP addresses, recognized protocol of each connection, connection start and end time, etc. More importantly, if the process of protocol recognition is finished, he/she can see the resolved transferred data through addresses regardless of the port information. This process includes two major subroutines that are pattern recognition and capturing packets. For capturing packets input must be selected by the user. The input can be a Pcap file (for offline application) or a network device (for real-time application). If a problem occurs in these subroutines or if the user does not select an input and does not attempt to catch packets then it is impossible to view any identification results.

Save Summary to Database:

The user can save summaries, namely details of the connection and transferred data that are shown in the summary window. This data is stored in the database. This process includes the preparation of the summary. As it is obvious, the user cannot save a summary before it is prepared. The user may want to do this action in several scenarios. For instance; during the work of the program many connections occur and as time passes the number of connections increases rapidly. The user may not be able to look at all summaries in a small amount of time. So he/she may save some of them for analyzing later on. Another scenario for this action is that the user may want to view statistical information and this action is probably performed with the stored data. For example; he/she may want to compare who uses which protocol most and etc.

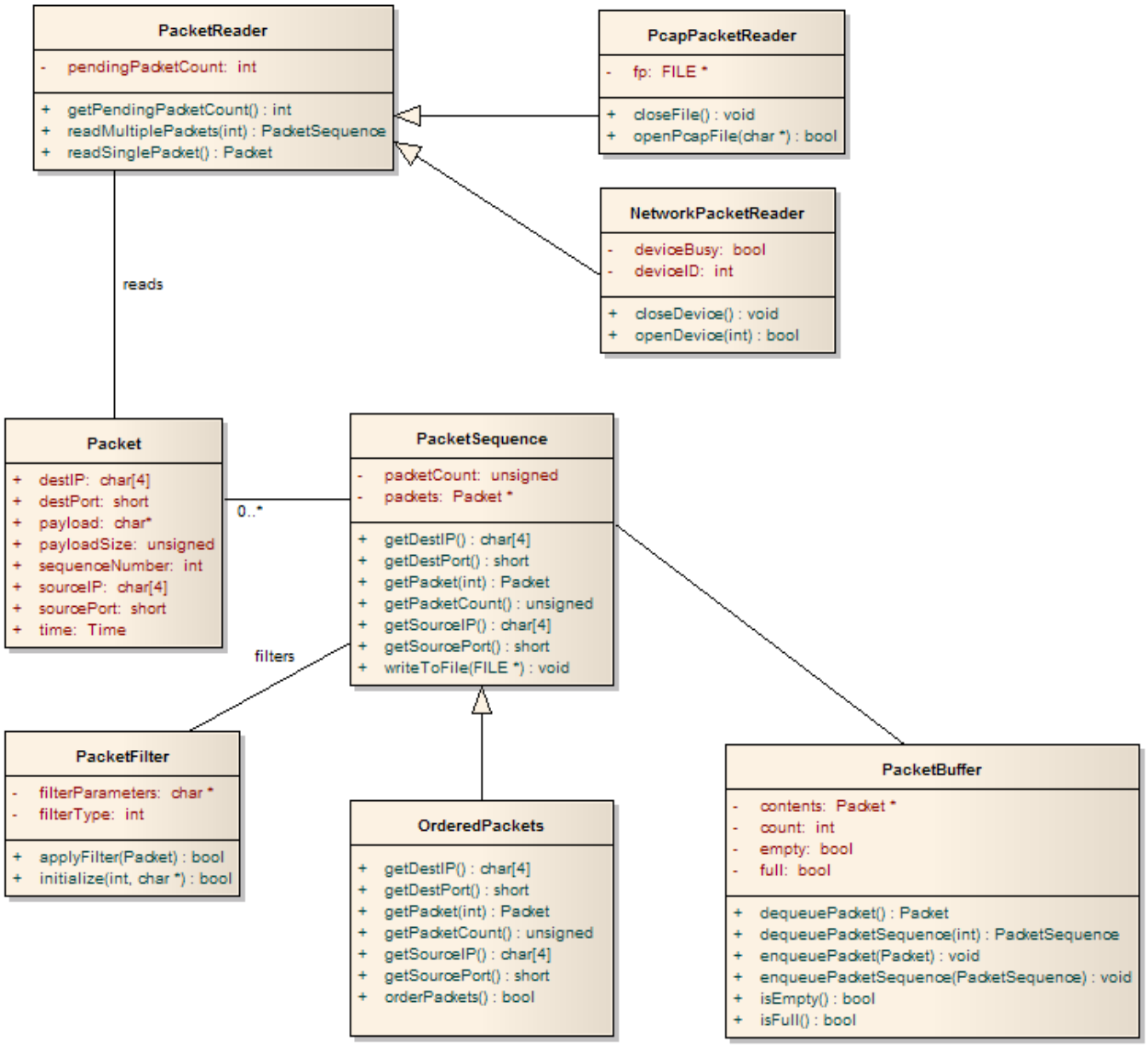
View Past Summaries:

The user can view past summaries. Whenever the user wants to retrieve a summary from the database, reading summary from the database must be performed. The scenarios for this action presumably show similarities with the above action. As it is mentioned above, the user may be obliged to view some summaries in the future because of limited time. Besides, he/she may want to work over an old summary, so this action would satisfy the user's will.

7.2 Class Diagrams

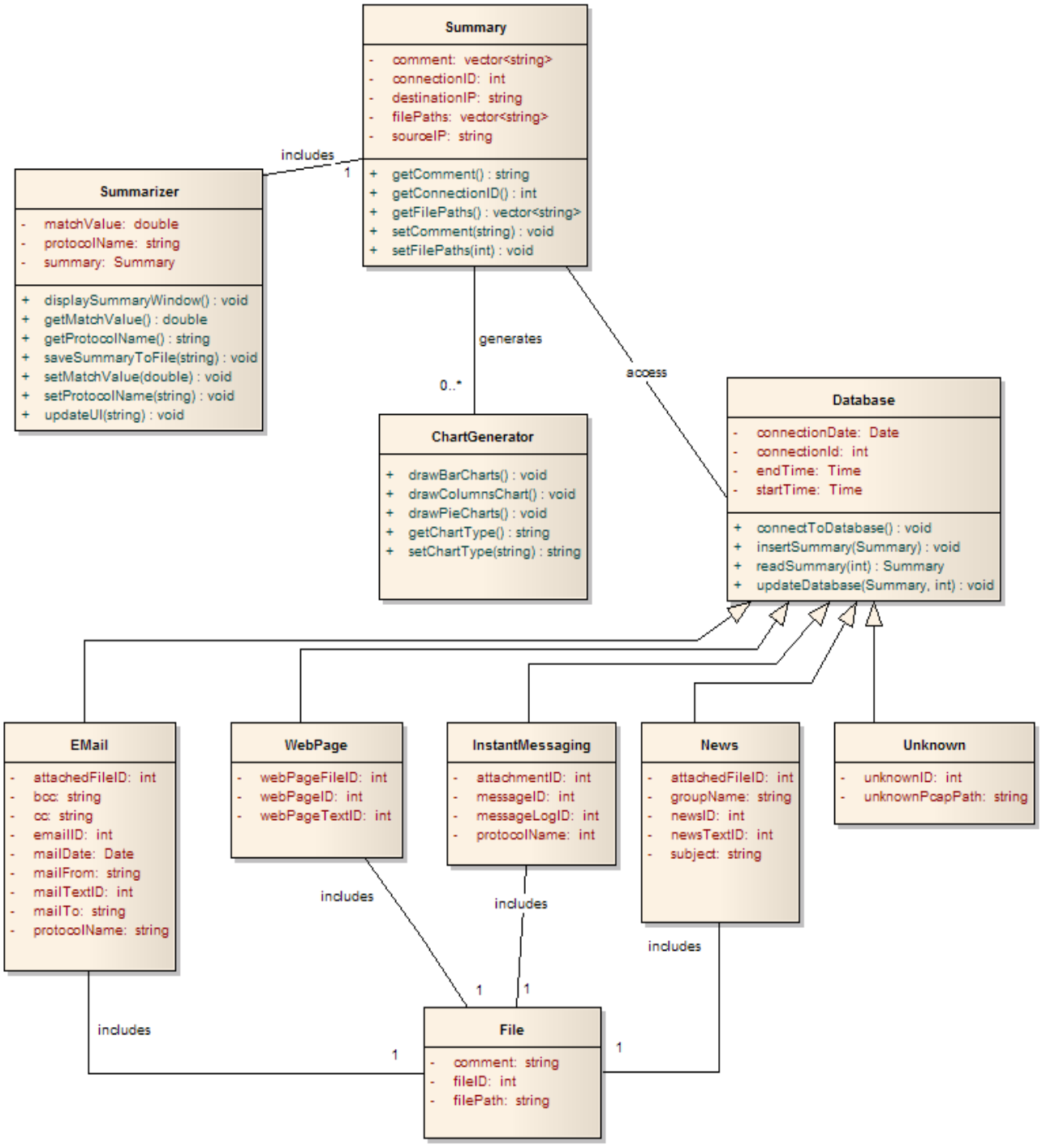
Class diagrams for the Decoder and Output Modules and the related descriptions can be seen below.

7.2.1 Decoder Module



- **PacketReader** is an abstract class that is responsible for reading packets from an input source. It includes methods for reading either a single packet or multiple packets at once. The pendingPacketCount member returns the unread packet count waiting at the input source.
- **PcapPacketReader** is derived from the PacketReader class and implements functionality to read packets from a Pcap file.
- **NetworkPacketReader** extends the base PacketReader class, and includes functions to read packets from a network device.
- **Packet** class is the data structure that is used to define a single packet. The member variables of this class are filled by the object that reads the packet from the input source.
- **PacketSequence** is the collection class for packets that have the same source and destination addresses and same ports. It includes methods that provide random access to packets stored in the collection. This class is also responsible for dumping its contents to a Pcap file.
- **PacketFilter** is the class that is responsible for eliminating packets that are not TCP or UDP, and the ones with invalid checksum values.
- **OrderedPackets** extends the PacketSequence class to add functionality that orders the packets in the sequence based on their sequence number.
- **PacketBuffer** implements a simple FIFO queue mechanism that is able to store a predefined number of packets in a queue data structure.

7.2.2 Output Module



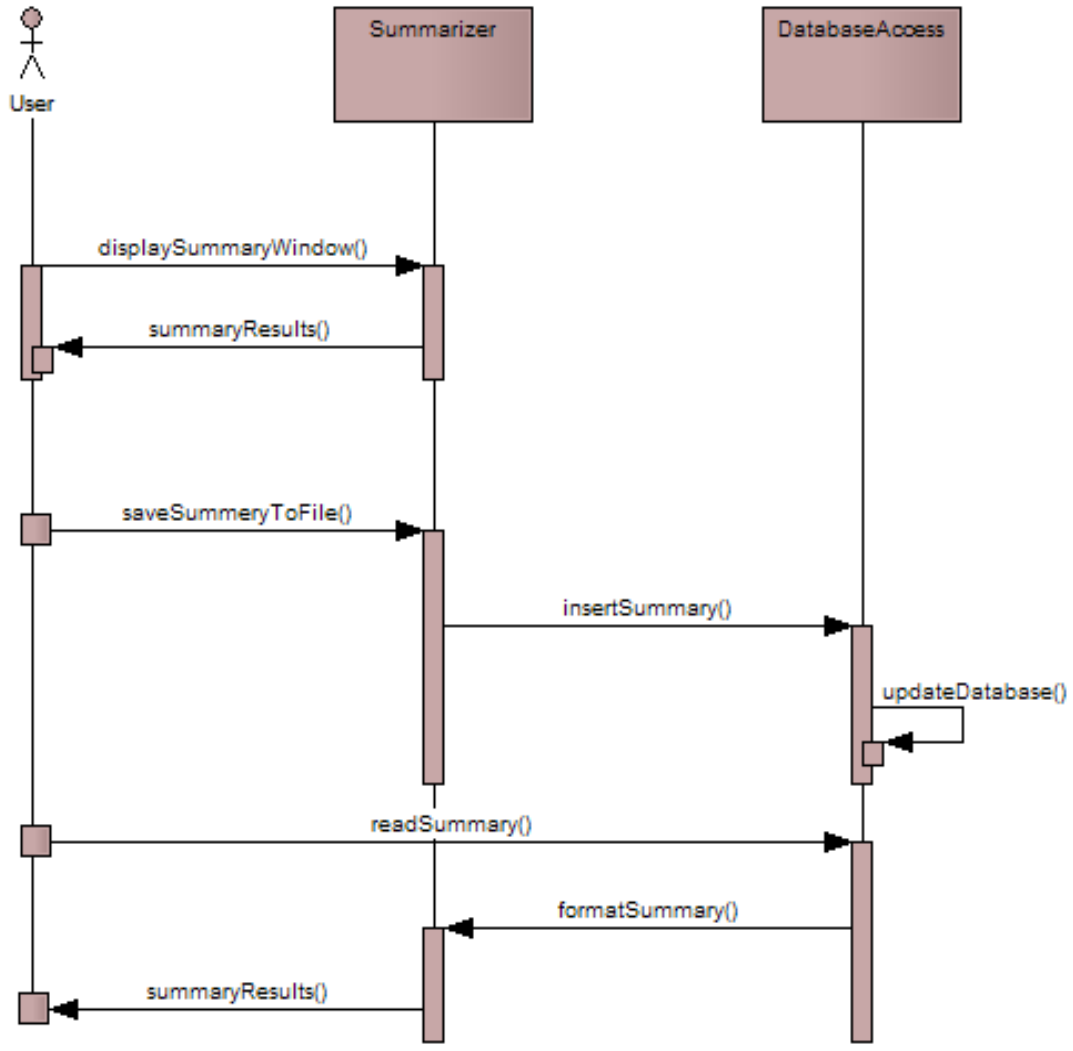
- **Summarizer** class generates user-friendly summary of the connection data received from the AutoSensing mechanism. When it receives a new connection, it calls Summary class.
- **Summary** class is actually a data structure for storing the summaries generated by the Summarizer class. It calls ChartGenerator class only if requested by the user and calls the Database class at all cases.
- **ChartGenerator** generates bar, column and pie charts for visual interpretation of summaries formed from the connection data received by the Summarizer class.
- **Database** class establishes connection with the ACCIPP database and creates queries in order to retrieve data from, insert data into and update fields of the database. Its methods use these queries to add all connection data received and eventually calls the EMail, WebPage, InstantMessaging, News and Unknown classes for further classification of the connection data.
- **EMail** class is a type of a Connection class and is used for storing summaries related to E-mail protocols such as POP3, IMAP and SMTP.
- **WebPage** class is a type of a Connection class and is used for storing summaries related to Web Page protocols such as HTTP.
- **InstantMessaging** class is a type of a Connection class and is used for storing summaries related to Instant Messaging protocols such as MSN, YMSG and JABBER.
- **News** class is a type of a Connection class and is used for storing summaries related to News protocols such as NNTP.
- **Unknown** class is a type of a Connection class and is used for storing summaries that cannot be classified into one of the four classes, i.e EMail, WebPage, InstantMessaging and News. As the AutoSensing feedback mechanism operates, instances of the Unknown class will eventually be deleted from the Unknown class and added to one of the four other classes mentioned above.

- **File** class is called whenever the need for storing the attached files and/or contents of the EMail, WebPage, InstantMessaging and News classes arises.

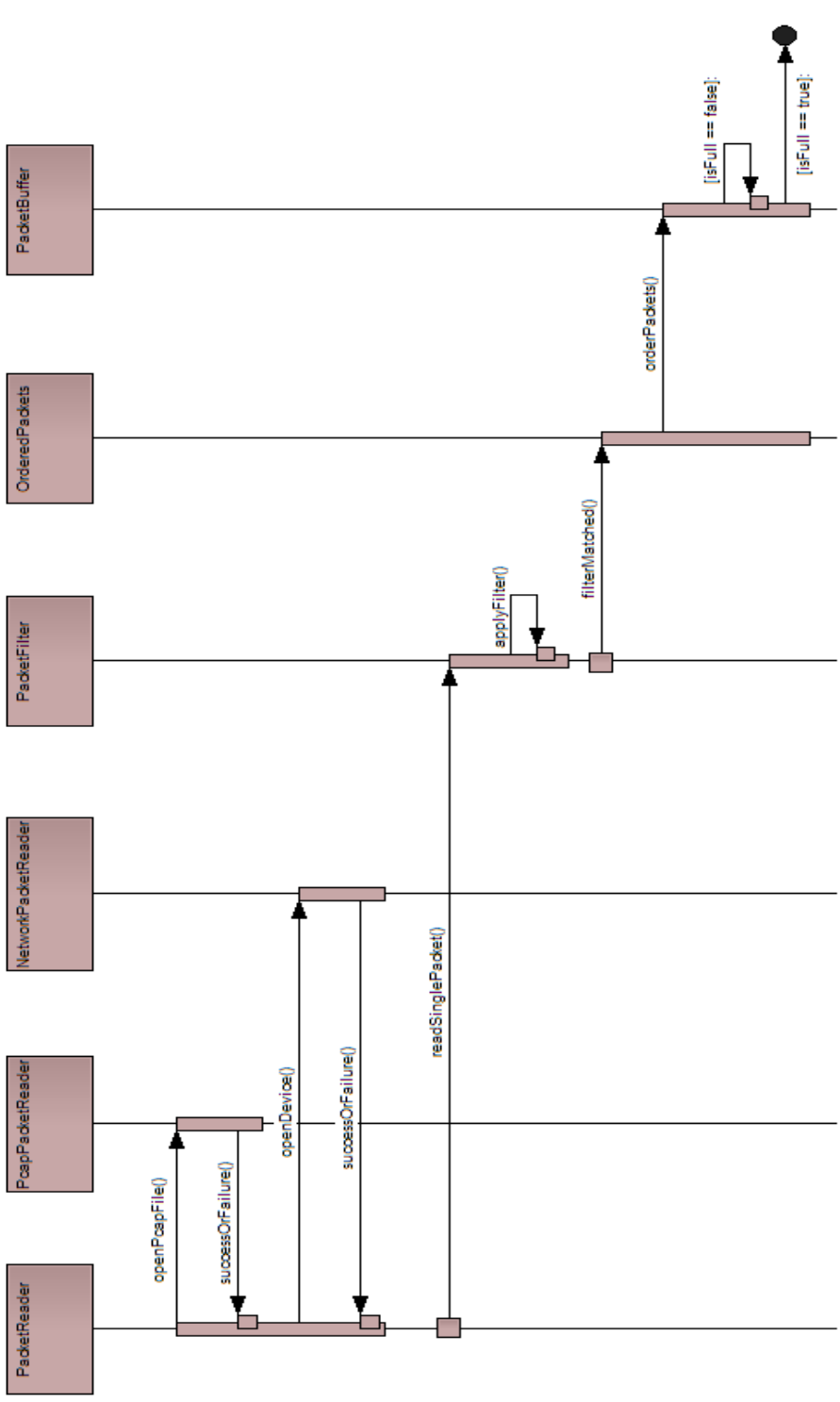
7.3 Sequence Diagrams

Sequence diagrams for Output and Decoder Mechanisms are below.

7.3.1 Sequence Diagrams for Output



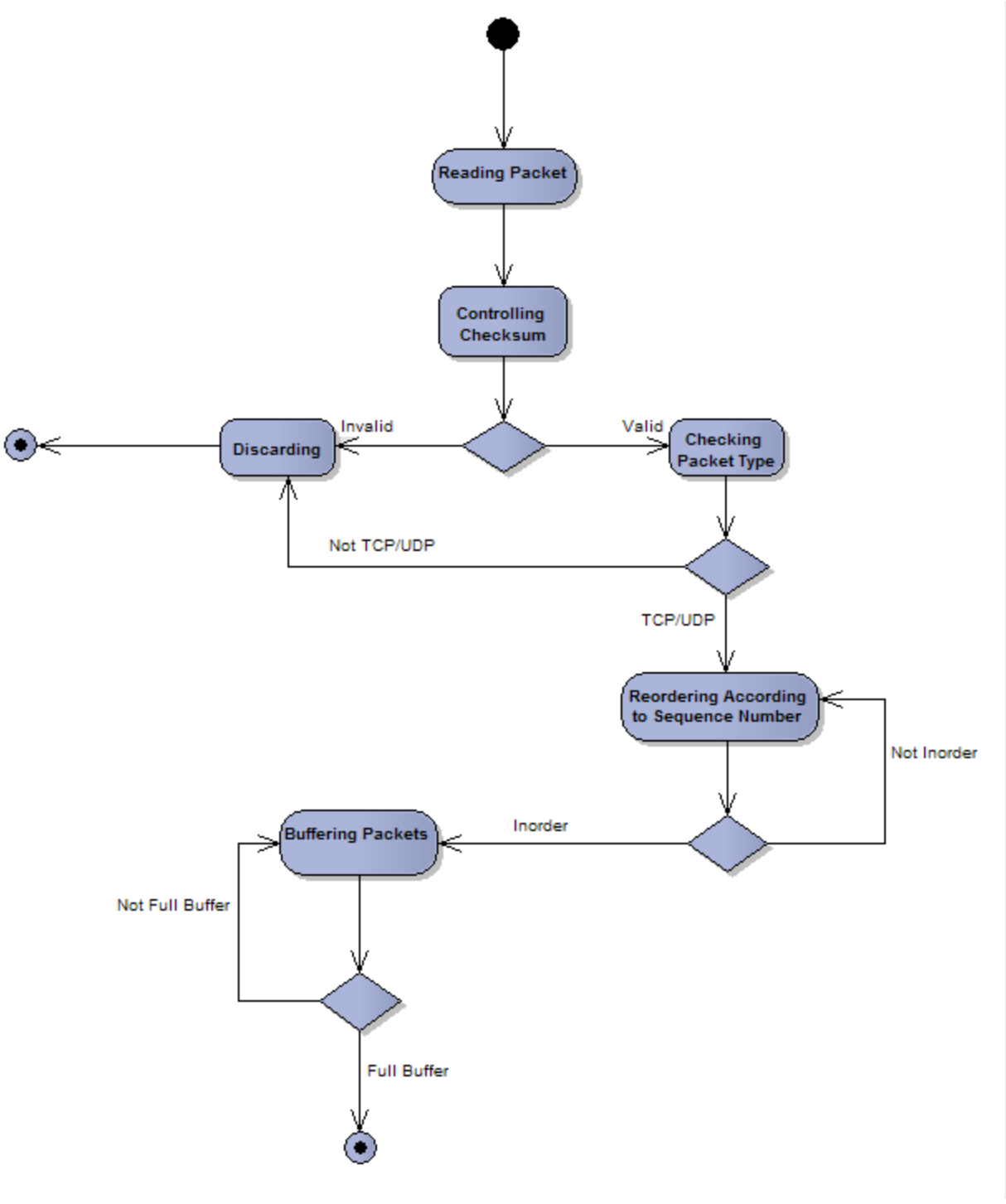
7.3.2 Sequence Diagrams for Decoder



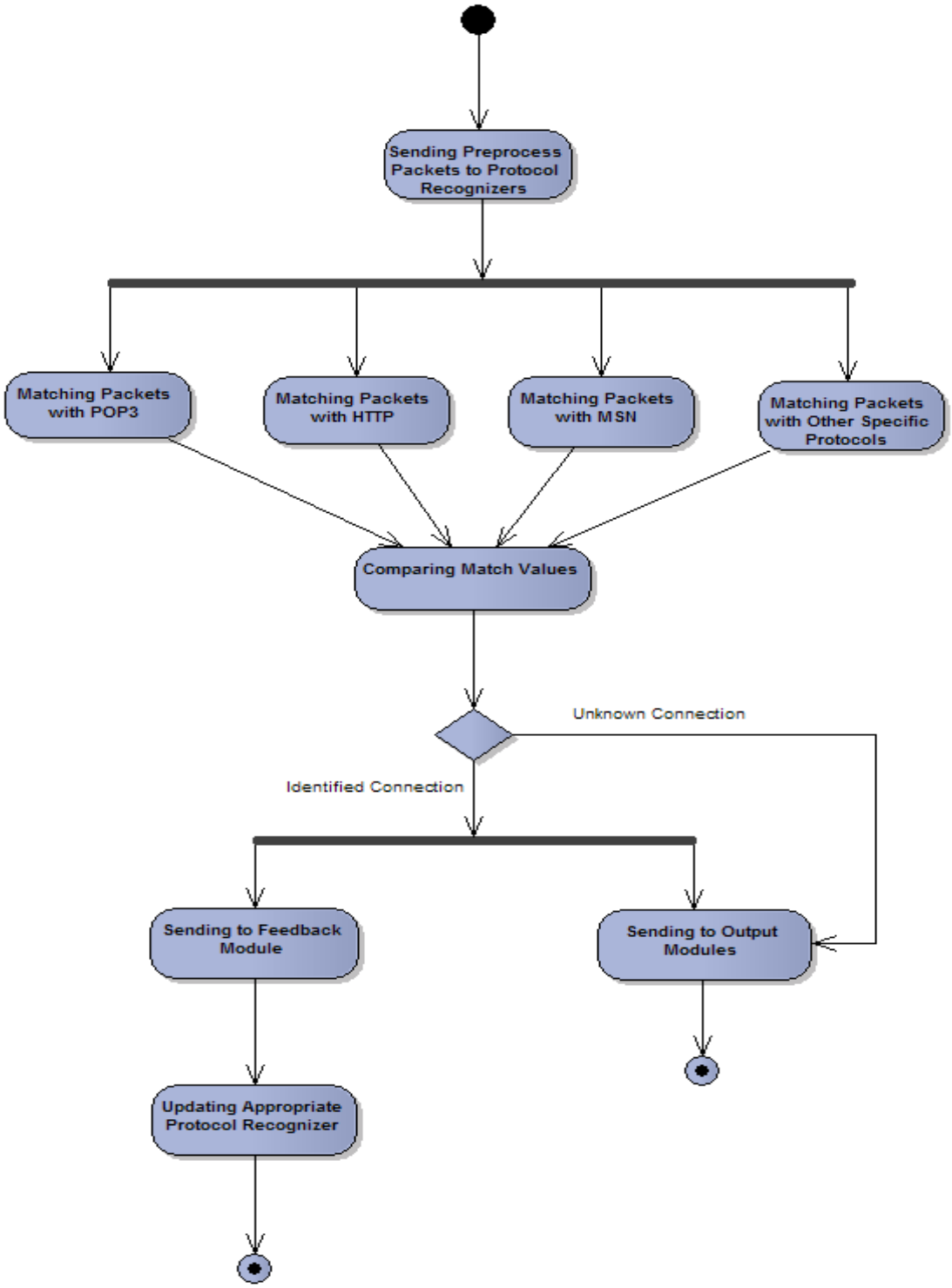
7.4 Activity Diagrams

Activity diagrams for Decoder and Auto-Sensing Mechanisms can be found below.

7.4.1 Activity Diagram of Decoder



7.4.2 Activity Diagram of Auto-Sensing



8. Testing Strategy and Procedures

As Edsger DIJKSTRA states, "Program testing can be used to show the presence of bugs, but never to show their absence!". Keeping this in mind, in order to have a program as free of bugs as possible, it is a must to have a good testing plan. Testing is the vital tool for quality assurance, validation and verification procedures. By validation it is meant if what has been specified is what the user actually wanted whereas by verification it is meant if the software is conformed and consistent with an associated specification. The testing before development of the software consists of deciding upon a testing strategy and future testing procedures. Although it is practically impossible to prove that no more errors exist, the more errors will be found as more tests are conducted and the rate of finding new errors will decrease as the testing process continues, to ensure the quality of the developed software in terms of correctness, reliability and efficiency, testing plan and procedures have been developed.

8.1 Testing Strategy

When designing test cases not regarding the database part, *white box* point of view has been taken since the tester, actually being the team members, has access to the internal data structures, code and algorithms. Thus, although ideally meaning to test every branch in the code with every combination of input values, it is planned to do a reasonable amount of testing while trying to cover a meaningful representation of the complete picture. On the other hand when designing test cases regarding the database parts *grey box* testing will be used since the tester has control over the input, inspects the value in a MySQL database, and the output value, and then compares all three (the input, mysql value, and output), to determine if the data got corrupt on the database insertion or retrieval.

Since ACCIPP has different layers and modules, testing phase should be conducted in a *bottom-up* manner as the project is concerned as a whole. However, testing each module in each layer separately requires a different testing strategy, i.e. *top-down* testing.

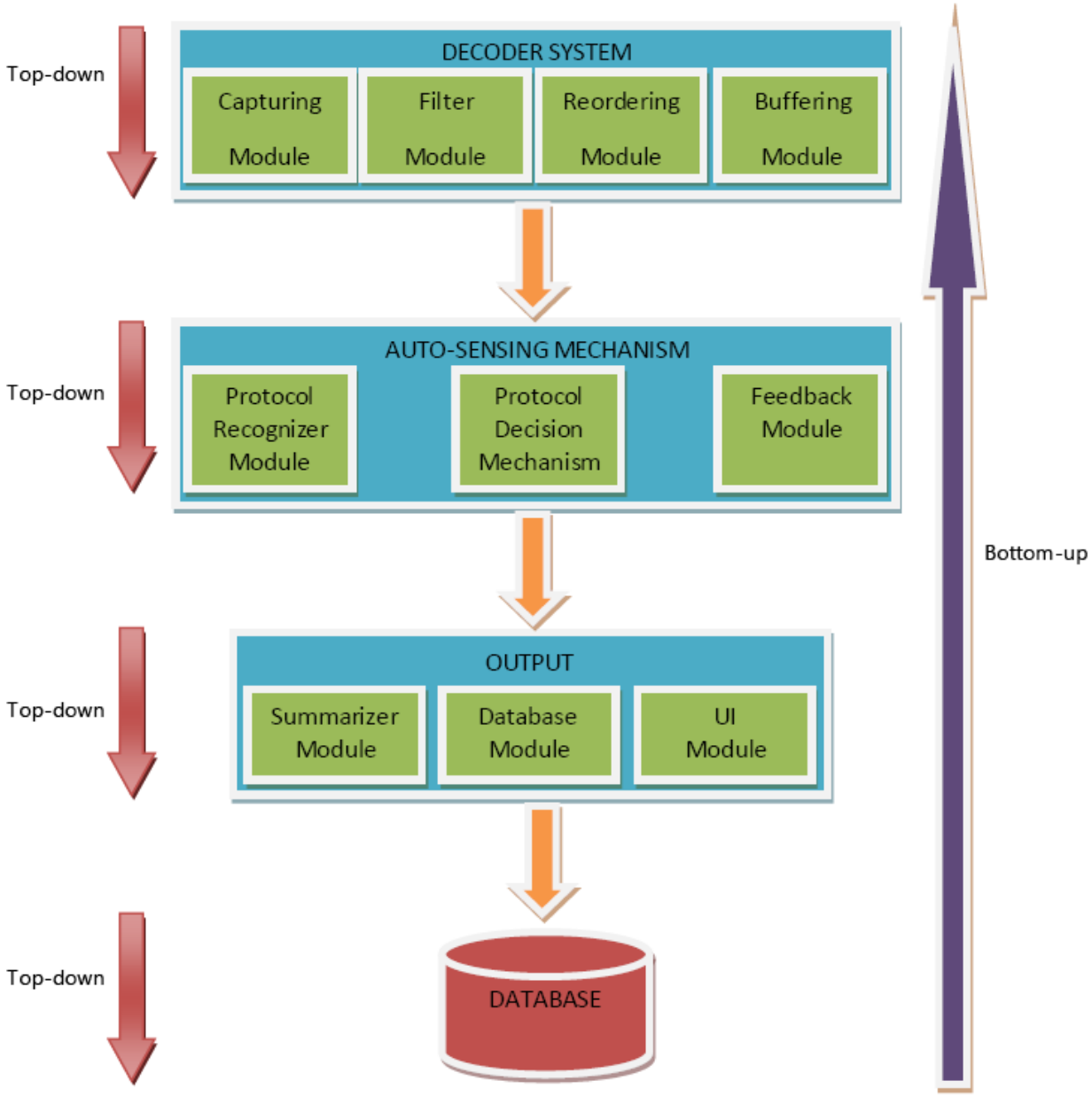


Figure : Testing Strategy of ACCIPP

8.2 Testing Procedure

The following procedures are applied in accordance with the testing strategy.

8.2.1 Unit Testing

In unit testing, minimal software component, i.e. module, is tested by *white-box testing* to verify that the detailed design for the module has been correctly implemented. This way, since the internal coding structure is visible, the tester is able to optimize the code and decide upon which type of input is more helpful in testing the application effectively. As each module or a sub-module is developed the unit testing will be carried out. Usually the member who developed the specified module or sub-module is in charge. However as being a software developer group, everybody is an end-tester to each module or sub-module developed by the other group members.

8.2.2 Integration Testing

Integration tests are different from unit tests in that it includes the testing of flow of operations, whole-part structure during lifetime and appropriate delegation and cascading behavior using infrastructure conditions and failure scenarios. The general aim of this type of testing is to determine if simultaneously running modules function together correctly.

9. Syntax Specification

Projects are not daily, simple work. So any software project should be coded properly. The word proper does not only stand for working good but also easy to read and understand, add to, maintain and debug.

There may be cases where one project member may stop developing his/her part and decide to return to it several weeks later or hand development over to another member. In

these cases both that member and the other developers will want to be able to understand the code.

As a result, after consulting with all team members and compromising and incorporating elements of everyone's style a group of coding standards have been decided upon. These standards help the readability and maintainability of the code by basically enforcing syntactical constraints and forbidding the use of complex language functions/construct that are quicker to write but affect the mentioned factors.

Consequently, with the help of the CVS and the following syntax specifications, these aims are planned to be achieved.

9.1 Naming Classes

All classes will have names beginning with capitalized letters, and the classes with names containing more than one word will have names where each word's first letter will be capitalized. Some example class names are as follows: "News", "NewsProtocol".

9.2 Naming Functions

The functions will be named so that each function name starts with a lower-case letter, until a new word starts. Each new word in the variable name, starts with a upper-case letter. For example "getConnectionId()" is suitable for a function name.

9.3 Naming Variables

Appropriate choices for variable names are seen as the keystone for good style. Poorly-named variables make code harder to read and understand. As a result, all variables begin with a lower-cased word, and if consisting of multiple words, the rest is capitalized. Some variable examples are: "protocolName", "comment".

9.4 Comment Conventions

Commenting is also a vital issue considering the understandability of the code. Since each C++ class is defined in separate files, detailed information about each class is included at the beginning of each file in the following format:

```
/*****************************************************************************
File Name:
Author:
Date/Time: (Date - DD/MM/YYYY , Time - HH:MM:SS)
Modified By:
Modified At: (Date - DD/MM/YYYY , Time - HH:MM:SS)
Description:
*****************************************************************************/
```

In addition to this, end of line comments which describe the code on that line only are written in accordance with the following convention: [2]

```
xIncrement *= -1;   // change horizontal direction
```

On the other hand, line comments that describe the purpose of a number of lines of code are written in accordance with the following convention: [2]

```
// Move the point in the current direction
y += yIncrement;
x += xIncrement;
```

9.5 MySQL Conventions

ACCIPP database and the related queries will be coded using MySQL, so some simple rules for MySQL conventions have also been decided upon. Basically, these rules are:

1. Avoid keywords in field names at all costs which will probably simplify the queries and save rework later.

2. Use case sensitivity in MySQL statements where key words are always capitalized and non-key words are cased as appropriate to the field names.
3. Using stored queries and procedures wherever possible since they are designed for optimal use and will help us save time.
4. Field names consisting of a single word are lower-cased and the ones with multiple words are also lower-cased with the words other than the first word being capitalized.

10. Project Schedule

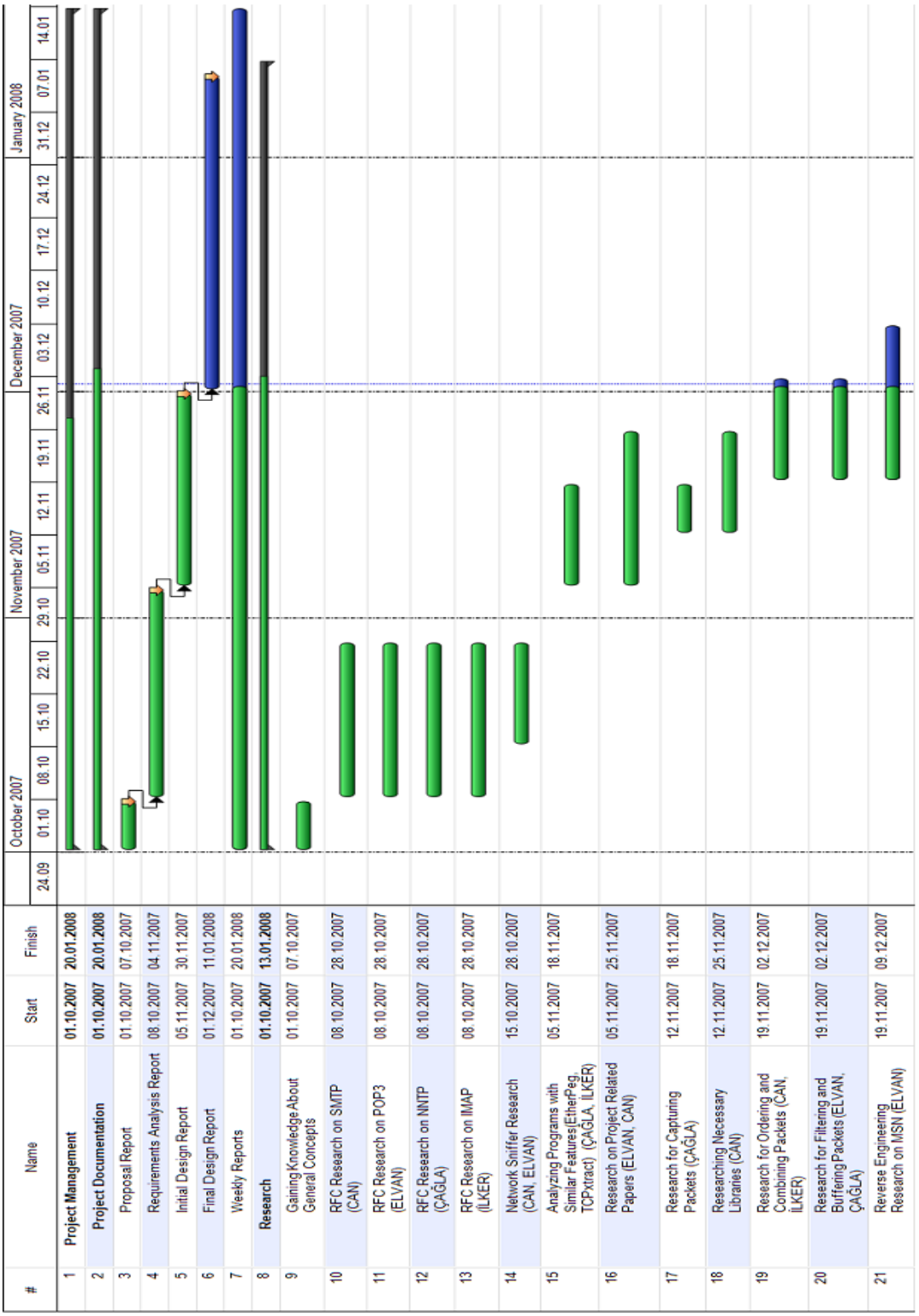
Gantt chart is used to visualize the schedule of ACCIPP including only the first term of the project.

10.1 Gantt Chart

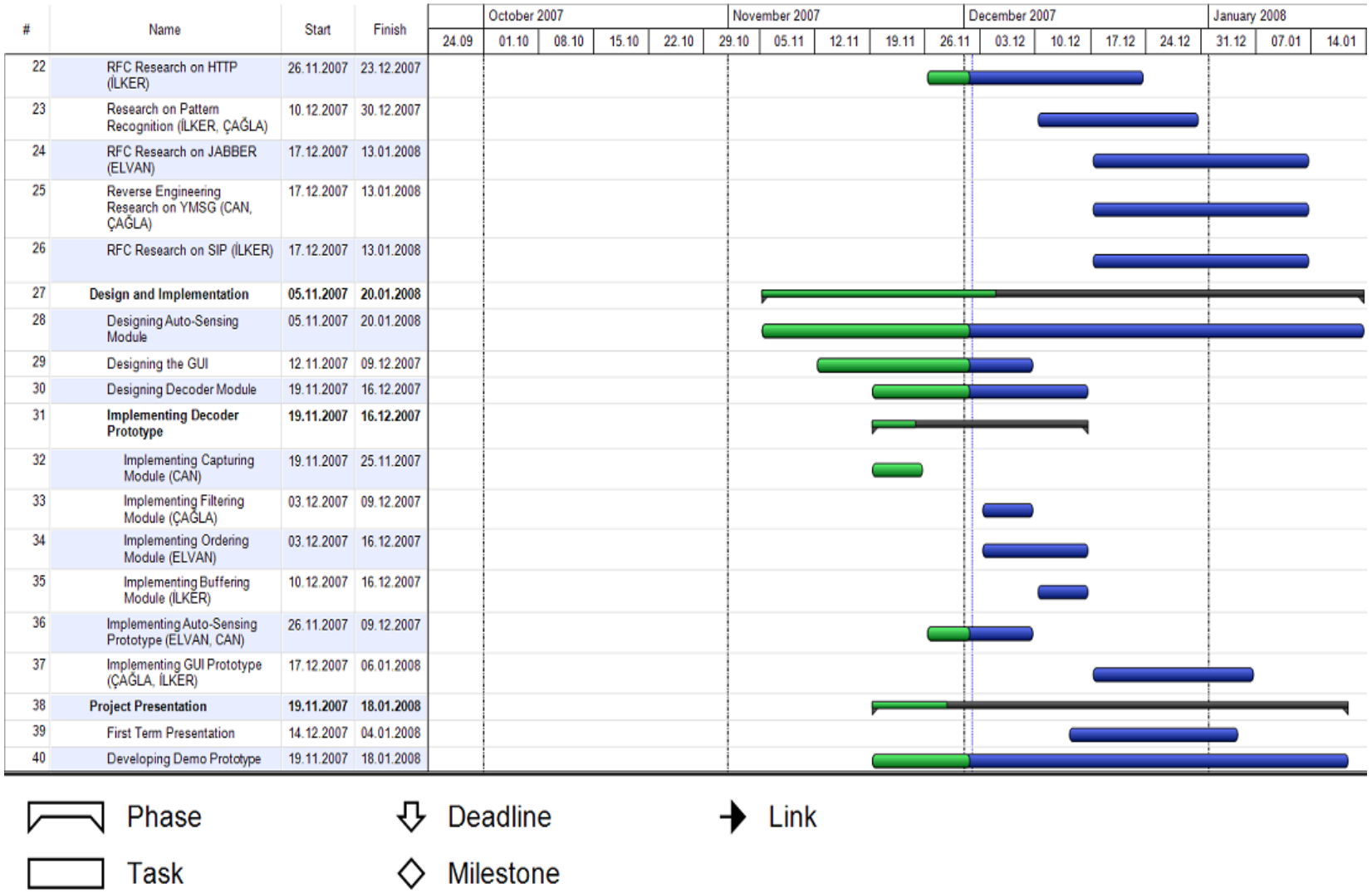
The Gantt chart of the project can be found in Appendix, 11.1.

11. Appendix

11.1 Gantt Chart



#	Name	Start	Finish
1	**Project Management**	01.10.2007	20.01.2008
2	**Project Documentation**	01.10.2007	20.01.2008
3	Proposal Report	01.10.2007	07.10.2007
4	Requirements Analysis Report	08.10.2007	04.11.2007
5	Initial Design Report	05.11.2007	30.11.2007
6	Final Design Report	01.12.2007	11.01.2008
7	Weekly Reports	01.10.2007	20.01.2008
8	**Research**	01.10.2007	13.01.2008
9	Gaining Knowledge About General Concepts	01.10.2007	07.10.2007
10	RFC Research on SMTP (CAN)	08.10.2007	28.10.2007
11	RFC Research on POP3 (ELVAN)	08.10.2007	28.10.2007
12	RFC Research on NNTP (ÇAĞLA)	08.10.2007	28.10.2007
13	RFC Research on IMAP (İLKER)	08.10.2007	28.10.2007
14	Network Sniffer Research (CAN, ELVAN)	15.10.2007	28.10.2007
15	Analyzing Programs with Similar Features(EtherPeg, TCPxtract) (ÇAĞLA, İLKER)	05.11.2007	18.11.2007
16	Research on Project Related Papers (ELVAN, CAN)	05.11.2007	25.11.2007
17	Research for Capturing Packets (ÇAĞLA)	12.11.2007	18.11.2007
18	Researching Necessary Libraries (CAN)	12.11.2007	25.11.2007
19	Research for Ordering and Combining Packets (CAN, İLKER)	19.11.2007	02.12.2007
20	Research for Filtering and Buffering Packets (ELVAN, ÇAĞLA)	19.11.2007	02.12.2007
21	Reverse Engineering Research on MSN (ELVAN)	19.11.2007	09.12.2007



#	Name	Start	Finish
22	RFC Research on HTTP (İLKER)	26.11.2007	23.12.2007
23	Research on Pattern Recognition (İLKER, ÇAĞLA)	10.12.2007	30.12.2007
24	RFC Research on JABBER (ELVAN)	17.12.2007	13.01.2008
25	Reverse Engineering Research on YMSG (CAN, ÇAĞLA)	17.12.2007	13.01.2008
26	RFC Research on SIP (İLKER)	17.12.2007	13.01.2008
27	**Design and Implementation**	**05.11.2007**	**20.01.2008**
28	Designing Auto-Sensing Module	05.11.2007	20.01.2008
29	Designing the GUI	12.11.2007	09.12.2007
30	Designing Decoder Module	19.11.2007	16.12.2007
31	**Implementing Decoder Prototype**	**19.11.2007**	**16.12.2007**
32	Implementing Capturing Module (CAN)	19.11.2007	25.11.2007
33	Implementing Filtering Module (ÇAĞLA)	03.12.2007	09.12.2007
34	Implementing Ordering Module (ELVAN)	03.12.2007	16.12.2007
35	Implementing Buffering Module (İLKER)	10.12.2007	16.12.2007
36	Implementing Auto-Sensing Prototype (ELVAN, CAN)	26.11.2007	09.12.2007
37	Implementing GUI Prototype (ÇAĞLA, İLKER)	17.12.2007	06.01.2008
38	**Project Presentation**	**19.11.2007**	**18.01.2008**
39	First Term Presentation	14.12.2007	04.01.2008
40	Developing Demo Prototype	19.11.2007	18.01.2008

Name	Start	Finish
Project Management	18.02.2008	15.06.2008
Project Documentation	18.02.2008	15.06.2008
Weekly Reports	18.02.2008	15.06.2008
Implementing ACCIPP	18.02.2008	30.05.2008
Implementing Decoder	18.02.2008	29.02.2008
Capturing Module(CAN)	18.02.2008	25.02.2008
Filter Module(ÇAĞLA)	18.02.2008	25.02.2008
Reordering Module(ELVAN)	18.02.2008	29.02.2008
Buffering Module(ILKER)	18.02.2008	25.02.2008
Implementing Auto Sensing	25.02.2008	30.05.2008
Protocol Recognizers	25.02.2008	09.05.2008
SMTP (CAN)	25.02.2008	16.03.2008
POP3 (ELVAN)	01.03.2008	20.03.2008
NNTP (ÇAĞLA)	25.02.2008	16.03.2008
IMAP (ILKER)	25.02.2008	16.03.2008
HTTP (CAN,CAGLA,ELVAN,ILKER)	16.03.2008	10.04.2008
MSN (ELVAN, ILKER)	10.04.2008	25.04.2008
YMSG (CAN, ÇAĞLA)	10.04.2008	25.04.2008
SIP (ILKER, CAGLA)	25.04.2008	09.05.2008
JABBER (ELVAN, CAN)	25.04.2008	09.05.2008
Protocol Decision Mechanism (CAN, ÇAĞLA, ELVAN, ILKER)	17.03.2008	30.05.2008
Feedback Module (CAN, ÇAĞLA, ELVAN, ILKER)	17.03.2008	30.05.2008
Implementing GUI	17.03.2008	30.05.2008
Main Window (ÇAĞLA)	17.03.2008	30.05.2008
Short Summary Pane (ELVAN)	17.03.2008	30.05.2008
Summary Popup Window (CAN)	17.03.2008	30.05.2008
Chart Generator Module (ILKER)	17.03.2008	30.05.2008
Help (CAN, ÇAĞLA, ELVAN, ILKER)	17.03.2008	30.05.2008

12. References

[1] Dreger H., Feldman A., et.al, "Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection"

[2] http://virtual.parkland.edu/sbadman/00000007Fall/SuperSymplifiedCSyntaxSpecification.htm