Visualization of Passively Extracted HL7 Production Metrics

Ricardo Jorge Teixeira Ferreira
Mestrado Integrado Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência dos Computadores
2014

Orientador
Prof. Dr. Manuel Eduardo Correia, Professor Auxiliar, FCUP

Coorientador
Prof. Dr. Ricardo Cruz Correia, Professor Auxiliar, FMUP

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

This project is dedicated to all my family and friends, for their presence and unconditional love have been essential in allowing me to reach this moment...

Ricardo Jorge Ferreira

June 2014

Acknowledgments

I would like to thank my supervisors and mentors, Professors Manuel Eduardo Correia and Professor Ricardo Correia for all the support given during the development of this work.

Also, a word of appreciation is due to all the working team at C3P/HLTSYS for giving all sorts of support for this project.

Resumo

Os centros hospitalares têm vindo a assistir a um enorme desenvolvimento ao nível das suas infra-struturas informáticas, o que levou à criação de uma panóplia de diferentes aplicações que são, hoje em dia, essenciais para o bom funcionamento das instituições de saúde. Contudo, inerente a cada uma dessas aplicações, existe um conjunto muito considerável de informação que está constantemente a ser criada e posteriormente arquivada pelos mais diversos sistemas de informação hospitalares. Essa mesma informação permite, de uma forma priveligiada aferir importantes métricas relacionadas com o nível de produtividade dos vários serviços de cada centro hospitalar.

Nesta tese apresentamos uma proposta para um sistema capaz de apresentar métricas relacionadas com o nível de produtividade de um centro hospitalar através da extracção e reconstrução passiva de fluxos TCP que contenham mensagens HL7 ou outros tipos de protocolos utilizados em eHealth. Com base nessas mensagens, o nosso sistema é capaz de extrair informação útil e com ela, construir uma base de dados de conhecimento relativo à infra-estrutura hospitalar em análise.

As várias mensagens HL7 presentes na rede informática hospitalar contêm informação útil com a qual é possível produzir importantes dados estatísticos relativos à produtividade dos processos de negócio. A dificuldade de extrair dados de um grande conjunto de sistemas heterogéneos pode assim ser contornada através da extracção passiva de pacotes IP, que contenham mensagens HL7, directamente e de uma forma não intrusiva, a partir da rede hospitalar.

O nosso sistema foi colocado numa infra-estrutura hospitalar de grandes dimensões localizada na cidade do Porto, em Portugal, onde foram extraídas mensagens HL7 directamente de rede hospitalar. O nosso sistema extrai e analisa uma média diária de 44.000 mensagens HL7 com vários picos na ordem das 1.100 mensagens por minuto. Com base neste tráfego, o nosso sistema é capaz de determinar e apresentar de forma gráfica a distribuição temporal de várias actividades hospitalares como pedidos de

análise, marcações de consultas ou ainda informação relacionada com facturação, entre outros.

Abstract

Healthcare facilities have been improving their information systems over the past few years. Such improvements led to the creation of a multitude of different applications essential to the facilities services. Associated with the various applications, there's also a considerable amount of information being produced and stored throughout the facility. Such data constitutes a privileged way of inferring past and current performance metrics of a given healthcare facility for it's different activity domains. However, complex challenges arise when trying to gather all the different data from all the systems scattered throughout the facility.

We present a proposal for a system capable of displaying production metrics in a healthcare facility by passively extracting IP packets from the network and reconstruction TCP streams containing HL7 compliant messages and other eHealth relevant network protocols. Based on those messages our system is able to extract meaningful data and with it, it is possible to produce a knowledge database for a given healthcare facility.

The HL7 messages moving over the network contain information that can be used to assess many relevant production metrics for a given infrastructure. The challenge of having to query a considerable amount of different systems in order to gather such data can be solved by passively extracting packets containing HL7 standardized messages or other eHealth related protocols directly from the network.

We have deployed our system in a large healthcare facility located in Porto, Portugal where we've been passively extracting HL7 messages from their network infrastructure. Our system extracts and analyses a daily average of 44,000 HL7 messages with several peaks of 1,100 messages per minute. Based on such network traffic, our system has been able to infer the daily distribution of healthcare related activities such as lab orders, appointment scheduling and also billing information, among other relevant business metrics.

Acronyms

AJAX Asynchronous JavaScript and XML. 42

ANSI American National Standards Institute. 6

API Application Programming Interface. 13, 39, 41, 42

ASCII American Standard Code for Information Interchange. 21, 25

DICOM Digital Imaging and Communications in Medicine. 9, 56

DPI Deep Packet Inspection. 11

EHR Electronic Health Record. 1, 6

FTP File Transfer Protocol. 8

HIS Hospital Information System. 1, 2, 5, 6, 8

HL7 Health Level Seven. 3, 6–9, 13–15, 18, 20–22, 24–40, 42–45, 47–50, 52, 54, 56, 57

HTML HyperText Markup Language. 39

HTTP Hypertext Transfer Protocol. 8, 9, 13, 38–40

IDS Intrusion Detection System. 10, 11, 55

IPS Intrusion Prevention System. 10, 11, 55

IT Information Technologies. 6, 7, 53, 55

JSON JavaScript Object Notation. 39–41

MSH Message Header. 6, 7, 28

NCPDP National Council for Prescription Drug Programs. 9

NIC Network Interface Controller. 17, 24, 45, 46, 55, 56

OSI Open Systems Interconnection. 17, 18

PACS Picture Archiving and Communication System. 1

PID Patient Identification. 6

PV1 Patient Visit. 7

RESTful Representational State Transfer. 13, 38–41, 45

SSD Solid-State Drive. 56

TCP Transmission Control Protocol. 8, 9, 11, 15, 16, 18–24, 54–57

URI Universal Resource Identifier. 38, 40

XML Extensible Markup Language. 7

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rapid development of new information technologies and its adoption by healthcare facilities has made way to the rise of eHealth as a mature new area of research. EHealth technologies are radically transforming healthcare facilities by revolutionizing the way they produce and process useful business information, which allows them to improve their service efficiency, reduce internal costs and more importantly help to provide better service to its patients [30, 9]. This type of developments are also strongly backed up by governments searching to invest in ways to improve their healthcare systems and at the same time reduce their maintenance costs [8, 14].

Recent investments made in eHealth have allowed the development of many Hospital Information Systems (HISs). Those same technologies have been crucial in helping deploy tools such as the Electronic Health Records (EHRs), Picture Archiving and Communication Systems (PACSs) or even electronic prescription systems. While there is undergoing research evaluating the real impact of these new technologies in healthcare systems [35, 14], their mere usage has left hospital facilities highly dependent on numerous different information systems, each playing a different role in the everyday activities of the facility

Because of their criticity, these systems require effective monitoring mechanisms in place so that in case of a failure, it is possible to quickly restore them to its normal performance levels. Monitoring mechanisms also need to be placed and configured accordingly to the needs of each different system. Bearing in mind that healthcare facilities already possess a considerable amount of heterogeneous systems, each working in completely different ways, the development of such a comprehensive monitoring system constitutes a very costly and complex task.

There is however and indirect complementary way of monitoring those systems. It consists of directly analysing the network messages they exchange between them in an unobtrusive way and therefore extract several meaningful metrics that can be used to build an historic perspective for normal system behaviour. The monitoring of these systems can then be made by determining whether in a certain point in time, the current values for production metrics fall within the average values for the historic values of these metrics.

1.1 Motivation

With each different system, the healthcare facility is also left with a big amount of heterogeneous data scattered throughout its HISs. Important data such as application logs, can also be considered as an important source of metrics to assess the good functioning of the healthcare facility.

Such data can also be used to deduce meaningful information and trends about the daily activity of the healthcare facility. For example, suppose one of the deployed systems is used to schedule medical appointments. We could easily use the system's logs to try and identify the number of appointments scheduled during a certain period of time. Such methods when generalized to a considerable number of different systems, could easily produce significant data that could be used to build a knowledge database from which meaningful performance metrics for the healthcare facility could be easily produced [31].

The existence of a knowledge database for healthcare facilities containing performance metrics for their everyday activities can be considered an important asset not only from a monitoring point of view, but also from a management perspective. Such information can be used to support more informed decision making about hospital day to day management.

In order to build a knowledge database with more meaningful data, one could take advantage of the logs produced by each system present in the healthcare infrastructure. However, we would need to develop a system capable of extracting the log files produced by all the HISs. However, this approach presents some problems. The data used to build the database would be completely dependent of the quality of the log files produced, and the extraction system would need to be able to interact with all the different systems present in the healthcare facility, many of which have very poor log facilities and cannot be easily improved.

Moreover, since heterogeneous systems use the network to exchange relevant business related information, this presents a valuable extraction point for the data we can use to build a more meaningful knowledge database from which new system with much more useful metrics can be derived

1.2 Proposed Solution

The work detailed in this thesis aims to provide healthcare facilities with a system capable of producing meaningful metrics by passively extracting network packets directly from the network infrastructure and build a knowledge database with the collected data from which new overall business metrics can be more easily derived.

Towards this goal, we take advantage of the integration techniques typically employed by healthcare facilities to promote interoperability amongst their heterogeneous systems. By directly analysing network IP packets carrying HL7 and other eHealth protocol messages, our system is capable of extracting meaningful data and build a knowledge database rich with performance indicators about the healthcare facility. Based on that same database, our system is then able to produce a series of charts that can be used to support more informed decision making.

1.2.1 Objectives

The main goals of this thesis is to develop a system capable of extracting relevant data from an hospital core network, and with it produce a knowledge database about the performance of the facility. We have the following objectives:

- **Technology Research.** Identifying the current state of the art technologies employed by hospital facilities and understanding the way their different components interact and communicate.

- **Architecture Design.** Design a system architecture capable of efficiently extract relevant data from the core network of an healthcare infrastructure in a more unobtrusive way.

- **Implementation.** Develop and code the necessary components in order to deploy the proposed architecture in a real healthcare facility.

- **Testing.** Deploy our system's implementation in a real large use case scenario and analyse the results thus obtained.

1.2.2 Features

The main features of our proposed system are as follows:

- **Dynamic Data Extraction.** The information we are trying to acquire from the network is gathered and archived in a dynamic way.
- **Network Independent.** The deployment of our systems' architecture is independent of the underlying network infrastructure of the facility. That is, as long as we can place a node directly connect to a network where important hospital business metrics are transmited, our system is capable of extracting the necessary information without the need to reconfigure any internal hospital system.
- **Centralized Data Access.** The gathered data can be accessed from a single point of our infrastructure, thus opening the possibility to create a series of other subsystems that could use the meaningul data of our system to produce useful services to the hospital institution.
- **Graphical Display.** Our system produces a series of actionnable charts from the gathered information, thus allowing a quick and direct analysis of the collected business related metrics from several hospital systems.

1.3 Outline

The next chapters of this thesis are organized as it follows:

Chapter 2 presents a brief overview of the current state of the art on technologies typically used in modern healthcare facilities, as well as a series of challenges felt by such institutions and their approach to solve them. In Chapter 3 we present a proposal for the architecture of our system followed by its implementation details and approaches used. The testing scenarios and the results obtained are detailed in Chapter 4. Lastly Chapter 5 presents some final remarks and lays the ground for future work.

Chapter 2

State of the Art

In the following sections we present an overview of the set of technologies that play an important role in the core development of our system and in its architecture. We present a series of standards used in the HISs and describe their importance for developing a more integrated monitoring and business intelligence system in the healthcare infrastructures.

2.1 Towards Integrating Healthcare Systems

Information systems present in modern healthcare facilities are essential tools to increase the proficiency of medical care services. In fact, Buntin et al. shows that for considerable larger healthcare organizations, the early investment in new health information technologies has led to numerous benefits such as cost savings, improvements in the overall performance of the facility physicians and even has shown encouraging potential to increase patients empowerments by promoting the engagement in their own treatment in more meaningful ways.[12].

Moreover, for academic research, Goldzweig et al. refer a substantial increase in publications related to the development and impact of healthcare information systems [24]. On the other hand, the authors in [24, 26], also refer the existence of an increasing number of publications related to “patient-focused applications” which represents a new side to the HIS, where the usage of such systems can fall almost entirely within the responsibility of the patient instead of the healthcare professional, thus fully promoting patient empowerment[20].

On another work, the authors of [10] detail the effort made by the United States of America to improve their own healthcare service. In fact, the author explains the substantial monetary investment made to take advantage of the HISs present in each healthcare infrastructure in order to build meaningful EHRs capable of gathering each patients clinical history. According to the author, the existence of such an electronic repository of data promotes each healthcare institution to engage in more communications and information exchanges processes, therefore allowing the institution, as well as its workers to share meaningful information, and increasingly become more efficient.

Recent efforts made by worldwide healthcare facilities have also promoted an increase in the parallel development of tools aiming to assist healthcare professionals in their everyday activities[18, 32, 26]. However, this particular tendency is seen as the cause for some problems in the long term. Namely, in [7], Barbarito et al. refer that healthcare facilities experience major difficulties when trying to exchange data among so many different and heterogeneous systems.

2.1.1 HL7 Standard

In order to try to solve the interoperability problems being felt in healthcare facilities, developments were made in order to advance and adopt medical Information Technologies (IT) standards for those type of services [7, 38].

One of the main standards employed for interoperability in medical IT infrastructures is the HL7 standard [17]. The development of HL7 began in 1987 by an American National Standards Institute (ANSI) accredited organization whose main goal, among others, was to implement a standard for the management of patient data that could be easily used by heterogeneous healthcare systems as a way to exchange data in more meaningful ways [19].

```
MSH|^~\&|FRP||||20130205180519||OML^021|68|T|2.5
PID|||495426445||Doe^Jon^|||M|||Street^^City^^^
OBR||11112||6^proteins
OBR||11111||4^urea
```

Figure 2.1: HL7 Version 2 Message Sample

Figure 2.1 represents an example of an HL7 message. According to the standard [27], each HL7 message is composed by a set of segments (Message Header (MSH), Patient

Identification (PID), Patient Visit (PV1), etc). Subsequently, each message segment is composed by a series of fields and sub-fields, that should contain the actual data the systems are trying to exchange. The HL7 functionality [27] is based on the existence of certain types of events that trigger the creation of new messages. As such, each new message created should somehow be the product of an action triggered either by a healthcare professional or by different systems trying to exchange information between them. Looking back to Figure 2.1 we can observe that one of the main fields present in the MSH segment is the message type and message *trigger event* sub fields, in this case OML and O21 respectively. By looking at the referred fields, one can easily determine what type of event triggered the creation of the message. In this specific case, a laboratory order triggered some system to create the HL7 message and then, that same message can be used to inform other systems inside the healthcare facility of the requested order.

Currently, version 2 of the HL7 standard is the most employed version among healthcare facilities worldwide [19]. When employing HL7 V2, the content of each message is encoded in ASCII, the standard also allows for the creation of a certain level of flexibility on what type of information passes on the HL7 segment fields. As a result, many fields are allowed either to contain vague information or contain no information at all. Although this type of flexibility can sometimes be desirable, when used without care, it can sometimes invalidate interoperability between different systems and therefore jeopardize the main intent for why HL7 was initially created. In order to rectify this situation, the new HL7 version 3 standard now uses a Extensible Markup Language (XML) language with much more precise syntax and semantics.

2.2 Healthcare Integration Engines

Assuming the existence of a standard for message exchanges in an healthcare IT infrastructure, the integration between heterogeneous systems can be made in two different ways. Either the sending system communicates directly to end receiver using a message standard or an interface integrating engine is introduced at the healthcare facility in order to interconnect several different systems.

In the first case, the usage of a message standard such as the HL7 may not be sufficient to assure that heterogeneous systems can exchange information. Such challenge arises due to the fact that even if software vendors in an healthcare facility both use HL7 as the message standard in their applications, they will hardly agree on the specific message and semantics format to use[15]. In practice, it is extremely difficult to implement an end-to-end integration model that encompasses all software vendors within an healthcare facility.

To allow different legacy systems to interconnect, healthcare facilities often employ interface engines[15] to solve interoperability issues between systems even if they use the same message exchanging standard. Interface engines are deployed as an intermediary between different systems. Their method of operation can often be resumed according to Figure 2.2. The interface engine starts by receiving an HL7 message from any HIS present in the healthcare infrastructure, applies a predefined transformation to the message based on its source and destination and finally proceeds to send it to one or more receiving entities.

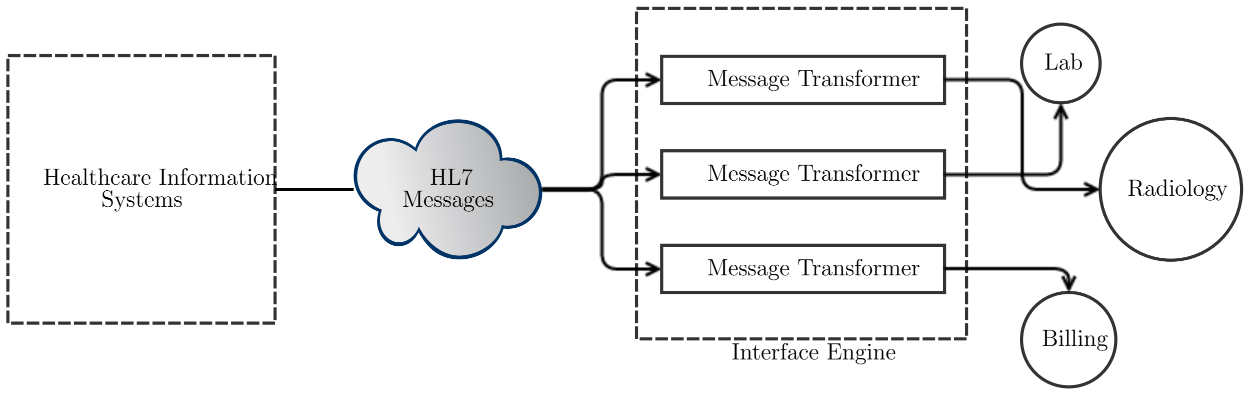

Figure 2.2: Interface Engine Operation

2.2.1 Mirth Connect Solution

An example of such interface engine is the Mirth Connect interface engine [34]. The Mirth Connect project is mainly supported by a company whose main goal is to develop health information systems capable of empowering hospital facilities with the latest trends in technology and health standards. Also, as an open-source based system, apart from the official company paid support, the Mirth Connect engine also has the advantage of being supported by a large worldly community of users and developers.

Among others, Mirth Connect is able to operate under several different network protocols such as Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP). More importantly, the system is able to understand several eHealh relevant protocols such as the HL7, Digital Imaging and Communications in Medicine (DICOM) or even the National Council for Prescription Drug Programs (NCPDP) standards.

One of the main advantages of using the Mirth Connect engine is the fact that apart from supporting several “official” eHealth standards, the system is also able to operate with any user customized message standard [11]. The feature of supporting raw ASCII text allows a developer to create several custom text delimited standards and have the Mirth connect reading those same messages. In doing so, the interface engine allows the exchange of meaningul information to be made in a more standard independent way.

Related to the message transformation and translation capabilities, when using the Mirth Connect engine users are presented with numerous possibilities [11]. Due to the fact that the system allows for the direct injection of JavaScript code to read, alter, and finally rebuild a new message, developers are presented with an easy and efficient way to apply any necessary transformations to any eHealth standard or custom set of messages.

2.2.2 Microsoft Biztalk Solution

Another interface engine typically used in eHealth scenarios is the Microsoft Biztalk server[33]. This server mainly functions as a business to business transaction tool, contrary to the Mirth Connect interface engine that was especially developed for eHealth scenarios.

As for supporting network communication protocols, like the Mirth Connect engine, the Biztalk server is also able to cope with the main protocols used like the TCP and HTTP protocols. However, one of its main disadvantages is the fact that supporting the different eHealth protocols is not a functionality provided by its base system. Instead, in order to provide the necessary support for eHealth related protocols like the HL7, the Biztalk server needs the instalment of additional modules often called “accelerators”.

Apart from the monetary costs associated with acquiring the platform, there’s also the costs of substantially losing the freedom to develop and adapt the base system according to the needs of hospital facilities since the Biztalk server was not built using and open-source philosophy.

2.3 Data Collection On Healthcare Facilities

Apart from difficulties in making different systems to interact together, healthcare facilities also faced other types of challenges. Namely Konrad et al. in [29] describe a recommendation system capable of automatically measuring the patients' variance from a clinical pathway which consists on the predefined course of treatment a given patient should take depending on its pathology. As such, in order to track the clinical pathway followed by each patient, the proposed system refers the need for the existence of a reliable source of data where clinical information about each patient is fetched and then analysed by the system. However, the authors point out that one of the main challenges to the implementation of such a system was precisely encountering a reliable source of information that already had all the necessary data aggregated for each patient. Instead, what the authors encountered was a multitude of sources containing the needed data, each containing the information in several different heterogeneous structures.

In respect to the collection and aggregation of patient data present throughout an entire healthcare infrastructure, authors in [36] address the challenges felt when trying to acquire meaningful data from across multiple systems' databases. The main challenge resided in the fact that each system stores the exchanged information with data schemas, models and even the query languages for databases, which vary widely according to each system.

2.3.1 Data collection By Passive Recollection of Data from the Network

By abstracting ourselves from the healthcare infrastructures and their own integration systems, there are several areas where data collection is also necessary in order to feed a given system with information. Such an example may be seen in areas such as network traffic classification or even in Intrusion detection Systems (IDSs) or Intrusion Prevention Systems (IPSs) where data is analysed through network "sniffing" techniques so that IP packets can be dynamically checked for suspicious content and therefore avoid potential network attacks.

In fact, passively extracting packets from a network and storing its data may also be considered as a way to aggregate and process data from different given systems, assuming they require network connectivity to exchange data.

Academic literature provides an extensive overview of traffic classification techniques employed by IDSs and IPSs[6, 13]. Such systems often rely on Deep Packet Inspection (DPI) techniques in order to classify data traffic flowing through a network. In respect to DPI techniques, one of the main challenges is associated with the processing costs of having to analyse each packet structure as well as its contents in order to accurately classify a given flow.

Cascarano et al. in [13] provides an insightful overview of several DPI techniques as well as a set of optimizations that aim to reduce the costs of thoroughly analysing each packet flowing through network without having to loose traffic classification accuracy. In respect to the evaluation of each packet, [6] claims that the usage of finite automata is the most widespread technique for packet pattern evaluation. Among others, string matching and regular expressions may also be employed.

However efficient, when receiving considerable amounts of out of order TCP packets, IPSs tend to start discarding packets since the actual reconstruction of the data flow may be considered too computationally expensive. On the subject of passive network traffic reconstruction there is an interesting report [22] describing a proposal for a system capable of relieving the need to create log files on end nodes of a messaging system . In order to implement such system, the authors placed an instance of the Snort IDS / IPS tool, logging packets from specific TCP sessions on the network. Those same log files were later used to reconstruct the set of messages transmitted between the different end systems in the network, extract a predefined set of metrics and then create a new better set of log files for the system.

Still on the subject of TCP stream reassembly tools, [23] refers that current implementations of stream re-assemblers are often seen on IPSs systems. However, from an architectural point of view, an IPS is placed in-line with the TCP flows on the network, so that connections can be cut off in case of an attack. The authors then proceed to explain the usage and improvements made to the *tcpflow* tool in order to develop a forensic tool capable of passively sniffing and reconstructing TCP streams.

By looking at the previous examples, methods that allow the TCP flow reconstruction are often employed in architectures where a dump file is created containing the sniffed packets from the network and is later used as an input to other systems responsible for the analysis and reassembly of each TCP stream.

Chapter 3

System Architecture

In this chapter, we describe the architectural components of a system capable of extracting performance metrics by passively monitoring the network traffic of an healthcare institution.

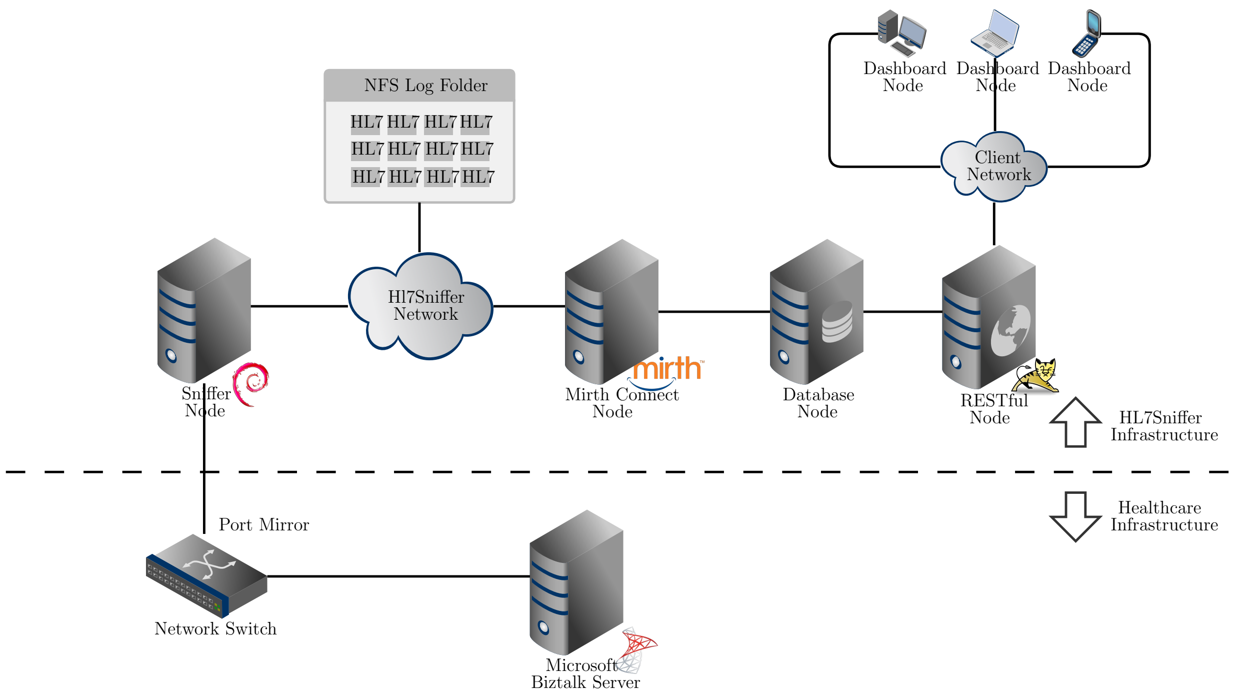

Figure 3.1: System Infrastructure

Our system is composed by five different nodes that work independently of each other. Figure 3.1 depicts the overall interactions between the different components of our architecture. They are:

- **Sniffer Node:** Responsible for passively extracting network packets, re-assemble HL7 messages and then store them in log/data files.
- **Mirth Connect Node:** Reads the log files produced by the "Sniffer Node", extracts a set of predefined fields from the HL7 message and stores them in the "Database Node".
- **Database Node:** Stores all the meaningful data extracted from each HL7 message gathered from the network infrastructure. The details related to the data model used are further detail in Section 3.3.
- **RESTful Node:** Receives HTTP requests and responds to the clients with information fetched from the database. In a general way, this node acts as a proxy for our knowledge database. As for the RESTful API we further detail its implementation in Section 3.5.
- **Dashboard Node:** Requests information from the "RESTful Node" and display a set of charts with the information received from IP packets extracted from the network.

By looking back to Figure 3.1 we can observe that our system's source of information is actually based at the network switch that connects to the hospital's main interface engine. That being the case, the only configuration requirement our system actually needs from the institution services is a simple *port mirror* at the network switch that will ensure that all the network traffic directed to the interface engine will be replicated to the switch port where our "Sniffer Node" is connected.

In the following sections, we start by showing our approach in adapting an existing tool which, in its final form, allows us to passively extract and reconstruct HL7 messages. The next step in our system consists on configuring an installation of the Mirth Connect application that can poll the data previously gathered by the "Sniffer Node", extract the appropriate metrics contained on each segment of the HL7 message and store them in the "Database Node". Finally, we have developed a dashboard application especially useful for monitoring purposes, capable of displaying visual information directly provided by our RESTful API. Examples of such visual information are the number of HL7 messages received during a given period of the day or even a weekly comparison for the number of received messages in each day of the week.

3.1 Data Collection

Modern healthcare facilities rely on interoperability architectures to improve the functionality of their services[37]. The use of such techniques aims to provide means that allow for different healthcare applications and systems to exchange meaningful data. As such, in order for this exchange of information to happen, we need a standard that different applications must use in order to communicate with each other.

In order to achieve such level of interoperability in eHealth, information systems rely, among others, on the use of the HL7 standard as one of the main tools used in order to allow different systems to exchange meaningful data[36]. The usage of a well defined message standard between several different application, presents an opportunity for us to obtain the necessary data that allows us to build a performance measuring system for healthcare institutions with many different applications.

On considerable sizeable infrastructures, the usage of such interoperability standards relies on the presence of an interface engine. Such piece of software is responsible for the transformation or translation of HL7 messages between different systems, therefore allowing different end systems to communicate using a standardized message pattern. The usage of such software presents a valuable opportunity for the extraction of the required data from the network for our system. Considering the fact that the HL7 message traffic needs to pass through and be logged by the interface engine, we could use those same logs to collect the data our system needs. However, such approach presents some potential disadvantages. Namely, by following this approach the quality of the data collected by our systems would always be subject to the quality or even the existence of the logs produced by the interface engine. Another potential downside to this approach concerns the fact that the extraction of the metrics our system needs would imply a direct intervention on the interface engine itself. Assuming that these particular points of the system, on considerable sizeable infrastructures are typically subject to strain due to the amount of computational processing they have to do, the positioning of our data collection point on these specific nodes would have a negative impact on the overall system performance. Even assuming that a particular interface engine is not subject to an excessive amount of computational strain and that it could endure the placement a data extraction mechanism, the fact that these systems are running in a production environment makes them a potential "untouchable" system since the slightest miscalculation in their configuration could lead to disastrous outcomes to the healthcare infrastructure normal functioning.

Since our initial goal aimed to provide a system for data extraction that would allow us to maintain a certain level of independence from the healthcare applications, a log file bound approach did not present itself as capable of complying with our requirements. However the existence of such interface engines on the network still represented our best line of approach for efficient data collection points. As such, we decided that the usage of network sniffing techniques on the network, where the interface engine receives all the HL7 messages, represented our best hope in achieving a data collection point that would comply with our initial goals. As such, this approach would still allow us to extract an accurate image of the data the interface engine has to process without the disadvantage of having to rely on the quality of its own logs or even having to add additional layers of processing to the interface engine that could potentially translate to an excessive strain to its often already strained processing capabilities.

3.1.1 Sniffing Process

Extracting the required data directly from the network at strategic points using sniffing techniques fills the necessary requirements for our system. However, this particular approach also means that our system needs to take additional steps in order to guarantee that the information extracted translates itself into an accurate copy of what the HL7 transformer receives. Namely, by using this approach, we need to reassemble out of order TCP packets in order to acquire HL7 messages exactly as seen by the facility integration engines.

In fact, the task of having to reorder TCP packets that are directly sniffed from the network presents itself as one of the biggest challenges in the design of our system. When trying to reconstruct HL7 messages from TCP packets we had to keep in mind that the HL7 standard may be used to transfer potentially large sets of data such as *pdf* files or in worst cases, heavy radiology imagery. This particular fact, causes a significant restriction in the design of our system. The reconstruction of all the TCP packets sniffed from the network could not be done in memory since we could easily end up consuming all the memory of the system in such cases where the HL7 message we were trying to reassemble consists of a extremely large set of data.

We have therefore decided to use a slightly modified version of the *tcpflow* [28] tool. This particular tool in its unmodified form, allows a user to sniff TCP packets directly from the network, at the same time that it reconstructs and logs all the data from the different TCP connections it detects flowing through the network. One of the most attractive features of *tcpflow* is its ability to reconstruct and log the data transmitted in a TCP connections without having to maintain all the data from each packet in memory.

Although *tcpflow* is able to provide us with the requirement of not using the system memory for the reconstruction of the data transmitted through different TCP connections, in its unmodified version, this tool also presents some potential drawbacks. Namely, *tcpflow* was developed so it could serve as a debugging tool. As such, it lacked certain dynamic aspects to its behaviour. For example, in its base implementation, *tcpflow* is able to determine the start of each new TCP connection but in contrast, it is unable to determine the end of a connection. As such, each file descriptor associated with the log file in question is always kept open in the underlying operating system. Therefore, as long as there was an instance of *tcpflow* running on the operating system, the log files produced by this tool could never be used by another entity and as such, we would never be able to give further treatment to the data extracted from the network. It was precisely this static aspect that we tried to change with our modified version of *tcpflow*.

In what follows, we present a more detailed description of the overall architecture and techniques used in the extraction and reconstruction of the necessary data flowing through the network and also all the main modifications we have made to the base implementation of *tcpflow*.

3.1.1.1 *tcpflow* Data Extraction Pipeline

tcpflow makes use of a small data structure that it calls a *flow* (Code Block 1) to track each active TCP connections on a given network. Each *flow* keeps information about each TCP connection such as its source and destination ip addresses, its source and destination ports, the first sequence number detected and also the path to the log file allocated for this TCP connection. Concordantly, all of these *flow* structures can be accessed by consulting an hash table where all the *flow* structures are kept in memory.

tcpflow also makes use of the popular *libpcap* library to be able to function as a network traffic capturing tool. By using such a library, *tcpflow* is able to capture all the packets in its raw form directly from the Network Interface Controller (NIC) connected to a given network. In a general way, each packet captured by the NIC, is passed to a specific function that is responsible for analysing the packet at a specific layer (network, transport, datalink, etc).

Figure 3.2 further illustrates how each packet captured is handled by the *tcpflow* tool.

```
typedef struct {
  u_int32_t src;    /* Source IP address */
  u_int32_t dst;    /* Destination IP address */
  u_int16_t sport; /* Source port number */
  u_int16_t dport; /* Destination port number */
  tcp_seq isn;      /* First seq number we've seen */
  FILE *fp;         /* Pointer to log file */
} flow_t;
```

Code Block 1: *flow* structure

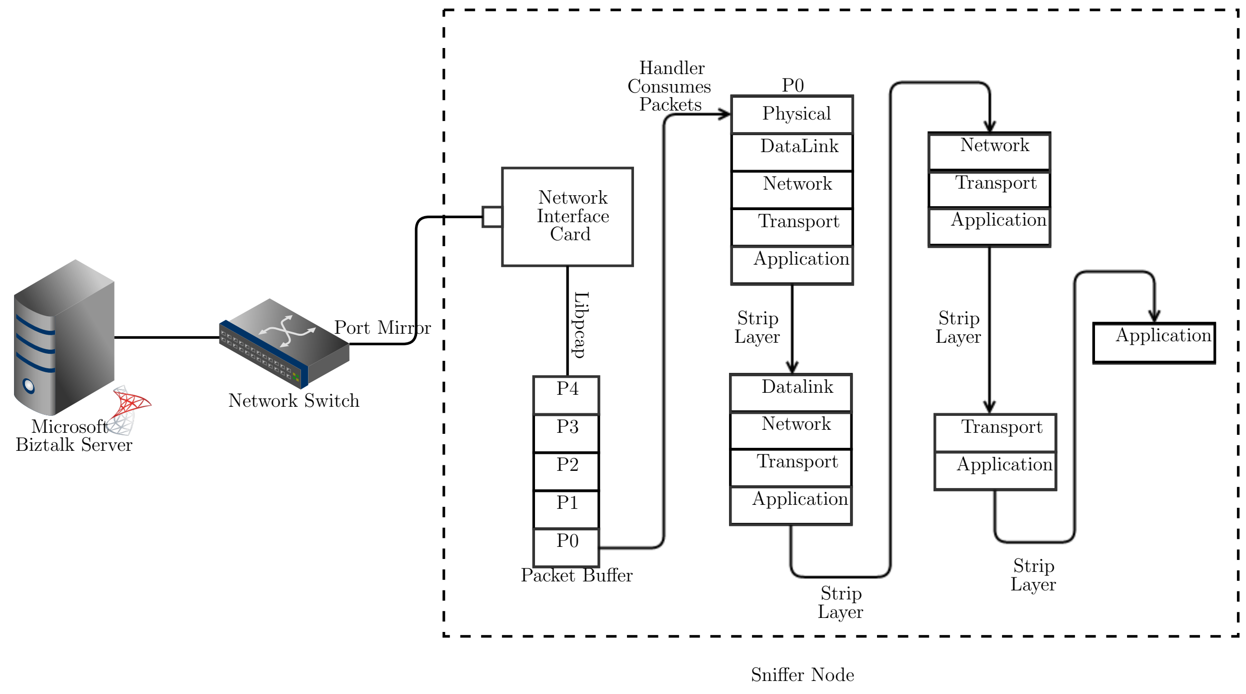

Figure 3.2: General packet handling

Each packet starts by being captured by the NIC and then passed to a series of functions each responsible for striping down a specific Open Systems Interconnection (OSI) layer of the packet while the core of the packet processing is then done at the transport and application layers. In respect to the transport layer, this precise point will contain all the necessary information our system needs to re-assemble out of order packets. That is, since TCP is the main protocol used to exchange HL7 messages between different systems, at the transport layer of the OSI model, where the TCP resides, our system is able to fetch the TCP sequence number so that we can determine the position of the data the received packet is transporting in the overall data set.

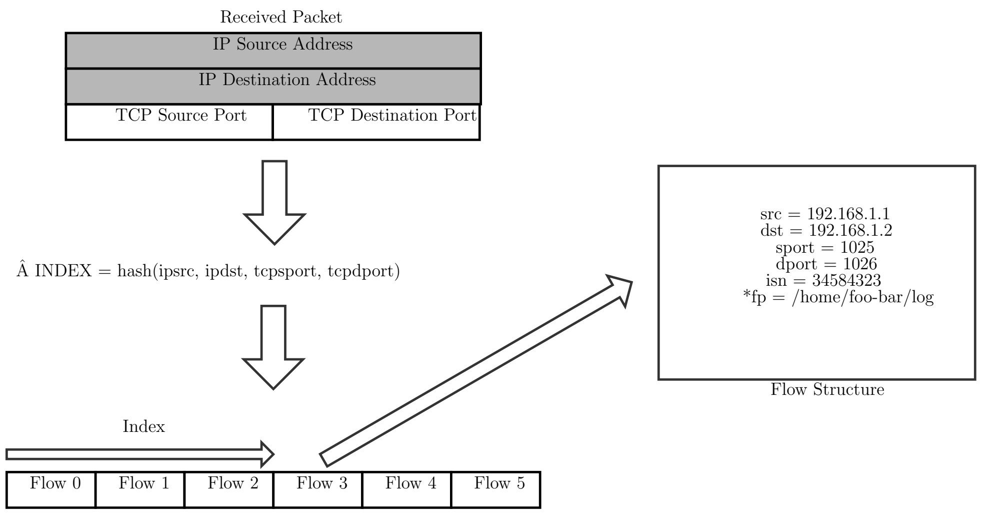

Figure 3.3: Transport layer processing

Figure 3.3 gives us a more insightful vision of the processing done to the captured packet at the transport layer. *tcpflow* starts by extracting both the source and destination IP addresses, as well as the source and destination ports given from each TCP packet. Based on this extracted information, a unique hash index is calculated. This hash index is then used to access the hash table of *flow* structures in order to verify if we already have a *flow* allocated for that stream. In the next step we verify which flags are active in the transport layer header. By looking at the flags, we can determine if we are in the presence of a packet that contains any payload with relevant data. If the packet simply contains a SYN flag, then we can assume that this will be the start of a new TCP connection and all we have to do is just allocate memory for a new *flow* structure that represents that same connection. On the other hand, a packet containing an active FIN flag triggers the necessary functions that aim to close the file descriptor associated to its log file as well as the removal of all the structures associated with that respective TCP connection. At last, a packet containing an active PSH flag means that the packet in question contains useful data in the payload segment. In that case, the payload segment of the packet is subject to further processing in order to extract and save all the necessary information our system needs.

Figure 3.4 further details the processing done to the payload layer of the captured packet. At this stage, *tcpflow* is responsible for extracting and writing the data

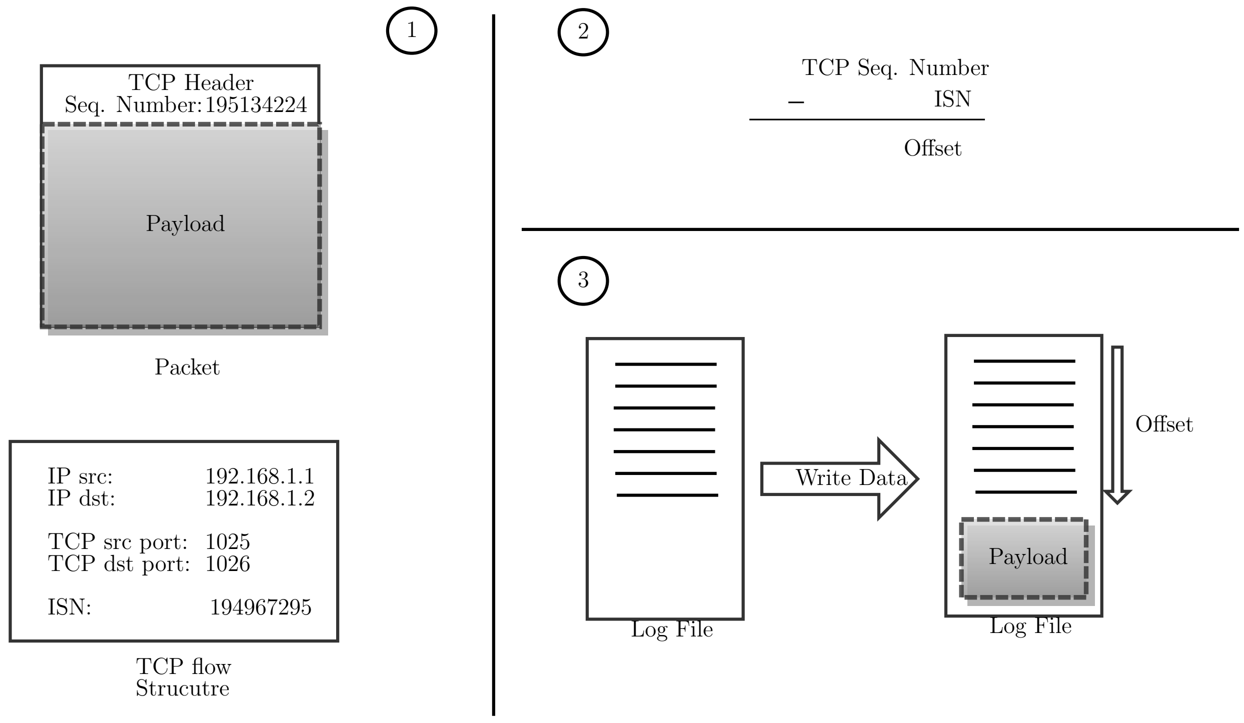

Figure 3.4: Data payload processing

contained in the payload segment of the packet. Given that the tool does not maintain in memory any other data other than a small amount of information that allows it to make a distinction between different TCP connections, *tcpflow* needs to write the data contained in the packet already in the correct place in the log file even if the packet arrived out of order. Such task is possible due to the fact that each TCP packet contains in its header a sequence number that allows us to identify the correct place where the data we received is located in the overall stream of data transmitted. Therefore, all *tcpflow* needs to do is to store the first seen sequence number in a given TCP connection. After that, for each packet received, we check the sequence number contained in the TCP header of the packet and subtract it to the first seen sequence number. The number then obtained indicates where in the log file the data we just received belongs. By using this approach, *tcpflow* avoids having to store in memory any data contained in the packet payload and instead it calculates where in the file each piece of data must reside, uses the *seek* Linux system call to adjust the current writing position of the log file and finally writes the data in its proper place.

3.1.1.2 *tcpflow* Modifications

Given that our overall system is based on the extraction of HL7 and other eHealth interoperability protocol messages, our custom modifications to the behaviour of *tcpflow* were made so that its final version could produce log files containing a series of HL7 messages, each associated with the timestamp of its capture. Also, since the extraction of these messages aimed to produce a system that could provide almost real time information about the state of an healthcare institution. Instead of keeping each file descriptor open indefinitely to keep adding new received data to it, our data extraction architecture needed to improve *tcpflow* so that it could perform the additional functions:

- Create a new log file for each new TCP connection detected;
- Recognize the beginning and the end of different HL7 messages transmitted within the same TCP connection;
- Timestamp each HL7 message with the time of the capture
- Close the file descriptor associated with the log file when a TCP FIN packet is detected;
- Close the file descriptor associated with the log file when it contains a predetermined number of HL7 messages;
- Close the file descriptors associated with the files for which their respective TCP flows have been inactive for a given amount of time;
- Move all the closed completed log files to a given directory of the system for further processing;

Based on these requirements, we needed to implement additional steps in the overall *tcpflow* packet processing pipeline.

To associate the a timestamp with each HL7 message we first had to look at the first bytes of each message in order to determine an expression that could unequivocally indicate that we were in the presence of a new HL7 message. As such, after analysing a sample of the log files produced by the unmodified version of *tcpflow* we determined that each new HL7 message is preceded by the vertical tab American Standard Code for Information Interchange (ASCII) control character ("\x0b") followed by a "MSH" segment defined in the HL7 standard. As such, in the packet processing pipeline, before the writing to the log file, we firstly need to introduce a small verification of the first 4 bytes of data and compare them to the following string: "\x0bMSH". If the data corresponds to the expression we are looking for, then we extract the current epoch value in milliseconds directly from the *gettimeofday* system call implemented in the GNU/Linux kernel. We then use that same epoch value and write it to the log file together with the data contained in the packet. By doing so, we can easily associate each message with a timestamp that corresponds to the approximate time when the packet arrived at the HL7 transformer without introducing too much overhead to the packet analysis pipeline.

Another set of modifications made to the *tcpflow* consisted in developing means for it to stop functioning solely as a debugging tool and to start having a more dynamic role in the extraction of the HL7 messages. Namely, we needed the tool to start closing the file descriptors of the log files so that they could be further processed by other tools. We thus started by stating a set of conditions where *tcpflow* should consider a given TCP connection as finished and proceeded to close all the file descriptors associated with that connection. Namely, the conditions to close a given file descriptor are:

- A TCP FIN packet is received;
- The log file of a given *flow* exceeds a predefine amount of HL7 messages;
- A given TCP connection exceeded a predefined amount of time without transmitted any data;

By developing a way for *tcpflow* to close the file descriptors of the log files when the previous conditions are met for a given TCP connection, we can guarantee that the modified tool does not stay indefinitely writing data to a given file without ever closing it.

Again, when adding these features to *tcpflow* we need to keep in mind that we can eventually end up losing packets if we cause too much overhead during packet processing, and as such our modifications need to be as efficient as possible.

In its original implementation, *tcpflow* never checks for the presence of an active FIN flag in the TCP header since it never needs to give any special treatment either to the log file nor to its internal structures. It simply ignores packets in that condition. Since we required the closing of the file descriptors associated with a log file when an active FIN flag was detected, we needed to add a small verification for any special active flag in the TCP header of the packet. To do this flag verification as efficiently as possible we use bitwise operators to find any active flag in the TCP header. As such, suppose there is packet with the TCP header flag segment w, and the variables $f = 0\text{x}01(hex)$ and $a = 0\text{x}10(hex)$ respectively represent the FIN and ACK flag values for any given packet, then $w \oplus (f \vee a)$ yields us the value 1 if the TCP header has the FIN and ACK flags set as active. By adding this small verification to the packet handling pipeline we were able to filter any packet containing an active FIN flag. For the packets that verified the previous condition, we close the file descriptor associated with its log file, moved the respective file to a predefined directory and proceeded to free any memory associated to that particular TCP connection.

In order to keep track of the number of HL7 messages contained in a given log file we decided to take advantage of the verification we were forced to do in order to associate a timestamp to a given message. To do so, we reserved another 4 bytes to the *tcpflow flow* structure implementation in order to store an integer value. As such, every time we tested the condition to verify if the packet contained the start of a new HL7 message, we firstly verified the value contained in those 4 bytes and if that same value exceeded a preconfigured threshold, then we close the file descriptor associated with the respective TCP connection log file and restart its *flow* structure before logging the data contained in the packet currently being processed. Otherwise, we increment the value contained in those 4 bytes in order to keep track of how many HL7 messages we already stored in the log file.

Finally, keeping track of the TCP flows that haven't sent any data in a predefined amount of time proved to be the biggest challenge in our set of modifications made to the original *tcpflow*. In fact, having an hypothetical solution where we would need to iterate over all the *flow* structures and check the timestamp of the last known transmission against the current system time could prove to be too computationally intensive when the number of active *flow* structures was too large. Our solution to the problem consists in taking advantage to the features of multi-threading implemented in the *pthread* [3] library. By doing so, we are able to run all the necessary verifications in order to look for any inactive TCP connection separately from the main *tcpflow* thread of execution. Although the use of multithreading allows us to run a certain part of the code concurrently, we also inherit some of concurrency problems. We now have to take into account problems that arise by lack of synchronization from both threads when trying to access critical sections of the code and that could ultimately lead to *deadlocks* in the program. To solve such problems, we use mutual exclusion functions and structures implemented directly in the *pthread* library.

To find inactive TCP connections, we decided to keep a concurrent thread periodically running that would inspect the modification date of all the log files currently being processed. By looking at the modification time of each log file, we were able to determine if there had been any recent writings to the file, which would mean that there had been recent data being transmitted in that given flow. This approach however leads to one particular problem. During the time that our thread was running and inspecting the modification dates of our log files were not able to give further treatment to new packets that had been sniffed from the network since this would imply that both threads were trying to access critical sections of the code at the same time. In order to solve this problem we had define a synchronization mechanism on both threads that could allow us to inspect the log files without having to drop any packets while waiting for that inspection to finish. We therefore took advantage of the fact that the *libpcap* implementation makes use of a buffer in which it stores the packets sniffed from the network until a certain function is called to consume the packets.

The synchronization of both threads can be achieved by using mutual exclusion (*mutex*) techniques directly implemented in the *pthread* library. In our system's implementation we use a mutex as flag which only one executing thread can hold at any given time. That is, suppose we have several running threads in one application. A mutex can be seen as a variable shared among the different threads and it can only have two different states, either it is in an unlocked or in a locked state. As such, when trying to synchronize concurrently running threads, each thread will first try to acquire the lock on the mutex. From this action we can have two possible outcomes, either the mutex is unlocked and therefore the running thread can go ahead and acquire the lock on the mutex and continue executing or the mutex is being held by a different thread which will cause the thread that is trying to acquire the lock to wait for an opportunity to hold the lock on the mutex itself.

In the specific case of our system, the main thread contains a function where packets are sniffed from the network and then added to a buffer where those same packets are subsequently consumed and its data logged into a given log file. This behaviour continues until the periodic thread checks the modification time for each log file. When the program reaches this state, the periodic thread attempts to acquire the lock to a given mutex, the result of that attempt can lead to two possible outcomes. Either the mutex is already locked by the main thread or the mutex is free and can therefore be locked by the periodic thread. In the first case, if the mutex is already locked by the main thread, the periodic thread will enter a state where it will wait for the lock to be released by the main thread. On the other hand, if the mutex is already available, the periodic thread immediately acquires its lock and then proceeds to evaluate each log file modification date to determine if a given *flow* is to be considered closed. In doing so, we create a well defined mechanism in which both threads first need to agree which one of them is to be given access to a critical section of the code. As for the problem of not losing any packets in the main thread while the periodic thread is running, the *libpcap* sniffing buffer solves that problem automatically for us, since even if the periodic thread takes too long to run, the main thread keeps waiting for a lock in the mutex in a state where it keeps adding the sniffed packets to a buffer and when the lock is finally acquired, the packets contained in that buffer are going to be consumed and its data logged in the respective log files.

This solution, however simple is not without problems and as such in using this previous thread synchronization we introduce another problem. The buffer for the sniffed packets implemented in the *libpcap* is implemented in memory and as such we have to guarantee that the buffer has enough space to hold all the necessary packets while the periodic thread is running. Otherwise, if the buffer if full the new packets will start to be discarded by the NIC and we lose the possibility to log its data. This particular problem can be solved, at the cost of consuming more system memory, by using the *pcap_set_buffer_size()* call, where we have the possibility to set a custom size for our buffer, therefore guaranteeing that we have enough space to hold all the packets received while periodic thread is running. Another downside to the use of such approach concerns the timestamping of each HL7 message. As previously stated, the timestamp associated to each message is set at the time of the logging and as such, if the periodic thread runs for too long, the packets waiting in the *libpcap* buffer will experience a small delay in its associated timestamp. As such, the timestamp of each message will tend to represent the time of the logging instead of sniffing. One possible solution to this problem would be to force the usage of NICs that allow hardware timestamping of each packet sniffed from the network. By using such devices, the time that each packet spent on the sniffing buffer would no longer influence the timestamp our system uses to determine the time of the message.

As a final result, our data collection tool, dynamically logs all HL7 messages in different files, one for each TCP connection detected in the network. Figure 3.5 shows a sample log file generated by our modified version of tcpflow. As we can see, each message starts by a thirteen digit timestamp representing the number of milliseconds passed since the 1st of January of 1970. The timestamp is then followed by the contents of the HL7 message, each separated by a vertical tab ASCII control character (^K) and ending with a file separator ASCII control character (^\) followed by a carriage return

```
 1398186096506^KMSH|^~\&|PSCRIBE||RDHL||20021021070646||ORU^R01||P|2.3
 PID|||12345||Doe^Jon|...
 PV1||I|IQ^363^07||...
OBR|1|P123|F123|502^CHEST XRAY^L|||||||||||||||||||||||1
OBX|1|ED|502^CHEST XRAY^L||Word^TEXT^^Base64^/9j/4AAQSkZJR...
 ^\^M
 1397579547939^KMSH|^~\&|PATHNET|||STJO_AZ|20040718235800||ORU^R01||T|2.5
 PID|||1194200||Doe^Jon|...
 PV1|1|GENERAL|EMRW^EMRW^01|...
OBR|1|000000002|0000420002354^LA|...
OBX|1|NM|1000050^BUN||9|MG/DL|8-24|...
```

Figure 3.5: Log file Sample

character (^M).

Since the log files produced by our sniffing tool do not represent a valid HL7 message format, before sending each log file for the next processing, we first need to apply some transformations to its contents. Namely, the placement of the capture timestamp within the HL7 message would need to be corrected in order to produce a log file that could be directly accepted by our Mirth Connect node. As such, we took advantage of the fact that the HL7 message standard provisions a set of custom segments called Z segments that allow for the placement of any kind of data within the HL7 message structure. Therefore, in order to produce a valid HL7 message based on each log file, we developed a small tool we called *hl7_transformer* that simply reads each log file in a specified directory, extracts the timestamp at the start of each HL7 message and places it in a valid Z segment at the end of the message.

Figure 3.6 represents the final format of the HL7 log files produced by our Sniffer Node.

3.2 Integration With Mirth Connect

Another component in our infrastructure is the HL7 log files analyser. The log files are generated during the data collection phase and are composed by a multitude of HL7 messages. We need a tool to parse each message and extract the relevant data fields. We thus decided to take advantage of the Mirth Connect integrator engine whose main advantage consists on its interoperation engine which already possesses numerous tools that can efficiently handle the different types and flavours of HL7 messages.

```
MSH|^~\&|PSCRIBE||RDHL||20021021070646||ORU^R01||P|2.3
PID|||12345||Doe^Jon|...
PV1||I|IQ^363^07||...
OBR|1|P123|F123|502^CHEST XRAY^L|||||||||||||||||||||||1
OBX|1|ED|502^CHEST XRAY^L||Word^TEXT^^Base64^/9j/4AAQSkZJR...
ZTS|1398186096506
^\^M
MSH|^~\&|PATHNET|||STJO_AZ|20040718235800||ORU^R01||T|2.5
PID|||1194200||Doe^Jon|...
PV1|1|GENERAL|EMRW^EMRW^01|...
OBR|1|000000002|0000420002354^LA|...
OBX|1|NM|1000050^BUN||9|MG/DL|8-24|...
ZTS|1397579547939
```

Figure 3.6: Final Log File Sample

A Mirth Connect engine typically serves a connector between different eHealth systems. Namely, this tool allows different applications that use the HL7 standard to communicate with each other by applying transformations to the HL7 messages that are necessary for data to be transferred in a meaningful way between those applications.

In our monitoring infrastructure, Mirth Connect plays a much simpler role. As illustrated in Figure 3.7, its main use is to read each of the log files produced by our modified version of *tcpflow*, extract the necessary metrics according to the type of each HL7 message and insert that data into a MySQL database.

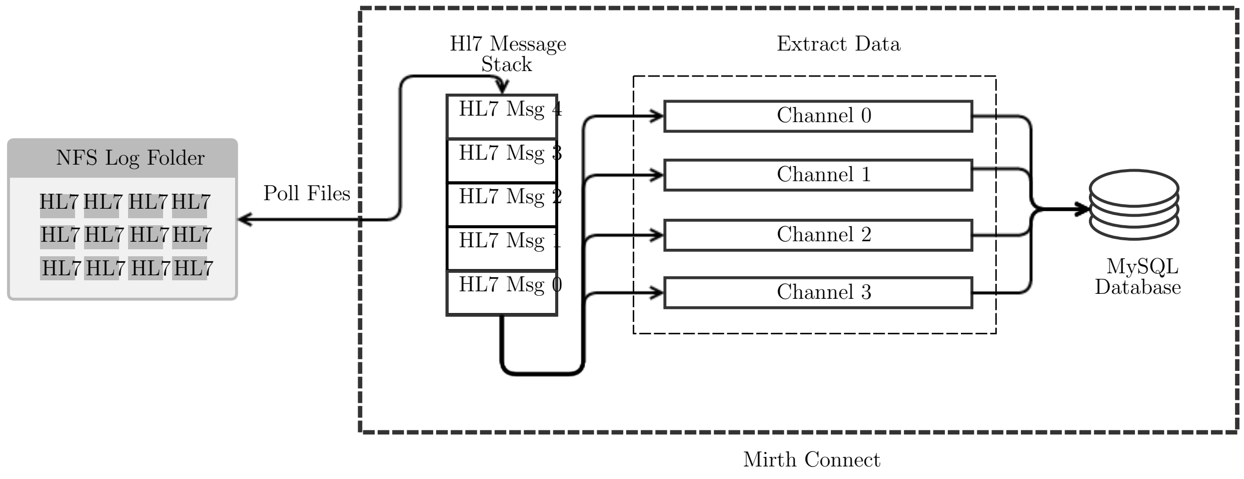

Figure 3.7: General Mirth Connect usage

To accomplish such task we used what Mirth Connect designates as channels. A channel can be seen as a pathway for any message in which we configure any given source of data, a set of transformations to apply to the data received and finally we store or send to a predefined destination the transformed result. In a sense, a Mirth connect channel acts as a specialized router for standardized messages where we have the possibility to apply transformations to the contents of the data received and then reroute it to any given address.

We thus configured a Mirth Connect server with a series of channels each of which acts similarly to a pipeline, where each channel routes, transforms and/or extracts some data from each HL7 message. Our main goal in adopting this strategy was so that we could have a single channel with the responsibility of polling all log files previously generated by our system, queue them and then send a copy of each HL7 message present in that file to a series of different channels where each one of them would do a specific function to the message. We then take advantage of concurrency by having each Mirth Connect channel running on separate thread and therefore we can a take advantage of a multiprocess system.

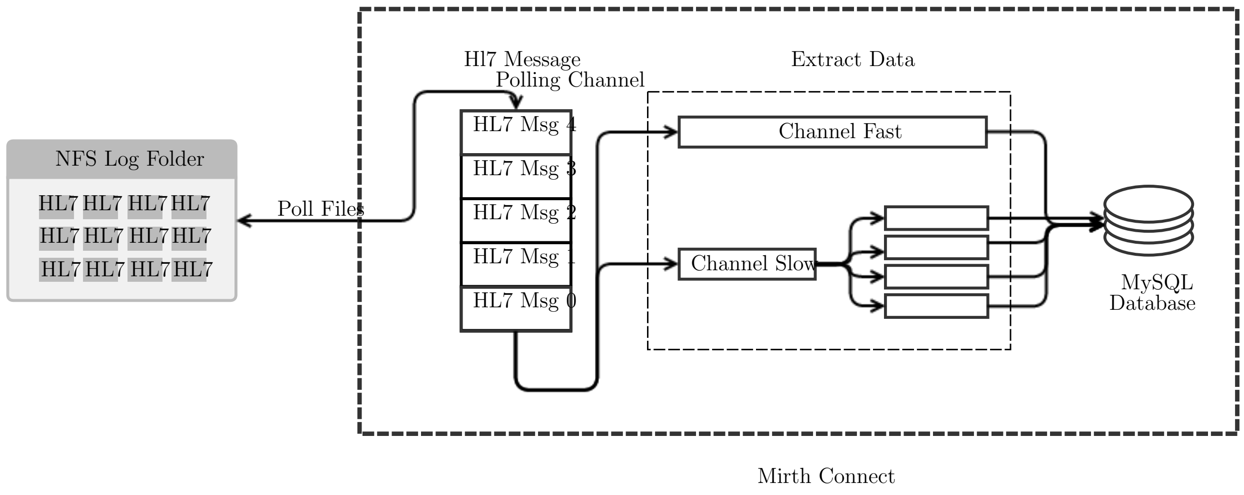

Figure 3.8: General channel pipeline

Figure 3.8 shows our Mirth Connect channel structure for our monitoring infrastructure. We start by having a channel that keeps polling all log files present in a given directory. Each file is then opened and a copy of each HL7 message present in that same file is sent to two different channels.

Since our system needs to display real-time performance metrics, the data extraction process requires a channel configuration capable of dispatching each HL7 message as quickly as possible. On the other hand, the extraction of all statistical useful data requires a more thorough processing for each message and therefore it is much more computationally intensive and more time consuming. Hence the separation of the data extraction process in two different channels.

In the fast channel, the system is responsible for extracting simple data present in all HL7 messages independently, of their type. In our specific case we extract the following fields present in the MSH segment for each HL7 message:

- **Sending Application.** Uniquely identifies the application that sent the HL7 message;
- **Sending Facility.** Identifies the organization of the application that sent the HL7 message;
- **Receiving Application.** Uniquely identifies the application that received the HL7 message;
- **Receiving Facility.** Identifies the organization of the application that received the HL7 message;
- **Message Type.** Identifies the type of the message and therefore allows the system to recognize a set of possible next segments;

The fact that, in the fast channel, our system only stores this small amount of data makes the gathering of meaningful information a really fast process and in doing so, we allow our system to display real-time performance metrics related to the number an type of HL7 messages exchanged between the different systems in the institution. This particular functionality can be especially useful in a system monitoring situation where a dashboard can be dynamically updated with the number of received messages in the last minutes. In doing so, we can quickly assess if any given hospital system has stopped functioning if for instance, the dashboard stops reporting exchanged HL7 messages.

As for the second channel, its main objective is to thoroughly parse each HL7 message and extract all the necessary data. Depending on the type of each HL7 message, we might extract different parts of the message and store them in the database. In order to efficiently differentiate the way we handle each type of message, we decided to apply the usage of *channel filters* directly implemented in the Mirth Connect engine.

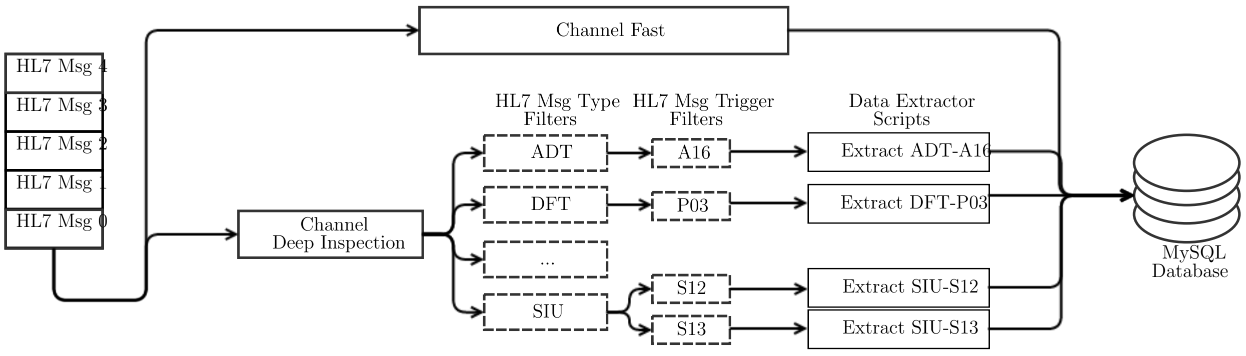

Figure 3.9: Second channel network

A channel filter can act as an controlling mechanism that can either accept or reject a given message. As such, when a message arrives to a channel, before applying any transformation and/or routing to the message, the interface engine passes the message through a filter. When configuring a channel filter a user can essentially specify a set of rules that any given messages must have in order to be accepted for further processing. One of the most basic channel filters that can be applied consist on creating a set of rules that aim accept only messages of a given HL7 type, therefore dropping any undesirable message types.

We can thus create an even more specific network of channels where each one of them will handle a different type of message. Figure 3.9 shows our channel infrastructure. We take advantage of the fact that each Mirth Connect channel is able to run concurrently and in doing so, we may have different types of messages being handled at the same time. Each message starts by being transferred from our file polling channel to our deep inspection channel. After receiving a new message, the deep inspection channel passes the message through a series of filters where it will look at the HL7 message type field and according to its value it redirects it to the appropriate channel. Since the HL7 standard defines each message with a *type* followed by a *trigger event* field, it is possible that messages with the same type field have the same type of data but on different segments of the message. Therefore, after the HL7 message arrives to its appropriate channel, we use another set of filters, this time to differentiate messages with the same *type* but with different *trigger events* fields. After all the filtering and redirection process, we can now be sure that each message that arrives in a specific channel will have the data in the correct positions we are expecting and as such, all we need to do is write the desired data to the database.

3.2.1 Post-Extraction Process

Apart from the data extraction process, we also felt the need to have a set of different debugging and validation tools that can track any problems encountered during the data extraction process, as well as a backup system for each HL7 message received.

We relied on some of the already implemented functionalities on the Mirth Connect engine. As such, we configured each Mirth channel to log all the HL7 messages. This configuration was especially useful in determining the effectiveness of the filters installed in some channels. As such, by analysing the logs produced by the Mirth Connect engine, we were able to assert that all the messages received were following the desired path in the channel network and therefore they would arrive at our desired extraction point.

In respect to the archiving of each log file received, Mirth Connect allows the possibility to make a copy of each file received and store it in a predefined directory. After processing all the messages in a give log file, we store its contents in a given filesystem directory and add a timestamp to its file name. We also wanted to have the possibility to analyse problematic messages that produce errors during the processing stage and segregate erroneous log files to a different archiving directory. This archiving strategy duplicates the storage space needed for the node where Mirth Connect runs. We thus decided to use the *sniffer* node as a backup system for our archived HL7 messages. As such, we created a simple *cron* job that every day during night time would compress all the archived HL7 messages in a *tarball* and send it to the *sniffer* node using *rsync*. We were thus able to create an archiving system for the processed HL7 messages, at the same time that we have a backup on another node that would help us restore our system in case of a catastrophic database loss.

3.2.2 Scaling

The Mirth Connect engine presents a great advantage for our system. Not only it facilitates the whole process of data extraction from HL7 messages but also presents some possibilities if the system needs to scale up in order to support an increase of network traffic related to meaningful eHealth data. Since our system currently relies on the existence of an HL7 message connector infrastructure where all HL7 traffic passes through, our system could evolve alongside with the healthcare infrastructure in two different ways. Either the HL7 connector suffers an upgrade and begins processing more traffic or a different HL7 connector is installed in the infrastructure and the traffic gets divided between the two connector nodes. In either case, we believe that our infrastructure is able to adapt to these changes without having to redesign most of its implementation.

If the healthcare infrastructure increases the traffic flowing through the HL7 connector then the adaptations in our system infrastructure should be relatively simple to implement. Namely, assuming that our "Sniffer Node" is able to withstand the traffic increase and capture all the packets in the network all we need to do is add another channel responsible for polling log files. so, However, the latest version of the Mirth Connect engine does not allow the existence of multiple channels polling files from a single shared directory. This behaviour is due to the fact that Mirth Connect implementation does not have any mechanism responsible for syncing multiple channels when reading from the same directory and in lacking such a mechanism we might end up having different channels both reading the same file and therefore we could contaminate the database with repeated messages. One possible solution for this problem is to configure our modified version of *tcpflow* to maintain a list of possible log directories and when writing a file, all we needed to do is to randomly choose from a predefined set of directories to create the log file. Assuming those directories are made accessible to the Mirth Connect Node through a network protocol capable of sharing directories between two different nodes like for instance the NFS protocol, we would then be capable of having multiple channels responsible for polling the log files, given that they are configured to use different directories.

As for the second case, the introduction of multiple HL7 connectors in the network infrastructure of the healthcare facility could also be easily adapted in our infrastructure. The solution is to have multiple *sniffer* nodes on the network, one for each new HL7 connector. In the Mirth Connect all we have to do is add new channels for polling log files produced by the multiple *sniffer* nodes.

3.2.3 Additional Usages and Advantages

The Mirth Connect engine also presents some additional advantages, we can use that same interface engine instance as a source of HL7 messages for other systems residing at the healthcare infrastructure and make it act as another integrator.

One potential problem that such an integration model could help to resolve, resides with the fact that each HL7 message may contain susceptible information about patients of the healthcare infrastructure and as such, the disclosure of such information may be considered a severe breach of privacy. It is possible to develop a network of Mirth Connect engines similar to the one presented in Figure 3.10 to create an anonymization channel that would remove any type of sensitive data and proceed to send the anonymized HL7 message to a given set of different destinations, therefore guaranteeing that the next nodes do not receive any kind of patient sensitive data.

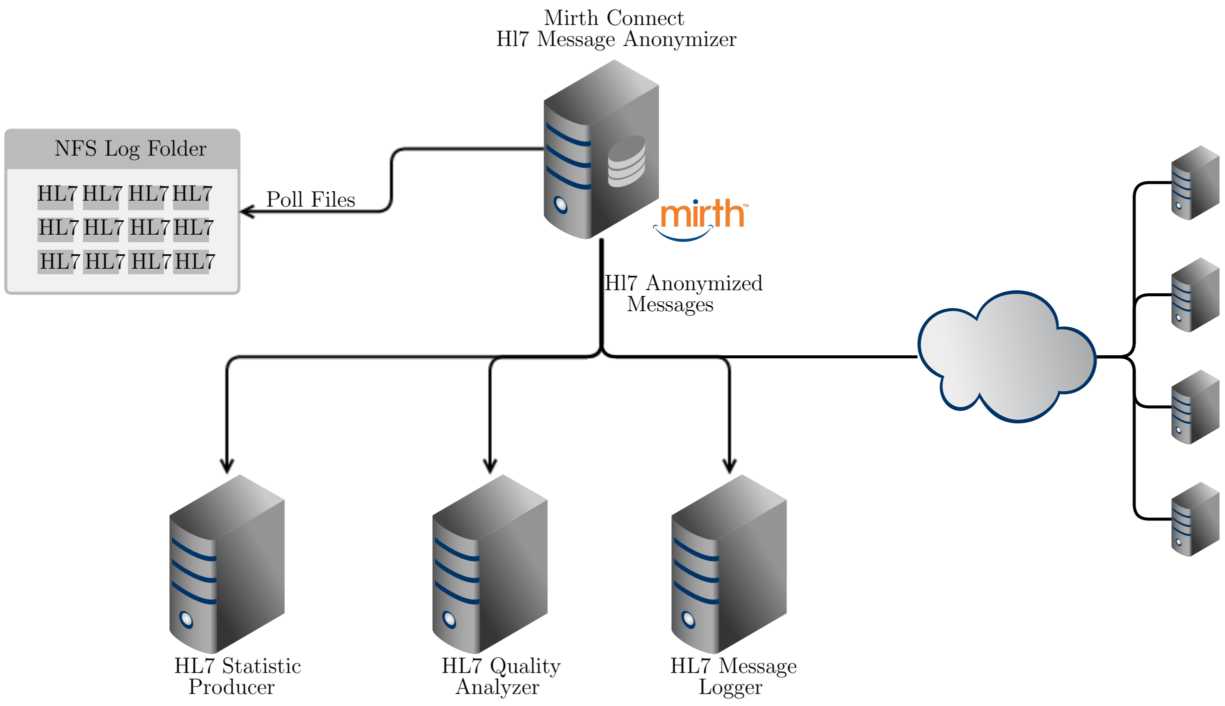

Figure 3.10: Mirth Network

According to the HL7 message standard, patient related information should be placed in a specific segment called “Patient ID” (PID) segment. This segment often transports information like the patient name, address, sex, date of birth etc. Since this type of sensitive information is gathered in a single HL7 segments it is possible to simply remove that segment from the HL7 message thus removing any type of sensitive information related to the patient.

3.3 Production Database Model

In order to have a database that can efficiently store the data our system collects we developed the model presented in Figure 3.11. As illustrated, the main “Message”

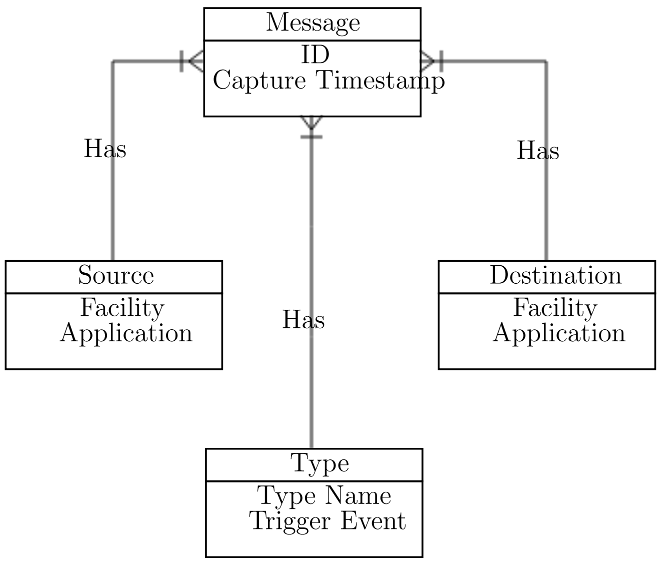

Figure 3.11: Database Model

entity contains an incremental id that uniquely identifies each entry in our database followed by a timestamp related to the time of extraction from the network. The "Message" entity is also related to three different other entities, them being:

- **Source.** Determines the source of the HL7 message. It can be uniquely identified based on its facility and application fields;
- **Destination.** Similarly to the "Source" entity, the "Destination" is also composed by a facility and an application field;
- **Type.** The type of the HL7 message can be decomposed by its message type and trigger event fields;

The previously enumerated data consists on a set of information that can be directly extracted by any given HL7 message and thus can provide our system with useful information.

3.4 Statistic Production

In order for our system to be able to display useful information about the level of performance of a given healthcare institution we firstly needed to find a reliable base of comparison for each metric. That is, suppose we are evaluating the number of medical appointments performed on a given day, in order to understand if the current metrics are within expectable ranges, our systems first needs to have a reference value that provides a base of comparison for the number of expected appointments on a given day.

3.4.1 Data Warehouse Approach

In order to obtain the previous requirements, we decided to apply some data warehouse techniques to the database containing all the captured data. By doing so, our system is able to gain some advantages.

The creation of a data warehouse model for a given production database is able to provide meaningful aggregated data about the stored information. That is, in the case of our system, the construction of a datawarehouse model whose main aggregation value is the time of extraction of each message is able to directly provide useful statistics like for instance the number of HL7 messages exchanged between two different systems in the last hour. Although this type of information can be calculated from the production database, the main advantage in this approach is that these kinds of information can be directly fetched from our datawarehouse, thus relieving the main database from unnecessary heavy queries. Also, deriving from the fact that useful timely aggregated data can be gathered by querying our datawarehouse, our system is able to respond much more quickly to information fetches since it can obtain all the necessary information directly by querying our datawarehouse.

When using a datawarehouse, we can also obtain other advantages apart from having the ability to quickly obtain meaningful data. Based on the information our system gathers from the HL7 messages, we can also build a model containing information related to what can be considered as the normal functioning of the hospital institution. Suppose we want to find out the average number of laboratory orders requested in a single day. If again, we build our datawarehouse it containing information in a timely aggregated manner, we can, for instance, show the number of laboratory requests made each day in the last month. Based on the results obtained we can thus obtain an average number of messages the hospital facility is expected to receive in any given day of the week.

3.4.2 Datawarehouse Fact Tables

Using aggregation operators as one of the main tools used to produce datawarehouse models [25], the quality and the relevance of the information contained in it is mainly subject to the schema used in the datawarehouse fact table.

In a datawarehouse, the fact table represents the main table where information is stored. Its schema can be divided in two parts:

- **Dimensions.** The dimension fields in the fact table mainly represent the fields we are trying aggregate our data. For instance, in our scenario the time of capture and HL7 message type can be two interesting metrics for which we may desire to have an aggregated view of our data;
- **Measures.** The measures provide the actual numbers we wish to obtain. That is, suppose we choose the total number of messages as one the measures for our fact table and that the dimensions are the time of capture and the HL7 message type, then our datawarehouse model would be capable of answering questions like the total number of laboratory orders requested yesterday;

In the topic of datawarehouses, an important factor to take into consideration is related to the *grain* of the data we are trying to gather. The grain, in datawarehouse model can be seen as the minimum level of detail we want our data to have. This specific concept is especially important when using "time" as one of the dimensions of the fact table. In that context, the grain will represent the minimum accuracy we want in our data. So for instance, in our systems' scenario we may choose to have a grain of seconds, minutes or even hours.

3.4.2.1 Building the Datawarehouse

In order to build our datawarehouse for the information gathered from the HL7 messages we started by creating the structure of our fact table (Figure 3.12).

In our particular case, we decided to use three fact tables so that we could have multiple timely aggregated perspectives into the data we collected. To achieve that, each fact table has a different grain, namely we used 15 minutes, 30 minutes and 1 hour as the grains for our different fact tables. As for the dimensions, we used the time of capture, the type and trigger event of each HL7 message and finally for the measures fields we used the minimum, average and maximum number of messages.

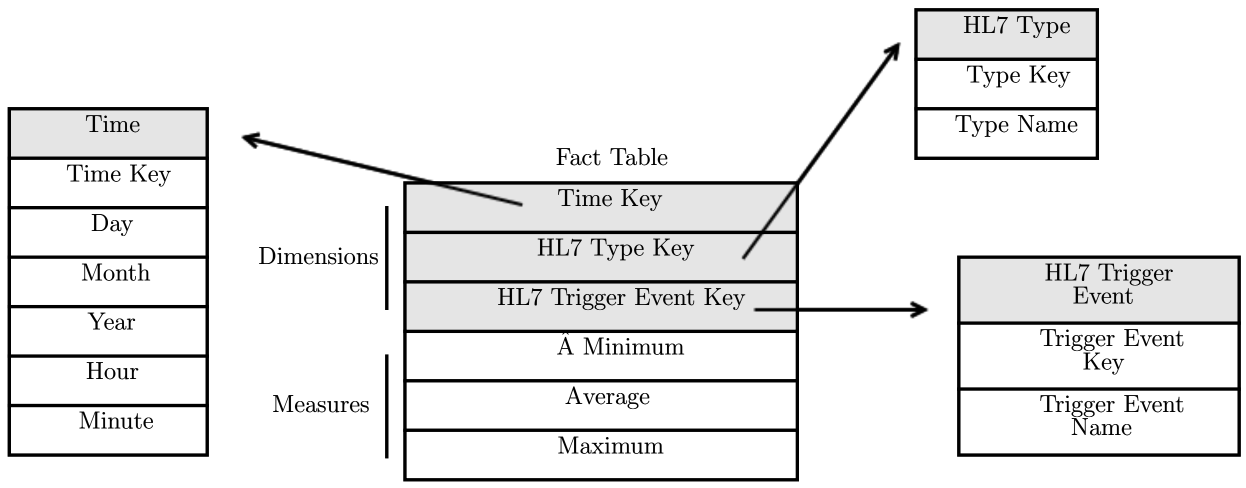

Figure 3.12: Fact Table Structure

Looking back to Figure 3.12 we can observe the relational model of our fact table. The dimensions are essential foreign keys that point to other simple MySQL tables where the actual data resides. On the other hand, the measures represent the actual metrics we desire for any given combination of dimensions. For instance, Figure 3.13 provides an illustration of how the data is actually connected and we can actually draw meaningful information from it. By analysing Figure 3.13 we can assess that on the 2nd of May of 2014 at 10:30 AM our system received a minimum, average and maximum of 123, 156 and 170 messages respectively.

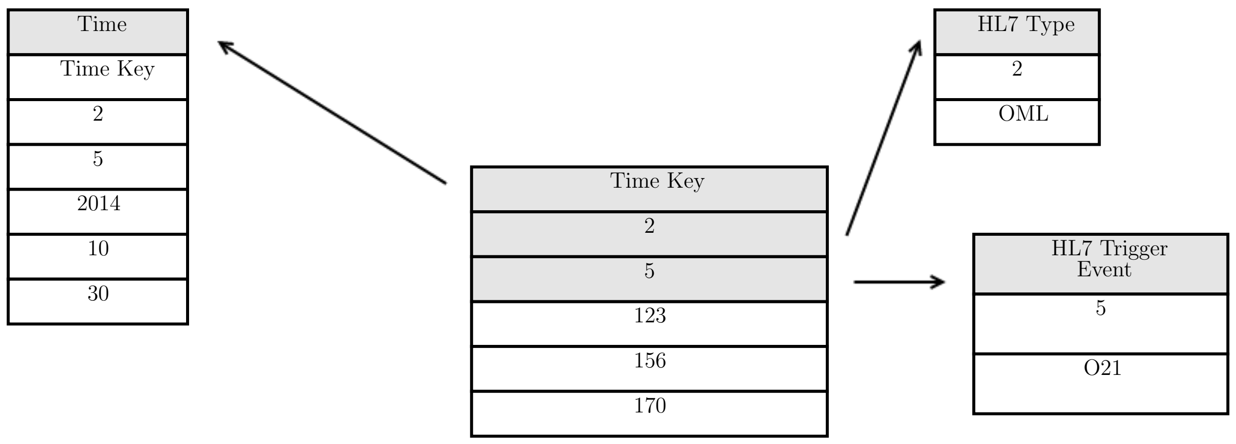

Figure 3.13: Fact Table Example

With this kind of schema for our fact table, our system is thus prepared to potentially

Table 3.1: Aggregation Example

HL7 Msg Type	**Day**	**Period**	**Nr. Messages**
ADT-A16	4	09:00 - 10:00	1
ADT-A16	2	09:00 - 10:00	5
ADT-A16	1	09:00 - 10:00	7
ADT-A16	5	09:00 - 10:00	8
ADT-A16	3	09:00 - 10:00	20

answer some of the following questions:

- Number of CAT scans ordered;
- Number of exams of each type;
- Top of physicians ordering exams;

We have implemented a small program written in C that we called *db_worker* (Database Worker) whose main function is to query the production database, gather the necessary data and create or update our fact tables with the calculated values.

Suppose we are trying to create the 1 hour grain fact table, then, our *db_worker* would need to iterate over every 1 hour segment of each day and query the production database for the aggregated number of messages during that period. Table 3.1 presents an example of the data our *db_worker* builds. In this case, we have the number of messages between 09:00 and 10:00 aggregated by the HL7 message type and the day of capture. After obtaining the data in such structure, our *db_worker* simply gets the 15th, 50th and 85th percentile of the number of messages (in this case 5, 7 and 8 respectively). The choice to use the percentiles instead of the minimum, average and maximum values is based on the work of Cruz-Correia et al. in [16]. By using such approach, we can safely eliminate any outliers derived from situations like holiday periods or even server downtimes that can cause a system malfunction and thus reduce the number of exchanged HL7 messages in the network.

To determine the minimum, average and maximum expected values for a given time segment, our *db_worker* is responsible for inserting into our fact table the values obtained for each HL7 message type within the correct time segment. The *db_worker* will continue this process iterating over hour segments, and inserting into the fact table the values thus obtained.

In the end, the *db_worker* produces a fact table as the one presented in Table 3.2 and, from which, our system is now able to have an internal base of comparison for the number of messages that should be expected during a certain period of time.

Table 3.2: Fact Table Sample

Msg Type	**Period**	**Minimum**	**Average**	**Maximum**
ADT-A16	09:00 - 10:00	5	7	8
ADT-A16	10:00 - 11:00	15	22	30
ADT-A16	11:00 - 12:00	18	25	28
ADT-A16	...	...	...	...
DFT-P03	09:00 - 10:00	8	10	15
DFT-P03	10:00 - 11:00	1	4	8

3.5 RESTful Service

Since our initial goals determined that the display of the information produced by our system needed to be made available to a browser, we took an approach to develop a system capable of abstracting the client (browser) from having to actually fetch the necessary information directly from the database.

In order to do that we decided to use a RESTful approach so that we could retrieve the necessary data from the database in a structured and systematic way. architecturally speaking, a RESTful service can be described as a web-service that treats data as a resource that can be accessed through an Universal Resource Identifier (URI). Typically HTTP is used in order to have the clients communicating with the server and as such, the interaction and manipulation of the resources is mainly achieved by using HTTP GET or POST operations.

Apart from being able to abstract the complexity of querying the database to obtain information, one of the main advantages in using a RESTful web service approach for the server-client interaction is the fact that the resources returned to the client can take many different formats ranging from simple HyperText Markup Language (HTML) to a complex JavaScript Object Notation (JSON) structures.

3.5.1 RESTful API

In order to abstract the requesting browsers from the complexity of having to query the database to obtain information, we introduced in our system a simple RESTful service capable of receiving HTTP GET requests, select the appropriate data from the database and deliver the requested information back to the client in a format that can be directly used to produce data charts in the client's browser.

The RESTful service is based on the the Jersey framework [2] to create a server written in Java capable of processing RESTful interactions. Also, apart from the Jersey framework, we also resorted to the use of the JDBC API [5] in order to interact to the database. As for the output sent to the client, we decided to use the JSON format to wrap the desired data in a way the browser could then directly use to display a set of charts with meaningful information.

Table 3.3: RESTful API

Request Type	**URI**	**Arguments**
GET	hl7sniffer/rest/msg_types	NONE
GET	hl7sniffer/rest/msg_ranges	HL7 Message Type
GET	hl7sniffer/rest/back_info	HL7 Message Type Starting Hour Ending Hour
GET	hl7sniffer/rest/live	HL7 Message Type Current Time
GET	hl7sniffer/rest/week_comp	HL7 Message Type Days to Compare

Table 3.3 summarizes the developed API for the clients' interaction with the database. Our systems' RESTful API is essentially composed by five different resources that can be requested to the server, them being:

- **Message Types.** Enables the client to request a list of all different HL7 message types our system captured;
- **Message Ranges.** Returns to the client a range for the number of expected messages for a given HL7 message type;
- **Number of Past Received Messages.** Allows a client to obtain the number of received HL7 messages during a predefined time frame;

- **Number of Currently Received Messages.** The server generates a response containing the number of received HL7 messages between midnight and the time of the clients' request;

- **Weekly Comparison.** Retrieves information related to the number of HL7 messages received in the specified days of the week;

3.5.2 RESTful Invocation and Response Format

As previously explained, we wanted to keep the interactions to the database separate from the client and by using the RESTful approach, we were able to hide all the database querying complexity from the client, as such, in order for a client to obtain a certain type of information from the database, all it needs to do is a simple HTTP GET request to the server.

Figure 3.14 depicts a typical interaction between the clients' browser and our systems' RESTful web-service. The client starts by requesting a given resource by accessing its respective URI, providing the server with all the necessary information so it can complete the request. The server then responds with a JSON structure containing the requested information.

Figure 3.15 shows an example of a response from the server when a client requests the number of messages received in the current day. When the client makes this type of request, the server is responsible for querying the database and return back to the client the number of received messages ranging between the midnight of the current day and the hour of the clients' request. The server then wraps the requested data in a JSON structure consisting on an list where each element provides a Cartesian coordinate.

In using such approach, the client receives the requested information in a way that can be efficiently handled by the browser. By looking back at Figure 3.15 we can see that each coordinate refers to a specific point in time. That is, the second coordinate of the example ([0.5, 6]) actually represents the number of HL7 messages received between midnight and midnight and a half. In this case, we can see that between that time a total of 6 messages were received, while the total number of messages received between 1:00 PM and 1:30 PM is 303 and so on.

The choice behind the usage of the JSON format to transport the requested data from the server directly to the client is mainly due to the fact that this specific format can

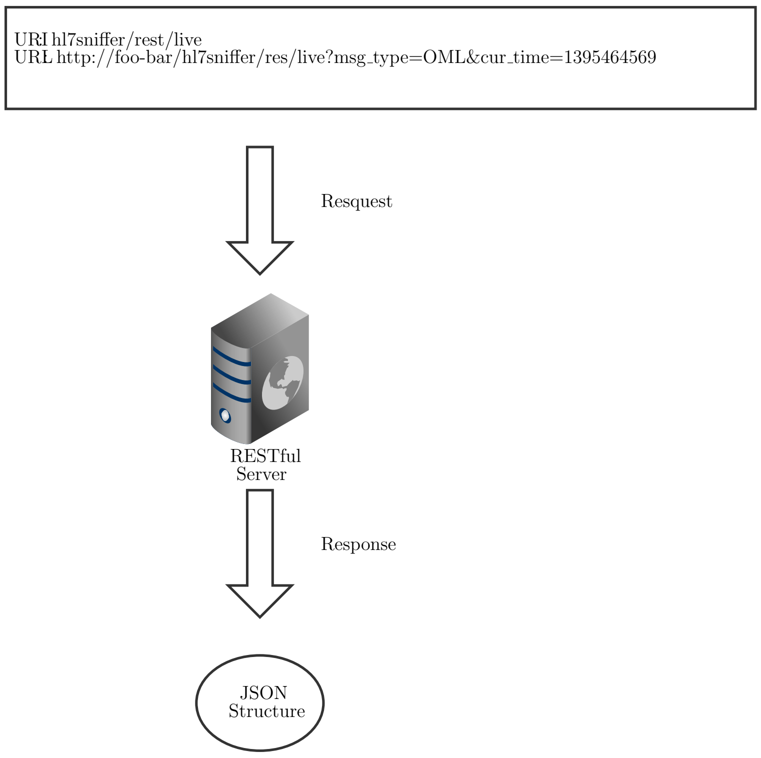
URI hl7sniffer/rest/live
URL http://foo-bar/hl7sniffer/res/live?msg_type=OML&cur_time=1395464569
Resquest
RESTful Server
Response
JSON Structure

Figure 3.14: RESTful Invocation

```
?([[0.0,0],[0.5,6],...,[13.0,303],[13.5,350]]);
```

Figure 3.15: RESTful Response Sample

be easily consumed by JavaScript and therefore browsers should not have difficulties in interpreting the results. Also apart from the efficient browsers' capabilities to consume JSON structures, the charts API used by our system is implemented using JavaScript which also contributed to our choice in delivering the requested information in such a format.

3.6 Metric Display

The final step in our infrastructure is related to the way we can graphically present the data being collected and calculated. To provide clients with an actual display of the metrics captured from our system, we decided to use a series of charts that could easily represent several interesting production metrics.

3.6.1 Highcharts Library

In order to produce the referred charts, we used the Highcharts library [1] so that we can dynamically present the extracted metrics directly in a browser using JavaScript.

The Highcharts library is essentially a framework that extends the main functionalities of the Asynchronous JavaScript and XML (AJAX) API so that it becomes possible to draw charts directly in a browser application. The library uses a JavaScript object called *Chart* and allows the developer to append a series of another objects that help define and draw the chart. Among all the possible objects used to define a chart, one of the most important is the *Series* object. This object contains the actual data we are trying do graphically represent. As such, the *Series* object simply consists on a set of coordinates similar to the ones presented in Figure 3.15.

As for the types of charts the Highcharts API allows to, among others, create the following:

- **Line Charts.** Used to create a line connecting a predefined set of coordinates;
- **Area Charts.** Creates a chart displaying a shadowy delimiting a certain range of values;
- **Combination Charts.** Allows the creation of chart that may be composed by several other secondary types of charts;
- **Dynamic Charts.** Can be used to dynamically update any given chart with new sets of data over time;

The previous enumerated types of charts present the ideal characteristics to support our systems requirements. Namely, the line charts can be used to display the evolution over time of the number of HL7 received while the area chart can be used to draw an area for the expected number of HL7 messages. When we combine the two previous charts, when trying to assess the number of HL7 messages received in a given day, we create a clear representation of what the system is expected to receive and what it has actually received.

3.6.2 Graphical Output

Since our system aims to dynamically present business metrics directly extracted from the network in such a way that would be easy to understand if the values are within normal ranges. In order to achieve that, we decided to use two different kinds of charts embedded in a single chart:

- **Area chart.** Based on the information present in our fact tables, we are able to draw this type of chart displaying a range of values that correspond to the expected values for a given metric.
- **Line chart.** Using the production database, we are able to draw a line chart representing the actual values detected for a given metric.

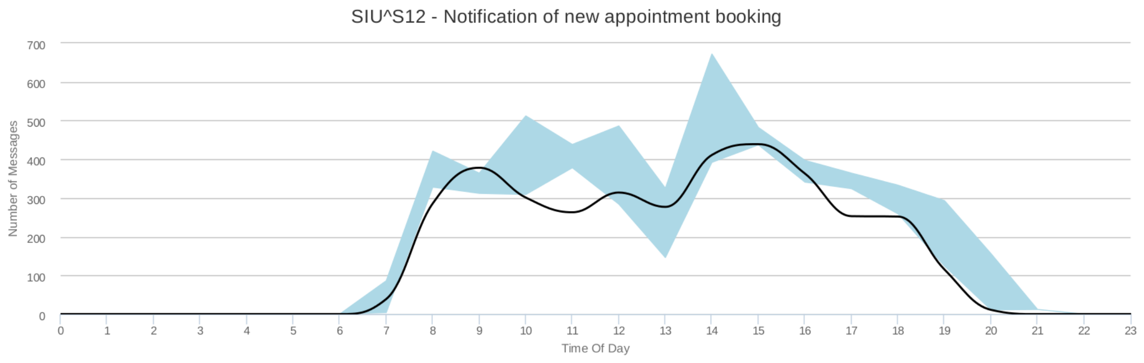

Figure 3.16: Chart sample

Figure 3.16 presents an example of the charts our system is able to produce. The blue area represents the range chart where we aim to present a visual display of the expected values for a given metric, while the black line presents the actual values at a certain period of time extracted from the network. This allows a user to quickly verify if any type of metric is within its expected values for a certain time interval.

For example, from a monitoring point of view, our system is able to show unexpected fluctuations in the number of HL7 messages passing through the network and in doing so, it can easily detect failures in a given service if, for example, the number of detected HL7 messages falls outside the expected range for a certain amount of time. Another type of application that can be given to these types of charts is related to administrative tasks. By displaying meaningful information in a temporal span, we allow the possibility to identify potential "dead" or "overloaded" periods of time. From an administrative point of view, such charts could present a powerful tool to support readjustments hospital services.

Chapter 4

Experimental Results

In the this chapter we detail a real test deployment scenario for our system, some of its hardware specifications and describe in detail the results obtained.

4.1 Prototype Architecture

We deployed our system architecture in an healthcare facility in the northern region of Portugal. As for the systems' architecture, due to lack of resources we weren't able to build an infrastructure exactly as the one presented in Figure 3.1. Instead, we were forced to place the "Mirth Connect Node", "Database Node" and the "RESTful Node" on the same physical machine.

As for the machine specifications used in our testing scenario we have used the following:

- **Sniffer Node.** Intel(R) Core(TM)2 Duo CPU E4600 2.40GHz with 2995 Mb of RAM memory and 100Mb/s NIC running the GNU/Debian system with a Linux kernel version 3.2.0-4
- **Mirth Connect / RESTful Node.** Intel(R) Core(TM)2 Duo CPU E4500 2.20GHz with 1985 Gb of RAM memory running the GNU/Debian system with a Linux kernel version 3.2.0-4

In respect to the healthcare facility infrastructure, at their core network they employ a Microsoft Biztalk server as their interface engine for HL7 compliant messages. As for the traffic captured, since we only had access to a standard NIC, we started by capturing only a small part of the traffic that passed through the interface engine. As such, we applied a *libpcap* filter capable of discarding all packets that weren't directed to a predefined port of the interface engine.

To measure the performance of our Sniffer Node at the network level ran a series of tests to assess not only the network load our system could endure, but also the amount of packets the NIC could eventually loose. Preliminary results are presented in Table 4.1.

Table 4.1: Packets Lost

Hour Of Day	**Packets Received**	**Dropped By Kernel**	**Dropped By Interface**	**Loss Percentage**
09:00	1587	0	7	0.44 %
10:30	1983	0	10	0.50 %
11:30	624	0	3	0.48 %
13:00	441	0	6	1.36 %
15:00	460	0	8	1.73 %
20:00	164	0	2	1.22 %

These numbers are obtained by running an instance of the *tcpdump* [21] tool. *Tcpdump* is able to provide the user with statistical information about the number of packets captured, as well as the ones that were lost. For this scenario, we ran an instance of the *tcpdump* tool during five minutes on different periods of the day so that we could observe the behaviour of our Sniffer Node and in particular, the performance of our NIC.

One of the drawbacks in our systems resides precisely within the NIC hardware. During periods of high traffic a standard network card can't endure the network load experienced by our test scenarios and therefore, it starts discarding packets before they reach the system's kernel level. Although the packet losses are not extremely high and can easily be solved by employing a better NIC, they still have a negative impact at a critical level of our infrastructure. One interesting fact about the previous results is related to the values obtained for the packets dropped by the kernel. According to *tcpdump* there were no packets dropped at the kernel level which might hint us that the packet buffers implemented in the Linux kernel have enough space to hold the packets while the user space applications handle each packet. Note however that since we are filtering packets whose destination is a predefined port on the interface engine, it is expected that when we decide to capture all traffic directed to the interface engine, the packet buffers at the kernel level may not have sufficient space to hold all the incoming packets.

4.2 Obtained Results

After the deployment of our infrastructure we began extracting and analysing all the HL7 messages destined to a predefined port on the Microsoft Biztalk server present at the healthcare infrastructure where we have tested our metric extraction framework.

On a daily basis, we have processed an average of 44,500 HL7 messages with rates of 932 messages per minute, reaching peaks of 1,200 messages per minute on critical hours of the day. Since the start of the data collection process on the 26th of April 2014 until the 28th of June 2014, approximately 1,300,000 HL7 messages were successfully extracted from the network by our Sniffer Node.

4.2.1 Metrics

Table 4.2: Message Types Weekly Results

Message Type	**Description**	**Number of Messages**
OML^O21	Laboratory Order	71020
ORU^R01	Unsolicited Transmission of an Observation Message	28253
ORL^O22	General Laboratory Order Response Message to any OML	25493
SIU^S13	Notification of Appointment Rescheduling	20598
SIU^S12	Notification of New Appointment Booking	18007
ADT^A16	Pending Discharge	11195
SIU^S15	Notification of Appointment Cancellation	3397
DFT^P03	Post Detail Financial Transactions	1275

Table 4.2 represents the number of HL7 messages received by our system from the 28th of April to the 2nd of May, aggregated by the type and trigger event of the HL7 message. As observed, the majority of the HL7 messages present in the network refer to laboratory orders as well as requests for patient observations. On the other hand, the smallest percentage of the HL7 traffic refers to billing processes.

Table 4.3: Message Types Daily Results

Message Type	**Description**	**Number of Messages**
OML^O21	Laboratory Order	18318
ORU^R01	Unsolicited Transmission of an Observation Message	6869
ORL^O22	General Laboratory Order Response Message to any OML	6306
SIU^S13	Notification of Appointment Rescheduling	5363
SIU^S12	Notification of New Appointment Booking	4563
ADT^A16	Pending Discharge	2484
SIU^S15	Notification of Appointment Cancellation	454
DFT^P03	Post Detail Financial Transactions	316

Table 4.3 presents the number of HL7 messages obtained on a sample normal production day at the healthcare facility.

Table 4.4: Message Types Holiday Results

Message Type	**Description**	**Number Of Messages**
SIU^S13	Notification of Appointment Rescheduling	2904
OML^O21	Laboratory Order	2837
SIU^S12	Notification of New Appointment Booking	2818
ADT^A16	Pending Discharge	2118
ORU^R01	Unsolicited Transmission of an Observation Message	868
ORL^O22	General Laboratory Order Response Message to any OML	471
SIU^S15	Notification of Appointment Cancellation	74

One interesting comparison that our system has allowed to infer is related to the difference between the infrastructure production level on a holiday and a normal day of work. Table 4.4 presents the number of messages collected by our system on the 1st of May 2014. In this specific case, we have a very different view about the number of HL7 messages exchanged. In this case, we seem to verify a lot more traffic related to administrative tasks such as scheduling or changing medical appointments and a significant reduction on the number of messages containing requests for patient analysis or laboratory orders.

Table 4.5 presents the busiests hours of the day related to HL7 traffic exchange through the network. As we can observed, the morning period is usually associated with a

Table 4.5: High Load Hours

Hour Of Day	**Number Of Messages**
09:00 - 10:00	7131
08:00 - 09:00	6282
10:00 - 11:00	6272
11:00 - 12:00	5028
13:00 - 14:00	3684

higher HL7 traffic load when compared to the afternoon periods.

Another series of results obtained by our system is related to the deeper analysis of each HL7 captured by our infrastructure. This type of information tends to yield more value when used to support decision makings from an administrative point of view.

Table 4.6: X-Rays by Physicians

Physician Name	**Number Of X-Rays**
Frieda Paige	11
Bill Stevens	8
Cydney Church	7
Jorja Walton	6
Duana Battle	5

An example of such data is presented in Table 4.6. The data represents real values collected on a single day by our system. As it can be observed, this type of information can be helpful when trying to determine which person requests which type of services.

One can also present patient related statistics such as the number of the number of lab analysis patients are subjected during their admission.

From an administrative point of view, it can thus be fairly easy to keep track of the performance of each employee as well as create a relation between methods employed by each physician and the results obtained by the patient.

Although this type of information may present itself as one of the most valuable sets of knowledge our system can produce, when used for decision making, the data gathered must always be subject to a previous thorough analysis in order to assess its veracity.

4.2.2 Interesting Dashboard Charts

Based on the data collected we were able to draw a series of charts that can be aggregated into a dashboard, for example to quickly evaluate the production level of the healthcare facility at a certain point in time.

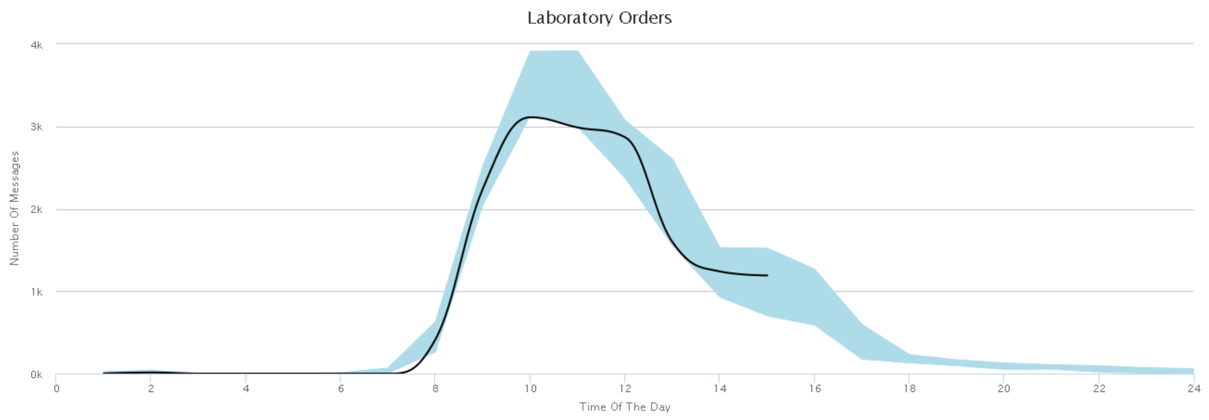

Figure 4.1: Laboratory Orders 1 Hour Aggregation

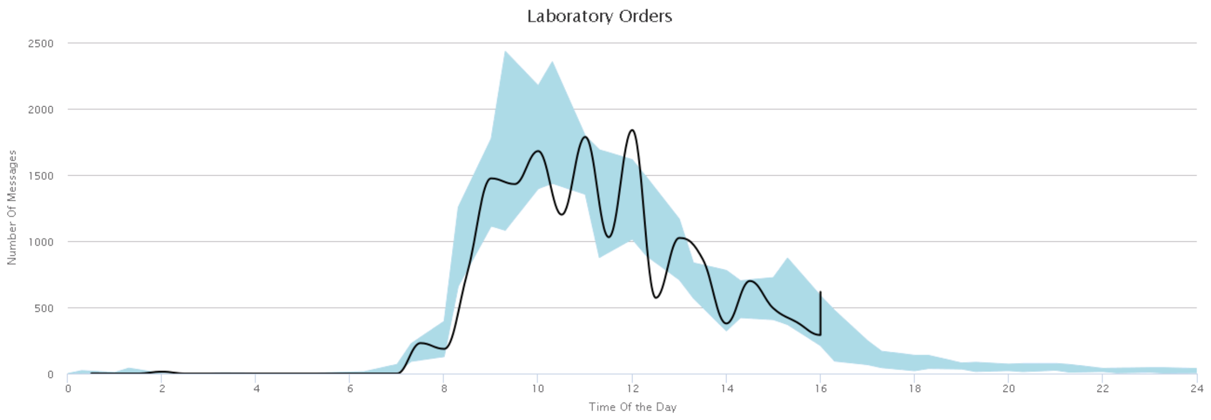

Figure 4.2: Laboratory Orders 30 Minutes Aggregation

Figures 4.1, 4.2 and 4.3 shows the number of laboratory orders collected on the 26th of June 2014. As expected, the number of laboratory orders starts increasing at the start of each work day around 08:00 hours, continuing to increase until reaching the 12:00 hours. After the morning period, the number of HL7 laboratory orders continues to decrease until the end of the work day around 20:00 hours.

By comparing the previous figures, we can also have different views into the data our system collects. As such, while on Figure 4.1 we have the total number of messages

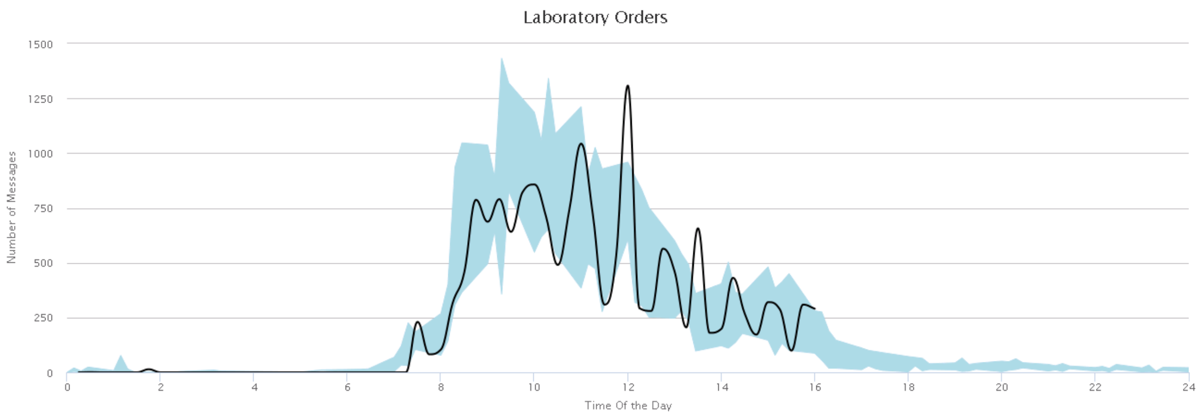

Figure 4.3: Laboratory Orders 15 Minutes Aggregation

collected during periods of one hour, Figure 4.2 and 4.3 present the total number of messages accumulated during periods of thirty and fifteen minutes respectively.

As expected, when we decrease the aggregation period for the number of messages, the displayed data tends to be increasingly susceptible to small variations in the number of messages. As such, for the extreme case of a fifteen minute aggregation, the slightest variation to the number of messages detected by our system can easily lead the current message curve to fall outside the expected values. Therefore in this particular case, the fifteen minute aggregation chart does not present a good choice for the data grain since from a monitoring point of view the data is easily subject to deviations from the expected ranges. As for the thirty minute aggregation, this chart also presents some deviations from the expected values, however, in this particular case we expect that our system will be able to adapt the message range based on the increasing history of the collected data.

4.3 Results Analysis

We expect the data collected and produced by our system to be used for mainly two different purposes.

From a service monitoring point of view, the proposed charts can be used to identify potential malfunctioning services when for instance, the number of messages either drops or grows too much outside their expected range. By using such approach we lay the groundwork for the creation of an alert system based on the number and type of

HL7 messages flowing through the network. Such an alert system could easily support different levels of severity based on the level of discrepancy from the expected number of messages.

As from an administrative point of view, the information gathered can potentially be used in order to assess the production level at individual levels and at different data dimensions. For example, by taking advantage of some types of HL7 messages containing data that can uniquely identify an individual in the healthcare facility we believe that our system would be capable of producing information related to the production levels at an individual level.

The systems' architecture also presents a great value as a starting point for the creation of several other parallel and independent services. Namely, in using the Sniffer node as a starting point, we open the way for the possibility of having several different systems using the gathered HL7 messages in order to provide other given services. For instance, a service that could assess the semantic and syntactic quality of the HL7 messages exchanged at an healthcare facility or even a service that dynamically searches for incoherent or erroneous data present the HL7 messages. An even more elaborate system that can be built around the existence of our architecture is a patient registry system capable of encompassing all the patients currently "active" at a given hospital facility. Such system could even include information related to the pathologies and courses of treatment the patient is currently undertaking.

The proposed system can also be seen as an opportunity for the creation of an integration system completely independent of any software vendor. Assuming our system is able to collect all HL7 messages associated with a given software vendor, since we use an instance of the Mirth Connect engine, our system would be able to read those same messages and create a channel capable of applying any type of transformation the hospital services would require, therefore eliminating the necessity of having software vendors altering their own channels in order to fulfil a specific institution requirement.

Chapter 5

Conclusion And Future Work

The improvement of the healthcare IT infrastructures has led to the creation of multiple applications aiming to provide physicians and healthcare institutions with the necessary tools to improve their individual performance and level of care. These systems are highly heterogeneous and are responsible for the creation of big pockets of data that end up being scattered throughout the healthcare infrastructure.

Such pockets of data contain very valuable information that could be put to use, for example they could be employed to assess the levels of performance of each healthcare infrastructure at different levels, ranging from the institutional level to each individual healthcare professional. It all depends on the quality and detail of data that is being produced at the institution. However helpful this information may be, very few hospitals are prepared to take advantage of every source of potential piece of information the IT infrastructure produces. As such, every day valuable information ends up being lost before it can be properly analysed and integrated into some useful metric.

We believe that our proposal takes one step further and allows healthcare facilities to recover such pockets of data and put them to good use by producing useful statistics about daily basis activities.

5.1 Research Summary

In this thesis we studied a series of challenges derived from the development and deployment of a multitude of heterogeneous systems. We looked for the solutions implemented when trying to solve the system integration problems. The communication standards and interface engines used by health providers in their network infrastructure present a good opportunity for performance metric extraction since much of the everyday activities require some kind of network communication between different systems.

We also looked for metric extraction techniques on the academic literature. However, we didn't find much useful information regarding systems capable of dynamically gathering data directly from the network and with the constraint of being completely separated from the actual TCP data streams.

We have described and implemented an architecture for a system capable of incrementally building a knowledge database for an healthcare facility based on standard protocol messages transmitted through the network. We were able to efficiently extract HL7 messages directly from the network with the additional advantage of not having to depend on physical memory in order to reconstruct out of order packets since we use the information contained in TCP headers in order to calculate the precise point where each piece of data fits in the content.

We have been able to use this data mainly for two different goals. From a monitoring point of view, the data gathered can be used to find normal levels of performance for a given healthcare facility and with that information, one can easily detect outliers that result from malfunctioning sections of the healthcare infrastructure. A deeper analysis of this data can also be used to support decision makings from an administrative point of view.

We have also described a set of other uses for our system architecture. Namely, after the message extraction from the network, one can also build a network of systems that could receive anonymized HL7 messages and produce a for example a new service based on the data received such as HL7 semantic and syntactic quality assessment.

We believe we have achieved our initial goals with some additional advantages of being able to design and deploy a system capable of easily scaling and general enough so it can be placed at several different healthcare facilities without the need to reconfigure or modify any of the existing institutional systems.

However, our project still needs improvements at different levels of the infrastructure as well as a thorough validation of the data gathered and the statistics produced.

5.2 Main Findings

Integration techniques based on the usage of message standards represent one hot topic when related to healthcare facilities. There is a significant number of recent publications trying to assess the effective advantages and gains when healthcare facilities invest on the development of their IT infrastructure [19].

However, apart from the integration techniques, literature related to TCP stream reconstruction is somewhat scarce when we tried to look for applications of known tools outside IDS or IPS scenarios. Although we found that there is a considerable amount of academic literature bent on analysing and describing new sets of algorithms for TCP flow reconstruction without considerable packet loss rates [13, 6]. Unfortunately, the fact that many of the referred literature is based on the assumption that the sniffing node is placed in-line with the TCP stream invalidates the usage of such tools in our systems.

Related to eHealth, we also learned that the development of hospitals' IT infrastructures has left these institutions with a multitude of different systems incapable of exchanging information without external intervention. To solve such problems, hospital facilities started employing integration engines capable of translating messages so that different systems can interact together and exchange meaningful information. However, one interesting point related to this topic is connected to the fact that only a small set of healthcare facilities applies this type of approach to solve their interoperability challenges. Instead, the majority of healthcare institutions still rely on software vendors agreeing on changing their products in order to meet a predefined set of requirements.

5.3 Current Limitations

Our system has already yielded some interesting statistics, however there is still space for several improvements, both from the hardware and software point of view.

In terms of hardware, the system is heavily limited by the processing capabilities of the Sniffer node at several levels. That is, starting on the NIC, we believe our overall system would greatly benefit from the usage of a hardware capable of automatically associate each packet with an extraction timestamp directly calculated from hardware [4]. With this, the cost of timestamp association with each message could be greatly reduced since in our current implementation, such timestamp can only be calculated in user space.

Apart from the NIC on the sniffer node, one could also benefit from using a CPU capable of offering more computational power in order to reduce the amount of time each packet needs to remain in user space to be analysed. Also from an hardware point of view, the usage of Solid-State Drive (SSD) hard disks could also improve the overall performance of our sniffer node, since the TCP stream reconstruction is made directly on the hard drive in order to reduce amount of physical memory needed.

From the software point of view, the current deployed version of our system is unable to deal with fragmented IP packets. As for now, our system simply discards any packet fragmented at the network layer. Tests have already been made in order to provide the Sniffer node with the capability to reconstruct fragmented packets, however, the reconstruction of such packets proved to be too slow when using the hard disk.

5.4 Future Work

During the development of this thesis, interoperability became one of the main topics when searching for information related to healthcare facilities. Hospitals have adopted the usage of a set of standards in order to have heterogeneous systems exchanging information. The work developed under this thesis uses the HL7 standard as the ground basis to build a knowledge database and produce all the related statistical information. However, apart from the HL7 standard, healthcare facilities use a multitude of other messaging standards. Standards such as the DICOM used in order to transfer an store heavy radiology imagery in healthcare facilities.

As future work, we want to concentrate our efforts in supporting more healthcare standards and be able to draw significant information based on the collected data. The support for different standards should enable us to empower our knowledge database with sufficient data to produce more well grounded statistics with more incisive views on several business processes.

5.5 Conclusion

The proposed infrastructure implements a systems capable of extracting performance metrics on a given healthcare infrastructure as well as produce visual data about the collected data. Our systems improved a known tool for dynamically reconstructing TCP streams empowering it with the ability to dynamically create and release the log files containing the payload for the TCP packets according to the “life” of the stream.

The architecture employed also possesses the advantage of allowing an easy scaling of the infrastructure without needing to reconfigure any special part of the healthcare infrastructure. Therefore, our system presents a light and scalable method for extracting performance metrics on any given healthcare infrastructure by taking advantage of the HL7 message standard.

References

[1] Highcharts. `http://www.highcharts.com/`. [Online; accessed 2014/02/8].

[2] Jersey framework. `https://jersey.java.net/`. [Online; accessed 2014/01/18].

[3] pthread library. `https://computing.llnl.gov/tutorials/pthreads/`. [Online; accessed 2014/02/12].

[4] D. Agarwal, J. M. González, G. Jin, and B. Tierney. An infrastructure for passive network monitoring of application data streams. *Lawrence Berkeley National Laboratory*, 2003.

[5] L. Andersen and S. Lead. Jdbc 4.2 specification. 2014.

[6] R. Antonello, S. Fernandes, C. Kamienski, D. Sadok, J. Kelner, I. Gódor, G. Szabó, and T. Westholm. Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. *Journal of Network and Computer Applications*, 35(6):1863–1878, 2012.

[7] F. Barbarito, F. Pinciroli, J. Mason, S. Marceglia, L. Mazzola, and S. Bonacina. Implementing standards for the interoperability among healthcare providers in the public regionalized healthcare information system of the lombardy region. *J. of Biomedical Informatics*, 45(4):736–745, Aug. 2012.

[8] A. D. Black, J. Car, C. Pagliari, C. Anandan, K. Cresswell, T. Bokun, B. McKinstry, R. Procter, A. Majeed, and A. Sheikh. The impact of ehealth on the quality and safety of health care: a systematic overview. *PLoS medicine*, 8(1):e1000387, 2011.

[9] J. A. Blaya, H. S. Fraser, and B. Holt. E-health technologies show promise in developing countries. *Health Affairs*, 29(2):244–251, 2010.

[10] D. Blumenthal. Launching hitech. *New England Journal of Medicine*, 362(5):382–385, 2010. PMID: 20042745.

[11] G. Bortis. Experiences with mirth: an open source health care integration engine. In *Proceedings of the 30th international conference on Software engineering*, pages 649–652. ACM, 2008.

[12] M. B. Buntin, M. F. Burke, M. C. Hoaglin, and D. Blumenthal. The benefits of health information technology: A review of the recent literature shows predominantly positive results. *Health Affairs*, 30(3):464–471, 2011.

[13] N. Cascarano, L. Ciminiera, and F. Risso. Optimizing deep packet inspection for high-speed traffic analysis. *Journal of Network and Systems Management*, 19(1):7–31, 2011.

[14] L. Catwell and A. Sheikh. Evaluating ehealth interventions: the need for continuous systemic evaluation. *PLoS medicine*, 6(8):e1000126, 2009.

[15] Corepoint Health. Hl7 interface engine. `http://www.corepointhealth.com/whitepapers/why-do-i-need-hl7-interface-engine`, 2010. [Online; accessed 2014/04/10].

[16] R. J. Cruz-Correia, J. C. Wyatt, M. Dinis-Ribeiro, and A. Costa-Pereira. Determinants of frequency and longevity of hospital encounters' data use. *BMC medical informatics and decision making*, 10(1):15, 2010.

[17] P. De Meo, G. Quattrone, and D. Ursino. Integration of the hl7 standard in a multiagent system to support personalized access to e-health services. *Knowledge and Data Engineering, IEEE Transactions on*, 23(8):1244–1260, Aug 2011.

[18] L. Duan, W. N. Street, and E. Xu. Healthcare information systems: data mining methods in the creation of a clinical recommender system. *Enterprise Information Systems*, 5(2):169–181, 2011.

[19] M. Eichelberg, T. Aden, J. Riesmeier, A. Dogac, and G. B. Laleci. A survey and analysis of electronic healthcare record standards. *ACM Computing Surveys (CSUR)*, 37(4):277–315, 2005.

[20] F. Falcão-reis and M. E. Correia. Patient empowerment by the means of citizen-managed electronic health records: web 2.0 health digital identity scenarios.

[21] F. Fuentes and D. C. Kar. Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose. *Journal of Computing Sciences in Colleges*, 20(4):169–176, 2005.

[22] S. L. Garfinkel and M. Shick. Applications of passive message logging and tcp stream reconstruction to provide application-level fault tolerance. `http://www.cs.cornell.edu/courses/cs717/2001fa/reports/Passive%20Message%20Logging.pdf`, Dec. 2001. [Online; accessed 2014/05/08].

[23] S. L. Garfinkel and M. Shick. Passive tcp reconstruction and forensic analysis with tcpflow. `http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA585499`, Sept. 2013. [Online; accessed 2014/05/09].

[24] C. L. Goldzweig, A. Towfigh, M. Maglione, and P. G. Shekelle. Costs and benefits of health information technology: New trends from the literature. *Health Affairs*, 28(2):w282–w293, 2009.

[25] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[26] K. Häyrinen, K. Saranto, and P. Nykänen. Definition, structure, content, use and impacts of electronic health records: A review of the research literature. *International Journal of Medical Informatics*, 77(5):291 – 304, 2008.

[27] ISO. Data exchange standards – health level seven version 2.5 – an application protocol for electronic data exchange in healthcare environments. ISO/HL7 27931:2009, International Organization for Standardization, Geneva, Switzerland, 2009.

[28] Jeremy Elson. tcpflow. `http://www.circlemud.org/jelson/software/tcpflow/`, 2003. [Online; accessed 2013/10/25].

[29] R. Konrad, B. Tulu, and M. Lawley. Monitoring adherence to evidence-based practices. a method to utilize hl7 messages from hospital information systems. *Applied Clinical Informatics*, 4(1):126–143, 2013.

[30] G. L. Kreps and L. Neuhauser. New directions in ehealth communication: Opportunities and challenges. *Patient Education and Counseling*, 78(3):329 – 336, 2010. Changing Patient Education.

[31] P. Li, T. Wu, M. Chen, B. Zhou, and W. Xu. A study on building data warehouse of hospital information system. *Chin Med J*, 15:2372–2377, 2011.

[32] I.-C. Lin, Y.-H. Hou, H.-L. Huang, T.-P. Chu, and R.-E. Chang. Managing nursing assistants with a web-based system: An empirical investigation of the mixed-staff strategy. *Journal of Medical Systems*, 34(3):341–348, 2010.

[33] Microsoft. Biztalk server. `http://www.microsoft.com/en-us/server-cloud/products/biztalk/default.aspx`. [Online; accessed 2014/03/28].

[34] Mirth. Mirth connect. `http://www.mirthcorp.com/community/wiki/`. [Online; accessed 2014/03/28].

[35] J. Schweitzer and C. Synowiec. The economics of ehealth and mhealth. *Journal of health communication*, 17(sup1):73–81, 2012.

[36] S. Umer, M. Afzal, M. Hussain, K. Latif, and H. Ahmad. Autonomous mapping of hl7 rim and relational database schema. *Information Systems Frontiers*, 14(1):5–18, 2012.

[37] J. Weber-Jahnke, L. Peyton, and T. Topaloglou. ehealth system interoperability. *Information Systems Frontiers*, 14(1):1–3, 2012.

[38] M. Yuksel and A. Dogac. Interoperability of medical device information and the clinical applications: An hl7 rmim based on the iso/ieee 11073 dim. *Information Technology in Biomedicine, IEEE Transactions on*, 15(4):557–566, July 2011.