Reverse Engineering Packet Structures from Network Traces by Segment-based Alignment

Thesis submitted for the degree of

Doctor of Philosophy

at the University of Leicester

by

Othman Esoul

Department of Informatics

University of Leicester

July 2018

To my loving parents

Reverse Engineering Packet Structures from Network Traces by Segment-based Alignment

Othman Esoul

Many applications in security, from understanding unfamiliar protocols to fuzz-testing and guarding against potential attacks, rely on analysing network protocols. In many situations we cannot rely on access to a specification or even an implementation of the protocol, and must instead rely on raw network data "sniffed" from the network. When this is the case, one of the key challenges is to discern from the raw data the underlying packet structures – a task that is commonly carried out by two steps: *message clustering*, and *message Alignment*.

Clustering quality is critically contingent upon the selection of the right parameters. In this thesis, we experimentally investigated two aspects: 1) the effect of different parameters on clustering, and 2) whether suitable parameter configuration for clustering can be inferred for undocumented protocols (when messages classes are unavailable). In this thesis, we have quantified the impact of specific parameters on clustering, and used clustering validation measures to predict parameter configurations with high clustering accuracy. Our results indicate that: 1) The choice of the *distance measure* and the *message length* has the most substantial impact on cluster accuracy. 2) The *Ball-Hall* intrinsic validation measure has yielded the best results in predicting suitable parameter configuration for clustering.

While clustering is used to detect message types (similar groups) within a dataset, sequence alignment algorithms are often used to detect the protocol message structure (field partitioning). For this, most approaches have used variants of the *Needleman-Wunsch* algorithm to perform byte-wise alignment. However, they can suffer when messages are heterogeneous, or in cases where protocol fields are separated by long variable fields. In this thesis, we present an alternative alignment algorithm known as *segment-based* alignment. The results indicate that segmented-based alignment can produce highly accurate results than traditional alignment techniques especially with long and diverse network packets.

Preface

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration, except as specified in the Acknowledgements.

The names of all products referred to in this dissertation are acknowledged as the trademarks of their respective owners.

Othman Esoul

July 2018

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Neil Walkinshaw. It is his advice, encouragement, and enormous patience that helped me get this far. Dr. Neil has taught me how do research, write an academic paper, and grow as a reseracher. His enormous energy for teaching, attending conferences, and performing research have made him a very busy person, but he has always had time for our meetings. I would like to thank him for his outstanding supervision, and the tremendous support he offered me throughout my PhD.

I also would like to thank all my friends who helped during this period especially Salah Albukhasheim and his wife for their encouragement and endless support. It is impossible to enumerate the generosity and ways in which they have helped during this journy.

Finally, I am grateful to my family in Libya. To my parents for their unwavering love and support for my education, and to my bothers and sisters for their encouragement and dedication to help.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Protocol reverse-engineering (or protocol inference) is concerned with the challenge of inferring a specification of a network protocol from its available artefacts. It is often that application protocols are targeted for reverse engineering because many application protocols are undocumented or have no publicly available specification (closed). Inferred protocol specification can be valuable in a multitude of scenarios, such as *intrusion detection systems* (IDS) [4], *protocol fuzz testing* [5–7], *application fingerprinting* [8], *traffic classification* [9, 10], *detecting implementation deviations* from original protocol specifications [11], and in generic *network protocol analysers* [12].

For such applications, it is generally necessary to have an existing model that describes the expected behaviour of the network protocol. In practice however, such models tend to be only readily available for generic protocols with well-established characteristics. Generating such specifications by hand can be an arduous, error-prone task, especially when the protocol in question is unfamiliar and not accompanied by detailed documentation (e.g. the implementation is provided in a third-party component). It took more than a decade for

a team of reverse engineering experts to infer specifications of the *Server Message Block* protocol (SMB) [13] into an open source project known as *SAMAB* [14]. Also, the process of protocol reverse engineering is not a once-and-done matter; existing protocols are often extended to support new commands and functionalities. Therefore, automating the process can successfully speed up the inference time and preserve the effort.

Specifications for open protocols such as the *File Transfer Protocol* (FTP) [15] can be retrieved by accessing public documents (e.g., *Request For Comments* (RFC) [16]), or in some cases, derived from the available source code. However, there are many closed and proprietary protocols that have no released specifications, such as the *Skpe* protocol [17]; protocols used by instant messaging clients such as AOL's *ICQ* [18]. Also, *Malware* (e.g., Botnets [19]) use undocumented command-and-control (C&C) protocols to facilitate stealthy communications between the controlling server and infected clients [20–22]. Earlier studies report that more than 40% of the internet traffic belongs to unknown protocols [23] and there is no sign that this trend is going to decrease in the years ahead.

Specifications for closed protocols need to be reverse engineered. Currently, there are two common approaches for this task: **(1)** by *reverse engineering executables* of the network protocol (e.g., sever), and **(2)** by *analysing captured network traffic*. Typically, the first approach is carried out through the *dynamic analysis* of the program which involves monitoring the execution of the protocol binary at run-time, i,e,.analysing the program execution in terms of machine entities such as instructions, registers and memory locations. The second approach is normally is based on the captured trace which is generated by the protocol program (client/server).

Typically, protocol reverse engineering comprises of two steps: **(1)** extracting the protocol *message formant*, which captures the structure of all message types in the protocol, and **(2)** constructing the protocol *state machine*, which describes the sequences of messages that represent valid protocol sessions.

This thesis focuses on inferring the protocol *message structure* using captured *network traces*, and leaves state machine inference to future work. Also, this thesis deals with application protocols that do not encrypt or obfuscate their communications.

1.2 Purpose of This Thesis

The process of protocol reverse engineering from network traces is complex. The foremost difficultly is that protocol reverse engineering is not a term for a single integrated technique with well-defined rules; rather it is an umbrella for a collection of heuristic procedures, diverse elements of data mining algorithms and applied statistics. Generally, protocol reverse engineering is a process often based on two aspects of data mining: *data clustering*, and *sequence alignment* algorithms. However, these two aspects have their own limitations (as discussed below) when applied to protocol inference.

A crucial step in protocol inference from network traces is to classify captured messages of the same type into separate groups. Most approaches [24–27, 23] accomplish this step by identifying common patterns within the data by way of an *unsupervised*[1] data mining technique known as *clustering* [28]. Clustering can empirically elucidate the "natural", unknown and ideally interesting groups of messages within the captured network trace. These groups can then be used to identify the possible structures of message types implemented in the protocol.

Most of protocol inference approaches that involve clustering follow a common sequence of steps, but vary substantially in terms of the specific methods or parameters that they adopt with respect to the clustering step. For example, they might pre-process the data in different ways (e.g. limit messages to the first *32*, *64* bytes etc., or tokenise messages into tokens such

[1] Unsupervised learning mostly refers to data mining techniques that group data items without pre-specified class labels.

as n-grams etc.). They might adopt different combinations of "distance measures". They might be tailored towards text-based protocols or binary ones.

It is important to understand that clustering is often combined with a set of factors applied on the natural structure of the dataset. The extent to which these factors distort or improve the structure is considered an ever present risk. The fact that different set of factors can suggest different clustering results when applied on the same dataset signifies the importance of understanding the influence of these parameters on the inferred model.

Most of the empirical results are presented with respect to a fixed configuration of clustering parameters. However, the sensitivity of clustering algorithms to their parameters suggests that performance could vary significantly [29, 30], depending on factors such as the type of protocol, the choice of distance measure, the amount of data, etc. A review of previous applications is not very helpful because authors typically give only their final selections and rarely provide any insight into the process leading to the choices. A reverse engineering analyst who has a set of messages to be clustered faces quite a number of questions such as: How does one choose among similarity measures, packet lengths and sample sizes? What is the effect of choosing certain factors on clustering quality? What is the effect of these factors on clustering time? What is the best factor configuration for clustering? Such vital questions are left unidentified by previous approaches.

This part of the thesis aims to provide an approach to answering such questions as well as an empirical evaluation on realistic protocols that demonstrates the effect of such factors on clustering as well as detecting the best factor combinations for clustering.

Another important step for a protocol inference technique is to infer the packet structures from the data – to identify within packets the various data fields and field headers. Current approaches [24, 31–33, 26, 34, 23] tend to identify common patterns by attempting to align classified (clustered) protocol packets. Aligning a large number of packet sequences can

identify commonalities and variances, which can in turn be used to identify, for example, the tokens that are used to delimit packets, the key field identifiers, and the data fields.

Although there has been a substantial amount of research in the area, most of the emphasis has been placed on either stages *prior* to the alignment [24, 23] or on challenges such as the inference of the protocol state machine (once packet types have been identified) [31, 27, 35–37]. The underlying algorithm that is used to align packets to identify their structure tends to be the same for most techniques – the *Needleman-Wunsch* algorithm [38].

Although well suited for its original purpose of protein sequence alignment, Needleman-Wunsch can become problematic when applied to sequences of bytes from network packets. For one, it is highly sensitive to various parameters (such as the "gap-penalty" parameter) that, though honed through decades of use on protein sequences, are far from straightforward to identify for network packets (and in all likelihood need to be varied on a per-protocol basis). Secondly, it can produce highly inaccurate results when messages have an identical packet structure, but happen to contain variable-length data fields.

In this thesis, we propose the use of *segment-based sequence alignment* [39] to align network messages. Instead of aligning messages on a character-by-character basis, segment-based alignment constructs alignments in terms of entire sub-strings. The algorithm is not dependent upon any user-defined parameters. Also, because it operates in terms of sub-sequences, it is more forgiving of slight discrepancies when comparing protocol messages that would confound Needleman-Wunsch. The approach has proven to be a successful replacement for Needleman-Wunsch within bioinformatics, and in this thesis we seek to show that it can provide a similar replacement with respect to protocol reverse-engineering.

1.3 Contributions

The purpose of this thesis is to investigate and improve the process of protocol reverse engineering from network traffic. Accordingly, this thesis makes the following contributions:

- It presents an empirical study investigating the impact of various process factors on clustering accuracy and clustering time. The study also demonstrates that intrinsic validation measures for clustering can be used to determine suitable factor configurations to achieve highly accurate clustering when message classes (types) are unknown, which is often the case for undocumented protocols.

- This thesis proposes the use of segment-based alignment to address some of the limitations of traditional alignment algorithms (e.g., Needleman-Wunsch), which is used to identify packet structures from network traces. The proposed technique depends on less user parameters and yields significantly higher accurate alignment especially with long and diverse protocol messages that contain different compositions of protocol fields.

- The proposed solution of segment-based alignment has been implemented into a framework that is capable of reverse engineering the message structure from captured network data. Unlike previous approaches, the framework works in a completely protocol-independent fashion, i.e., no information is used from other protocols in the protocol stack (other than the application protocol) and no assumption is made about the nature of protocol type (text/binary), or its behaviour (synchronous/asynchronous). Also, the framework does not assume that the first constant bytes of a packet describe the complete structure of an application protocol as described in previous projects, or assumes any prior knowledge about protocol delimiters that separate the different fields in a message. Generally, the framework takes the captured network traffic as the

input and automatically outputs the inferred protocol message structures in a form of message patterns.

- This thesis also proposes a novel approach to evaluate the accuracy of the inferred message structures. Instead of relying on the conventional analytical approach of scrutinising inferred packet structures, we use what we consider to be a more empirically valid approach. We use the inferred packet structures to synthesise protocol messages, which we send to servers, and track whether or not the packet is parsed as valid or not.

- Finally, this work offers preliminary insights (presented as a future work) into finding the hierarchical structure (context-free grammar) of the inferred message patterns. The contribution of this part demonstrates how the *SEQUITUR* algorithm [2] could be used to infer the hierarchical message structure for text-based protocols.

1.4 Thesis Structure

The rest of the thesis is laid out as follows:

Chapter 2 gives a background related to the thesis, and divides it into three sections: network protocols, data mining, and research design. The first section is served as an introduction to network protocols and discussed from two perspectives: protocol models for communications and the protocol message structure. Section 2.2 explains selected subjects on data mining and knowledge discovery that are associated with this work. This involves: data clustering, and sequence alignment. This chapter concludes with a background on experimental research and the statistical tests (effect size) that relate to the empirical study carried out in this thesis.

Chapter 3 consists of two sections. Section 3.1 provides a comprehensive review of previous protocol reversing approaches with special emphasis on the network-based inference approach. This chapter also highlights the common steps that is usually followed in protocol

[2] A recursive algorithm that infers hierarchical structure from a sequence composed of discrete symbols.

inference from network traces. The final section of this chapter discusses the motivations of the thesis.

Chapter 4 gives details of the design and implementation of a generic framework for reverse engineering the message format of application protocols from captured data. The first section in this chapter discusses the key desired properties for the envisioned framework. While section 4.2 gives details of the blueprints and building blocks of the framework, section 4.3 describes its implementation details. The last section highlights some of the known limitations of framework.

Chapter 5 addresses the first motivation of this thesis. It demonstrates how the constructed framework is used to conduct an empirical study on a number of protocol traces. Section 5.1 explains the different types of process factors and how they are generated within the inference process. Section 5.2 give details on the conducted experiment which includes: the subject protocols, selected variables for the experiment, methodology, and the results of the experiment. The final section of this chapter highlights some of threats to validity that might have affected the experiment.

Chapter 6 covers two closely related subjects within the inference process: message alignment using the segment-based alignment approach, and extracting message structure using an alignment generalisation technique. Accordingly, section 6.1 explains how the segment-based alignment approach is applied within the context of protocol reversing which includes: explaining the necessary modifications to the weighting scheme as well as the implementation details of the segment-based alignment algorithm. A case study aimed to demonstrate the operational aspects as well as the practicality of the approach is also explained at the end of this section. Section 6.2 explains how an alignment generalisation technique is used to extract message patterns which serves to describe the overall structure of the message.

Chapter 7 discusses how the quality of the inferred message structure is evaluated. Section 7.1 describes subject protocols, evaluation methodology as well as the results of the evaluation.

Section 7.2 conducts a quantitative and qualitative comparative evaluations between the segment-based alignment and another Needleman-Wunsch based alignment technique. The last section provides further discussions on the evaluation and the adopted inference approach.

1.5 Publications

This thesis comprises of a work appeared in one conference. The article appeared in the proceedings of the 2017 IEEE International Conference on Software Quality, Reliability & Security (QRS 2017). The paper entitled: *Using Segment-based Alignment to Infer Network Packet Structures from Network Traces* [40]. The paper highlighted the common weaknesses of current message alignment approaches and proposed the use of segment-based alignment to overcome these problems.

Chapter 2

Background

The purpose of this chapter is to establish a general background on the diverse subjects involved in this thesis. First, we begin with a general snapshot of network protocols focusing on the related technologies and terminologies related to our work. The second part of this chapter gives a general description of the process of data mining and the relevant techniques to protocol inference (data clustering and sequence alignment). Finally, the chapter gives a background on the experimental research design and the effect size statistical measure that is related to the empirical study conducted in this thesis.

2.1 Network Protocols

This section provides an overview of network protocols, protocol models and how they are used for communication. This section also explains protocol messages, types of protocol messages and how they are structured emphasising on the main elements connected to this thesis.

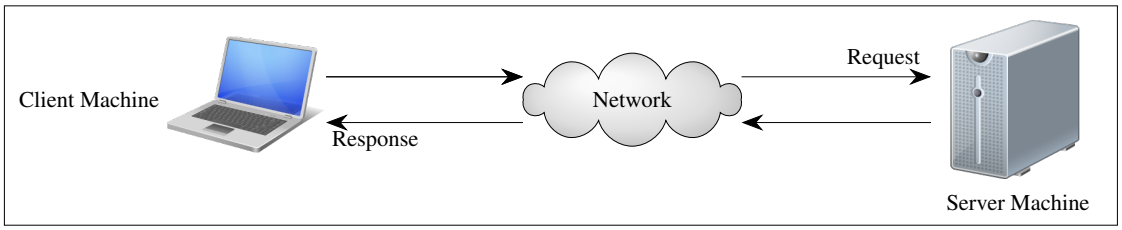


Figure 2.1 Client-Server Communication Model

2.1.1 Communication Models

A network protocol is a set of rules that control communications between two (or more) computer programs. Typically, these rules define the structure and the meaning of the exchanged messages as well as the correct order of the exchanged messages based on a well-defined model for communication. A communication model can be defined as an abstract representation that describes the communication infrastructure involving the communicating parties, means for communication, as well as the assigned roles to them.

There are a number of protocol communication models, such as the *Peer-to-Peer* [41] and *Client-Server* models [42]. We will be focusing on the client-server model since it is the most common model of communication. The client-server model consists of three elements: a *Client* program, a *Server* program, and a *network* medium that facilitates the communications between the client and server as illustrated in Figure 2.1. The client-server model is a two-way communication model where the client is the service requester and server is the service provider. The client and server programs are normally set-up on two separate machines connected by a network.

Protocol Hierarchy

Communication between client and sever programs is not direct. In order for the client and server to communicate, their exchanged messages have to go through several stages. There are

multiple intermediate communication interfaces that need to handle specific communication tasks.

Technically, each of these interfaces is implemented in a separate protocol and organised as layers of protocols where each layer is able to interact with the layer above or below it. The basic idea behind the layered protocol architecture is that each layer offers a different level of abstraction and performs a separate function. Each layer offers specific services to the layer above it through a well-defined interface. The interfaces are dependent on each other and collectively deliver the intended function of the protocol.

The number of layers and functions for each layer normally follows a specific model. The best known models are the *Open System Interconnection* model (OSI) [43], and the *TCP/IP* model [16]. The OSI model consists of seven layers and considered the standard for the layered protocol architecture. In practice, however, protocol implementations do not really adhere to the standard OSI architecture. Instead, they use the TCP/IP model, which is based only on four layers. Since our work is based on network protocols that have adopted the TCP/IP model, we will focus our background discussion on TCP/IP model.

The TCP/IP model is a combination of several protocols. The TCP and IP are only two of the protocols (layers) in the stack. Each layer hosts one or more protocols communicating with its peer at the same layer on the other side, as illustrated in Figure 2.2. The model of TCP/IP consists of the following layers:

1. **The Link Layer:** The link layer normally incorporates two elements: the device driver (e.g., Ethernet and Token ring) and the hardware interface. Together they handle the physical interaction with the network media. This layer is also responsible for masking any transmission errors and regulating transmission speed between connected computers.

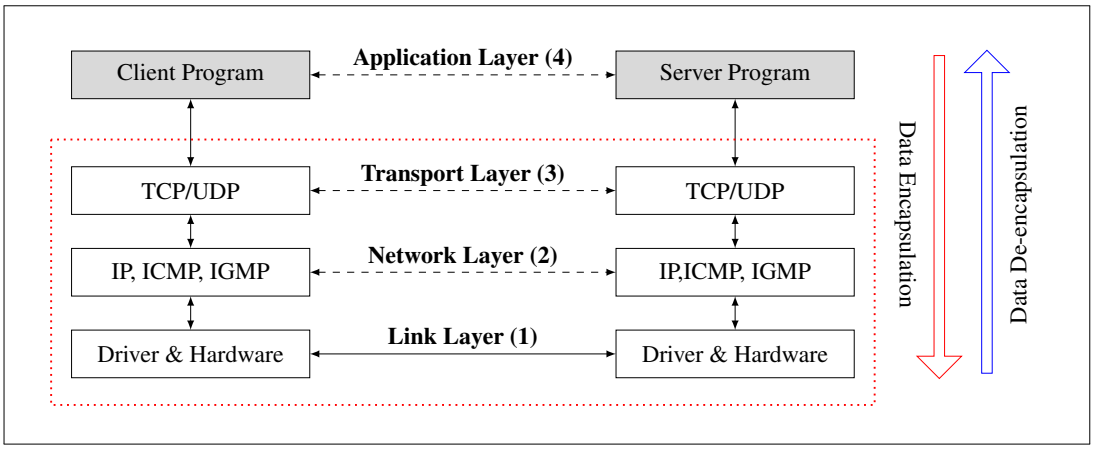


Figure 2.2 TCP/IP Protocol Suite.

2. **The Network Layer:** This is sometimes called the *Internet Layer*. This layer controls the flow of packets around the network by determining how packets are routed from the source computer to the destination computer. This layer typically includes the implementation of the IP (*Internet Protocol*), ICMP (*Internet Control Message Protocol*), and IGMP (*Internet Group Management Protocol*).

3. **The Transport Layer:** The basic function of the transport layer is to make certain that transmitted data between the computers are provided for the application programs above it (client/server) and communication port numbers are defined as well. The TCP/IP model includes two different mechanisms of transport implemented in two separate protocols: TCP (*Transmission Control Protocol*) and UDP (*User Datagram Protocol*). TCP provides a reliable exchange of application data by acknowledging received data and setting time-outs for the applications to acknowledge sent packets, fragmenting and assembling data into transmittable units, and verifying the integrity of the transmitted data (e.g., use of checksums). The UDP protocol on the other hand provides a more basic service to applications. It sends data known as *Datagrams* from one the source computer to a destination, but without making sure that these datagrams

are actually received on the other end, a task that is left to protocol applications to handle.

4. **The Application Layer:** The application layer is responsible for handling application details. This layer represents the implementations of a variety of user applications, such as accessing and retrieving documents, and emails, file sharing, video conferencing, etc.

Implementations of application protocols are commonly targeted for reverse engineering and testing because they tend to be undocumented, and more likely to contain bugs [5], whereas other protocols in the TCP/IP stack are well-documented and have been debugged for years.

Protocol Communications

In the TCP/IP model, when one application sends data to another, the data is passed down through each layer in the protocol stack until it is transmitted as a stream of bits to the other end. In this process, each protocol in the stack prepends specific information known as *headers*, and sometimes appends trailers as well to the data (e.g., the Ethernet protocol in Link layer [16]). This process is called *Data Encapsulation*. When the data reaches the other end, the encapsulation process is executed in reverse where the transmitted data starts its way up and all attached headers (and trailers) are parsed and removed by the appropriate protocol peer. This process is called *Data De-encapsulation* (or Demultiplexing [16]), as shown in Figure 2.2.

Typically, Each layer in the TCP/IP has its own *Data Unit*. The unit of the data that the transport layer passes to the network layer is called a *Segment*, and the unit of data that the network layer sends down to the data link layer is called a *Packet*. The unit of data that the data link layer sends to the hardware interface is often known as a *Frame*. Because there is

no common defined data unit for the application layer [16], we will refer to the data unit passed by application protocols to the transport layer as a *Message*.

The TCP and UDP transport protocols identify the application protocol by its port number assigned for communications. Typically, protocol servers are known by their associated port numbers. For example, the FTP (File Transfer Protocol) sever provides that service on TCP port 21, and TFTP (Trivial File Transfer Protocol) provides its service through UDP port 69. The standardisation of port numbers is managed by the *Internet Assigned Numbers Authority* (IANA) [44].

Network protocols are normally categorised into two categories: text protocols and binary-based protocols. In text based protocols, data communicated between the the client and server mostly fall within the printable ASCII characters and the exchanged messages are human-readable. There are several examples of text protocols such as the Hypertext Transfer Protocol (HTTP) [45] and the File Transfer Protocol (FTP) [15]. For binary protocols, on the other hand, data is communicated as stream of bits and the messages contain characters that are not particularly meaningful to humans. The *Trivial File Transfer Protocol* (TFTP) [46] and *Domain Name Service* (DNS) [47] are both binary protocols.

The way protocols communicate depends on the application. For certain applications the protocol may require the server to treat each request from the client as an independent transaction that is unrelated to previous requests so the communications between the client and server consists of separate pairs of request/response messages. In this case the protocol is called a *Stateless* (or *Connectionless*) protocol. For other applications the protocol may require the server keeping the internal state of previous sessions because recent message requests depend on it. In such case the protocol is known as a *Stateful* (or a *connection-oriented*) protocol [41].

2.1.2 Protocol Message Structure

To give a basic idea of the message structure and field definitions within the message header, we will be using the *Trivial File Transfer Protocol* (TFTP) [46] as an illustrative example.

TFTP is a trivial file transfer protocol (as the name suggests). It is also used for other purposes, such as the remote booting of disk-less devices, and even for malicious purposes [32]. TFTP is implemented on top of the *User Datagram Protocol* (UDP) transfer protocol.

The protocol supports five simple operations (five types of messages). In TFTP, each message is composed of a number of *fields*. Each message consists of an Opcode (*Operation Code*) to indicate the type of the operation and a few other fields for other purposes. The overall number of fields in each message varies from one message to another depending on the operation type.

For instance, the Read and Write messages are exactly the same apart from the value of the Opcode, as illustrated in Figure 2.3. The format of the Read (and Write) messages consists of five fields. The first field is of a two-byte length and used to indicate the operation type (01 for read, and 02 for write), immediately followed by another field for holding the file name that we want to read or write. At the end of the file name, a field of one-byte is used to indicate the end of the string (null character). The message also contains a field for the communications mode, which indicates the method that should be used for encoding data before transmission. TFTP supports three modes of communications: "ascii", "octet" and "mail". Each Read/Write message ends with a one-byte field contains a null character signalling the end of the message.

As shown in Figure 2.3, fields in a protocol message differ in length, type (numeric, string etc.), and purpose. Accordingly, a field in a protocol message may fall in one or more of the following categories:

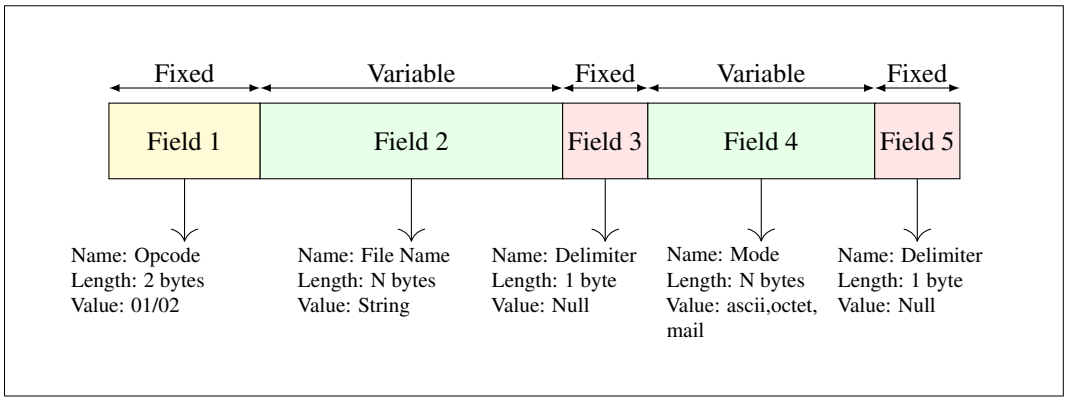


Figure 2.3 Simple illustration of the Read/Write message structure for the TFTP Protocol.

- **Fixed-length:** Each field in the protocol message is either of a fixed length or variable length. The length of the fixed-length field does not change across multiple instances of the same message. Normally, the lengths of fixed-length fields are specified in the protocol specifications and should be known to the protocol implementers. Fixed-length fields are typically used when maintaining certain field lengths is needed in a protocol message. In the TFTP example, the Opcode field is a fixed-length field.

- **Variable-length:** The length of a variable length field is dynamic, therefore, it changes across multiple instances of the same message. Protocol designers should explain how the boundary of a variable-length fields should be determined in the specifications. Typically, *Delimiters* or *Length Fields* are used to mark the end of a variable-length field. Variable-length fields are commonly used by the applications when the protocol data maintain no structure. The File-name field in the TFTP example is a variable-length field.

- **Length-Field:** For some protocols, a message may contain a field to describe another field such as a length field. A length field is a field that holds the length of another field of a variable-length. Typically, a length field always proceeds its variable-length field in the protocol message. The unit used to measure the length is in bits or bytes

(depending on what is described in the specifications). Length fields are commonly used in text-based protocols such as the HTTP protocol.

- **Delimiters:** Delimiters are special characters used to mark the end of variable-length fields. A delimiter may consist of a single byte or multiple bytes and always appears at the end of a variable-length field. Delimiters are part of the protocol specifications and known to the developers. The Null character in the TFTP example is a delimiter used to indicate the end of two fields of variable length: the file-name field, and the mode field.

- **Keywords:** Keywords are special strings or numbers (or sometimes a combination of both). Keywords are typically used as part of the protocol *Commands*, *Requests* and *Responses*. Keywords are sometimes also used for information purposes. They are determined by the protocol specifications and have to be known to the protocol developers.

Typically, the format of the protocol message is part of the protocol specification. Protocol specifications are normally documented in an official document know as *Request For Comments* (RFC)[1]. An RFC document normally contains information on the protocol elements such as the communication rules and message formats, as well as the design decisions and the correct implementation of the protocol. An RFC document also explains any security considerations that need to be addressed. The TFTP specification is publicly available in RFC-1350 [46].

2.2 Knowledge Discovery and Data Mining

In this section, we will provide an introductory overview to the general process of knowledge discovery using data mining techniques. First, we explain the main steps of the process,

[1]There are RFCs published for only information purposes and not considered official [16].

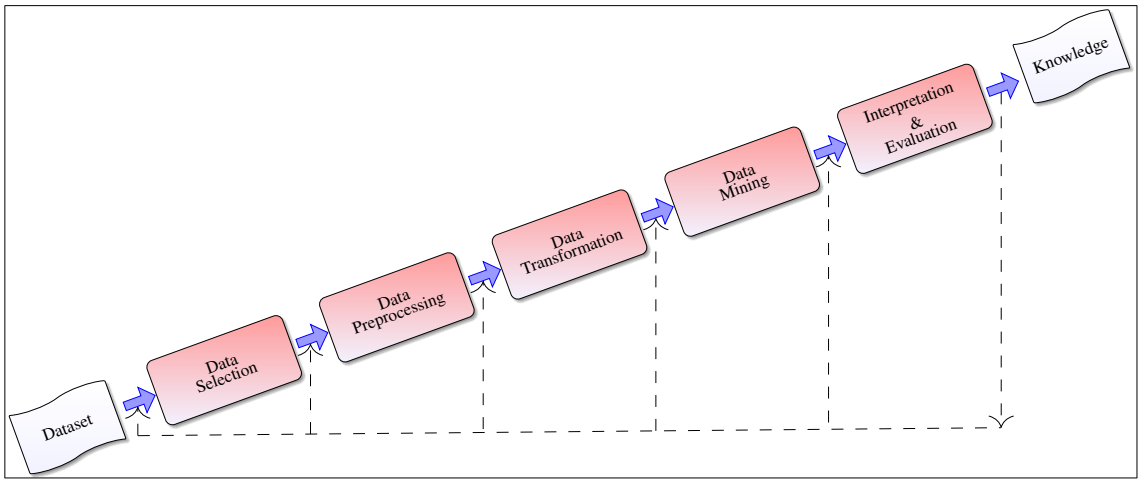


Figure 2.4 The process of knowledge discovery (From [2]).

then we narrow down our discussion on two data mining techniques (*Data Clustering* and *Sequence Alignment*) which are considered the foundation of many protocol reverse engineering approaches.

2.2.1 General Process

The process of knowledge discovery is defined as the extraction of useful and novel knowledge from large data sets using data mining algorithms [2]. The goal of the process is to turn large, unstructured and detailed volumes of data into concise, structured and more useful descriptions that can be interpreted and understood by humans or processed algorithmically. The process of knowledge discovery is often interactive and iterative, and involves a number of steps [2]. Figure 2.4 shows the main steps of the process.

1. **Data Selection:** After developing a good understanding of the application domain and the goals have been determined, data samples should be identified. This requires knowledge of the type and size of the data. This step is the foundation that we base or evidence upon to construct our patterns and models. Therefore, it is imperative to select a good representative data sample. However, collecting, organising, and operating

on complex and large data is expensive and time-consuming, and there should be a trade-off between the size of the sample and the other factors affecting the process.

2. **Data Preprocessing:** Having understood the goals, and selected a representative sample from the data set, the preprocessing step aims to enhance the reliability of the chosen sample. It does so by eliminating outliers (noisy elements), removing redundant data, and handling missing values (if any). This step may contain within it a variety of statistical methods for filtering data. This step is essential because it can reduce the processing time and significantly enhance the accuracy of the inferred pattern or model.

3. **Data Transformation:** Data mining algorithms expect the data sample to be in the shape and form that allows the algorithm to effectively produce the desired output. It is often that data needs to be reshaped or transformed prior forwarding it to data mining step (next step). For example, for some data mining algorithms (e.g., clustering), in order to work effectively, they cannot simply differentiate between raw data items within the sample unless these data elements are translated or fragmented into a set of discriminative tokens ('features' [48]). That enables the mining algorithm to determine the level of association between the identified features and data.

   It is also common that features are not all "important" for the inference - i.e., large parts of the features may be noise (irrelevant features). Therefore, eliminating such features can improve the accuracy of the algorithm and reduces the processing time [49]. The process of reducing (or transforming) an n dimensional data to an m dimensional representation (while $m < n$) is often known as *Dimension Reduction*. There are several techniques that are used to reduce data dimensions which generally fall within two categories: *feature selection* and *feature extraction* techniques [50, 51, 48, 52, 53]. Feature selection is a process that selects a subset of the original features, while feature extraction is a process of extracting a set of new features from the original features

normally through some functional mapping (transformation), such as *principal component analysis* (PCA) [54, 50]. The main idea of principal component analysis (PCA) is to reduce the dimensionality of a dataset consisting of many variables correlated with each other while retaining the variation in the dataset up to the maximum extent. The same is done by transforming these variables to a new set of variables known as the principal components (or simply, PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the first principal component retains the maximum variation that was present in the original components. In chapter 5, Section 5.2.4, we will use PCA to visualise the datasets involved in the empirical study. Typically, feature selection algorithms are preferred as they operate on the original attributes of the dataset [50]. Normally, this step is application-specific as we will elaborate more on it in section 3.1 and in Chapter 4, section 4.2.2.

4. **Data Mining:** This step is about identifying and using the appropriate data mining approach (e.g., Classification, Clustering , Regression etc.) and algorithm that suit our application at hand. This is mostly depends on the what want to infer from our data and the type of data available for the analysis .

   Typically, there are two broad goals from carrying out data mining: *Prediction* and *Description* [2]. Accordingly, most data mining algorithms are based on approaches where a model/pattern is inferred. Prediction is often based on a supervised learning process, where data items are labelled and assigned to classes, and the underlying assumption is that the inferred model is applicable to future cases. Descriptive inference, however, is often based on an unsupervised learning process, where data classes are unknown (e.g., clustering), and a pattern is inferred to describe the data.

   It is important to know how to employ the data mining algorithm and integrate it within the process. It is often the case that multiple data mining algorithms are used to carry

out a specific task. Choosing the right order and selecting the right control parameters for each algorithm is important as will be explained in more detail within the context of protocol reversing engineering ahead (section 3.1).

5. **Interpretation & Evaluation:** The last step in the process is how to interpret and evaluate the results. Results are typically evaluated against the goals set out prior to the commencement of the process. This step focuses on evaluating the inferred model (or pattern) with respect to its usefulness, and accuracy. This can involve evaluating the empirical prediction accuracy for an inferred model, or how well the inferred pattern describes the data included in the test.

2.2.2 Data Clustering

Clustering [29] is an important technique in data mining. It can be defined as the process of partitioning a dataset into distinctive groups under some criterion of similarity such that objects in each group are more similar to each other than objects in different groups [55]. Clustering is the subject of extensive research and, has been embraced in a variety of disciplines and applications, especially for pattern analysis and understanding correlations in large datasets [56].

The goal of clustering is often descriptive. Typically, it is used to empirically elucidate the "natural", unknown and ideally interesting groups of objects within a dataset as shown in Figure 2.5. It is normally applied when there are no predefined classes in the dataset, for this reason, it is known as an *Unsupervised Learning* [28]. In data mining, the information gained from clustering is either used separately or as a preprocessing step for further experiments. There are several approaches to clustering, such as *Hierarchical Clustering*, *Partitional Clustering*, and *Density Clustering* [28].

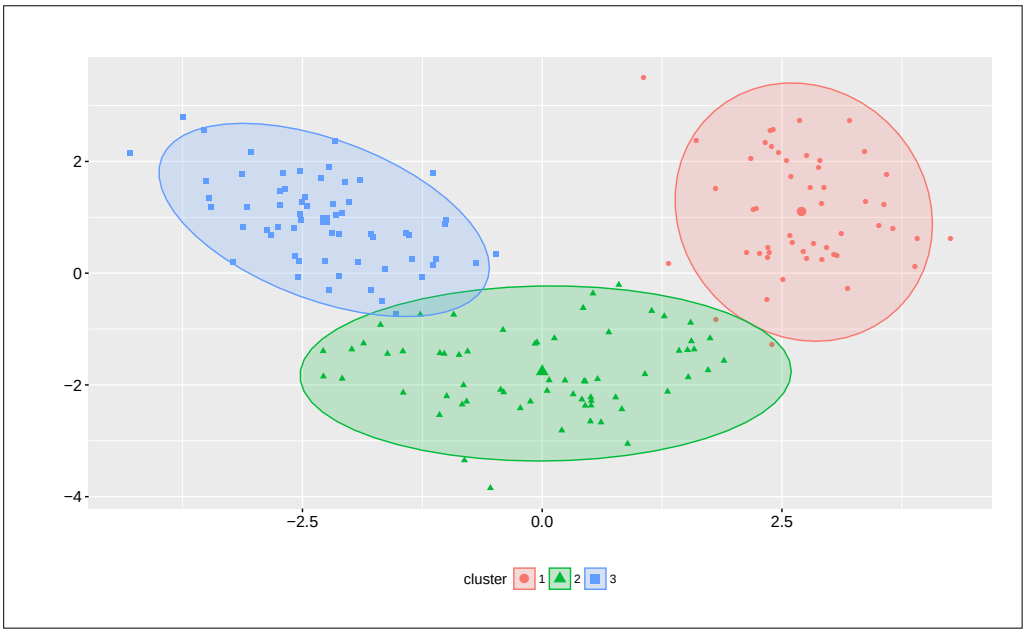


Figure 2.5 An example of a data set with a clear cluster structure

Hierarchical clustering can be used in a variety of applications. However, it is commonly used in biological and social sciences because of the need to construct taxonomies and understand relationships between the clustered objects. Partitional clustering, however, has been preferred in certain engineering applications [56]. In this section, we are restricting our discussion to hierarchical clustering since it is the approach used in this thesis. More detailed discussions on clustering approaches and algorithms are found in [56, 57].

Hierarchical Clustering

Hierarchical clustering can be performed either by recursively merging smaller clusters into larger ones, or starting with a large cluster and recursively splitting it into smaller clusters. The first is a bottom-up approach and known as *Agglomerative Hierarchical Clustering* (AHC), while the second is top-down and known by *Divisive Hierarchical Clustering* (DHC) [56].

The agglomerative hierarchical clustering starts by placing each data item into individual disjoint clusters. Algorithms in this category will merge (nest) the most similar clusters first to form a second cluster, and then the second cluster will be merged with another cluster to form a third cluster, and so on. While the process is repeated to form a set of nested clusters, the number of clusters decreases until a single cluster is created which contains all objects in the dataset. A divisive clustering algorithm will perform the task in reverse order.

The process of hierarchical clustering is normally captured by a special tree structure that provides a picture on how these clusters are formed. The generated tree is called a *dendrogram*. Cutting the dendrogram horizontally at a desired level (hight) defines clustering, and identifies clusters as shown in Figure 2.6.

Unlike other clustering approaches, the number of clusters is not required a priori for hierarchical clustering algorithms. Hierarchical clustering outputs a structure that that is more informative than the unstructured set of clusters that can be produced by partitional clustering because the visual impact of hierarchical clustering can provide invaluable information about the data being explored. A dendrogram enables us to see how objects are merged into clusters at successive levels of proximities. Also, we can determine whether the generated dendrogram describes our data at some fixed level that seems more sensible for the application at hand.

Clustering Method. In agglomerative hierarchical clustering, clustering proceeds according to the chosen *clustering method*. This is a merging method that determines which clusters to be merged to form one cluster. Many Methods for hierarchical clustering have been proposed, such as the *single linkage* method which merges clusters based on their nearest neighbours, the *average linkage* method merges clusters based on their centre neighbours, and the *complete linkage* which merges clusters based on the farthest neighbours.

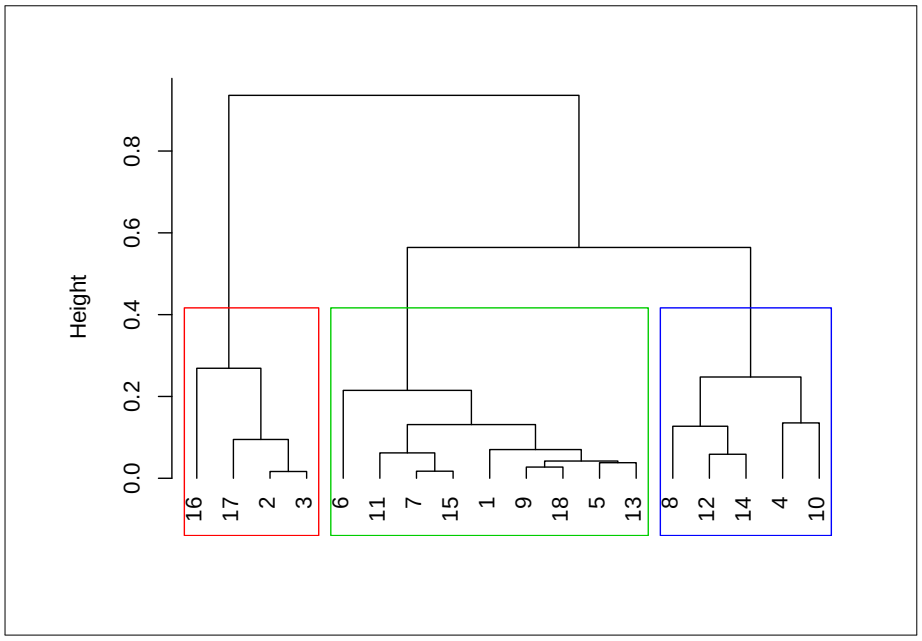


Figure 2.6 A dendrogram consists of 15 data objects, cut off at a level 0.4 creating 3 distinct clusters.

Different clustering methods can produce totally different clustering results. The choice of clustering criteria is difficult to determine. There is no list of characteristics exist that enable us to determine how and when to choose a clustering method in a rational manner [56].

Distance Measures

Most clustering algorithms require a measure of *similarity* to be defined between every pair of objects in the dataset. The similarity measure is often expressed as a *distance* (or *dissimilarity*) and the distance scores between objects are represented in a symmetric distance matrix in which rows and columns correspond to data objects. The more a and b data items resemble each other, the smaller the distance. A distance measure is a function $d(a,b)$ that takes two points in space as arguments and produces a real number reflects the distance between them. A distance measure is called a *true* distance measure (i.e., *metric*) only if

it satisfies certain mathematical properties [2]. There are several metric distance measures described in the literature, such as the *Euclidean* and *Manhattan* distance measures [49, 57]. Distance measures can also be calculated based on a variety of similarity coefficients such as the *Jaccard* index and the *Cosine* similarity measure [58]. Hierarchical algorithms can be seen as a way of transforming a distance matrix into a dendrogram. The hight of the cross-bar in the dendrogram reflects the contrast between clusters within dataset as shown in Figure 2.6. Choosing a suitable distance measure is very important; unless a suitable measure of a distance has been established, clustering results may have no real meaning [56].

Clustering Validation

Clustering validation is the process of evaluating the result of a clustering algorithm. For many applications, it is important to validate clustering results in terms of the 'goodness' of partitions. In general, clustering validation can be divided into two categories, *external validation* and *internal validation*. The main difference between the two categories whether external information is used in the validation process. External validation measures require the actual (ground truth) classes to be known to validate clustering. There are several external validation measures such as, the *Rand Statistic*, and the *Folks and Mallows index* (FM) [28]. Internal measures validate the goodness of clustering based on the intrinsic aspects of the data (e.g., compactness and separation) without the need to the external information. There are a number of internal clustering validation measures such as, the *Dunn index* [59] and *Davies-Bouldin index* [60]. External validation measures are mainly used for choosing between clustering algorithms that suits best for clustering. Internal measures can be used to determine the best clustering algorithm as well as the optimal number of clusters without the need for external information [61].

[2] As described in [29, 56], a true distance measure needs to satisfy three conditions: For all data items a & b, 1) $d(a,a) = 0$, 2) $d(a,b) = d(b,a)$, and 3) $d(a,b) \geq 0$.

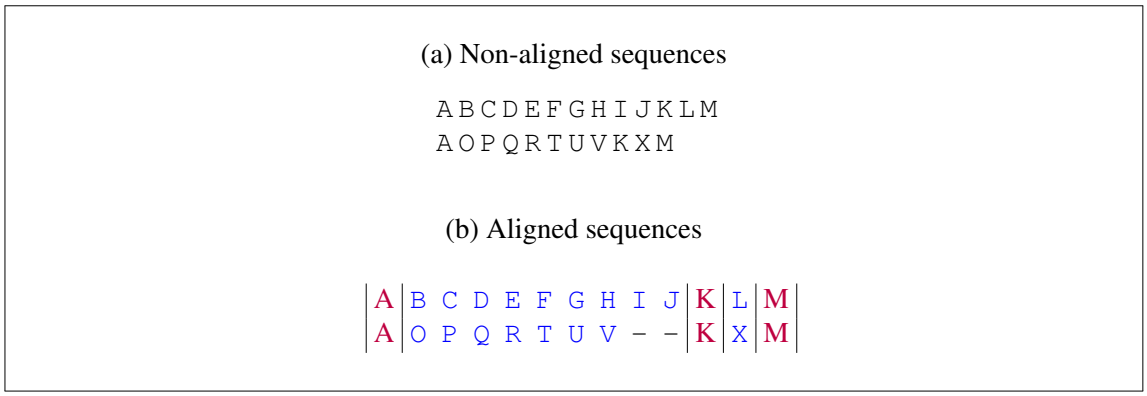


Figure 2.7 Alignment of pair of sequences using the Needleman-Wunsch algorithm.

2.2.3 Sequence Alignment

Sequence alignment algorithms have long been used in bioinformatics [62], for example to identify relationships between protein sequences. An alignment can show precisely where two sequences are identical – which zones of the two sequences match each other, potentially indicating that they are related in some way. The basic task for a sequence alignment algorithm is to determine how two sequences are related. The algorithm aligns the two sequences by comparing characters from both sequences revealing similarities, differences, and missing residues by inserting gaps if the sequences are not of the same length.

For example, in Figure 2.7, the alignment algorithm takes as input a pair of non-aligned sequences and produces an alignment that identifies the elements 'A', 'K' and 'M' as similar (marked as red). Also, it shows that there are two gaps inserted in the second string, because the first one has two additional elements. The alignment also shows where the two strings contain different elements at different positions.

Global vs. Local Alignment

There are two common types of alignments; *global* (e.g., Needlman-Wunsch algorithm [38]) and *local* (e.g., Smith-Waterman [62]). Global alignment algorithms have the goal of

matching entire sequences with each other (i.e. finding a corresponding position for every element from the start to the end). Local alignment algorithms on the other hand merely focus on identifying regions that are strongly similar. Global alignment algorithms tend to be better suited to pairs of sequences that are broadly of a similar content, whereas the latter is better suited to sequences that are more diverse in nature. The Needleman-Wunsch and Smith Waterman are both based on *Dynamic Programming* (DP). Dynamic programming is a method based on the *divide-and-conquer* principle used to solve a complex problem by braking it down into smaller sub-problems that can be solved separately; once a solution is found to a sub-problem, it can be used to solve other sub-problems [63].

Scoring Scheme

Most alignment algorithms employ some sort of a scoring scheme to calculate the similarity between sequences. Based on the scoring scheme, the alignment algorithm seeks to maximise the alignment score in order to find the best possible alignment between sequences. Alignment algorithms based on dynamic programming guarantee an alignment with optimal score [62].

The scoring scheme can be as simple as assigning 1 for a character *match*, and 0 for a *mismatch*. A gap is normally penalised by giving it a negative score. Typically, an *identity matrix* can be generated from this simple scoring scheme where, for example, similar characters are given positive scores, and dissimilar ones are assigned negative or no scores. For its use in bioinformatics, there is the additional complication that characters are not simply identical or different. Different pairs of characters (proteins) can share varying degrees of similarity which makes the task for coming up with a scoring scheme quite difficult.

Multiple Sequence Alignment

Traditional global and local alignment algorithms are based on *dynamic programming* which cannot be easily extended to align more than two sequences as it becomes prohibitively expensive. For this reason, various approaches have been developed to align multiple sequences, leading to a huge number of algorithms using fundamentally different approaches such as *progressive*, *iterative*, *hybrid*, etc. Traditionally, the most common approach has been the progressive alignment. This approach operates by initially aligning two sequences (typically the most similar pair) using Needleman Wunsch algorithm, and then 'progressively' adding additional (more distant) sequences to this fixed alignment. A number of programs based on the progressive approach has been developed [62].

Progressive alignment methods strongly depend on the initial alignments, and once a sequence has been aligned and added to the alignment list, its alignment is not considered again. While this approach offers speedy alignment for large data sets, it comes at the cost of sacrificing some accuracy [64].

Recently, several alignment algorithms have been proposed using an iterative (aggressive) procedure, and some times a hybrid of both. Iterative alignment works similarly to the progressive method but it aims to improve the accuracy of the alignment by repeatedly visiting and re-aligning initial sequences as well as adding new sequences to the alignment list. Iterative alignment algorithms can offer better alignment. However, they are slower than progressive alignments [64, 65]. *Segment-based* alignment (discussed below) is an alignment approach which is based on the iterative approach [39].

Figure 2.8 shows one possible multiple alignment of five different sequences using the progressive approach. Note that similar characters are aligned to one another. Gaps are also inserted into sequences 2,3,4, and 5 to align them with sequence number 1. The sequence shown below the alignment is the generalisation sequence which will be explained ahead.

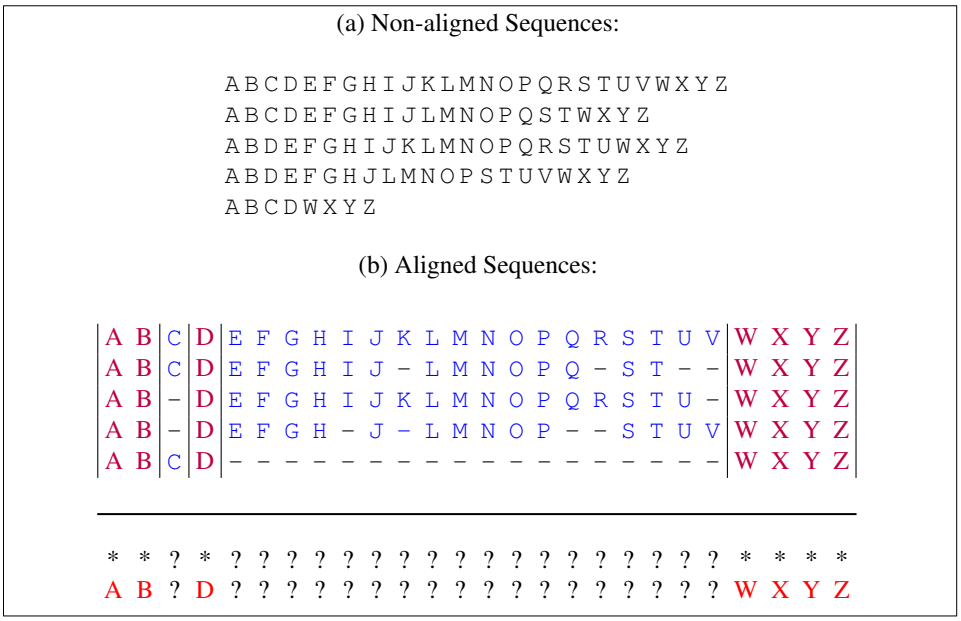


Figure 2.8 The alignment of multiple sequences using the progressive approach and the inferred consensus sequence (marked in red) below the alignment.

Alignment Generalisation

The outcome of the alignment step can be quite daunting to understand when large data sets are involved because the general structure cannot be easily observed. For this reason, it is important to produce a generalised result after the alignment step. The generalisation step serves as a summary of the alignment outcome.

A common method used to generalise the results of sequence alignments in bioinformatics is known as the *consensus sequence* [66]. A consensus sequences is a sequence that simplifies the alignment result by separates static regions from dynamic regions within the aligned sequences, underlying the similarities shared by all sequences. Revisiting the example shown in Figure 2.8, we notice that similar characters ("A","B","D","W","X","Y" and "Z") which appeared in the same positions of the five sequences are shown below the alignment, while different and infrequent characters are filled with gaps. Alignment generalisation gives an abstract description of how sequences are structured.

Another (more informative) method used for alignment generalisation is known as a *sequence logo* [67]. A Sequence logo is a graphical representation of generating the alignment in which the size of character is related to the frequency of that character occurring at certain position. Sequence logos use the information theory to quantify the information content (IC) for each aligned character [66].

2.2.4 Segment-based Alignment

In this subsection we provide a background on Segment-based alignment. First, we give an overview on the alignment procedure, then we explain the scoring scheme adopted by this approach. We conclude this subsection with a brief discussion on the time complexity involves the alignment procedure.

Overview

The basic idea of the segment-based approach is to align sequences by comparison of whole segments of the sequences rather than comparison by single characters. A *segment* is a contiguous sub-sequence of characters within a sequence. Segment-to-segment alignment operates by identifying similar pairs of segments of equal length within the sequences that it seeks to align. This local alignment (sub-alignment) of matched pairs of segments is commonly known as a *fragment*.

In segment-based alignment, each fragment is given a non-negative *weight* (score) that reflects its significance among other fragments (detailed below). The approach then seeks to find a collection of fragments that produce optimal (or near optimal) alignment. Fragments which are chosen to construct optimal alignment must satisfy two conditions: 1) produce the highest alignment score when their weights are summed up, and 2) meet certain *consistency* criteria. A collection of fragments is called consistent if the overall order of the positions in each sequence is respected, i.e., there is no conflicting double or cross-over assignment of

characters between the compared sequences. Fragments may overlap only when different pairs of sequences are involved [68, 69]. The concept of consistency is explained in more detail ahead.

Mismatches are allowed within fragments, however they should not contain gaps in them. When the alignment involves multiple sequences, the segment-based approach offers an extra (optional) weighting mechanism known as an *overlap weight* which reflects the fragment weight as well as the degree of overlap with other fragments, that is to favour patterns occurring in more than two sequences in the alignment process. Although, overlap weights improve the alignment quality, however, this step is time consuming (and normally switched off) when the number of sequences exceeds certain threshold.

In segment-based alignment, the quality of the alignment largely depends on the fragments selected for the alignment. However, similar to the progressive approach, once a fragment is selected, it becomes part of the alignment and cannot be removed at later stages [70].

In the early versions of segment-based alignment which is implemented in a project known as *Dialign-1* [39], fragments were also known as *diagonals* since pair of segments appear as diagonals in the *dot matrix* [71]. The dot matrix is a visualization method used to compare and observe potential matches between two sequences. Figure 2.9 shows fragments (diagonals) in a dot matrix created from aligning sequence `ABABABCA` against sequence `ADBABABA`. For simplicity, a *dot* is placed where characters match, otherwise is left blank.

The segment-to-segment approach is especially suitable when sequences involved are not globally related but share only local similarities [72]. Also, It is clear from the procedure outlined ahead that gaps are not considered in the calculation of the alignment score which avoids the well-known difficulties concerning choosing appropriate gap penalty parameters in classical alignment approaches.

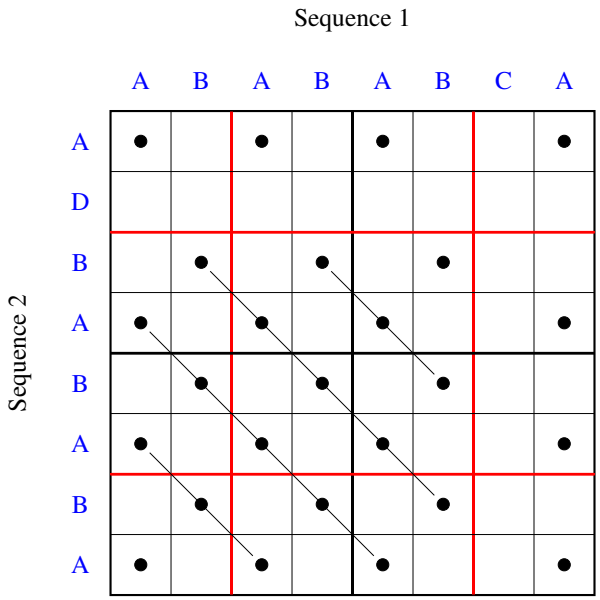


Figure 2.9 Fragments (aka diagonals) as they appear in the dot matrix.

Similar to the Dialign project [39], there is another project known as *Blast* [73, 74] (Basic Local Alignment Search Tool) that uses a similar approach to calculate the significance of fragments.

The Algorithm

In the segment-based approach [39] (shown in algorithm 1), the optimal pairwise alignment (maximum sum of weights) is determined using a modified version of conventional dynamic programming known as *fragment-chaining* [39, 75, 76] where the optimisation problem of aligning pairs of sequences is to find a chain of fragments f_1,f_2,,$_3$,. . . , f_k such that the sum of these fragments is maximal, and in both sequences, the end positions of a fragment f_i are strictly smaller than the respective beginning positions of a fragment f_j (For mathematical definition, see [75, 76]).

Because direct extension of the pairwise alignment increases the computational complexity of the algorithm exponentially, a *greedy* heuristic (based on the pairwise alignment) is used

to align multiple sequences [39, 70]. The multiple-sequence alignment steps (for a set of N sequences) can be summarised in algorithm 1 and explained as follows:

First, for each pairwise comparison, weights for all possible fragments are calculated. Based on these weights, all optimal pairwise alignments are computed, i.e., for every pair of sequences, a collection of fragments with maximum sum of weights is determined. We refer to the set of the identified fragments from this step as L_1 in algorithm 1. Then, the overlap weights for these fragments are calculated (if enabled). To this end, L_1 is sorted according to their fragment weights and their overlap weights (again if enabled). Staring with the fragment of maximum weight, fragments are incorporated one by one into another set L_2 provided they are consistent with fragments already added.

The above steps (weight computation, optimal pairwise alignment, calculation of overlap weights, sorting fragments as well as filtering inconsistent fragments) are iterated until no more fragments can be found. When the alignment involves only two sequences, the alignment requires only one iteration to complete, however, the alignment process takes maximum of three iterations when the number of sequences are more than two.

The final step in the alignment procedure, gaps are inserted into the input sequences until all positions of the selected fragments (contained in L_2) are matched. The above steps for segment-based alignment are outlined in algorithm 1.

Clearly from the algorithm, the segment-to-segment approach depends on a number of steps. However, the approaches needs to handle three major steps: 1) an efficient segment-based scoring method to reflect the similarity between pairs of segments (fragments), 2) an alignment algorithm that is able to find a set of fragments to produce the optimal (or near optimal) alignment, and 3) finally, a mechanism to check that the selected fragments are consistent. The following subsections explain more of these steps.

```
input :Sequences, N, Probs, Overlap
/* N=Number of Sequences, probs=probability estimates &
   overlap= boolean variable                                  */
output :AlignedSequences
repeat
    for all ½N(N−1) pairwise comparisons do
        W ←ComputeWeights(Seq1,Seq2,Probs)
        L1 ←ComputeOptimalPairwise(W)
        if Overlap then
            Wo←ComputeOverlapWeights(W)
        end
        Sort(L1,W,Wo)
        foreach fragment ∈L1 do
            if consistent(fragment) then
                L2 ←Accept(fragment)
            end
        end
    end
until no additional fragments found
Insert gaps into sequences until selected fragments in L2 are matched
```

Algorithm 1: Segment-based Alignment.

Weighting Fragments

In this section we give more details on how weights for fragments are computed. The scoring scheme is based on two essential aspects: i) establish a measure of similarity between characters within each pair of segments (fragment), and ii) defining a weighting function that assesses the overall significance of each fragment.

Measuring Similarity. To compute fragment scores it is first necessary to provide a matrix that defines the similarity between any given pair of characters in the set of characters being considered (for all characters expected to appear in the input sequences). In bio-informatics, similarity between single residues in sequences are represented in many different formats such as *identify scores* where score 1 is assigned for a match and 0 for a mismatch in the

matrix, and *substitution scores* where the similarity scoring between residues is based on the observed substitution or mutation of these residues in the sequences.

In segmental alignment, the scoring for a particular fragment (consisting of a pair of segments x and y) is computed first of all by summing up the similarity scores (according to the aforementioned scoring matrix) for every pair of characters. This score is denoted $f(x,y)$.

Weighting Function. Once the similarity score is computed, each fragment f is assigned a *weight* $w(f)$. The weight function for fragments is based on a probabilistic approach [70, 68]. This is computed by establishing the probability $P(f)$ of the *random occurrence* of a fragment of the same length that results in the same score. The intuition behind this is that the less likely a given collection of fragments is to occur *just by chance*, the more likely it is to be *related* so the higher its score should be (for a mathematical definition on the measure see [77, 39]).

Although many fragment properties could be considered in finding interesting fragments, only two properties have been considered: its length and cost (similarity). This in fact makes the measure for fragment similarity more generic and less dependent on any specific properties of the biological residues (e.g., protein, nucleic acid, etc.).

In segment-based alignment, sub-strings (fragments) in the diagonal path does not require gap insertion and deletion, thus the algorithm cleverly avoids dealing with determining gap costs. The key question is how one determines "interesting" fragments when we have a collection of fragments with different lengths and scores, ie., which fragment is more important to us, a fragment of length 25 and 5 mismatches or a fragment of length 50 with 18 mismatches? Segment-based alignment answers this question by calculating the *significance* of the fragment and determining which one is less likely to occur by chance.

An important advantage of segment-based alignment is that mathematical results show is that the statistical significance of pair of segments (fragment) can be estimated using an

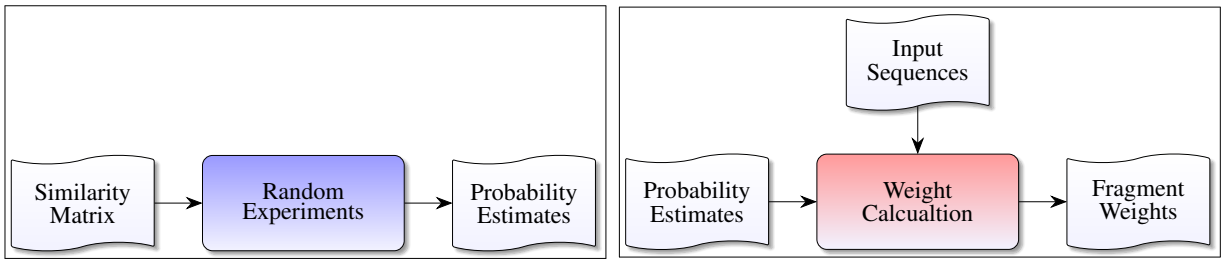


Figure 2.10 Assessing significance of fragments in segment-based alignment.

appropriate random sequence model [77, 73]. A random model means (in this context) that these sequences are *independent* and *identically distributed* sequences (*iid*) where each symbol occurs at any position within a sequence has the same probability as the others [78]. Given a random sequence model, and a set of similarity scores, it is simple to calculate the probability that two random segments of length l will have a score at least S. i.e, the probability of a hit is arising from an arbitrary pair of segments in the input sequences.

Since the introduction of the Dialign project [39], it has gone under several improvements [72, 70, 78, 69], particularly, concerning computing fragment probabilities. To overcome some of the shortcomings of the probability function adopted in *Dialign-1* [39], The probability formula was modified to take into consideration the length of the input sequences as well and introduced in a new version known as *Dialign-2* [68]. In later versions, *Dialign-T* [78] and *Dialign-TX* [69] they have included the length of the fragment as well to be taken into consideration. In this thesis, we use the refined formula adopted in later versions of *Dialign-T* and *Dialign-TX*. The probability for a fragment is calculated on two (separate) steps as follows:

- **Step A**: The probability of a fragment of a particular length obtaining a given score is established experimentally [79, 78]. Specifically, the probability $P'(s,l)$ of finding a fragment f' of length l and with a score $\geq s$ in random sequences is computed

where the length of these randomly generated sequences is twice the maximum length assigned for the fragment [78].

The calculation of the probability estimates depends on the similarity matrix which is used to generate the random sequences from its characters set as well as determining fragment scores as illustrated in 2.10 (Step A). The random experiments are carried out by starting with the trivial case for a given score s and length l=1, and then for l=2, and so forth, up to the maximum length assigned for the fragment. Generally, the probability estimates are computed using the equation [78]:

$$P'(s,l) = \begin{cases} P_1(s,l).(l+1)^2, & if \quad P_1(s,l).(l+1)^2 < P'(s,l-1) \\ P_{\exp^{(s,l)}}, & \text{otherwise} \end{cases} \tag{2.1}$$

From this step, a *probability table* is generated, whereby the probabilities for large numbers of fragments of a given length and score are computed. Since this step is computationally expensive, and the calculation of these probabilities do not depend on the actual input sequences, they are pre-calculated (off-line) and saved externally. Typically, this step is only required once per each similarity matrix.

- **Step B**: Probability estimates produced in the previous step (step A) are used to calculate the probabilities for the actual fragments as shown in Figure 2.10 (Step B). The probability P'(s,n) is used to estimate the probability $P(s,l)$ for finding a fragment f' of length l and with a score $\geq s$ in the input sequences using the equation [78]:

$$P(s,l) = \begin{cases} 1-(1-P'(s,l))^{n_1 n_2/(4l^2),} & \text{if} > P_T \\ P'(s,l).n_1.n_2/(4l^2), & \text{otherwise} \end{cases} \tag{2.2}$$

, where n_1 and n_2 are the lengths for both sequences involved in the alignment, and P_T is a probability threshold which normally fixed to a specific value (e.g., 10^{-9}).

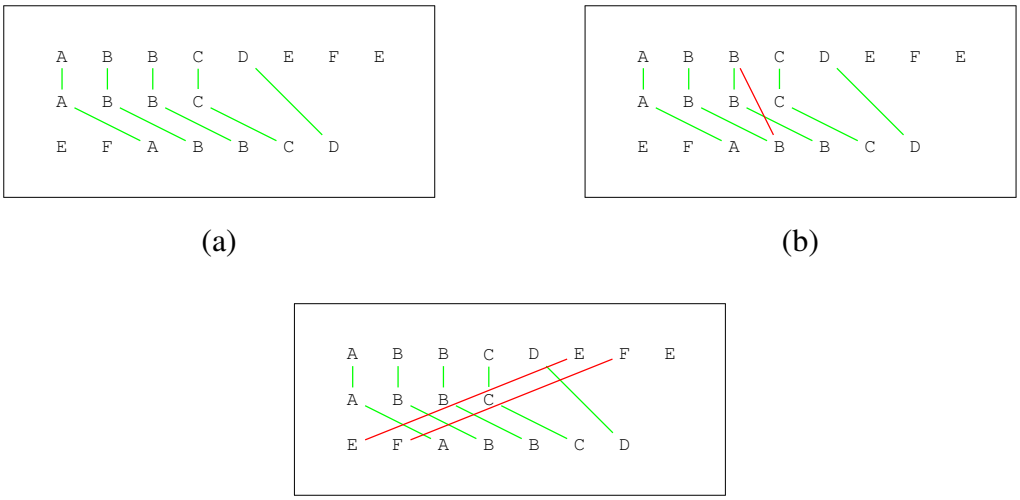


Figure 2.11 Consistent and non-consistent collection of fragments. Figure (a) shows a consistent set of fragments composed of three sequences while (b) & (c) show non-consistent fragments. In (b), the first 'B' in the third sequence is assigned to two different characters in the first sequence while (c) shows a cross-over assignments of characters between the first and the third messages.

Once $P(s,l)$ is computed for all possible fragments identified in the input sequences, the weight for a fragment f can be defined as:

$$w(f) = -log(P(s,l)) \tag{2.3}$$

Because this step is not computationally demanding, it is performed during the alignment process.

Checking Fragment Consistency.

An important step in segment-based alignment is the concept of *consistency*. As shown in algorithm 1, the algorithm needs to decide whether a fragment is consistent with the fragments already added into the alignment. Making sure that fragments participating in the

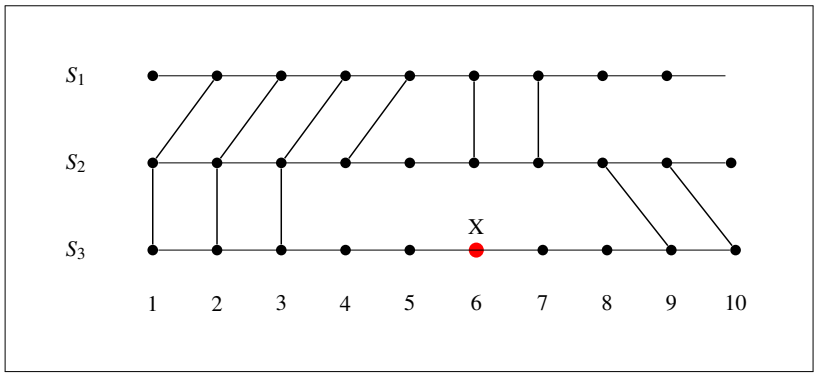


Figure 2.12 Consistency bounds for character x (sequence 3, position 6) given a set of fragments (bold lines) that are already accepted in alignment procedure. $b_1(x,1) = 5$, $b_2(x,1) = 9$, i.e., character x can be aligned with all characters between position 5 and 9 in sequence S_1. For sequence S_2, $b_1(x,2) = 4$, and $b_2(x,2) = 7$.

final alignment are consistent is part of most segment-based alignment approaches, i.e., every iterative alignment approach has to resolve the consistency problem [80].

In segment-based alignment, an alignment is defined as a *consistent equivalent relations* applied on all positions of all sequences involved. It simply means that the overall order of the positions in each sequence is maintained, i.e., a collection of fragments is called consistent if there is no conflicting double or cross-over assignment of characters (see Figure 2.11).

To determine whether a fragment is consistent with other fragments already included in the list, the so-called *consistency bounds* need to be recorded and updated. For example, for a character x and a sequence s, $b_1(x,s)$ and $b_2(x,s)$ need to be calculated, where $b_1(x,s)$ is the position of the left-most character in the sequence s that can be aligned with x without causing inconsistencies, and $b_2(x,s)$ is the position of the right-most character (see Figure 2.12 for more details.). For a complete mathematical discussion of the consistency problem, see [39, 80].

Chaining Fragments

In segment-based alignment, the pairwise alignment is a fragment-chaining procedure where the optimisation problem is to find a *chain* of fragments that yields the maximal overall score. The chain may contain fragments of different lengths, and fragments may contain mismatches.

A number of solutions have been proposed to solve the fragment chaining problem when the set of fragments are known. However, in this thesis, we will be explaining the solution followed by the Dialign project. The concept was originally introduced in [39] and later revised in [75, 76] to improve its space efficiency. One of the main objectives of this approach was to propose a solution that solves segment-to-segment alignment where gaps within segment pairs are not allowed.

The concept of fragment-chaining is based on a modification of the conventional dynamic programming followed in traditional alignment algorithms (e.g., Needleman-Wunsch). The idea is to form an optimal alignment using previous solutions for optimal alignments of small sub-sequences.

In Needleman-Wunsch, finding an optimal alignment for a pair of sequences $X = (x_1, \ldots, x_{L_1})$ and $Y = (y_1, \ldots, y_{L_2})$ is normally performed in three steps: First, a comparison matrix of size $L_1 \times \mathrm{L}_2$ is constructed where L_1 and L_2 are the length of sequence 1 and sequence 2 respectively. Both sequences need to be placed at the both edges of the matrix and perpendicular to each other. Second, for all positions (i, j) in the comparison matrix (where $1 \geq i \leq L_1$ and $1 \geq j \leq L_2$), the score $Scr[i, j]$ is recursively computed. Initially, we fill the matrix from the top left to bottom right according to the similarity scores between characters. If we know the scores of $Scr[i-1, j-1]$, $Scr[i-1, j]$ and $Scr[i, j-1]$, it is possible to calculate $Scr[i, j]$ using the equation shown in 2.4.

$$Scr[i,j] = max \begin{cases} Scr[i,j-1] - g, \\ Scr[i-1,j] - g, \\ Scr[i-1,j-1] + S(xi,yi) \end{cases} \tag{2.4}$$

where S is the similarity score between character xi and yj in sequences X and Y, and g is the gap penalty. The equation is applied repeatedly to fill in all matrix positions. As we fill in the $Scr[i,j]$ values, a pointer to each cell from which $Scr[i,j]$ is derived is stored. The value in the final cell of the matrix $Scr[L_1,L_2]$ is the best score for the alignment which is what we are after. The final step is the back-tracking procedure by building the alignment in reverse starting from the final cell in the matrix and tracing our way back (using the pointers that we stored when building the matrix) up to the starting position. (see [62, 24, 76] for more detailed discussion on how Needleman Wusnch aligns a pair of sequences).

The fragment chaining procedure used in segment-based alignment is similar to the Needlman-Wunsch method. However, the comparison is between pairs of segments rather than single characters and the scoring is based on fragments' weights. The procedure is summarised in the following steps:

1. First, for every pair of positions (i,j) in the comparison matrix, starting at position $(1,1)$ and for all possible fragment lengths, positive weights are determined.

2. The score $Scr[i,j]$ of the prefixes $x_1,\ldots,x_i$, and $y_1,\ldots,y_j$ are recursively calculated using the equation 2.5

$$Scr[i,j] = max \begin{cases} Scr[i,j-1], \\ Scr[i-1,j], \\ max \begin{cases} Scr[i-l,j-l] + w(f_{i,j,l}) & ,l \geq 1 \end{cases} \end{cases} \tag{2.5}$$

where l denotes to the length of the fragment, and $w(f_{i,j,l})$ is the weight of the fragment ending in position i and j.

3. The *last* fragment $Pr[i,j]$ in the optimal chain of the prefixes $x_1,\ldots,x_i$ and $y_1,\ldots,y_j$ is computed using the equation 2.6:

$$Pr[i,j] = \begin{cases} Pr[i,j-1] & ,if \quad Scr[i,j] = Scr[i,j-1], \\ Pr[i-1,j] & ,if \quad Scr[i,j] = Scr[i-1,j], \\ f_{ij} & ,if \quad Scr[i,j] = max\{Scr[i-1],j-1] + w(f) : f \text{ ending in}(i,j), \end{cases} \tag{2.6}$$

4. Once $Scr[i,j]$ and $Pr[i,j]$ have been calculated for all positions (i,j) in the comparison matrix, a backtracking process is carried out in order to retrieve an optimal alignment. At position (L_1,L_2), there is a pointer to the last fragment of an optimal alignment of x and y which, in turn, has a pointer to the second-last fragment in this optimal alignment etc.

Time Complexity

As described in section 2.2.4, segment-based alignment is performed in multiple steps. If the algorithm is used to align pairs of sequences, an optimal pairwise alignment can be found in $O(L^3)$ time where L is the maximum length of the two sequences that is because there are $O(L^3)$ possible fragments in the fragment comparison matrix and all of these fragments need to be considered. However, this time is reduced to $O(L^2)$ time because the length of the fragments is restricted to a specific length [70, 69].

When multiple sequences are involved ($N > 2$), all optimal pairwise alignments are computed first, that is in a $O(N^2)$ time, then for checking and filtering inconsistent fragments this step

requires $O(N^2 \times L)$ for every fragment added to the new set. If the average number of fragments in these pairwise alignments is denoted by n_a, then the first set S_1 consists of $O(N^2 \times n_a)$ fragments. The time needed for calculating the overlap weights (if enabled) is $O(N^4 \times n_a^2)$, and the gaps insertion in the final step requires $O(N^2 \times L^2)$.

The algorithm time crucially depends on the average number of fragments used in the optimal pairwise alignment, and the number of fragments considered for alignment depends on the degree of similarity between the input sequences.

The overall time complexity (worst case) for the algorithm is $O(N^4 \times L^2)$ where N is the number of sequences, and L is the maximum length of sequences [70].

2.3 Research Design

This section provides an introduction to the subjects used in the empirical study presented in Chapter 5. It begins with a general overview on the experimental design, then explains the experimental variables that needs to be identified, and how these variables are used in a statistical technique known as the *Effect Size*.

2.3.1 Experimental Design

There are two types of research design: *Experimental*, and *Observational* (aka quasi-experimental) [81, 82]. In experimental, design some degree of manipulation is involved because the intention is to exert some control over as many experimental factors as possible, on the other hand, observational research design is less invasive where the experimenter can only observe and interpret what is present in the experiment.

In this thesis, we are concerned with experimental design. Specifically, our focus is on identifying a relationship between a *cause* and its *effect* as well as whether we are able to *quantify* this effect. This type of experimental research is part of a larger subject known as

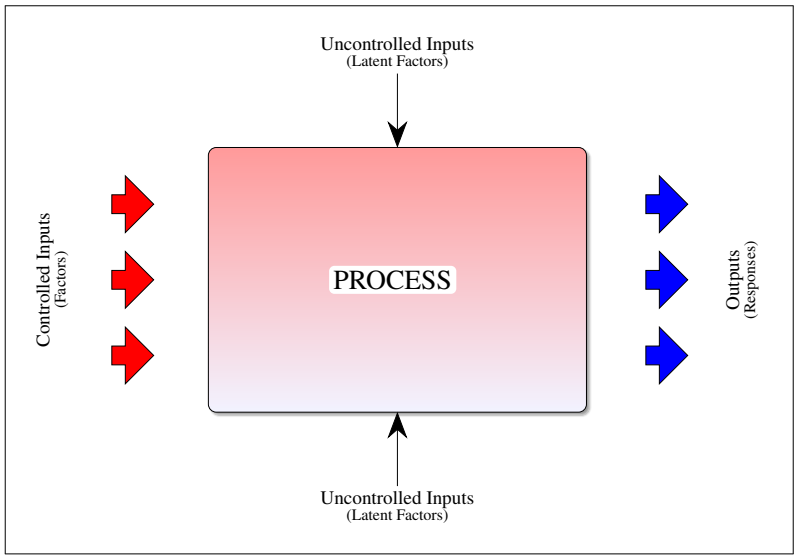


Figure 2.13 A generic cause-effect experimental model with controlled inputs, and outputs, and possibly uncontrolled (latent) inputs.

causal inference [83, 84] which is the act of using evidence to make an inference about a cause. However, in this thesis, we merely interested in investigating whether an independent variable (a cause) can have an effect on a dependent variable and measuring this effect.

Experimental Variables

A key step in experimental design is known as *Operationalisation* [82]. This is the step that links scientific concepts to the experimental data. It defines the *variables* and the *measures* which are the quantities of interest. In cause-effect experimental design, there are at least three type of variables that need to be considered. These types of variables are shown in Figure 2.13 and explained as follows:

- **Independent Variables:** For the experiment, we need to identify the variables that we would like to measure its effect. Typically, these variables can be varied at will by the experimenter (controlled inputs) and able to manipulate them in order to observe the effect and the changes that can cause on one or more *dependent variables* (explained

below). The independent variable may also be referred to as a *factor* and its possible values as *levels*.

- **Dependent Variables:** In addition to the independent variables, we need to identify a dependent variable (aka response variable) to act as an *instrument* to measure the amount of variation we apply on the independent variables. The variable should be a valid measure of the behaviour of interest and sensitive enough to the changes we apply on the independent variables.

- **Other Variables:** In some experiments, we should carefully consider the presence of other possible variables that might distort the results. These variables can be unknown (*latent*), or known to the experimenter but they are uncontrollable. These variables sometimes are known as *nuisance variables*. Typically, such variables are "irrelevant" to our interest, however they can be associated with other independent variables and *confound* the results by causing unaccounted variability on the response variable. For example, consider the process of measuring the impact of using different types of fuel on the speed of a car. In this example, the process consists of two variables, one independent variable (X) which is the *fuel* type, and a response variable (Y) which is the *speed* of the car as shown in Figure 2.14. When the type of fuel (X) is not associated with other parts in the car (i.e., normally not used with it), then the dependent variable (Y) should give us correct readings reflecting the choice of the fuel. However, when the type of fuel is regularly linked to the usage of another component such as the *fuel filter* (Z), then we cannot conclude that the speed readings are affected by the fuel type alone without considering the effect of the filter as well. The filter in this example is considered a *confounding variable* as illustrated in Figure 2.14.

  When a process variable is situated between the independent variable and the dependent variable (i.e, in the investigated path), it is known as an *intermediate variable*. Typically, an intermediate variable causes variation on the dependent variable and it itself caused

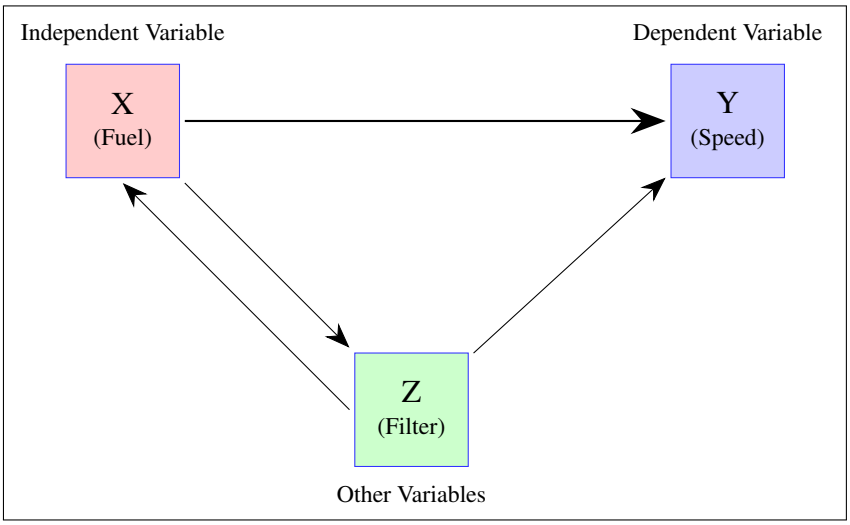


Figure 2.14 The cause-effect relationship between an independent variable X (e.g., fuel type), and a dependent variable Y (e.g., car speed), and the possible existence of a third confounding variable Z (e.g., fuel filter) associated with the independent variable.

to vary by the independent variable. However, not all intermediate variables are considered confounding variables.

In addition to identifying experimental variables, data samples (experimental subjects) need to be gathered for the experiment. The collected data is used to derive the empirical results linking outputs observed dependent variables with inputs caused by the independent variables. Assuming the investigated question can be answered by statistical analysis techniques, the second part of the experiment is to decide which statistical technique that can be applied to measure the effect.

2.3.2 Effect Size

In experimental design, it is often that we require a technique or a method to measure the impact of an independent variable may have on a response variable. The statistical test that is usually applied is known as *effect size* [85].

Effect size is a statistical test used to quantify the difference between two groups of data. Some researcher have used effect size within the context of *power analysis* to determine the needed sample size for a research [86, 81, 82] while others used it as part of *Meta-analysis* [87] where the aim is to quantitatively summarise the numerical results of several research studies on a specific topic to determine whether the finding holds generally. Also, effect size is extensively used in social science and medical experiments to determine the effect of a new drug or treatment on different subjects. In addition to the above applications, effect size has been used to evaluate the effect of different factors on the *adequacy score* [3] in software testing [88].

Measures of Effect Size

There are a number of statistical measures for estimating effect size such as, *Cohen's d* (aka *Standardised Mean Difference*), the *Correlation Coefficient*, *Odd's Ratio* (OR), and *Risk Ratio* (RR) [85]. The choice of the measure largely depends on the type of data being analysed (e.g., continues, nominal, categorical etc.). In this thesis, we will be referring to Cohen's d [4] , further details on other measures of effect size can be found in [82].

Cohen's d. The basic use of Cohen's d is to measure the mean difference (standardised) between two groups of data, i.e., given a pair of groups X_1 & X_2, Cohen's d gives the standardised difference between the mean of the two groups using the following equation:

$$d = \frac{\overline{X}_1 - \overline{X}_2}{S_p} \tag{2.7}$$

Where $\overline{X}_1$, and $\overline{X}_2$ are the sample means in the two groups, and the S_p is the within-groups standard deviation pooled across the groups.

[3] A score that can be used to measure the effectiveness of a test set and its ability to discover faults.
[4] The estimate of d is the statistic denoted by unbiased standardised mean difference or Hedge's g.

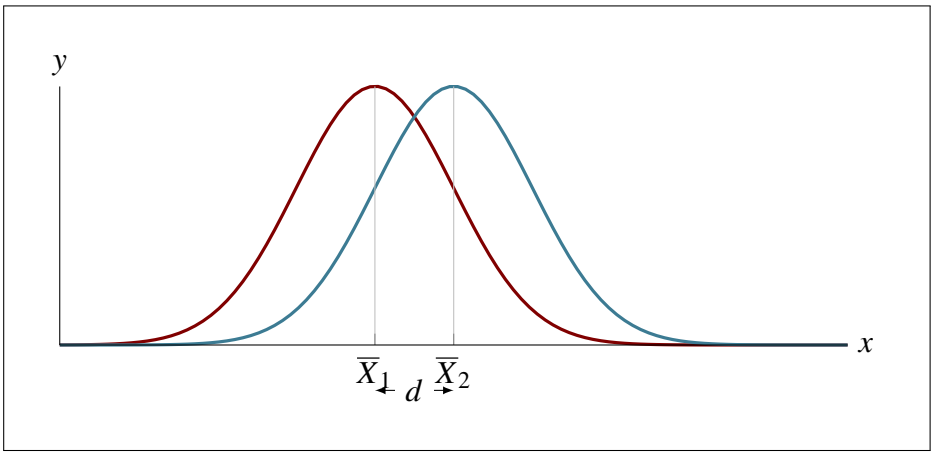


Figure 2.15 Mean difference as a measure for effect size.

$$S_p = \sqrt{\frac{(n_1-1)S_1^2 + (n_2-1)S_2^2}{n_1+n_2-2}} \tag{2.8}$$

Where n_1 and n_2 are the sample sizes in the two groups, and S_1 and S_2 are the standard deviations in the two groups. The idea is that the bigger the value of d, the larger the contrast between the two groups. When d is negative this indicates that the probability of all of the scores of X_1 group are smaller than X_2, and when d is positive this indicates the probability that all of the scores of X_1 group are greater than X_2. When d equals to zero this indicates the distribution of the two groups are overlapping. The basic concept of is illustrated in Figure 2.15.

For interpreting the magnitude of the effect, Cohen nominated *0.2*, *0.5* and *0.8* as the *small*, *medium*, and *large* reference values, respectively [86, 89]. However, he urged that these values should be referenced carefully and interpreted within the context in which these experiments are applied.

The Confidence Interval. When we use the sample mean as an estimate (magnitude) of the effect size, then we should also recognise the error associated with the sample mean.

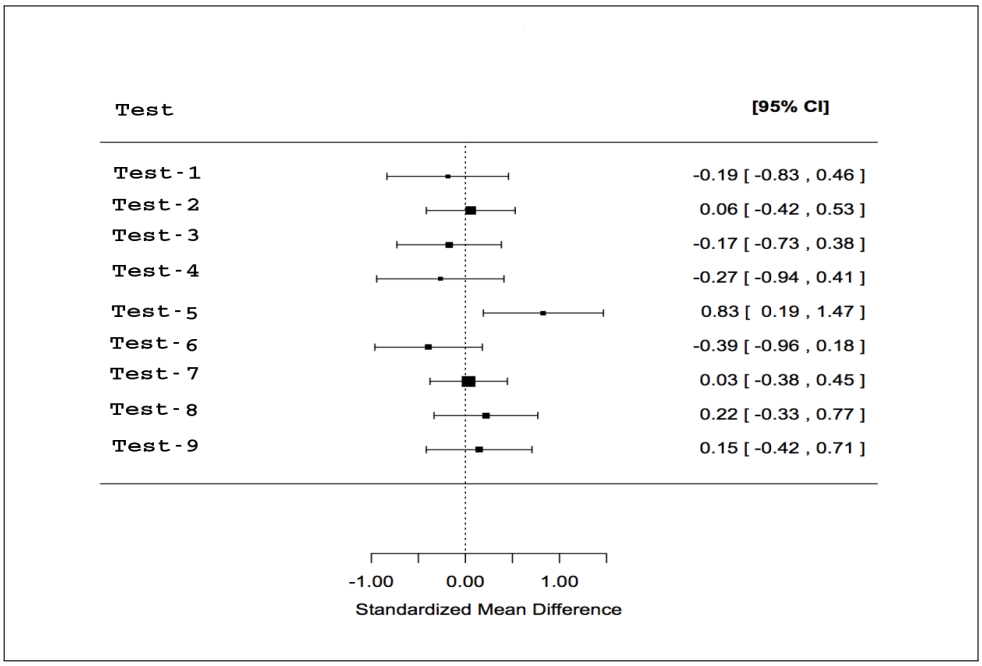


Figure 2.16 A Forest Plot shows 9 fictitious tests and their effect size mean estimates. It also shows the confidence intervals for each test as well as the level of confidence under which these tests are carried out.

This is done by calculating a "margin of error" and reporting it with the test estimate which known as the *Confidence Interval*. A confidence interval is used to indicate the range within the estimate of a test may fall and typically associated with a *confidence level* indicating the significance of the test. For example, when tests are conducted with 95% confidence level, this means if a test was repeated an infinite number of times, each time drawing different samples from the data and constructing a confidence interval based on these samples, 95% of the time the confidence interval would contain the estimated figure. A magnitude of an effect and a confidence interval can provide useful and complementary information about the result of the test.

Visualising the Effect

The result of the effect size is normally illustrated in a graphical representation called a *forest plot*. Typically, a forest plot consists of two columns. The left-hand column lists the names of the tests, and the right-hand column is the result of the test (effect size). Each of these test estimates is represented by a square incorporating confidence intervals represented by a horizontal line. The sold vertical line is the *no-effect* line. If the confidence interval for the test overlap with this line, it demonstrates that the effect size of that test do not differ from no effect at that confidence level. Figure 2.16 shows a dummy example of 9 tests using the standardised mean difference and 95% CI. The example shows that the results of Test 5 is more significant than others, however, the point estimate carry some uncertainty due to the wide confidence interval.

2.4 Summary

This chapter has discussed the main aspects of network protocols and data mining techniques used in protocol reverse engineering. First, the chapter has explained the "standard" models for protocol communications with a special emphasise on the client/server model and TCP/IP protocol suite as well as a description of the structure of a protocol message, then it has provided a general background on clustering and sequence alignment algorithms focusing on the hierarchical clustering approach, and the segment-based alignment. Finally, this chapter concluded with a background on aspects of experimental design and effect size that is linked to the experiment carried out in Chapter 5. This should set the necessary background for the next chapter and the rest of the thesis.

Chapter 3

Related Literature & Motivations

The purpose of this chapter is twofold. First, it familiarises the reader with the state-of-art protocol reverse engineering approaches focusing on inferring protocol message structure from network traffic. Second, it elaborates on some of the limitations exist in previous state-of-the-art techniques as well as the motivations behind this work.

3.1 Protocol Reverse Engineering

Protocol reverse engineering (or protocol reversing) is the process of extracting protocol specifications from protocol artefacts. Protocol reverse engineering can reveal a significant amount of information on the protocol design, implementation, and functionalities. Network protocol specifications are the backbone of several security applications, such as *Smart fuzz testing* [24, 103, 5] to generate test cases that uncover potential vulnerabilities, *protocol fingerprinting* [8] to identify specific protocol features on a remote computer, *intrusion detection systems* (IDS) [4] that perform deep packet inspections, *network traffic classification* [9, 10], detecting *redundant network traffic* [104], *protocol replay* [31, 33, 105, 32] to emulate protocol services for Malware detection, *generic protocol analysers* [12, 106] that require

	Project	Domian	Tested Porotols	Anlysis Technique	Other Artefacts	Inferred Specification
01	Polyglot [90]	Generic	Text & Binary	Executable	Network Data	Message Format
02	Rosetta [91]	Protocol Replay	Text & Binary	Executable	Network Data	Message Format
03	Dispatcher [21]	Botnet Infiltration	Text & Binary	Executable	Network Data	Message Format
04	Prospex [27]	Generic/Fuzzing	Text & Binary	Executable	Network Data	State Machine
05	Tupni [92]	Generic	Text & Binary	Executable	Network Data	Message Format
06	ReFormat [93]	Encrypted Protocols	Text & Binary	Executable	Network Data	Message Format
07	Wondracek *et al.*,2008 ([34])	Generic	Text & Binary	Executable	Network Data	Message Format
08	AutoFormat [94]	Generic	Text & Binary	Executable	Network Data	Message Format
09	He *et al.*,2009 ([95])	Generic	Text & Binary	Executable	Network Data	Message Format
10	ScriptGen-1 [31]	Protocol Replay	Text & Binary	Network Data	None	Complete
11	ScriptGen-2 [33]	Protocol Replay (Real-time)	Text & Binary	Network Data	None	Complete
12	Discoverer [26]	Generic	Text & Binary	Network Data	None	Message Format
13	PEXT [25]	Generic	Text only	Network Data	None	State Machine
14	PRODECODER [23]	Generic	Text & Binary	Network Data	None	Message Format
15	Antunes *et al.*,2011 ([96])	Complementing Specification	Text Protocols	Network Data	None	State Machine
16	ReverX [97]	Generic	Text only	Network Data	None	State Machine
17	PRISMA [37]	Generic	Text & Binary	Network Data	None	State Machine
18	Veritas [36]	Generic	Text & Binary	Network Data	None	State Machine
19	Kannan *et al.*,2006 ([98])	Anomaly Detection	Text & Binary	Network Data	None	State Machine
20	Bipriminer [99]	Generic	Binary only	Network Data	None	State Machine
21	PI [24]	Generic	Text & Binary	Network Data	None	Message Format
22	RolePlayer [32]	Protocol Replay	Text & Binary	Network Data	None	Complete
23	Trifilo et al.,2009 ([100])	Generic	Binary only	Network Data	None	State Machine
24	ASAP [101]	Anomaly Detection	Text only	Network Data	None	Message Format
25	Netzob [102]	Generic	Text & Binary	Network Data	Executable	Message Foramt

Table 3.1 A selected list of 25 state-of-the-art projects. It shows the adopted inference scenarios, application domain as well as the inferred specification.

protocol specification as an input to dissect messages, and for detecting *deviations from specifications* among different implementations of the same protocol.

The goal of protocol reverse engineering is commonly to extract a complete description of protocol specification. This mainly entails inferring the *message format*, which captures the structure of all types of messages, along with the rules that govern the order in which these messages can be sent and received (often modelled as a *state machine*) as illustrated in Figure 3.1.

There are mainly two common approaches for inferring protocol specifications [107, 108]. The first approach is by *reverse engineering* protocol executables (e.g., server) using *Dynamic Binary Analysis* techniques (DBA) [109–113, 108]. Specifically, the DBA engine is used to monitor and record the execution of the protocol application while processing received or sent messages, and then various intuitions are applied to extract protocol specifications from

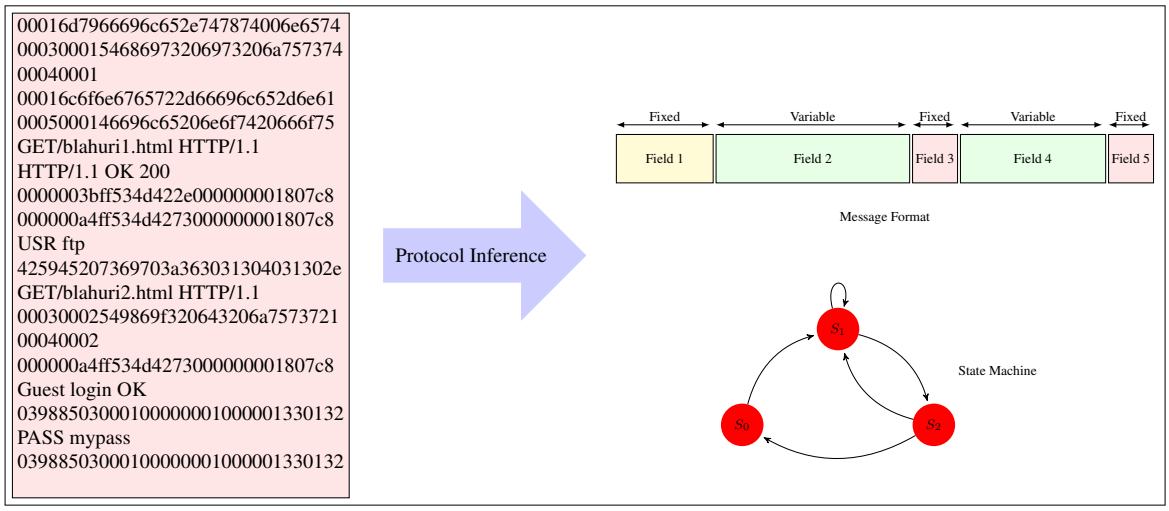


Figure 3.1 The inference of protocol specifications (the message format & state machine).

the execution trace. Detailed information on how the approach is implemented in previous systems can be obtained from various sources [90, 94, 92, 34, 27, 21, 93, 114, 99, 22]. DBA assumes the availability of protocol executables which is not the case in some situations. Also, the dynamic analysis of protocol binary tends to be platform specific (coupled to specific hardware) and generally entails huge execution traces generated even from relatively small programs. Furthermore, these execution traces may not generalise to future program executions especially when the test cases (set of inputs) do not trigger all possible program executions [115].

The second approach used to infer protocol specifications is based on *analysing captured network data* [24, 26, 25, 97, 36, 37, 23, 108, 116] where is the analysis is focused on the captured protocol traces. Typically, the approach has its own advantages in terms of no protocol executables are required for the analysis as well as completely independent from specific platforms.

There is a hybrid approach that combines both scenarios where the analysis is based on the captured network data as the main protocol artefact, and the protocol executable is used to support the inference process by monitoring and extracting contextual information that relates

to the execution of the program to enhance message clustering and alignment as followed in [102]. Table 3.1 shows detailed information of some of the previous state-of-the-art projects and the inference scenarios they followed in reverse engineering network protocols.

In addition to the machine-learning approaches discussed above, there is another common approach that is based on inferring a finite-state automata of the protocol [98, 97, 96, 27]. In this approach, the inferred protocol specification is modelled as a finite-state machine (FSM) that captures both the language (i.e., the formats of the messages) and the state machine (i.e., the relation between the different types of messages). To infer the protocol language, the approach constructs the state machine from the captured traces, then generalises the FSM so that it can accept similar message types with different payloads. However, to construct the FSM, this approach needs to decompose each message to a sequence of fields and delimiters using pre-defined (known) field delimiters (e.g., space, tab etc.) which are normally not available in case of reverse engineering undocumented protocols. Furthermore, the approach is quite difficult to apply on binary-based protocols where the notion of a delimiter is not used and instead they resort to fixed-length fields in their implementations [97]. Also, in many cases, the inferred FSM requires generalisation and minimisation which involves several steps as well especially in complex protocols which contain several types of messages composed of different fields - (e.g., SMB [13]), it is often difficult to infer a concise automata that accurately describes the protocol language [97, 27].

In this thesis we follow a machine-learning approach to discover the protocol message formats from network traces and does not consider other approaches any further.

3.1.1 Protocol Reversing form Network Traffic

Reverse engineering network protocols from captured traffic mainly relies on analysing the captured network traffic generated by the applications (client/server) of the target protocol. Most previous approaches have adopted (more or less) a common sequence of steps. The

process of knowledge discovery (illustrated in Figure 2.4) has fed into a significant amount of research on inferring protocol specifications form network traffic.

In this section, using the process of data mining and knowledge discovery as a guide, we provide an overview of the steps and how sequence clustering and sequence alignment algorithms are integrated into the process to infer message structures of network protocols from captured network data. Figure 3.2 illustrates the common sequence of steps that tend to be adopted. Generally, the approach consists of the following steps:

1. **Traffic Collection:** Collecting network traffic is the process of capturing data as it travels across a network. For traffic collection, there are plenty of powerful off-the-shelf tools that could be used to collect and record network data such as *TCPDump* [117] and Wireshark [118] network analysers.

2. **Message Pre-processing:** Normally the collected traffic consists of different protocols. Therefore, certain techniques have to be used to extract only traffic that belongs to a specific protocol. In this step, various techniques are applied to pre-process collected traffic; this includes filtering out what is known as *background noise* [36, 102] and selecting a healthy sample of the protocol of interest. There are many methods that can be used to classify and filter network protocols from the rest of the traffic [10, 119].

   Since we intend to cluster our data, it is necessary to re-code the data in such a way that a clustering algorithm is going to be able to identify similarities and differences between messages. As we discussed in Chapter 2, Section 2.2.2, most clustering algorithms require some sort of similarity (dissimilarity) measure to be established between data items.

   This similarity measure is often based on a common feature (or features) that could be identified among data items. It is common that some form of tokenisation is carried out on application messages to produce features which are subsequently used by the

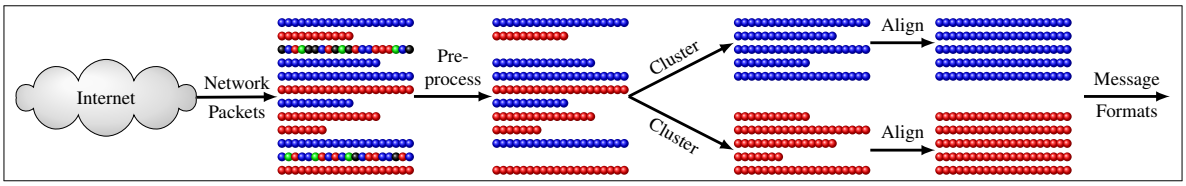


Figure 3.2 A common approach of inferring protocol message structures from network Traffic.

clustering algorithm to group similar messages into groups. Some of the previous systems have tokenised messages based on message data types (binary/ASCII tokens) [26], some have used *n*-grams [23, 36] (discussed in more detail in Chapter 4), and some have used external information as features (outside the trace) such as, information extracted from intercepted actions triggered by running protocol application [102, 27]. Normally, unrelated features are filtered out using the data dimensionality techniques as explained in section 2.2.1. The tokenisation step is important because clustering accuracy and time can be improved by considering only related features. The *Protocol Informatics* project (PI) does not use any tokenisation in its inference [24]. The distance between protocol messages is quantified using edit-distance sequence comparisons (local alignment, entropy etc.). However this approach can have huge impact on clustering results and the time required for clustering especially with large data sets and long protocol messages.

3. **Message Clustering:** Protocols involve multiple different types of messages, where each type has its own format. The *clustering* step serves to identify and group messages of the same type together, so that they can be subjected to a more further analysis to extract their shared format as previously discussed in Section 2.2.2. Normally, partitional or hierarchical clustering algorithms are applied at this step.

4. **Message Alignment:** Sequence alignment algorithms take as input the clustered protocol messages and align them, exposing the structural aspects of field similarities,

(a) Non-alinged messages	(b) Aligned Messages
GET / HTTP/1.0\r\n\r\n	GET /--------- HTTP/1.0\r\n\r\n
GET /indx.html HTTP/1.0\r\n\r\n	GET /indx.html HTTP/1.0\r\n\r\n
GET /k.ico HTTP/1.0\r\n\r\n	GET /---k.ic-o HTTP/1.0\r\n\r\n
GET /img.png HTTP/1.0\r\n\r\n	GET /i-mg.-png HTTP/1.0\r\n\r\n
GET /st.css HTTP/1.0\r\n\r\n	GET /--st.-css HTTP/1.0\r\n\r\n
	***** **********
	GET /--------- HTTP/1.0\r\n\r\n

Table 3.2 An alignment of a set of multiple HTTP messages using the Needleman-Wunsch algorithm and the Progressive Alignment heuristic.

differences, and gaps (as explained in Chapter 2, Section 2.2.3). Previous protocol inference techniques have sought to adopt this intuition to infer the message structure. The rationale is that messages that are related to each other have certain similarities; they share keywords and use the same symbols to act as delimiters between different parts of message fields. To illustrate this intuition, we can consider the set of packets in Table 3.2 (a), containing simple request messages from the HTTP protocol using the *GET* method. An alignment of the messages is shown in Table 3.2 (b). Similar characters (bytes) are aligned to each other. Because these messages have different lengths, gaps are inserted into each message in order to align it with other messages. As a result, the alignment clearly outlines the two keywords (GET & HTTP/1.1) and the variable field in the middle (in this case the URI). The lower-most sequence at the bottom is the consensus sequence, which summarises the alignment result.

The alignment shown in Table 3.2 is produced by the Needleman-Wunsch algorithm [38] (which is designed to align pairs of sequences) and the *progressive alignment heuristic*.

3.2 Motivations

This section discusses the limitations that we have observed in some of the state-of-the-art techniques. First, we discuss the effect of different parameters on message clustering as well as finding the best possible combination of these parameters for clustering. Second, we outline some of the drawbacks linked to traditional sequence alignment algorithms (i.e,. Needleman-Wunsch) that have been used by most state-of-the-art projects. Accordingly, we elaborate on the motivations behind our work as follows:

3.2.1 Investigating Clustering Factors

The choice of clustering configurations for protocol inference is commonly made on ad-hoc basis. By nature, the process of knowledge discovery depends on several decisions (e.g., choice of the data mining algorithms) which in turn depend on other parameters [2]. The choice of these is critical to the quality of the inferred set of clusters.

The clustering performance degrades significantly unless parameters are properly set, and yet it is difficult to set these parameters a priori. Previous projects realised that clustering is critical to the quality of the inferred specifications. For example, to enhance clustering (and other steps), some approaches relied on the contextual information (semantics) to be used as features for clustering [102]. However, these contextual information entails either the use of other layers in the protocol stack (i.e., inter-protocol dependencies) or the presence of the protocol executable to intercept its communications with the operating system (e.g., intercepting system calls while sending and receiving messages).

Clustering is arguably one of the most challenging problems in machine learning due to the lack of universal and rigorous mathematical definition [56, 30]. It is often considered an exploratory device because the actual classes of the protocol messages are unknown. Furthermore, clustering accuracy can be affected by a range of factors [29]. For example,

previous protocol inference approaches have arbitrarily chose to consider just the first *12* or *32* bytes of the message [36, 23], whilst other approaches decided to consider longer messages [26]. Other factors such as the size of the protocol sample is also a major decision that needs to be taken into account prior to inference.

Normally there are advantages and disadvantages associated with such decisions. For example, short messages normally lead to faster and better clustering quality since shorter messages would include less volatile data (payload) and less noisy features. However for many protocols (e.g., HTTP), a message header can span for hundreds (or even thousands) of bytes, and truncating messages to such drastic lengths often lead to incomplete inference of the message structure. Similarly, a protocol sample needs to be large and representative, that is to capture as many message types as possible. However, large sample sizes normally lead to complexities affecting clustering quality and time as well. Therefore, making informed decisions regarding such factors are significantly important to the quality and completeness of the inferred model.

The common process to protocol inference discussed in the previous section consists of multiple interdependent steps. Crucially, the choices that are made with respect to choosing the parameters for each of these steps can have a significant impact on the inference results. The ideal choices may depend to an extent upon the characteristics of the network data (the amount of data available or the nature of the data).

Choosing a suitable clustering configuration is ultimately a complex process. However, there is a dearth of guidance that can indicate how to choose different settings. Most protocol inference approaches are evaluated with respect to a static configuration. Thus, our first motivation is to provide an experimental framework, along with empirical data that can be used to determine the effect of certain factors on clustering and as a guide to the choice of suitable clustering configurations for packet extraction.

Sets	HTTP GET Messages
Similar	`GET /myimage.jpg HTTP/1.1\r\nHost: www.bbc.com\r\nUser-Agent: Dillo/3.4\r\n\r\n` `GET /thisisabitlongeruri/documents/index.html HTTP/1.1\r\nHost: www.le.ac.uk\r\n\r\n`
Less Similar	`GET /myimage.jpg HTTP/1.1\r\nHost: www.bbc.com\r\nUser-Agent: Dillo/3.4\r\n\r\n` `GET /thisisabitlongeruricontainsmanycharacters/documents/index.html HTTP/1.1\r\n\r\n`

Table 3.3 Two slightly different sets of HTTP GET messages.

3.2.2 Improving Message Alignment

Alignment algorithms used by current approaches often fail to identify suitable structures. Most protocol reverse engineering techniques are based - one way or another - on the Needleman-Wunsch algorithm (and the Progressive Alignment approach). As discussed in Chapter 2, this can work when sequences tend to share strong global similarities (from start to end). However, this is not necessarily the case with network messages where messages can share only a few patches of locally similar regions. Protocol messages tend to contain long variables and optional fields, as is the case in HTTP protocol for example. Additionally, with Needleman-Wunsch the quality of a sequence alignment critically depends on the judicious selection of user-defined parameters (e.g. a gap penalty) which can be very difficult to choose [32, 120] and can vary from one protocol to another. As a consequence, if the messages are heterogeneous, and the parameters fail to count for the specific characteristics of a message set or protocol, alignments can easily become highly inaccurate. Also, the quality and the performance of Needleman-wunsch algorithm can be very poor when protocol messages are long [39]. Several approaches in the past have observed such limitations and carried out a number of modifications to the Needlman-Wunsch algorithm in an effort to enhance the alignment step [32, 120, 26, 102].

Let us consider the pairs of messages shown in Table 3.3. Here we show two pairs of HTTP messages of the same type, each message consists of a request-line message and optional headers. One set consists of messages that share strong similarity (the stream of bytes is similar from start to the end). The other set contains request-line messages of the same

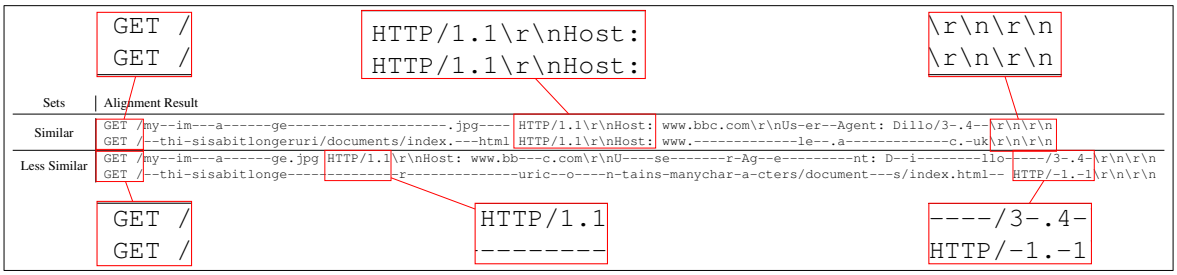


Table 3.4 Alignments of the HTTP messages by Needleman-Wunsch with the standard parameters (Match=1, Mismatch=0, Gap=0).

protocol that are less similar because the URI fields are of different lengths and contents, and omit some of the message headers.

We illustrate the aforementioned problems by aligning these messages with the Protocol Informatics tool [24], which is based upon Needleman-Wunsch. First it is necessary to select the Needleman-Wunsch parameters (scores for when a pair of aligned characters match and mismatch, and a penalty for any gaps that are introduced). We consider the default settings (match=1, mismatch=0, gaps=0) as used in the Protocol Informatics for protocol analysis.

The results are shown in Table 3.4. The alignment results show that when messages share significant similarity, the choice of the standard user-parameters can provide good alignment results. The message elements that we would expect to be aligned – '`GET /`', '`HTTP/1.1\r\nHost: www`' and the '`\r\n\r\n`' at the end of the message are all correctly aligned.

However, when the messages are dissimilar, the same algorithm with the same parameters fails to produce a suitable alignment. Although the first '`GET /`' and the final '`\r\n\r\n`' are aligned correctly, it fails to match the '`HTTP/1.1`' header. Whereas this example is necessarily small for the purpose of illustration, it is apparent the problems illustrated here can easily be exacerbated as the number of messages, their lengths, and heterogeneity increase.

In addition to the above limitations, because the quality of the alignment depends on clustering, previous approaches that use the Needleman-Wunsch algorithm are often required to cluster (split) messages of the same type that are not globally similar (composed of different optional fields) into different clusters to produce homogeneous clusters (messages composed of the same fields) [32, 120, 26, 102]. This type of clustering often leads to a problem known as *over-clustering* [30, 31]. Over-clustering is a common problem and often entails an extra step in the process, which is to *merge* the extracted fine-grained formats into the original format.

3.3 Summary

This chapter has presented state-of-the-art techniques used to reverse engineer network protocols. It also has discussed the common steps used to infer protocol specifications from network traces using clustering and sequence alignment algorithms. This chapter concluded with a discussion on the limitations of previous protocol reverse engineering approaches and the motivations to this work.

Chapter 4

A Framework For Experimentation

This chapter presents the design and implementation of a network-based protocol reverse engineering framework. The framework will be used to empirically address the limitations discussed at the end of the previous chapter and also to be used as a conceptual basis within which various techniques will be investigated. The chapter begins with an overview of the key desired properties for the framework followed by a detailed description of the design and implementation of the framework. The limitations of the framework are also discussed at the end of this chapter.

4.1 Properties

Although there is enough documentation in the literature regarding protocol inference from network traffic [26, 23, 36, 31], unfortunately there are few implementations available that we could employ out of the box without requiring significant modifications. That is because these systems follow specific inference scenarios (e.g., apply specific protocol inference such as the state machine [36, 37], executable-assisted inference [102] etc.). Additionally, several previous state-of-the-art projects do not include the key properties (discussed below) which we require in the framework.

In this thesis, we would like to construct a framework that supports specific properties. The key properties that we require in the framework are as follows:

- **Separation of Concerns (Modularity):** In order to experiment with different technologies, the framework needs to be modular where each module is assigned specific task so that each module can be easily replaced without significant changes to the module or its interface.
- **Configurability:** One of our primary objectives is to conduct multiple experiments on different network protocols and choose different settings of configurations mainly related to the sample and the inference process. Therefore, the framework should be configurable (when required) and represent a faithful implementation to the adopted approach.
- **Reproducability:** Since all required information is contained within the captured traces, data can to be analysed off-line (without live capturing mechanism). Off-line analysis has the benefit of allowing us to re-conduct experiments many times over the captured sample without new non-deterministic behaviour being injected into the sample. Also, off-line analysis allows the same data samples and results to be shared with other researchers for further evaluation.

4.2 Design

Our trace-centric approach to protocol reverse engineering requires the ability to perform data analysis on the captured traces. For this we investigate our problem from a data mining perspective. Accordingly, the framework for protocol reverse engineering will be based on a similar approach to the generic model explained in section 3.1.1, that is to address the

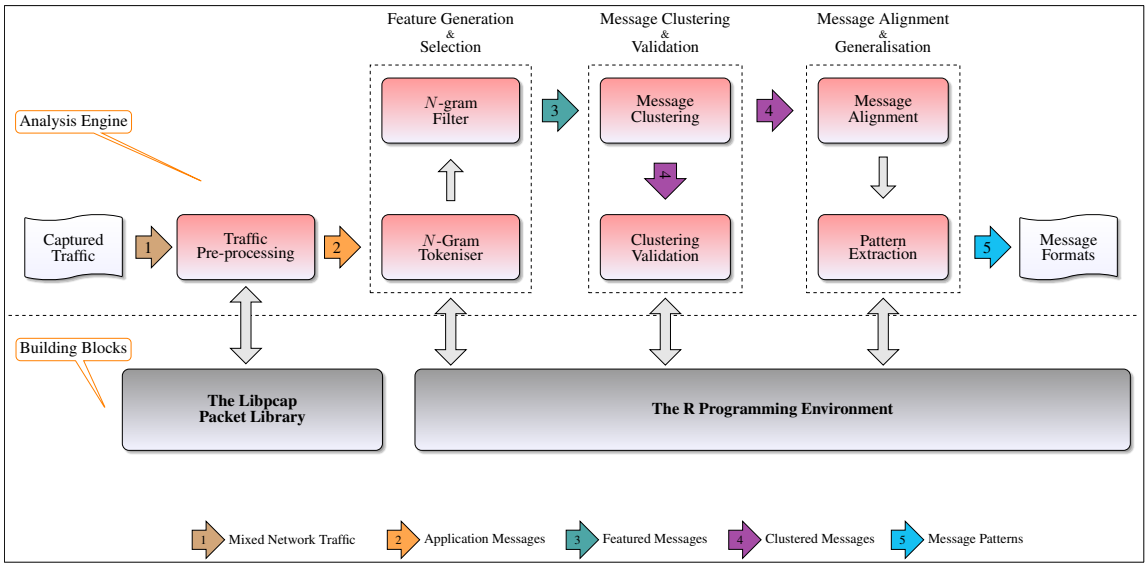


Figure 4.1 Architecture of application-level protocol reverse engineering framework. Modules in grey are the building blocks and were previously available. Modules above the building blocks are the analysis engine of the framework.

many limitations inherited in the process, and enhance the inference within the context of network-based scenarios.

Generally, our approach is based on four major steps: *Traffic Pre-Processing*, *Feature Generation & Selection*, *Message Clustering and Validation*, and *Message Alignment and Generalisation* step. These steps form the basis of the *Analysis Engine* as shown in the framework architecture (Figure 4.1) and will be explained in detail ahead.

As illustrated in Figure 4.1, The overall architecture shows the general structure of the framework and how the different steps are assembled to perform the inference. The architecture can be viewed as two levels into which different modules are fit. The bottom level of the architecture hosts what we call the *Building Blocks* (in grey) and they serve as a platform to the framework, and the upper level contains the *Analysis Engine* which is composed of several modules that follow the analysis approach.

4.2.1 Building Blocks

The framework is built on top of two openly available software systems: the *LibPcap* packet library [117], and the *R* programming environment [121].

LibPcap Packet Library: Filtering network traffic is a complex process. It often depends on multiple elements involving: the network interface, its driver, and the operating system's kernel. Typically, when we use a sniffer (network data capturing program), the process of capturing and filtering network traffic is performed by the operating system's kernel while the sniffing application runs as a user process. To filter the incoming traffic, the sniffer requests from the operating system's kernel to filter captured traffic so we just get a copy of the packets that match our filter. Every operating system implements its own packet filtering mechanism. However, most of kernel filters are based on the *BSD Packet Filter* (BPF) architecture [122]. BPF is a kernel agent that filters (unwanted) network packets mainly implemented to minimise the context switch occurring between the OS kernel and the capturing applications running in the user-space.

To avoid the complexity of filtering packets (briefly described above) and avoid direct dependencies on the operating system, we use the *LibPcap* packet library [117] (aka WinPcap on Microsoft Windows). LibPcap is an open-source library that provides a high-level and operating systems independent Application Programming Interface (API) to network capturing systems. In addition to its capturing capability, one of the most powerful features offered by LibPcap is its filtering mechanism. It provides a complete support for the BPF based packet filters for most Unix clones of operating systems (e.g., Linux, BSD etc.) as well as Microsoft Windows.

LibPcap filters network traffic based on three basic steps: construct a filter expression, compile the expression into a BPF program [1], and finally set the filter to the OS kernel. The

[1] A low-level code similar to assembly that can be interpreted by the operating system.

filter expression is a high-level language that allows to write and combine different filters according to a well-defined specification [117].

The R Programming Environment: The purpose of this module is to provide a data mining environment that offers efficient implementations for a variety of data mining techniques and algorithms [121].

R is a programming language and a data mining environment. It is available as free software under the terms of the Free Software Foundation (GNU General Public License). *R* is often used for data analysis by various scientific disciplines. It supports a wide variety of software packages geared towards different purposes in data mining, such as linear and non-linear modelling, statistical tests, classification, clustering, string operations etc. Additionally, *R* is equipped with a powerful graphical and plotting capabilities for data analysis and interpretation.

In addition to the data mining algorithms, *R* is a well-developed programming language that supports a wide range of the traditional programming capabilities (e.g., loops, conditions, functions etc.) as well as it is exceptional capabilities in efficiently storing and manipulating data sets through its support to a wide operations on arrays (vectors), matrices, and data frames. It is designed around a true computer language, and allows users to add additional functionalities by defining new functions. For computationally-intensive tasks, *C*, *C++* and Fortran code can be integrated and called at run time.

4.2.2 Analysis Engine

This section describes the analysis engine illustrated in Figure 4.1 and the techniques used in each step. It starts by describing how the network traffic is collected, classified and pre-processed in preparation for the inference process, then describes how discriminative features

are generated and selected from protocol messages prior to the clustering, and alignment steps.

Traffic Collection

There are many generic tools that can be used for capturing network traffic such as *Wireshark* and *Tcpdump* [123]. However, for collecting network traffic, we intend to use pre-captured protocol samples from on-line sources [124]. The captured traffic should be in the *pcap* file format [125] to be able to pre-process it by the Libpcap library (explained next). This file format is considered the "standard" file format for saving captured traffic and commonly supported by most traffic capturing tools.

Traffic Pre-Processing

Normally, the captured traffic belongs to different protocols where each protocol message is encapsulated with other protocol layers in the TCP/IP hierarchy (as discussed in Chapter 2 , section 2.1.1). Furthermore, it is often that these protocols are encoded with different encodings (e.g., ASCII, Binary). Therefore, it is important that we extract only data belongs to the application protocol of interest and encode the extracted traffic into a common format that enables us to apply our analysis approach on different types of protocols. The overall pre-processing steps are summarised in Figure 4.2 and explained as follows:

- **Traffic Filtering:** In this step, we filter raw traffic using a port-based filtering method [10] to separate out relevant network traffic. This separates out messages according to their destination port-numbers, which are typically standardised for a given protocol. We assume that there is no misuse of port numbers (e.g. use of non-standard port numbers for communication [9]) in the captured traffic.

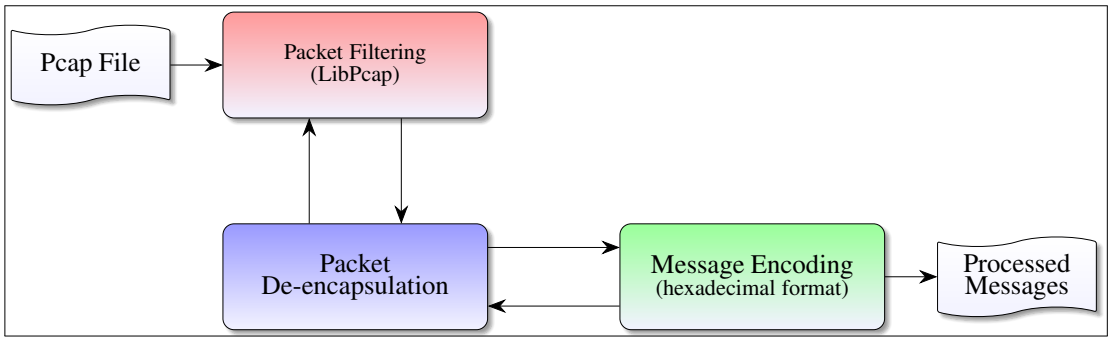


Figure 4.2 Pre-processing captured network traffic.

We use the LibPcap library [117] to filter only packets belong to a specific port number[2]. The filter expression and the captured packets are both have to be provided to library to process the request as shown in Figure 4.2. Packets that do not satisfy the filter criteria are discarded.

- **Protocol De-encapsulation:** The input to this step is a filtered packet by the LibPcap system as illustrated in Figure 4.2. The task of this module is to extract (parse) only data that belongs to the application protocol from the TCP/IP stack (i.e., perform protocol de-encapsulation). Data that belongs to the *transport layer*, *network layer* and *link layer* are excluded. Also, network packets that contain no application data (payload) are also discarded. Once the packet filtering and protocol de-encapsulation steps have been completed, the extracted application messages are forwarded to the next step for further processing.

- **Message Encoding:** Because our aim is to be able to work on binary protocols (as well as plain text protocols), we need to transform the captured traffic into a common format that could be used to represent any characters (including non-printable characters) that belong to these types of protocols. This is to be able to further analyse protocol messages in a textual (printed) representation in the next steps. To this end, we convert

[2]There are alternative filtering methods (expressions) that could readily be adopted using LibPcap when the port-numbers are not fixed.

```
ASCII:  GET / HTTP/1.0\r\n\r\n
HEX.:   474554202f20485454502f312e300d0a0d0a
```

Figure 4.3 A simple HTTP (GET) message and its encoding in Hexadecimal Format.

the captured packets into a *hexadecimal* format. The hexadecimal encoding indicates that every byte (character) in the packet can be represented as a hexadecimal pair such that there are 256 different possible characters that could be represented this way (0x00 - 0xff). Figure 4.3 shows a hexadecimal representation of an HTTP message in the ASCII format.

Feature Generation & Selection

In order to cluster protocol messages into distinctive groups (explained next), messages need to be described to the clustering algorithm in terms of their proximity (similarity/dissimilarity). This proximity is often established based on the characteristics and features of the protocol messages. Accordingly, this step consists of two parts. First, protocol messages are fragmented into tokens (features), then only tokens that carry "distinctive" features are selected. The feature generation and selection is as explained in more detail below.

- ***N*-gram Generation:** For feature generation, we use *n*-grams [58] to tokenise protocol messages. An *n*-gram is a sub-sequence of *n* consecutive characters from a longer sequence. The *n*-gram technique for message tokenisation has been successfully used in many domains such as *Natural Language Processing* (NLP) [126], *Protocol Inference* [23, 101, 37, 116], and *Botnet Detection* [127]. An *N*-gram of size one character (n=1) is called *unigram*; size two (n=2) is a *bigram*; size three (n=3) is a

trigram; size four (n=4) is a *quadrigram*, and so. For example, the list of trigrams that can be generated from the message `GET / HTTP/1.1` are:

`[GET]`, `[ET ]`, `[T /]`, `[ / ]`, `[/ H]`, `[ HT]`, `[HTT]`, `[TTP]`, `[TP/]`, `[P/1]`, `[/1.]`, and `[1.1]`.

The number of n-grams which can be generated from a message of length m using an n-gram of length n can be obtained using the following equation [23]:

$$m - n + 1 \quad (n \leq m) \tag{4.1}$$

To cluster messages, it is necessary to amalgamate all n-grams into a single index of similarity. Messages of the same type normally have similar *n-gram frequency distributions* [23], therefore, we use the n-gram occurrences as a feature to distinguish between protocol messages, i,e,. the set of possible n-grams are arranged in a frequency table where each message can thus be characterised as a sequence of numbers, where each number corresponds to the frequency of a given n-gram in the message. Finally, to normalise the amount of contribution of each n-gram, we apply the *Term Frequency-Inverse Document Frequency* (TF/IDF) as a weighting scheme [58, 128].

- **N-gram Filtering:** As a result of the tokenisation step, a large number of n-grams tend to be generated, this can significantly affect the accuracy of clustering (explained below) as well as the time required for clustering. As discussed in the background (Chapter2, section 2.2.1), the goal of feature selection is to find a small and representative set of features that best describes the "interesting natural" groupings in the dataset. To keep the number of dimensions (i.e. n-grams) as small as possible, we eliminate n-grams that carry no discriminative features, that is by removing n-grams which occur very infrequently [23, 128].

Because the generated frequency matrix is largely *sparse* (most of its elements are zero), we remove sparse n-grams (n-grams occurring 0 times in messages). Specifically, the adopted filtering method depends primarily on the maximal level allowed for sparsity, i.e., the resulting matrix contains only n-grams with a sparse factor of less than the allowed level. The intuition is that the filter selects (retains) only n-grams that occur more than a specific threshold f which is calculated using the following equation [128]:

$$f = m * (1 - sparse) \tag{4.2}$$

where m is the number of messages, and $sparse$ is the maximal allowed sparsity which ranges from > 0 and < 1. For example, if we have 10 messages, and the sparsity level is set to 0.7 (i.e., up to 70% sparse), only n-grams that occurred > 3 times are retained. If the maximal level of the sparsity is set to 0.8 (80%), only n-grams that occurred more than twice (> 2) are kept, and so forth. Clearly, the frequency threshold f is proportional to the number of messages involved, and the higher the sparsity, the more n-grams the filter tends to keep.

This adopted filtering method has been used in many applications of *Text Mining*[3] to eliminate infrequent terms [129]. This method can dramatically reduce the dimensionality of the generated space without losing significant information (frequent n-grams).

Before opting for this simple filtering method (described above), we had implemented and experimented with another feature selection method which is based on an *entropy* measure (information disorder) to determine important n-grams. Although the pre-

[3]The process of inferring useful information form text.

sented approach (described in detail in [50]) tends to reduce "noisy" *n*-grams, it failed to effectively filter large datasets and long messages.

Message Clustering & Validation

In this step, protocol messages are partitioned into distinctive clusters reflecting the message types in the trace, then the identified partitions are validated by a suitable clustering validation measure to determine the quality of the clustering results.

- **Message Clustering:** We use the *Agglomerative Hierarchical Clustering* (AHC) with the *complete linkage* clustering method [29]. This creates a hierarchy of clusters, where coarse, large clusters higher up are split up into more granular lower-level ones. Clusters are obtained by cutting the generated dendrogram at a given threshold (t) as discussed in Chapter 2, Section 2.2.2. Also, agglomerative hierarchical clustering requires a distance measure. Therefore, we need to use a suitable distance measure that can calculate message dissimilarities based on the features we have selected in the previous step.

- **Clustering Validation:** At this point it is important to note that the accuracy of clustering can be highly sensitive to the choice of different parameters in the process. This can include the choice of n when selecting n-grams, as well as the choice of parameters that are specific to the clustering algorithm in question. We use the clustering validation step to validate the output of the hierarchical clustering algorithm using external as well as internal validation measures (as explained in Chapter 2, Section 2.2.2). The clustering validation step as well as the choice of clustering parameters are all discussed in more detail in the empirical study presented in Chapter 5.

Message Alignment & Generalisation

In this step, clusters identified in the previous step are aligned using a multiple sequence alignment algorithm, and the result of the alignment is generalised to generate what we call a *message pattern* that is for each message type identified in the trace as explained below:

- **Message Alignment:** To identify the message structure (field partitioning), we use multiple sequence alignment algorithm. In this particular step, our intention is to work around the intrinsic limitations imposed by classic alignment algorithms and propose a novel approach within the protocol reverse engineering known as the *segment-based* alignment [39]. The approach has been successfully adopted within bioinformatics to improve the alignment accuracy for long and large corpora of protein sequences. Segment-based alignment and how it is integrated into the framework is explained in detail in chapter 6, Section 6.1.

- **Pattern Extraction:** In the context of network packets, aligned regions tend to correspond to one or more protocol fields. For each cluster of aligned messages, the messages are listed in aligned form (similar to the example alignments shown in Table 3.2 in the background). This means that every aligned position can be referred to in terms of a column (e.g. column 1 refers to the first character in every sequence etc.). This step is normally known as the *alignment generalisation* and will be explain in detail in Chapter 6, Section 6.2.

4.3 Implementation

We implemented the framework on the Linux operating system using *R*, and *C* programming languages. The front-end of the analysis engine (apart from the traffic pre-processing and the

alignment steps) is all written in *R* which is consistent with the back-end (*R* environment), and consists of a about 230 lines of code.

The front-end of the traffic pre-processing step is implemented in *C*, and consists of about 208 lines of code. As for the alignment, it is a re-implementation of the *Dialign-2* [70] and consists of about 2,700 lines of *C* code. The implementation of the segment-based alignment tool is discussed in more detail in Chapter 6, Section 6.1.2.

The framework is implemented on top of a number of R packages obtained from CRAN on-line repository. We have used the *n*-gram tokeniser embedded in the *RWeka* package [130] for message tokenisation, the *tm* (text mining) package [128] for filtering *n*-grams, the *proxy* package [131] for providing similarity and distance measures, and the *stats* package [132] for clustering which provides an efficient implementation of the agglomerative hierarchical clustering algorithm.

The framework takes a network capture file in the LibPcap format as an input and produces message patterns of an application protocol. The user interface for the framework is implemented as an *R* script and accessed from the command-line. We intend to build the project as a portable *R* extension package and make it available for future experimentations and evaluations.

4.4 Limitations

The design and implementation of the framework is driven by the motivations discussed in Chapter 3, Section 3.2. Therefore, analysis that intends to tackle protocol encryption or depends on protocol artefacts other than the captured network data falls outside of the scope of the platform.

Also, the framework employs per-packet processing and does not re-assemble fragmented messages that are larger than the allowed *Maximum Transmission Unit* (MTU) [16] which is

typically pre-set to 1500 bytes for the Ethernet interface. Bytes beyond the MTU limit are discarded.

The framework expects the "standard" TCP/IP protocol stack for correct de-encapsulation: the Ethernet protocol for the Data-Link, the Internet Protocol version 4 (IPv4) for the Network layer, and the UDP/TCP protocols for the Transport layer. Therefore, the captured traffic must have been encapsulated using the normal order of TCP/IP protocols. If additional layers of protocols are embedded within the TCP/IP stack, these layers need to be stripped off the standard stack (separately) using other tools such as the *Ipdecap* protocol de-encapsulation tool [133].

4.5 Summary

This chapter has introduced the design and implementation details of a generic framework for reverse engineering the message format of application network protocols. First, the key properties required for the framework have been discussed, then the inference approach and the architectural aspects have been explained. This chapter has also given details to the technologies and implementations of the framework. The limitations of the framework have been outlined as well.

Chapter 5

Investigating Clustering Factors

This chapter starts by a brief introduction on clustering factors and how they emerge within the inference process, then presents the details of an empirical study investigating the impact of certain factors on clustering, and whether suitable factor combinations can be extracted prior to clustering. This chapter concludes with a discussion on the threats to validity that might influenced the results of the experiment.

5.1 Introduction

In this section we give a basic introduction explaining common factors that can affect message clustering and how these factors appear within the process.

5.1.1 Clustering Factors

Successful clustering is highly dependent on parameter settings [29, 30]. Most all of protocol reversing approaches - one way or another - involve some factors (will be explained in more detail in the next section). The quality of clustering in protocol reverse engineering often depends on a range of factors impeded into the process. We define a factor as a variable (or a

Project	Parameter Name	Step	Method	Default Value
ScriptGen [31]	Macroclustering threshold (W)	Message Clustering	Hierarchical Clustering	[0-1]
	Microclustering threshold (w)	Message Clustering	Hierarchical Clustering	[0-1]
Discoverer [26]	Minum Length of text segments	Message Tokenisation	Tokenisation Control	3 letters
	Minimum Cluster Size	Message Clustering	Recursive Clustering	20 messages
	Maximum distinct value for FD	Message Clustering	Recursive Clustering	10
PEXT [25]	Similarity Measure	Message Clustering	Hierarchical Clustering	Longest Common Subsequence (LCSS)
	Clustering Termination Criterion	Message Clustering	Hierarchical Clustering	User-defined
PRODECODER [23]	N-gram Length	Message Tokenisation	N-gram Tokeniser	User-defined
	N-gram Filter threshold	Feature Selection	Frequent N-grams	User-defined
	Maximum Iteration Count (L)	Keyword Generation	Gibbs Sampling	Protocol dependent
	Problexity Score (P)	Keyword Generation	Generlisabity Threshold	Protocol dependent
	LDA Hyper-parameters α and β	Keyword generation	Gibbs Sampling	Protocol dependent
PRISMA [37]	N-gram Length	Message Tokenisation	N-gram Tokeniser	Protocol dependent
	N-gram Selection Threshold	Feature Selection	Binominal Test	Significance level α= 0.05
	Similarity Measure	Mesage Clustering	Weighted distance measure	Euclidean (modified)
Veritas [36]	Message Unit Length (l)	Message Tokenization	Message Unit Extraction	3 bytes
	Protocol Message Length (n)	Message Tokenisation	Message Unit Extraction	12 bytes
	Message Unit filter threshold (λ)	Feature Selection	Kolmogorov-Smirnov Test	Significance level α=0.05
	Similarity Measure	Message Clustering	PAM	Jaccard Index
PI [24]	Similarity Measure	Message Clustering	Hierarchical Clustering	Smith Waterman
	Cut-off threshold	Message Clustering	Hierarchical Clustering	0.6
ASAP [101]	N-gram Length (for Binary protocols)	Message Tokenisation	N-gram Tokeniser	User-defined
	N-gram Filter Threshold	Feature Selection	t-test and Pearson Corr.	Correlation=1
Netzob [102]	Similarity Measure	Message Clustering	Hierarchical Clustering	UPGMA (modified)
	Clustering Method	Message Clustering	Hierachical Clustering	Single Linkage

Table 5.1 A selected list of previous network-based projects shows the approach-dependent factors, their location within the inference process, and their default values.

parameter) that brings about certain effects on the inference process. In this thesis, we will be using the terms factor, variable, and a parameter interchangeably.

We divide factors that affect clustering into two coarse categories: *Sample-related* factors, and *Approach-related* factors. This basic categorisation is to provide a convenient structure for understanding process factors, their relationships, and identifying essential differences between various approaches of protocol reverse engineering.

Sample-related factors are part of the sample attributes such as, its size (number of packets), packet lengths, data type (binary vs ascii) etc. Sample-related factors play important role when the data sample is gathered.

Approach-related factors are linked to the choice of the reverse engineering approach. For instance, the length of the n-gram (in our approach) is a factor generated due to the tokenisation step. Additionally, approach-related factors may include algorithmic parameters, such as the similarity measure which is required by the clustering algorithm to identify message groups

Parameter	Name	Step	Method	Default Value
n	N-gram Length	N-gram Generation	N-gram tokeniser	Protocol dependent
f	N-gram Selection Threshold	N-gram Filtering	Select Frequent N-grams	$sparse = 0.90$
t	Cutting (tree) Threshold	Message Clustering	Hierarchical Clustering	0.97
d	Message Similarity	Message Clustering	Hierachical Clustering	Braun-blanquet
m	Clustering Method	Message Clustering	Hierachical Clustering	Complete Linkage

Table 5.2 A list of parameters used in our approach and their default values.

within the sample. Algorithmic parameters are normally controllable. As a consequence, any protocol inference technique that involves clustering is heavily dependent upon parameter choice.

5.1.2 Clustering Configurations

Sample-related factors normally get into the process as part of the research planing and data collection process. Probably the most common factor of sample-related factors is the sample size which is primarily under the control of the experimenter. On the other hand, approach-related factors are generated due to the adopted inference approach. Table 5.1 shows a list of previous network-based systems that followed different approaches. These systems involve a number of parameters prior and during the clustering step. We have excluded the data-related factors from this list since all network-based approaches have not escaped the dependence of the underlying model of the dataset anyway.

It is evident (from Table 5.1) that most of these factors come into existence either because of a pre-processing step (e.g., tokenisation , dimension reduction etc.), or as a parameter required by the clustering algorithm. As Table 5.1 shows, systems that do not include preprocessing steps (tokenisation, filtering etc.) tend to have less factors. For example, the *PEXT* and *PI* systems use sequence comparison algorithms to "directly" calculate the distance between protocol messages [25, 24]. However, this approach is a problematic because it increases the time complexities and often leads to poor clustering results [26]. On the other hand, authors

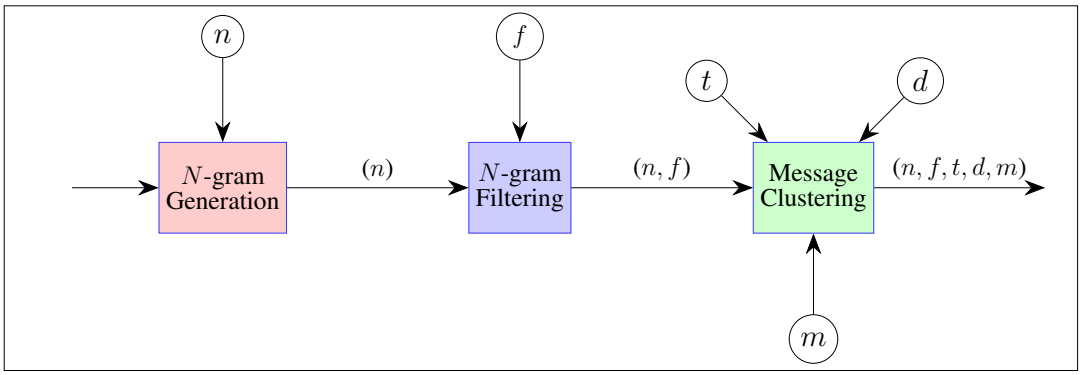


Figure 5.1 A correspondent path diagram of factors listed in Table 5.2.

of *PRODECODER* decided to tokenise protocol messages using n-grams. Furthermore, they have added another step to the inference which is known as the *Keyword Generation*, that is to construct protocol keywords from the selected n-grams, and then use the keywords as features for clustering. As a consequence, the keyword generation step have contributed to three more parameters in the process as shown in table 5.1. Normally, there is always a trade-off between solving a problem at the cost of introducing new factors.

Table 5.2 shows five factors used in the approach we explained in Chapter 4 and their default values. In protocol inference, the transition from one step to another normally defines a new parameter and new set of configurations. A configuration of an inference process involves a set of several factors (eg., n=2, f=0.92, t=0.90, d="Jaccard", m="Complete Linkage"). A simple and convenient way of representing these factors and their locations within the process can be demonstrated by what is commonly known as a *path diagram*. Figure 5.1 shows a basic illustration of the correspondent path diagram of factors listed in Table 5.2.

5.2 Experiment

In this section we empirically investigate the effect of some factors on clustering. As discussed in Chapter 2, Section 2.3.1,we will be using experimental research to determine whether a

dependent variable (cause) can have an effect on a dependent variable (measure) using cause-effect relationship. Most previous efforts to cluster network packets are presented with respect to a fixed configuration of parameters as shown in Table 5.1. However, because clustering algorithms are sensitive to these parameters (and their combinations), the clustering quality could vary significantly [29]. Therefore, an important step in this research is to investigate the effect of some factors on clustering, and whether we could derive a set of factors that produce "good" clustering. In this experiment we seek to answer the following questions:

RQ1 What is the effect of each factor on clustering quality?

RQ2 What is the effect of each factor on clustering time?

RQ4 Can we predict the best factor configurations for clustering?

Answering RQ1 and RQ2 will help identify *key factors* within the process, and make informed decisions about their selection while RQ3 will help determine whether the optimal (or near optimal) factor combinations for clustering could be predicted from the protocol sample.

As discussed in Chapter 2, Section 2.3.2, effect size has many applications. The idea behind RQ1 & RQ2 is similar to the example explained in Chapter 2, Section 2.3.1 where the objective is to determine the effect of specific car parts on the speed of the car. The focus in this example is not to improve the speed of the car or enhance its accuracy, but rather to determine the effect of these components on the speed using the right speed measure (speedometer). Similarly, the aim of RQ1 & RQ2 in this experiment is not to improve clustering (or its time), but instead to determine the effect of specific parameters (within the process) on clustering quality, i.e., measuring the effect is used as a mechanism to show the *difference* (distance) in *clustering scores* versus different *factor configurations* but not the actual clustering scores as will be explained in more detail in Section 5.2.3.

Protocol	Sample Size (# of messages)	# of Clusters	Data Type	Variables	
				n-gram	sub-sample
TFTP	2300	5	Binary	2,3,4	500,1000,2000
DNS	4000	5	Binary	2,3,4,5,6,7,8	1300,2600,3900
SMB	1600	5	Binary	2,3,4,5,6,7,8	500,1000,1500
HTTP	1100	6	Text	2,3,4,5	300,600,900

Table 5.3 Summary of protocol samples and trace-dependant variables.

5.2.1 Datasets

The protocols we have selected for the experiment are the *Trivial File Transfer Protocol* (TFTP) [46], *Domain Name Service* (DNS) [47], *Server Message Block* (SMB) [13], and *Hyper-Text Transfer Protocol* (HTTP) [45]. The main data set have been downloaded from an on-line source [124] which hosts large volumes of captured network traffic. The selected network protocols vary in terms of type of data (binary & text), and the complexity of their message structures. A summary of the collected network traces is provided in Table 5.3.

5.2.2 Experimental Variables

This section introduces the experimental variables involved in the experiment. It starts by describing the selected process variables (independent variables) and the rational for their selection, then it describes the response variables (dependent variables) and the criteria behind its selection.

Process Variables

For the experiment we have selected a list of factors (as explained in Chapter 2, Section 2.3.1). This list is not exhaustive, however the choice of these factors is based on a basic pilot study which included all five factors described in Table 5.2. From the conducted pilot experiments we chose what we perceived to be the most influential factors. Also, the selection of these factors is based on engineering judgement which consisted of various levels (values) and

from both categories we have outlined in section 5.1.1 (sample as well as approach-related factors).

We have selected to experiment with the following variables: length of the *n*-gram, length of the message, size of the sample, and choice of the distance measure. Additionally, the four variables have always been key technical questions in the literature [26, 99, 36, 23]. The rest of process factors are fixed to a specific values, as indicated in Table 5.2.

1. ***N*-gram Length:** We chose a range of values for the *n*-gram for each protocol trace. However, we have also observed the constraint indicated in Equation 4.1, which is the length of the *n*-gram should not exceed the length of the shortest message in the trace. A summary of the *n*-gram's range for each protocol is shown in Table 5.3 (column 4). In this step, we exclude the use of unigrams (n=1). A unigram (i.e., byte-level) analysis is possible, however, from previous experiments we have observed that higher semantic constructs ($n \geq 2$) tend to produce better clustering results.

2. **Message Length:** Three values are selected for the length of the message: *16 bytes*, *32 bytes*, and *64 bytes*. In the pilot study, we have experimented with different message lengths ranged from 3 bytes to 64 bytes, we have noticed that clustering scores, for all protocols, tend to be different and erratic when the length of the message is less than 12 bytes, and relatively similar when the length of the message lies between 12 to 16 bytes. We have also noticed that clustering scores gradually decline when the length of the message is greater than 16 bytes.

3. **Sample Size:** For each protocol, we have selected three sub-samples from three different positions of the total sample while maintaining the order of the messages in each sub-sample. The size of each sub-sample is trace dependent and shown in Table 5.3 (column 6). It is known that the reliability of the sample increases as its

size increases [82], however the aim in this experiment is to find out how this factor is affecting clustering accuracy.

4. **Distance Measure:** As explained in Chapter, 2 Section 2.2.2, most clustering algorithms require a distance measure to calculate distances between messages in a dataset. With respect to distance measures, we use five distance measures, four measures are based on the similarity coefficients of the *Jaccard* index, *Dice* index, *Braun-Blanquet* index and the *Cosine* similarity index [58] while the fifth is the *Euclidean* distance measure [57]. For the similarity coefficients, the distance is defined as $D(a,b) = 1 - S(a,b)$, where S is the similarity of two messages represented by a and b features respectively. The chosen distance measures are diverse and commonly used in the literature [58, 27, 36].

Response Variables

Since we are interested in how the above variables affect clustering, we use the *Adjusted Rand Index* (ARI) [134]. As discussed in Chapter 2, Section 2.2.2, clustering validation measures are divided into two categories: extrinsic and intrinsic validation measures. The Adjusted Rand Index is an extrinsic clustering validation measure and its score ranges from 0 to +1 where +1 indicates the two sets of clusters are identical, and 0 when the two sets are completely independent.

In a separate pilot study - conducted on the side of the experiment - we have experimented with a variety of extrinsic validation indices involved, the *Rand Index* (RI) [135], the *Adjusted Rand Index* (ARI) [134], the *Normalised Mutual Information* (NMI) [136], and the *Folks-Mallows Index* (FMI) [137]. We have observed that the Adjusted Rand Index is a suitable measure in particular for its clustering accuracy, and sensitivity to the changes we apply on the independent variables. The performance of the ARI as well as other external validation measures used in the study will be discussed in more detail in Section 5.2.4.

As for intrinsic validation measures, we use the following indices: *Ball-Hall* (BH) [138], *Calinski-Harabasz* (CH) [139], *Davies-Bouldin* (DB) [60], *Trace_WiB* (TW) [140], *SD* index [28], and the *S_Dbw* [141]. Similar to the choice of the external measures, the internal validation indices are chosen after a preliminary study investigated a large list of internal measures implemented in a dedicated software package [142]. The selected measures (listed above) are chosen based on different statistical measures as well as recommendations by previous studies [61]. We also ruled out internal measures that require intensive computations as well as measures that tend to generate infinite (or similar) results (e.g., Dunn index [59]) which is difficult to assess among other similar results. As briefly pointed out in Chapter 2, Section 2.2.2, internal validation measures are often based on two points of comparison: *Compactness* and *Separation* where the first measures the distance between data objects within a cluster (aka within-group scatter), and the later measures the distance between one cluster and other clusters (aka between-group scatter). The internal measures of *Calinski-Harabasz* (CH), *Davies-Bouldin* (DB), *Trace_WiB* (TW), *SD* index, and the *S_Dbw* consider both aspects in the evaluation (compactness and separation) in the way of ratio or summation. On the other hand, the *Ball-Hall* (BH) index considers only one aspect which is the compactness of the cluster as described below:

1. For the **Ball-Hall** measure, the compactness of a cluster is based on measuring the dispersion of the cluster using the *sum of squares* within the clusters [142]. It is based on the idea is that for each cluster, the mean of the squared distances between points and their centres is calculated, then the overall mean is computed by deviding these distances by the number of clusters. The optimal value for this index can be retrieved using the what is known as the *elbow method* [142] where the best value correspondes to the greatest difference (*max diff*) between two successive slopes. i.e., on a diagram representing the index values against the number of selected clusters, this corresponds to an elbow shape hence the name [61].

2. The **Calinski-Harabasz** index is based on the average between the sum of squares between clusters (separation), and the sum of squares within the clusters (compactness). The best partition is the one corresponding to the greatest (*max*) value of the index [61].

3. The **Davies-Bouldin** measure is calculated by computing the similarity for each cluster with other clusters, then the highest value is assigned to the cluster to represent its cluster similarity. The overall DB index is calculated by averaging all cluster similarities. By minimising this index, clusters are more distinct from each other. Therefore, we seek to retrieve the minimum (*min*) value for this index [61].

4. The **SD** index calculates the compactness of a cluster based on the average scattering of data objects which is based on variances between objects within a cluster. The separation between clusters is based on distances between cluster centres. The value of the SD index is the summation of the compactness and separation. The optimal number of clusters is obtained by selecting the minimum (*min*) value [142].

5. The **S_Dbw** index takes cluster density into consideration in calculating cluster separation. For each pair of cluster centres, at least one of their densities should be larger than the density of their midpoints. The S_Dbw calculates cluster compactness, and the overall value similar to the SD index (explained above). The minimum (*min*) value of the S_Dbw index indicates the optimal number of clusters [61].

6. The **Trace_WiB** index is based on the T statistic [142] where the sum of the scatter matrices in each cluster calculated (which measures the compactness) and multiplied by the between-cluster scatter matrix (which measures the separation). Simialr to the Ball-Hall measure, the index uses the *elbow method* (*max diff*) to retrieve the best clustering value [142].

For more details and complete mathematical definitions of the selected measures, we refer the reader to [142, 61].

Choice of the Clustering Algorithm

As we pointed out in Chapter 2, Section 2.2.2, there are many clustering approaches available for data mining. In cluster analysis, there is no "best" clustering algorithm [56, 143], therefore we need to choose a clustering algorithm that suits our application and the data samples. Choosing an appropriate clustering algorithm is often a challenging decision because it is normally based on various criterion related to the dataset, and the clustering algorithm such as, its time complexity, ability to discover clusters with arbitrary shapes, domain knowledge (e.g., known number of clusters), resistance to noise and outliers. Normally, there are advantages and disadvantages associated with each clustering algorithm and a smart choice should normally be based on the criterion that fulfils the requirements of the analysis.

Similar to many approaches (summarised in Table 5.1), the clustering algorithm used in our protocol analysis is the *Agglomerative Hierarchical Clustering* (AHC). We chose this approach because hierarchical clustering does not require us to specify the number of clusters, which is difficult parameter to estimate [30]. Also, an important feature of hierarchical clustering during the analysis is to provide a picture of the data that can easily be interpreted through the generated dendrogram.

As discussed in Chapter 2, Section 2.2.2, Agglomerative hierarchical clustering normally depends on three parameters: the clustering method, the distance measure, and the cut-off point of the tree. Different settings for these parameters may yield different clustering results, therefore we need to determine suitable settings for these parameters. In this experiment, we decided to determine the effect of the distance measure by experimenting with different choices of distance measures, and set other parameters to the values shown in Table 5.2. These values are not chosen arbitrarily, however they are set after observing from several

experiments that clustering results significantly improve at these settings. Section 5.2.5 presents further analysis with regard to the choice of the AHC algorithm and its performance.

5.2.3 Methodology

To carry out the experiment, we use the framework described in Chapter 4. We use the clustering validation step to validate clustering results using the intrinsic and extrinsic validation measures we have identified in the previous section. As explained in Chapter 2, Section 2.2.2, intrinsic measures do not require messages labels for validation. However, for the extrinsic measure (ARI), clustering is validated by comparing the generated partitions of the clustering algorithm with the *ground truth* labels (the actual message classes). We use message keywords and opcodes to define true message labels. For example, for the HTTP protocol, GET, POST, HEAD are all keywords that define three different message types. Similar to the approach followed in *Discoverer* [26], instead of manually extracting true message labels from the protocol documentation (e.g., RFC), we use of-the-shelf network analyser that is capable of correctly identifying and parsing the protocol traffic. We use *tshark* network analyser [144] (a command-line version of Wireshark) to automatically identify and extract ground truth labels (message types) and subsequently provide them to the external measure (ARI) for validation. The steps for clustering validation are shown in Figure 5.2. For each protocol trace, we use the framework (described in Chapter 4) to cluster protocol messages and validate the results using the internal and external validation measures. The process is systematically executed using all possible combinations of variable values. Each time the clustering validation scores as well as the clustering time (user application time) are recorded.

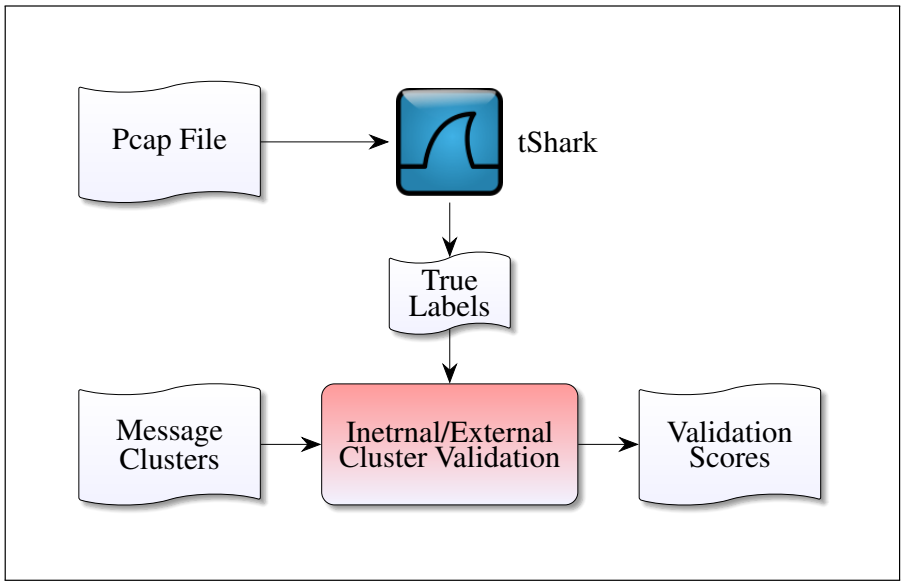


Figure 5.2 Validating clustering results using external/internal validation measures. It shows show how ground truth labels are extracted from the captured traffic using the tShark network analyser, and fed into the external measure (ARI) along with the message clusters produced by the clustering algorithm.

Effect of variables on clustering accuracy and time (RQ1 & RQ2)

To measure the effect of each variable (e.g., choice of the *n*-gram), we perform grouped statistical tests on the clustering validation scores produced by the external measure. Because we cannot presume normality of the distribution of the generated data (clustering validation scores), we resort to *non-parametric* statistical tests. We use *Cohen's d* (explained in Chapter 2, Section 2.3.2) to measure the effect size.

We use Cohen's *d* is to measure the mean difference (standardised) between two groups of ARI scores. Cohen's *d* is a pairwise test, therefore we carry out every possible pairwise test for each variable (e.g., we compare the ARI scores for every pair of *n*-gram values). The total number of pairwise tests that can be performed on each variable is: $m(m-1)/2$, where m is the number of values (levels) assigned to the variable.

Because we are mainly interested in the relative distance between variables and not the direction (which one is greater), we take the mean absolute value for all *d*'s to calculate the

aggregate effect of each variable. The effect estimate with the associated confidence interval (CI) are reported using 95% confidence level for all the tests. For clustering time, we follow the same procedure described above while substituting the ARI scores with the recorded time.

Finding best variable configurations for clustering (RQ3)

To answer RQ3, we could simply refer to the highest score returned by the ARI and retrieve the corresponding variable values. However, in practice, ground truth labels are often not available. Therefore, we would like to use an approach that does not rely on any external information. In this research question, we use the intrinsic validation measures. Internal measures have been used in previous applications to determine the correct (or an approximate) number of clusters in a dataset [61, 27, 36], which is ultimately to be provided to a clustering algorithm that requires the number of clusters beforehand. Similarly, our goal is to use the procedure described below to investigate whether internal measures can also be used to determine the optimal variable configurations that achieve best clustering:

- Step 1: For each protocol trace, use all possible variable combinations to get different clustering results.
- Step 2: Measure the clustering result obtained in step 1 using the corresponding internal validation index.
- Step 3: Choose the best validation result according to the criteria applied with the internal measure that retrieves the number of clusters [61]. (i.e., each internal validation measure has a rule which must be applied in order to obtain the optimal number of clusters, see [61, 142] for more details).
- Step 4: Finally, we retrieve values of variables corresponding to the optimal number of clusters obtained in step 3. Because detecting the number of clusters in a dataset

does not indicate that the clustering algorithm has classified these messages correctly (produced homogeneous clusters), we do not always expect the correct number of clusters "guessed" by internal validations matches the exact best clustering score produced by external validations (e.g., ARI). However, in the absence of external information (actual classes), internal measures can only be used as an "approximation" measure for clustering.

5.2.4 Results & Discussion

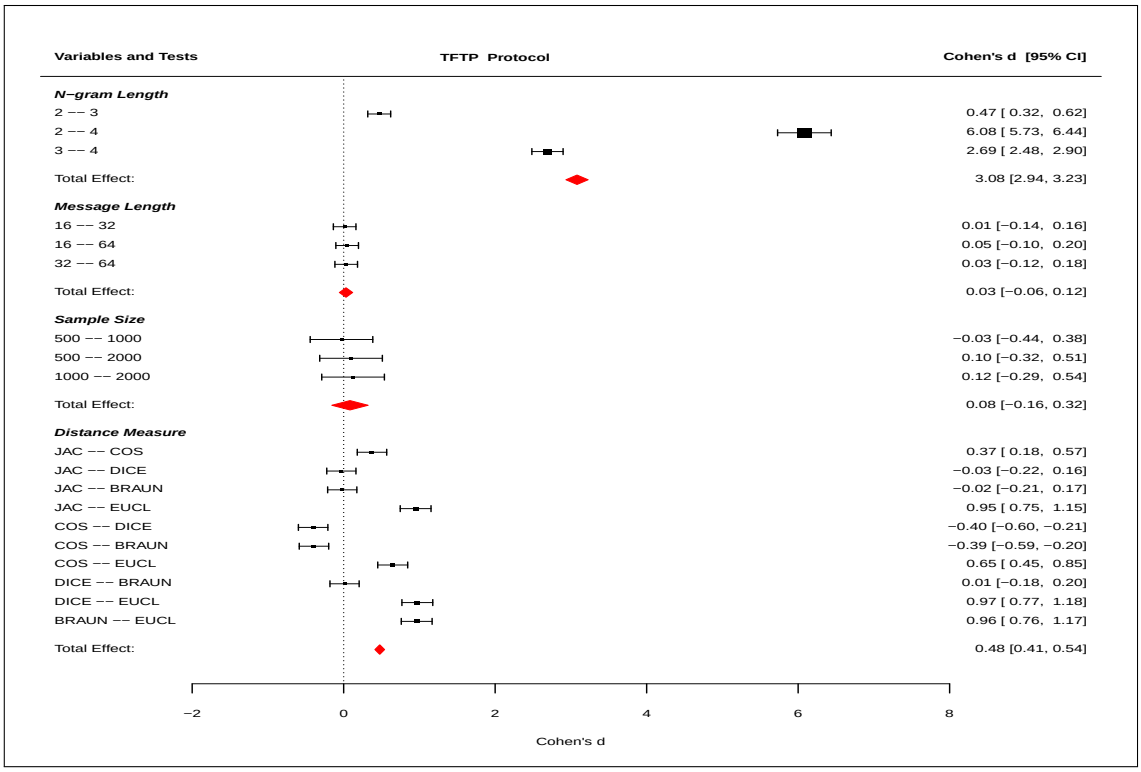
In this section we present the results of our experiment aiming at illustrating how the chosen process variables affect clustering accuracy and time.

RQ1 - What is the effect of factors on clustering quality?

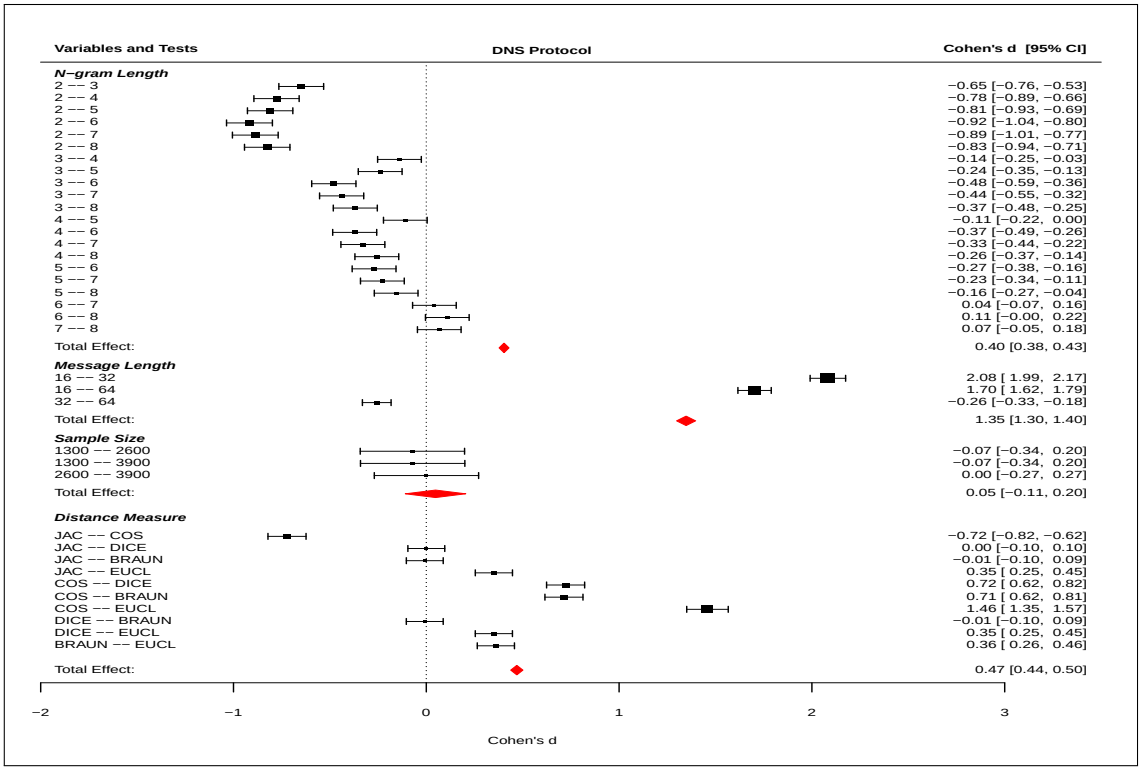
Figure 5.4 (a-d) shows Forest plots of four protocols illustrating the effect of variables. As explained in Chapter 2, Section 2.3.2, the left-hand column lists the names of the variables and the pairwise tests carried out between variable values. The right-hand column is a plot of these effects (shown as squares) within confidence intervals represented as horizontal lines where the size of these squares is relative to the estimated effects. The overall effect of each variable is shown as a diamond. A vertical line indicating no-effect is also plotted.

The overall results show that the choice of the distance measure and length of the message have a very large effect on clustering accuracy. Therefore, the choices of these variables are especially important. However, for TFTP, the choice of the n-gram seems to be the pivotal variable for clustering. The overall effect of the sample size is negligible. The results are explained in more detail below.

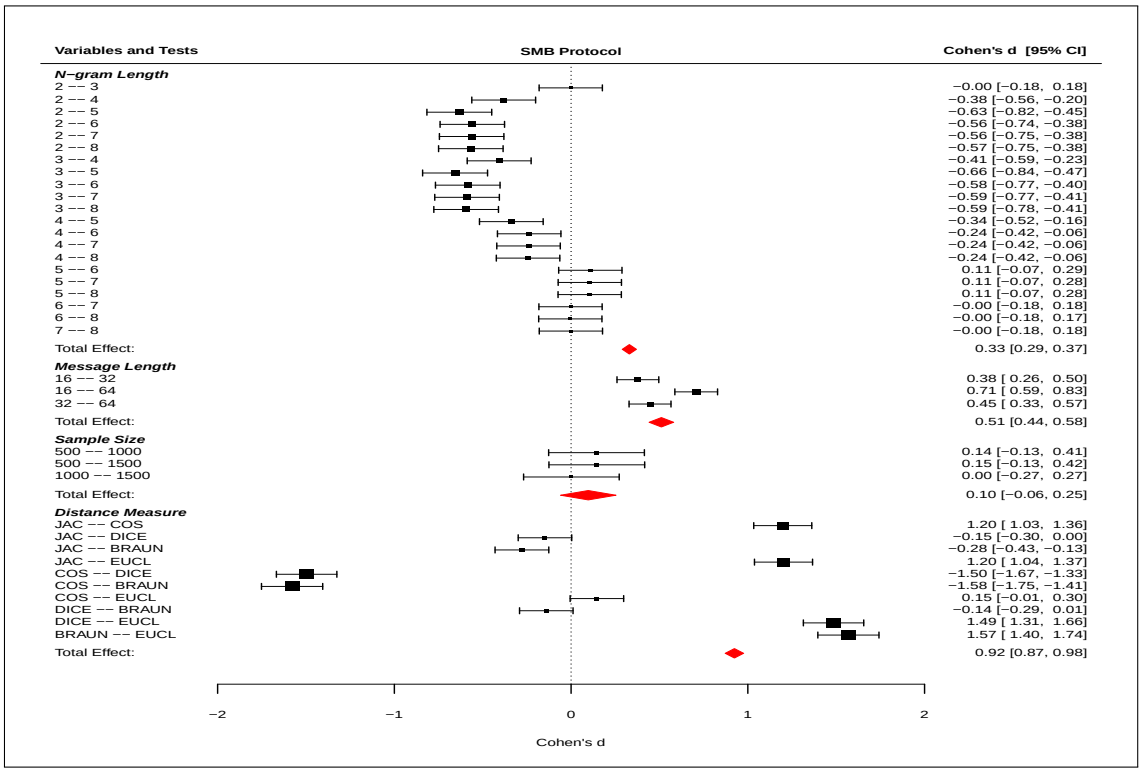
Distance Measure. The overall effect of the distance measure on SMB and HTTP is significantly *large*. The effect is clearly visible in Figure 5.4 (c-d) as point estimates and



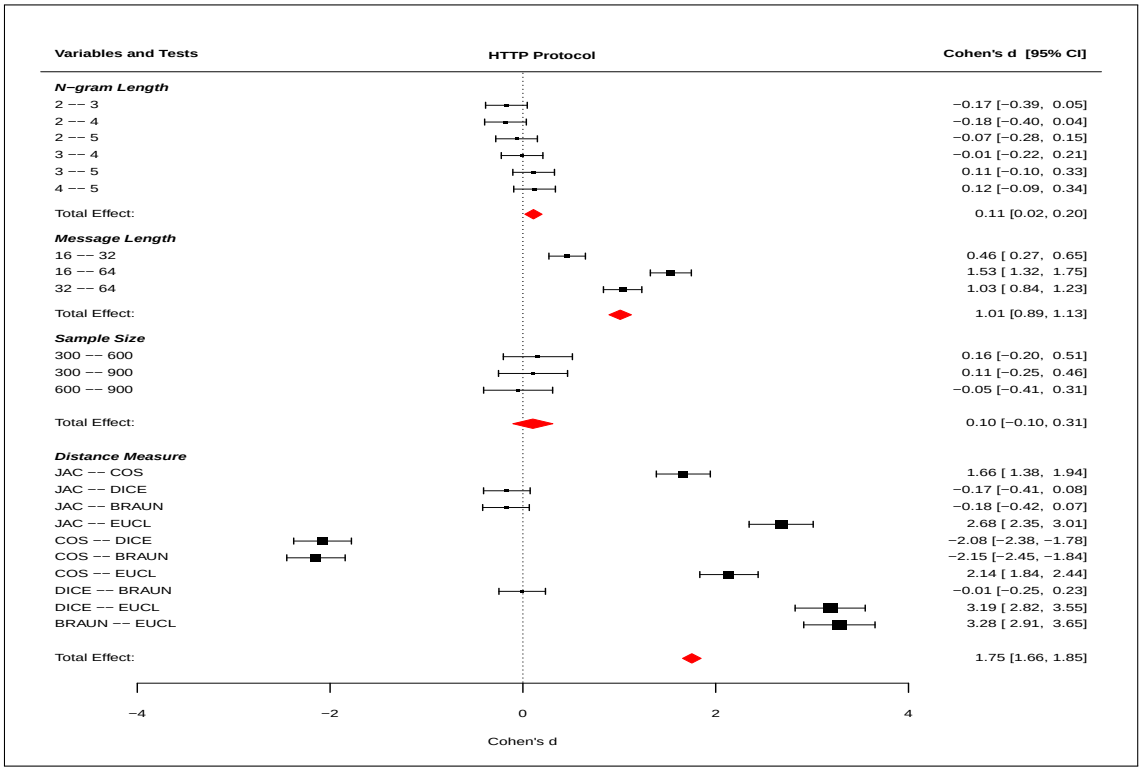
(a) TFTP Protocol



(b) DNS Protocol



(c) SMB Protocol



(d) HTTP Protocol

Figure 5.4 Forest plots showing the effect of variables on **clustering accuracy**. The figures show the estimated effects of the pairwise tests on the adjusted Rand scores between variable values as well as the aggregate affect of each variable. It also, shows the corresponding 95% confidence intervals for each test.

confidence intervals are shifted away from the no-effect line with an aggregate effect of *0.92*, and *1.75* standard deviations respectively. For the SMB and HTTP, we also notice (from the pairwise tests) that there is consistent and large contrast between the binary measures (Jaccard, Dice, Braun-blanquet) and the euclidean and cosine measure. Further, the difference among binary measures can be noted that the distance between the Jaccard and Braun-blanquet is observable which is indicative that binary measures do not produce the same clustering quality. In general, the Jaccard index seems to perform slightly less than other binary measures with the SMB and HTTP protocols.

As for TFTP and DNS, the effect of the distance measure is relativity less (about *0.5* standard deviation). However, judging the precision of the estimated effects corroborated by short confidence intervals, the effect is large enough to indicate the importance of the distance measure for these protocols as well. In the pairwise tests, the performance of the binary measures is quite similar hence we notice that the effect estimate and confidence intervals pass the no-effect line. For the DNS, the contrast between the cosine and euclidean measure is very large with about *1.48* standard deviation due to the fact that the cosine measure performed better than the euclidean measure.

Message Length. For DNS & HTTP, the average effect of the message's length is greater than *1.0* standard deviation which is very *large* for both protocols, while the effect on SMB is about *0.5* standard deviation.

For the HTTP and SMB, the contrast between message lengths is happening between l=16 and l=64, which quite natural, the longer the message, the larger the effect. However, for the DNS protocol, the biggest difference seems to occur between l=16 and l=32 (about *2.0* standard deviation). This is because noisy features have been generated within this range and could not be filtered. We could tell that the distance between l=64 and l=16 is reduced that is because more related features generated and compensated the noise.

For TFTP, the message length seems to have a *negligible* effect on clustering. This can be clearly seen in Figure 5.4 (a) as all confidence intervals cross the no-effect line. That is because the majority of protocol messages in the TFTP trace are either within a range of similar length (less than 16 bytes) and tend to generate the same set of $n-grams$. When messages are longer, they normally contain a payload (data packets). Most of N-grams generated from data packets are infrequent (noise) and get excluded by the filter. Therefore, the distribution of the selected n-grams does not change and the clustering scores relatively stay the same.

***N*-gram Length.** The effect of the n-gram length on DNS, SMB and HTTP ranges from *small* to *medium*. However, Figure 5.4 (b&c) indicate that the effect of this variable is much more significant for the DNS & SMB protocols than the HTTP as clearly shown by the individual tests as well as the overall effect of the variable.

For DNS and SMB, there is quite big contrast between the distribution of n-grams when n=2 and n=6,7 and 8, and a tiny difference between n=6, n=7 and n=8 which indicative that n-grams with length of 5 or above generate similar frequency distributions. The point of change is for the DNS is when n=6 which have about 0.92 standard deviation which is quite considerable effect. As for the SMB protocol, the estimates have wider confidence intervals suggesting the variability of the n-gram distributions as well. The turning point for the SMB protocol is when n=5. For the HTTP protocol, There is not seem to be significant difference between the n-gram lengths affecting clustering, hence we see all confidence intervals pass the no-effect line.

For TFTP, the effect of the n-gram is *very large* with *3.08* standard deviation as illustrated in Figure 5.4 (a). The difference between frequency distributions when n=2, and n=3 is rather small. It is quite evident that the n-gram distribution is significantly different when n=4

where the effect between n=2 and n=4 reaches to *6.08* standard deviation. Therefore, For the TFTP, n=4 is what we consider a turning point for clustering quality.

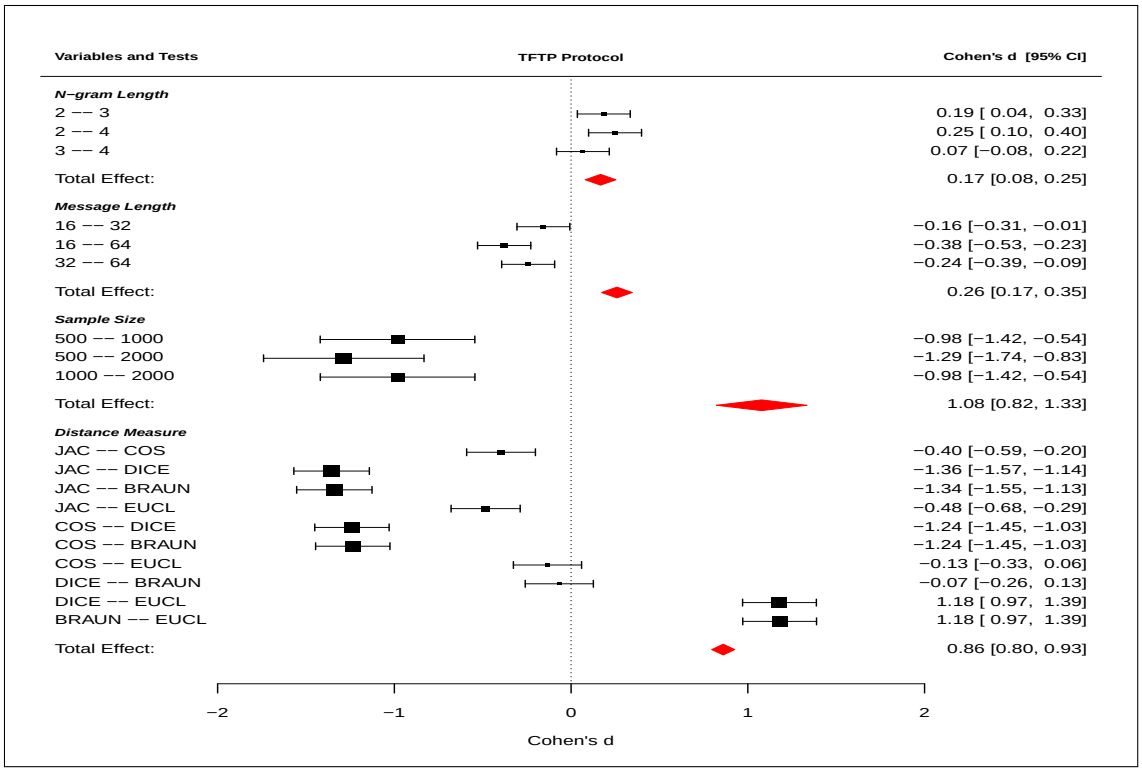
Sample Size. For all protocols, the overall effect of the sample size is *negligible*. This is clearly shown in Figure 5.4 since the effect of this variable situated within wider confidence intervals and all of these confidence intervals intersect with the no-effect line. The aggregate effect also indicates that the sample size has insignificant impact on clustering. The sample size does not seem to have an impact on clustering because clustering quality depends on the selected set of n-grams which generally indicates that the frequency distribution of the chosen set of n-grams does not significantly change by the number of messages within the sample.

RQ2 - What is the effect of factors on clustering time?

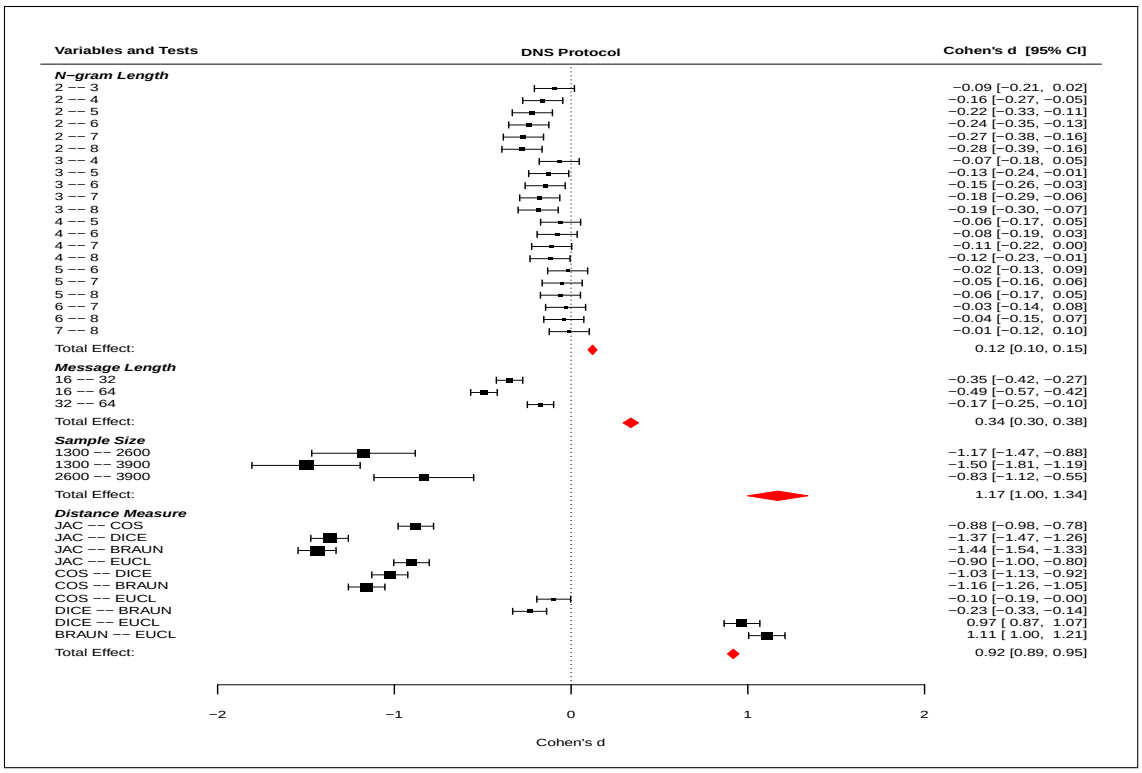
The overall results show that the sample size and the distance measure have significant effect on clustering time. Therefore, the choices of these variables are very important. The length of the n-gram and length of the message have an affect on clustering time with rather considerably less. The results are explained in more detail below.

Sample Size. For all protocols, the sample size seems to have very *large* effect on clustering time with an average above *1.0* standard deviation as shown in Figure 5.5 (a-d). Also, from the individual tests (and the aggregate effects) of these variables, we notice wider confidence intervals (and wider diamonds) amounting to the fluctuation of the recorded time between variable levels.

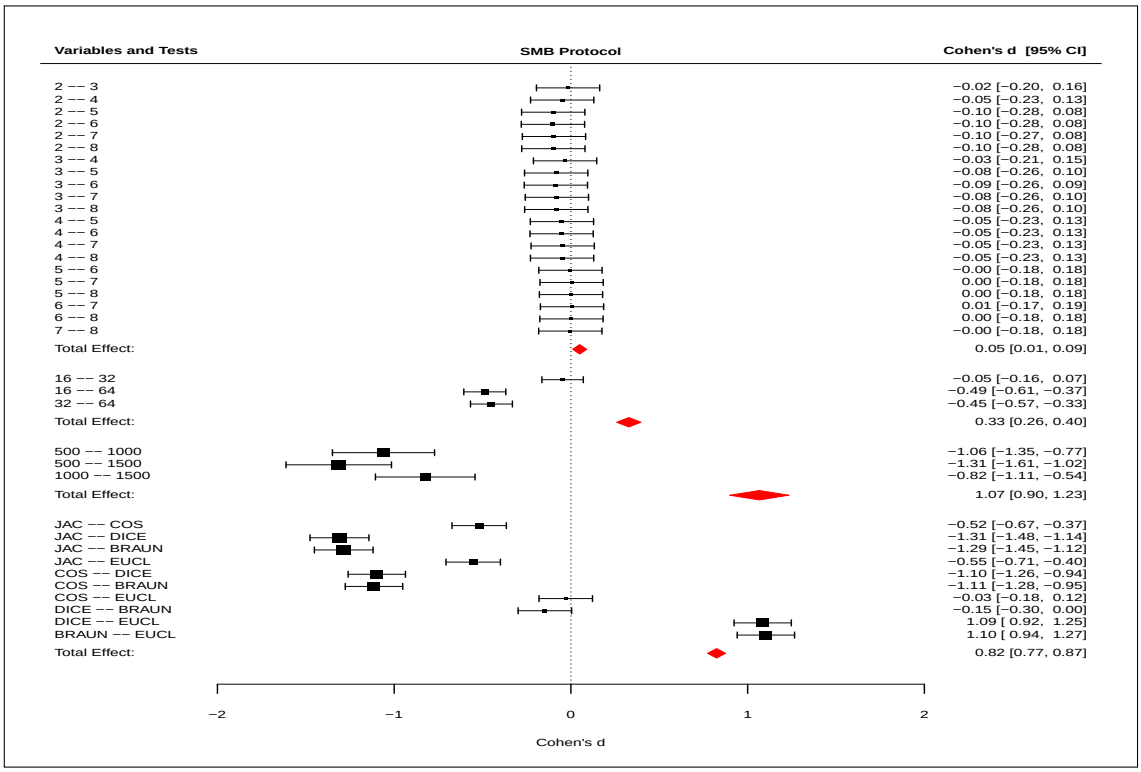
Distance Measure. For all protocols, the effect of the distance measure is *large*. We know that the calculation of the distance matrix depends on the sample size, hence the



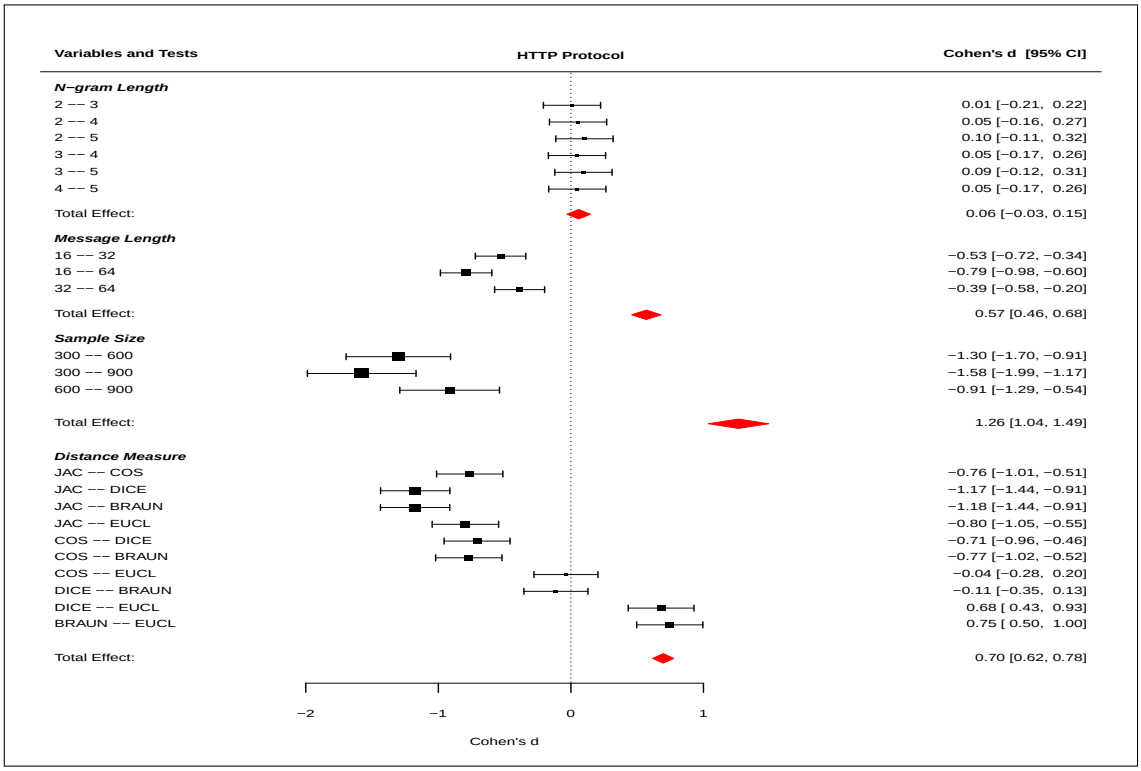
(a) TFTP Protocol



(b) DNS Protocol



(c) SMB Protocol



(d) HTTP Protocol

Figure 5.5 Forest plots showing the effect of variables on **clustering time**. The figures show the estimated effects of the pairwise tests on the recorded time between variable values as well as the aggregate affect of each variable. It also, shows the corresponding 95% confidence intervals for each test.

dependency is quite visible here, i.e., the larger the sample, the more time is required for distance calculations.

As Figure 5.5 (a-d) shows, it seems that the choice of the distance measure matters when it comes to which binary measure consumes less time for clustering. We have noticed in the previous question that binary measures have negligible contrast between them when it comes to affecting clustering accuracy. However, when it comes to affecting time, they are all the same. As Figure 5.5 shows, for all protocols, the Jaccard index seems to be faster than other binary measures by at least *1.0* standard deviation. Also, the euclidean and cosine measure seem to require less time for distance calculations than the Dice and Braun-blanquet measures. Cosine measure and euclidean measure seem pretty similar in terms to clustering speed.

***N*-gram Length.** For all protocols, the length of the *n*-gram has relatively *smaller* impact on clustering time as indicated by the forest plots shown in Figure 5.5 (a-d). For all protocols, the overall effect of the *n*-gram's length is no more than *0.1* standard devotion. For DNS and SMB protocols, we notice that high values of *n*-gram lengths (*n*=5,6,7,8) seem to have similar effect on time which is almost none, furthermore, lower *n*-grams (*n*=2, 3, 4 & 5) take less time to process. For TFTP lower *n*-gram lengths (*n*=2 & 3) seem to cause just the opposite by consuming more time to process than when *n*=4. For HTTP length of the *n*-gram seems to make very negligible difference on time.

Message Length. The length of the message for TFTP, DNS & SMB protocols has *small* effect on time and *medium* effect on HTTP. Although the influence of this variable ranges from small to medium it is a contributor factor to clustering time. That is due the generation of more features as the message increases in length which ultimately increases the dimension of distance matrix.

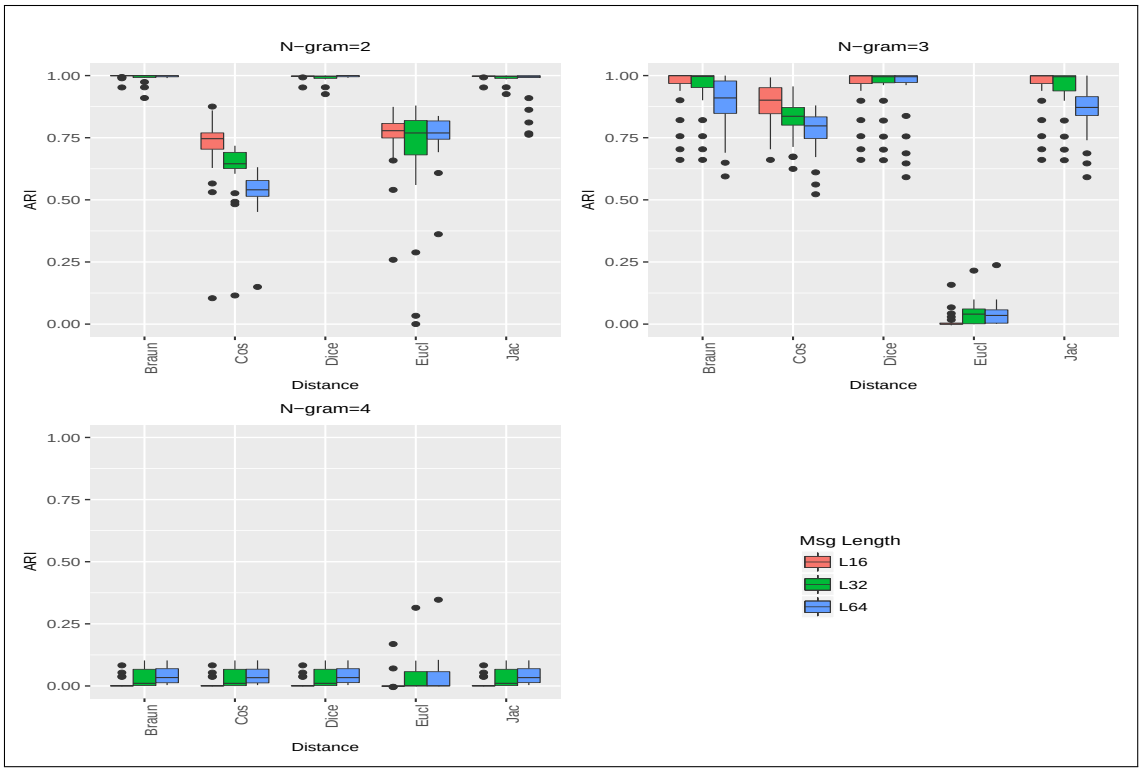
Dataset	Sample (k)	N-gram (n)	Message (l)	Distance (d)	ARI Score
TFTP	2300	2	32	Braun-Blanquet	0.999982
DNS	4000	6,7,8	16	Jaccard, Dice, Braun-Blanquet	0.989203
SMB	100	5	16	Jaccard, Dice, Braun-Banquet	1.000000
HTTP	700	4	16	Jaccard, Dice	0.962188

Table 5.4 Best variable combination for each protocol and their correspondent ARI scores using the Agglomerative Hierarchical Clustering (AHC) algorithm.

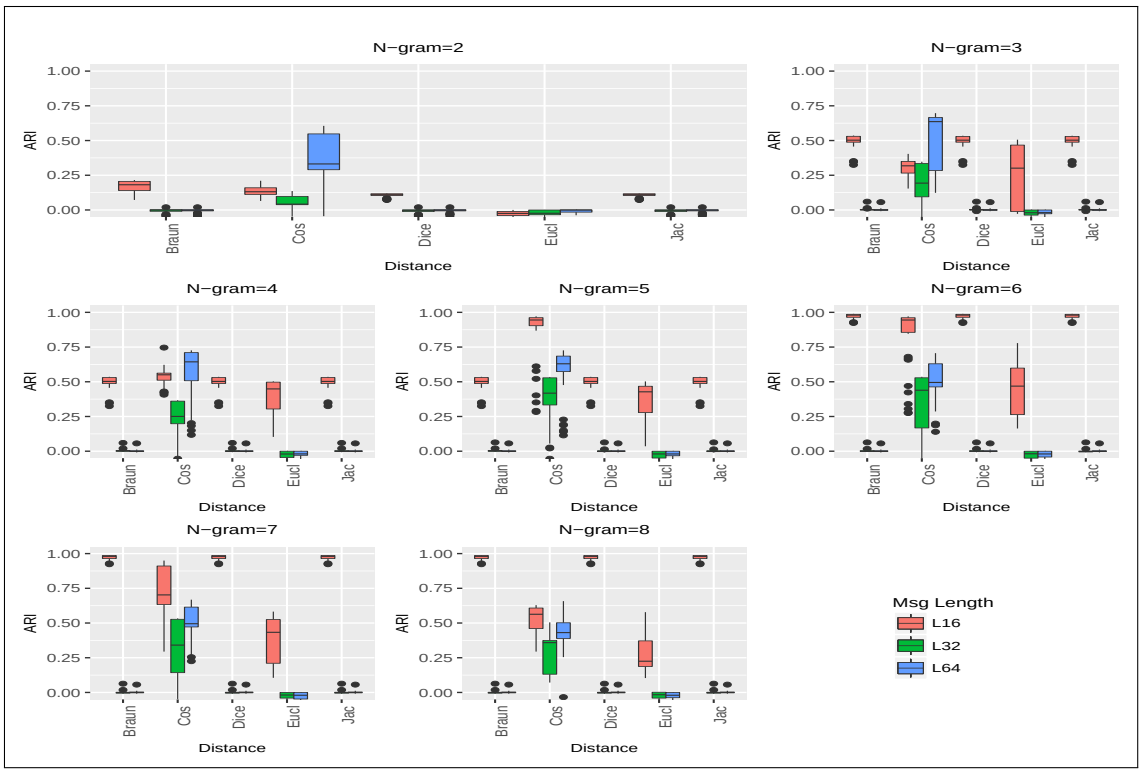
RQ3 - Can we determine the best factor configurations for clustering?

Using the recorded scores of the Adjusted Rand Index (ARI) as a reference, the optimal variable combinations that produced best clustering are listed in Table 5.4. The ARI scores against different variable configurations can also be visually corroborated by the box plots shown in Figure 5.6 (a-d). As as discussed in Section 5.2.3, we assume the best clustering configurations correspond to the predicated number of clusters returned by the internal measure. We understand that this may not always be the case because we have no idea how the clustering algorithm classifies these messages. However, it is considered a close approximation for the best clustering when extrinsic measures cannot be used (lack of true labels).

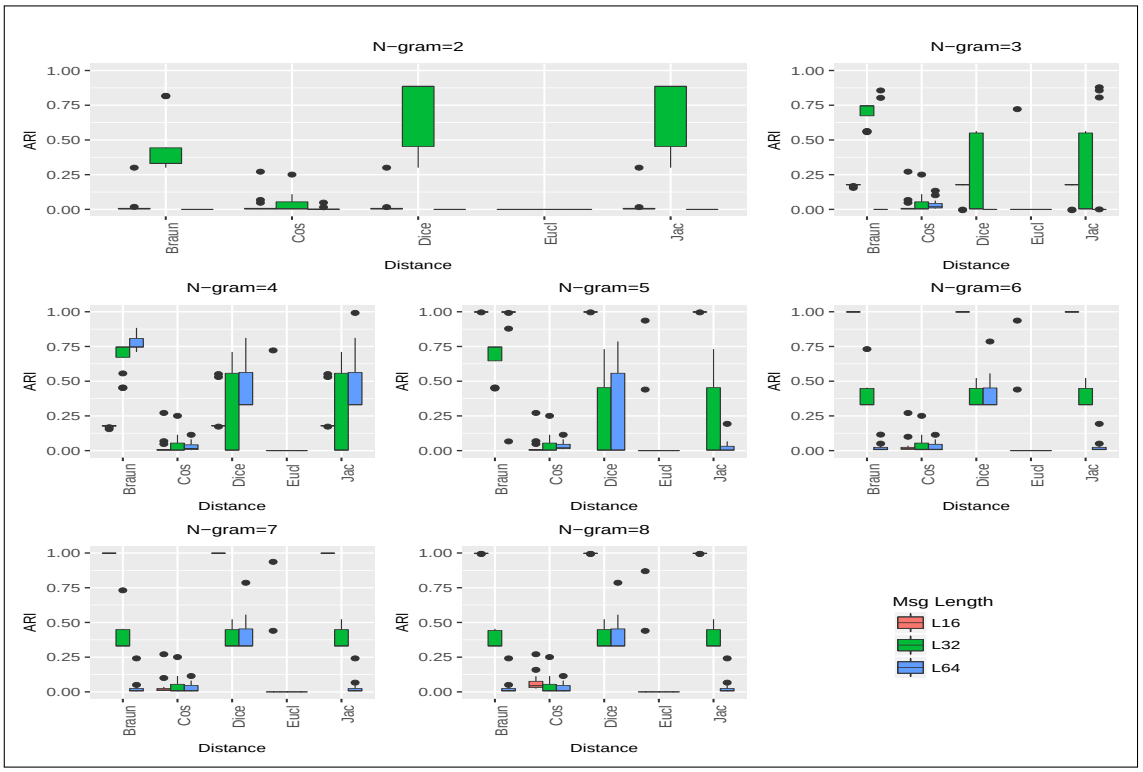
As for the performance of the internal measures in predicting the optimal variable configuration for clustering, the results show that the combination of the *Ball-Hall* validity index and the *Braun-Blanquet* similarity measure tend to give the best results. The results are shown in Table 5.5 (a-d). The table shows the experimental variables and chosen internal measures as well the score of the Adjusted Rand corresponding to each internal measure. For TFTP and DNS, the *Ball-Hall* index has predicted the "exact" best variable combination (best clustering score) recorded by the ARI as indicated in Table 5.5 (a-b), while the *SD* and *Calinski-Harabasz* indices have predicted the best variable combination for SMB and HTTP protocols respectively with the *Ball-Hall* index comes the second.



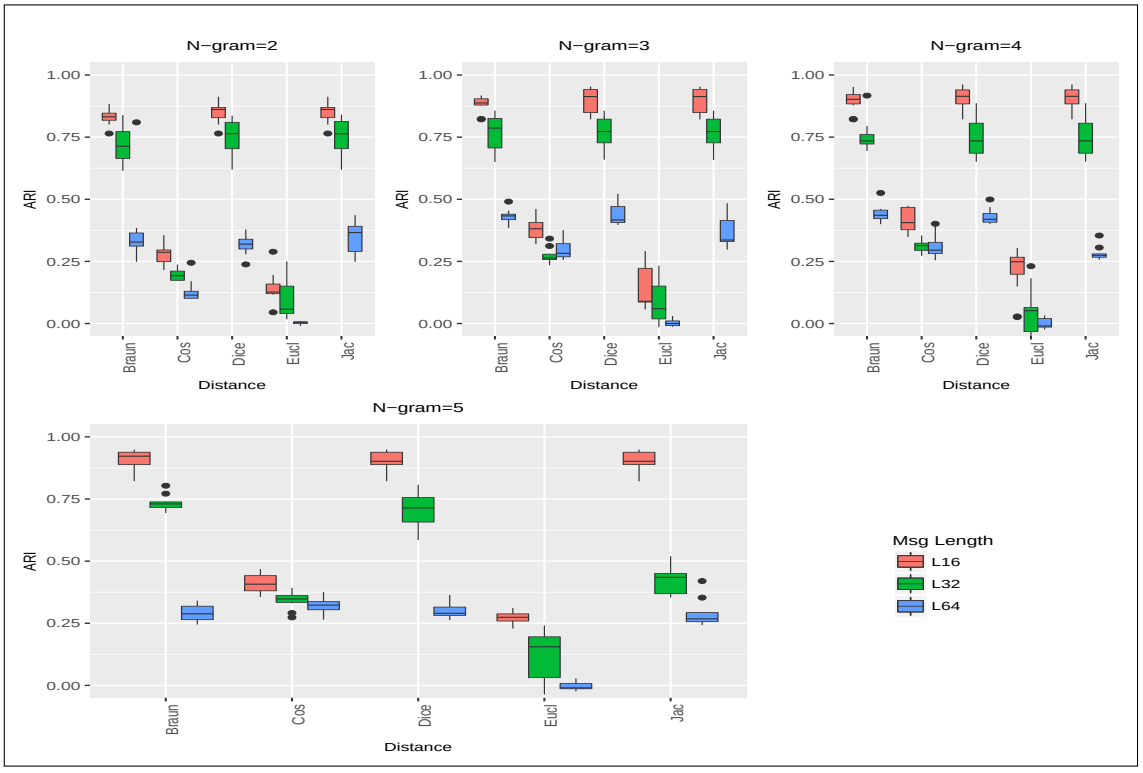
(a) TFTP Protocol



(b) DNS Protocol



(c) SMB Protocol



(d) HTTP Protocol

Figure 5.6 Box plot showing **clustering accuracy** for each protocol (a-d). Each plot shows the different combinations of variables and the correspondent clustering score of the Adjusted Rand Index (ARI).

Distance	Sample	*n*-gram	Message	adj. Rand	Internal Measure
Jaccard	200	2	16	0.998782	Trace_WiB
	2300	2	16	0.998254	Ball_Hall
	100	3	16	0.938577	SD_Dis
	1500	4	16	0.000214	Calinski_Harabasz
	1500	4	16	0.000214	Davies_Bouldin
	1100	4	16	0.000131	S_Dbw
Dice	200	2	16	0.998782	Trace_WiB
	2300	2	32	0.998254	Ball_Hall
	100	3	16	0.938577	SD_Dis
	1500	4	16	0.000214	Calinski_Harabasz
	1500	4	16	0.000214	Davies_Bouldin
	1100	4	16	0.000131	S_Dbw
Braun-Blanquet	2300	2	32	0.999982	Ball_Hall
	2300	2	32	0.999982	Trace_WiB
	100	3	16	0.938577	SD_Dis
	1500	4	16	0.000214	Calinski_Harabasz
	1500	4	16	0.000214	Davies_Bouldin
	1100	4	16	0.000117	S_Dbw
Cosine	1100	3	16	0.945071	S_Dbw
	800	2	64	0.629911	Trace_WiB
	100	2	16	0.104278	SD_Dis
	1700	4	64	0.003506	Ball_Hall
	1500	4	16	0.000214	Calinski_Harabasz
	1500	4	16	0.000214	Davies_Bouldin
Euclidean	100	4	32	0.314674	Ball_Hall
	2100	3	32	0.015229	S_Dbw
	1200	4	32	0.000971	Trace_WiB
	800	4	16	0.000610	SD_Dis
	2200	4	16	0.000230	Calinski_Harabasz
	1500	4	16	0.000000	Davies_Bouldin

(a) TFTP Protocol

Distance	Sample	*n*-gram	Message	adj. Rand	Internal Measure
Jaccard	4000	7	16	0.989203	Ball_Hall
	4000	4	16	0.531224	Calinski_Harabasz
	200	4	16	0.344495	SD_Dis
	2300	2	16	0.111911	Davies_Bouldin
	100	2	16	0.072107	S_Dbw
	800	2	32	0.019798	Trace_WiB
Dice	4000	7	16	0.989203	Ball_Hall
	4000	4	16	0.531224	Calinski_Harabasz
	200	4	16	0.344495	SD_Dis
	2300	2	16	0.111911	Davies_Bouldin
	100	2	16	0.072107	S_Dbw
	800	2	32	0.019798	Trace_WiB
Braun-Blanquet	4000	7	16	0.989203	Ball_Hall
	4000	4	16	0.531224	Calinski_Harabasz
	200	4	16	0.344495	SD_Dis
	2300	2	16	0.189199	Davies_Bouldin
	400	2	16	0.077443	Trace_WiB
	100	2	16	0.072107	S_Dbw
Cosine	4000	6	16	0.945626	Ball_Hall
	3600	3	32	0.346439	Trace_WiB
	300	8	16	0.297692	SD_Dis
	400	2	16	0.134757	S_Dbw
	4000	2	16	0.116275	Davies_Bouldin
	4000	2	32	0.044579	Calinski_Harabasz
Euclidean	2300	8	16	0.233399	Calinski_Harabasz
	1900	8	16	0.211938	SD_Dis
	800	7	16	0.175221	Trace_WiB
	300	3	64	0.027146	Ball_Hall
	3400	3	16	0.012933	S_Dbw
	100	2	32	0.000000	Davies_Bouldin

(b) DNS Protocol

Distance	Sample	*n*-gram	Message	adj. Rand	Internal Measure
Jaccard	100	5	16	1.000000	SD_Dis
	100	4	64	0.992056	Trace_WiB
	1600	3	16	0.180173	Ball_Hall
	1600	5	32	0.002438	Calinski_Harabasz
	1600	3	32	0.002438	S_Dbw
	100	2	64	0.000000	Davies_Bouldin
Dice	100	5	16	1.000000	SD_Dis
	100	4	64	0.710890	Trace_WiB
	1600	3	16	0.180173	Ball_Hall
	1600	5	32	0.002438	Calinski_Harabasz
	1600	3	32	0.002438	S_Dbw
	100	2	64	0.000000	Davies_Bouldin
Braun-Blanquet	100	5	16	1.000000	SD_Dis
	1600	4	32	0.747453	Ball_Hall
	100	4	64	0.710890	Trace_WiB
	1600	2	16	0.002438	Calinski_Harabasz
	1600	2	16	0.002438	S_Dbw
	100	2	64	0.000000	Davies_Bouldin
Cosine	100	5	16	0.271637	SD_Dis
	200	8	16	0.158804	Calinski_Harabasz
	200	8	16	0.158804	Davies_Bouldin
	200	8	16	0.158804	S_Dbw
	300	5	16	0.047732	Ball_Hall
	1000	4	16	0.003836	Trace_WiB
Euclidean	1600	7	64	0.000000	Ball_Hall
	400	2	16	0.000000	Calinski_Harabasz
	100	2	16	0.000000	Davies_Bouldin
	700	2	16	0.000000	SD_Dis
	1200	7	16	0.000000	S_Dbw
	200	2	16	0.000000	Trace_WiB

(c) SMB Protocol

Distance	Sample	*n*-gram	Message	adj. Rand	Internal Measure
Jaccard	1000	5	16	0.941975	Calinski_Harabasz
	900	5	16	0.941237	S_Dbw
	1100	4	16	0.924386	Ball_Hall
	100	5	16	0.821678	SD_Dis
	1100	5	32	0.448247	Trace_WiB
	700	3	64	0.335951	Davies_Bouldin
Dice	1000	5	16	0.941975	Calinski_Harabasz
	900	5	16	0.941237	S_Dbw
	1100	4	16	0.924386	Ball_Hall
	100	5	16	0.821678	SD_Dis
	700	3	64	0.471805	Davies_Bouldin
	100	2	64	0.238019	Trace_WiB
Braun-Blanquet	1000	5	16	0.942082	Calinski_Harabasz
	1100	4	16	0.902149	Ball_Hall
	100	5	16	0.821678	SD_Dis
	100	2	16	0.764644	S_Dbw
	700	3	64	0.453987	Davies_Bouldin
	100	4	64	0.403513	Trace_WiB
Cosine	800	4	16	0.465563	Calinski_Harabasz
	100	3	16	0.431648	Ball_Hall
	100	3	16	0.431648	Davies_Bouldin
	200	3	16	0.362845	S_Dbw
	100	5	16	0.355834	SD_Dis
	100	2	32	0.192357	Trace_WiB
Euclidean	1000	5	16	0.311167	Calinski_Harabasz
	700	3	16	0.274333	S_Dbw
	100	3	16	0.145950	Ball_Hall
	100	5	32	0.030939	SD_Dis
	100	2	64	0.000000	Davies_Bouldin
	100	4	64	0.000000	Trace_WiB

(d) HTTP Protocol

Table 5.5 Performance of internal validation measures in predicting optimal variable configuration for clustering.

There are many elements that can affect the performance of internal validation measures such as the clustering algorithm, structure of the data, and the statistic used by the internal measure to obtain the optimal number of clusters. The clustering algorithm used in this experiment is the Agglomerative Hierarchical Clustering (AHC) for the reasons explained in Section 5.2.2, therefore we restrict our analysis in this section on some of the observations related to the datasets, and the internal measures used in this experiment. To understand how the datasets affect the performance of the internal measures, it is important that we examine these datasets in terms of their internal shape and structure. Also, in order to carry out this analysis, we need a baseline for some of the factors (i.e., *n*-gram's length and message's length). We use the settings described in Table 5.4 for these factors as fixed baseline for this analysis.

Clustering Tendency. A preliminary assessment of the datasets can be done by assessing the *clustering tendency* [56]. Clustering tendency is used to explore whether datasets contain any clusters. We assessed the clustering tendency of the datasets using the *Hopkins* statistic [145]. This statistic assesses the clustering tendency by measuring the probability that a given data sample is generated by a uniform data distribution, i.e., it tests whether data can be clustered by checking its spatial randomness. The Hopkins test is stated in terms of the internal criterion of the data and no external information is required into the analysis. If the data were uniformly distributed, then the statistic should be about 0.5. However, if distinct clusters are present, the value should be almost to 1.0 [145]. In a previous study on random datasets, clustered datasets, and regularly spaced datasets, the Hopkins statistic has a value around 0.5, $0.7-0.99$, and $0.01-0.3$ respectively [146]. To calculate the Hopkins statistics for the datasets, the generated pattern matrix (*n*-gram frequency) for each protocol is normalised (as described in Chapter 4, Section 4.2.2) and used to calculate the statistic. Also, we have used suitable lengths for the *n*-grams and protocol messages (described in Table 5.4) to create the pattern matrix. The Hopkins statistic for the TFTP, DNS, SMB, and HTTP

samples are: 0.954, 0.781, 0.827 and 0.839 respectively, which indicates that the protocol samples (at the chosen baseline factor configurations) are highly amenable to clustering.

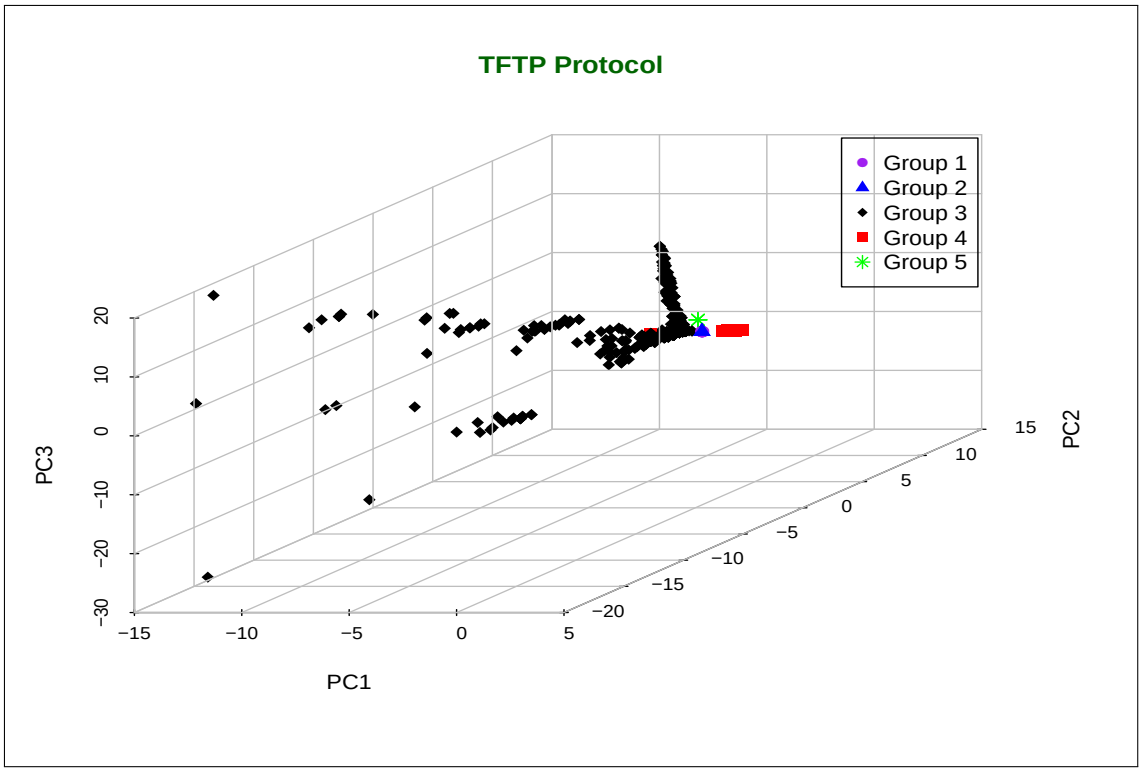
Visual Assessment. Although the Hopkins statistic shows that all data samples are highly "clusterable", unfortunately the statistic does not provide details for these clusters in terms of their size, shape, number of clusters etc. To understand more about the selected datasets, we need to visualise them. Because the data is multivariate (i.e., messages are described by more than one *n*-gram as explained in Chapter 4, Section 4.2.2), we need to reduce the dimensionality in order to visualise it. This is can be done using the *Principal Component Analysis* (PCA) algorithm [54]. As explained in Chapter 2, Section 2.2, PCA transforms the original variables (*n*-grams) to a new set of variables known as the principal components (PCs), such that the retention of variation present in the original variables decreases as we move down in the order, i.e., the first principal component retains the maximum variation that was present in the original components, and the second PC retains less variation than the first PC, and third PC retains less variation that the second PC, and so forth. To visually assess our samples using PCA, the normalised pattern matrix for each protocol is projected using the first three principal components. Also, the *ground truth* labels are used to highlight the actual groups within these samples.

As shown in Figure 5.8 (a-d), generally, all scatter plots confirm that there is a clustering structure in each dataset. However, the plots also show that the samples tend to exhibit different types of arbitrary shapes (e.g., non-spherical), sparse and occasionally overlapped clusters. Arbitrary shaped clusters are difficult to discover by traditional clustering algorithms (e.g., K-Means algorithm [147]) as well as some internal measures. In many cases this is because these algorithms are often operate on certain properties of the data that cannot be easily discovered, such as detecting cluster centroids as discussed in [148]. Additionally, the plots show that all protocol samples contain sub-clusters [1], which is difficult for some

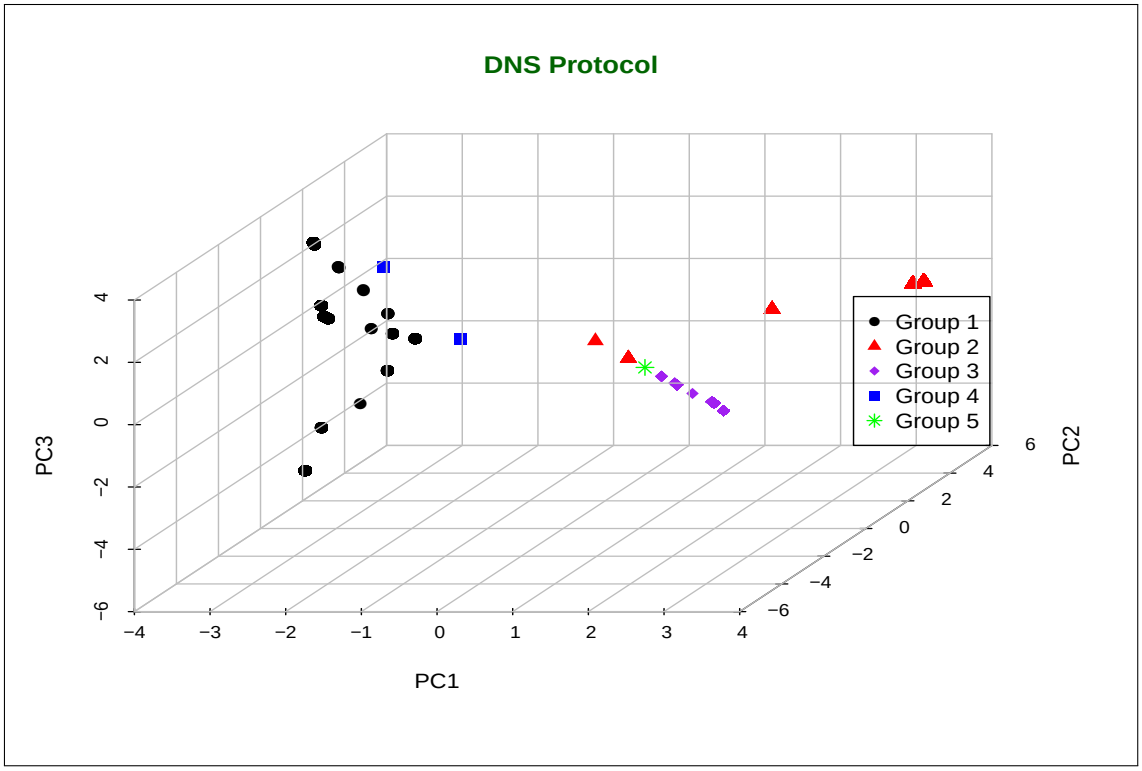
[1] Clusters that are close from each other and they can form one cluster.

internal internal measures to handle (e.g., DB, SD etc.) [61]. The scatter plots also indicate that these samples comprised of clusters with significantly various sizes. This is can be a factor in the performance of internal measures [61, 149]. For example, for the TFTP data (Figure 5.8(a)), this sample contains a large and arbitrary (v-shaped) cluster. The difference between the size of this cluster and other relatively smaller clusters is evident. Also, in the TFTP data, only four clusters are shown in the plot (out of five) due to the overlapping with another cluster. As for the DNS, all five clusters can be easily identified. However, one of these clusters is of an arbitrary shape (S-shape) as shown in Figure 5.8 (b). Also, similar members of one cluster are scattered apart (sub-clusters), which can be due to the presence of noisy *n*-grams in the pattern matrix. For the SMB data, only four groups of messages are shown (Figure 5.8 (c)) with two overlapping clusters. As for the HTTP, the sample contains several types of messages with relatively one large cluster. Similar to the TFTP, the cluster sizes in this sample seem to vary significantly form one another as shown in Figure 5.8 (d).

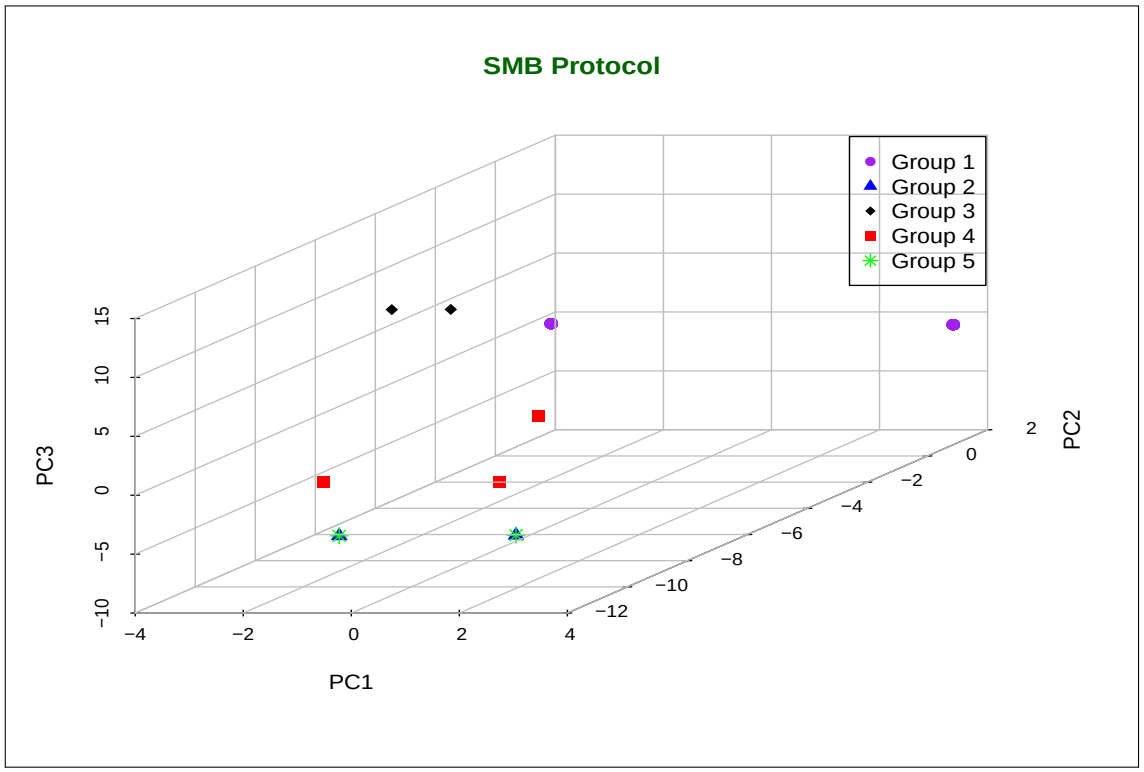
Internal Measures. As explained in Section 5.2.2, the chosen measures are based on different statistics and use different selection criteria to discover the optimal number of clusters. In this analysis, we divide internal measures into two types, one is compactness-based measures, and the other is compactness-and-separation based measures. The Ball-Hall index is a compactness-based measure while the rest of the measures are all based on both properties (compactness and separation). Also, the Ball-Hall and the Calinski-Harabasz measures are both based on the sum-of-square statistic which is used in these indices to measure the dispersion of the data items within the cluster as well as between the clusters respectively. As indicated in the Table 5.5, the performance of both measures varies significantly across the datasets. While the Ball-Hall index produced good results for all protocol samples, the Calinski-Harabasz index performed well only with regard to the HTTP sample as indicated in Table 5.5 (d).



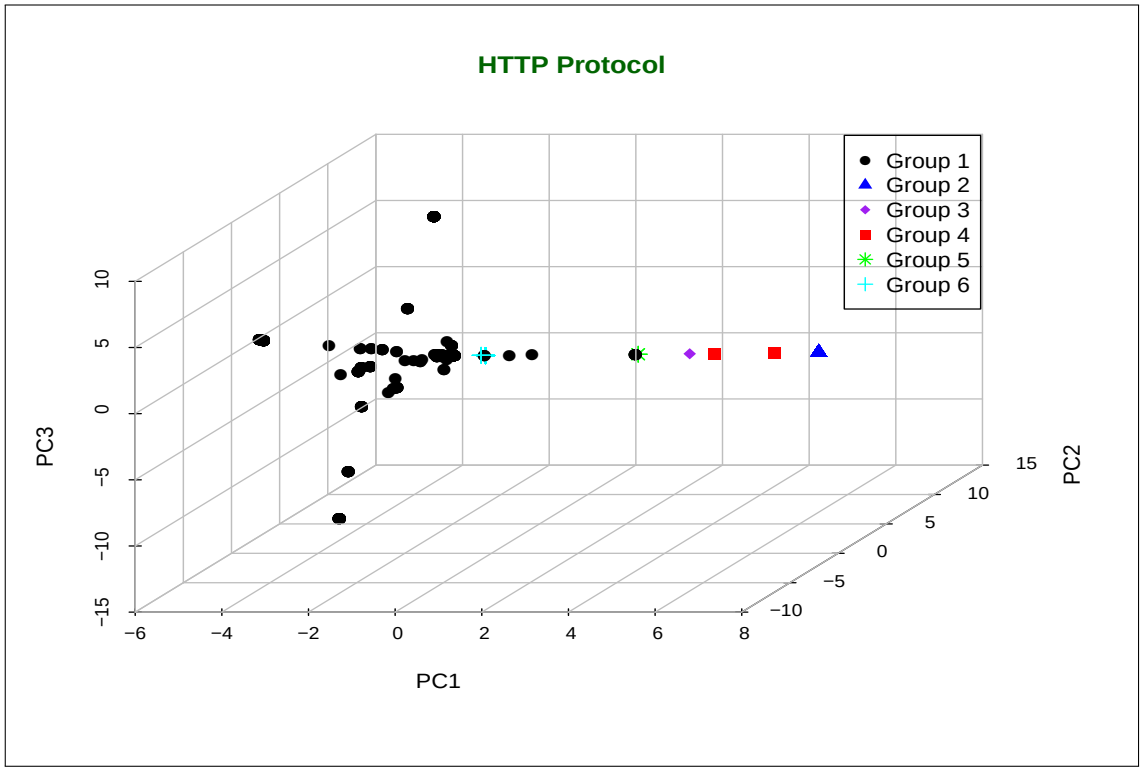
(a) TFTP Protocol



(b) DNS Protocol

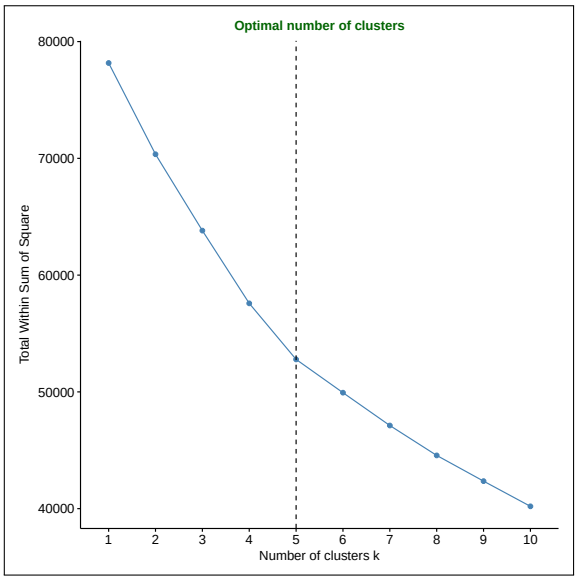


(a) SMB Protocol

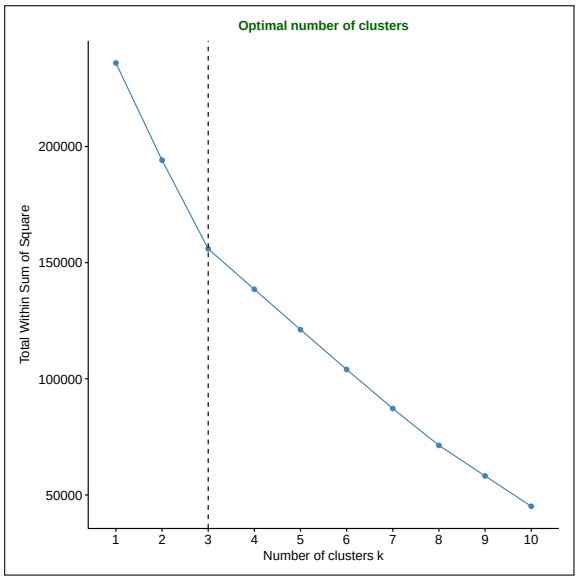


(b) HTTP Protocol

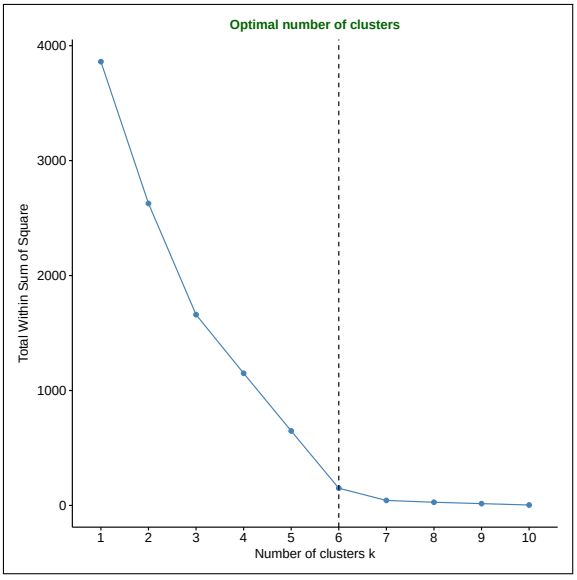
Figure 5.8 Three-dimensional projection of protocol samples on the first three principal components. The proportion of variance retained in the first three components for the TFTP, DNS, SMB and HTTP are: 31.0%, 43.0%, 72.2% and 28.8% respectively.



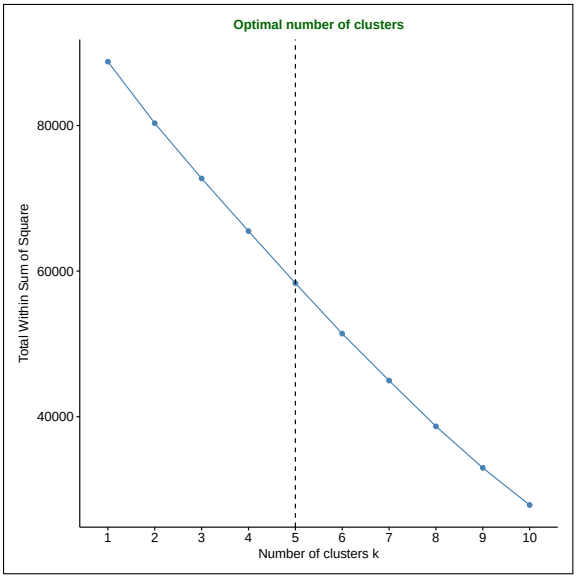
(a) TFTP Protocol



(b) DNS Protocol



(c) SMB Protocol



(d) HTTP Protocol

Figure 5.9 The total within sum of squares using the hierarchical clustering against the number of clusters. The so-called elbow method is used to select the optimal number of clusters for protocol samples.

Although the Ball-Hall index takes into account the within cluster variance only, the index produced better predictions than the Calinski-Harabasz measure, which considers both - compactness and separation properties. In the Ball-Hall index, we normalise the overall sum-of-square within the clusters (SSW) by the total number of clusters K ($\frac{WSS}{K}$) [142] while in the Calinski-Harabasz index the sum-of-square between clusters (SSB) is normalised by the within cluster sum-of-squares (SSW), which means that the factor of SSB has a more important effect in the ratio used in the index ($\frac{N-K\times SSB}{SSW\times K-1}$ where K is the number of clusters, and N is the number of data points) [142]. Therefore, the Calinski-Harabasz measure is mostly affected by the separation of the clusters and seeks to maximise this value. Because the selected samples are formed of arbitrary-shaped and largely scattered sub-clusters, the separation between clusters becomes difficult to determine especially among similar sub-clusters. This is particularly evident in the DNS and SMB datasets. In addition to the above observations, it seems that sample sizes can have an effect on how internal measures choose their best clustering point. For instance, measures which are based on the sum-of-square statistic (Ball-Hall and Calinski-Harabasz measures) tend to predict their best performance at large sample sizes (maximum size). On the other hand, other measures, such as the SD index tends to favour low sample sizes as shown in Table 5.5(a-d).

Figure 5.9 (a-b) shows the total sum-of-squares as a method to predict the optimal number of clusters for the protocol samples. The Ball-Hall measure uses this statistic and the elbow selection criteria to determine the optimal number of clusters as explained in Section 5.2.2. The elbow method looks at the difference of variance explained as a function of the number of clusters, i.e., we choose a number of clusters so that adding another cluster does not give much better modeling of the data. For the TFTP protocol (shown in Figure 5.9 (a)), it is clear that the best number of clusters is *five*. Similarly, for the DNS and SMB protocols (shown in Figure 5.9 (a-b)). However, for the HTTP, it is not clear as to which point is the best candidate due to the little difference in variance between these points as shown in Figure

5.9 (d). The Ball-Hall measure managed to retrieve a good clustering point for the HTTP protocol as indicated in Table 5.5 (d).

Distance Measures. As Table 5.5 (a-d) shows, internal measures tend to give significantly better predictions with the binary similarity measures of Braun-Blanquet, Dice, and Jaccard. The huge contrast of clustering scores between binary measures and the Euclidean measure is also shown in the box plots in Figure 5.6 (a-d). Specifically, it seems there is a close connection between the Braun-blanquet distance measure and the Ball-Hall index.

5.2.5 Performance of the AHC Algorithm

In this section we compare the performance of the Agglomerative Hierarchical Clustering (AHC) [56], against other clustering algorithms. In addition to the AHC algorithm, we have selected three other clustering algorithms: *K-Means* [147], Partitioning Around Medoids (PAM) [150], and Density-based Spacial Clustering of Applications with Noise algorithm (DBSCAN) [151]. The aim of this comparison is not to state one algorithm is better than the other. It is simply to determine whether other clustering algorithms can achieve similar (or better) clustering results than the AHC algorithm using the same datasets.

Methodology

In cluster analysis, according to Jain & Dubes [56], "*No generally accepted methodology for comparative analysis exists.*". However we use the following steps to evaluate the performance of each algorithm:

1. First, we configure each clustering algorithm with its "default" algorithmic parameters. For the Agglomerative Hierarchical Clustering, we use the settings shown in Table 5.2. For the K-Means and PAM algorithms, we use the default settings for both algorithms

in the original package [121, 152]. The DBSCAN algorithm requires two parameters: *epsilon* which determines how close points should be to each other to be considered a part of a cluster, and *minPts* which specifies how many neighbours a point should have to be included into a cluster. Although we have experimented with various settings for these parameters, the clustering scores didn't seem to be significantly affected. Therefore, we kept the common settings for the two parameters with $epsilon = 0.5$ and minPts=5 [153].

2. Using the best factor configurations for each protocol sample (described in Table 5.4), we set the number of clusters required by the algorithms to a range of numbers (e.g., from 1 to 10), each time we record the clustering score using the Adjusted Rand Index (ARI). Because DBSCAN does not depend on the number of clusters, we simply repeat using the default settings for the two parameters explained above .

3. Finally, we refer to the clustering score (ARI score) as well as the correspondent number of clusters (in comparison with the correct number of clusters) as a guide to assess the performance of each clustering algorithm. Using extrinsic measures to choose an optimal clustering algorithm on specific dataset is commonly used [61]. As discussed in Section 5.2.3, detecting the correct number of clusters alone cannot be used as an accurate measure for clustering because it does not indicate whether the clustering algorithm partitioned these clusters into homogeneous clusters (contain similar messages). However, detecting the number of clusters can be used in conjunction with the extrinsic measure to assess the overall performance of the algorithm.

Results

The results generally show that the Agglomerative Hierarchical Clustering (AHC) algorithm produced better clustering results than other clustering algorithms for all protocol samples.

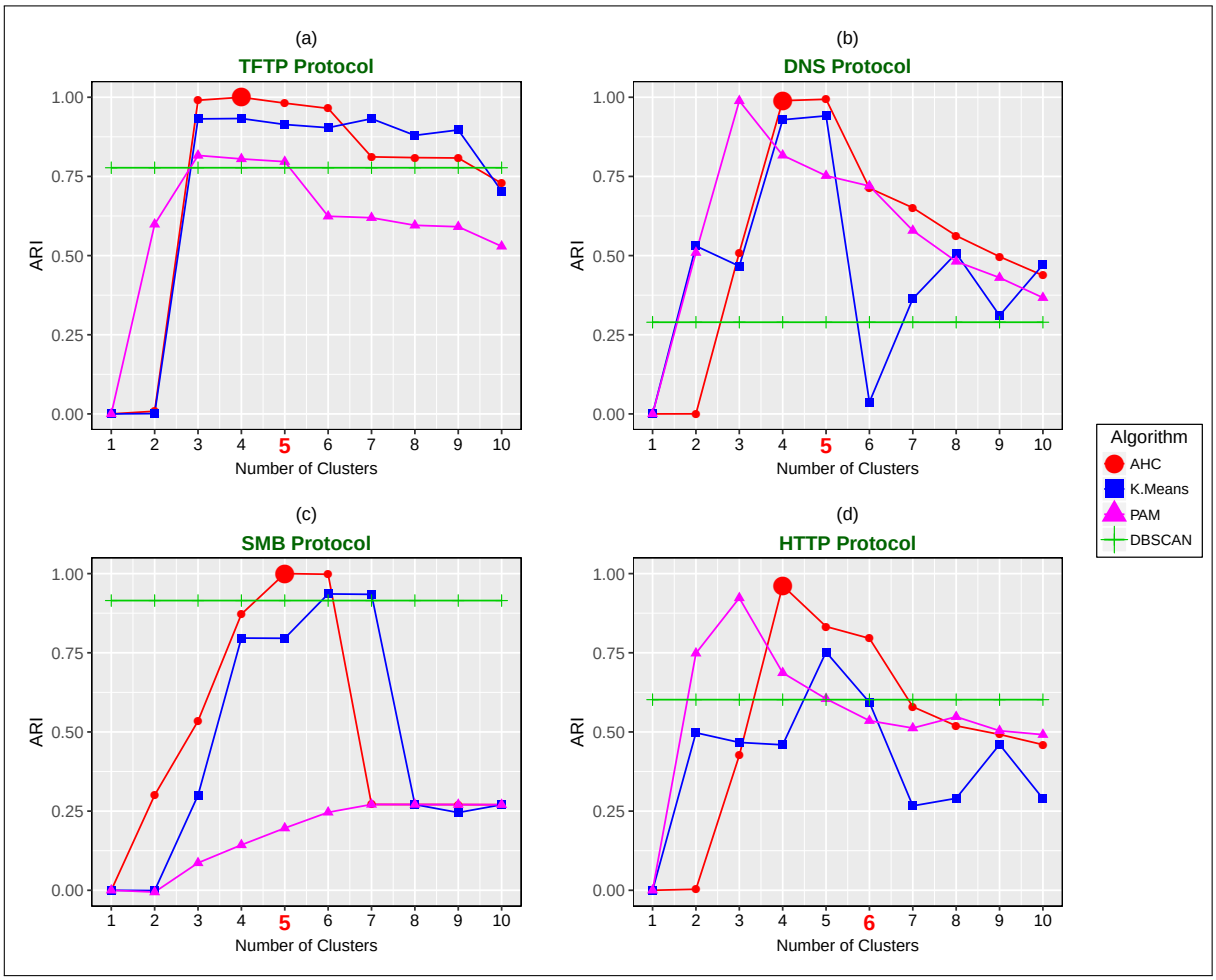


Figure 5.10 Performance of the Agglomerative Hierarchical Clustering (AHC) algorithm against other clustering algorithms. The best factor configurations extracted in RQ3 (shown in Table 5.4) are used as a baseline for the comparison. The performance is measured using the score of the Adjusted Rand Index (ARI) as well as the matching number of clusters. The actual number of clusters for each sample is marked in red in the horizontal axis, and the correspondent cutting point in the dendrogram (for the AHC) is indicated as a large red circle.

As shown in Figure 5.10 (a-d), none of the algorithms have managed to produce similar or higher clustering scores than the ACH algorithm. Furthermore, these clustering scores are matching a closer estimate of the actual number of clusters in the samples.

- **TFTP Protocol:** As shown in Figure 5.10 (a), the AHC has achieved the highest clustering score with 0.99982 and the correspondent cut-off point for the this score (shown as a large red circle) reveal that the number of clusters would be 4. On the other hand, K-Means managed to produce high clustering scores as well. However, the highest clustering score produced by K-Means is 0.932314 when the number of clusters is 7.

- **DNS Protocol:** The clustering score produced by AHC algorithm at the default cut-off point is 0.989203 with matching 4 number of clusters as illustrated in Figure 5.10 (b). However, it is clear that the best clustering score is 0.993632 which matches the correct number of clusters. PAM has managed to produce high clustering score (0.988661) only when the number of clusters is 3. The performance of the DBSCAN is significantly less for this sample. It seems the DBSCAN algorithm has trouble finding clusters of widely varying densities and with high dimensional data.

- **SMB Protocol:** The performance of the ACH algorithm for this sample is the best with a clustering score of 1.0, and matching the exact number of clusters in the sample. The clustering scores for K-Means and DBSCAN algorithms are also high with 0.914895 and 0.935816 respectively as shown in Figure 5.10 (c).

- **HTTP Protocol:** For the HTTP sample, the clustering score for the AHC is 0.962188 with 4 matching clusters as shown in Figure 5.10 (d). Although there are 6 clusters in the selected sample, the clustering score did not seem to be significantly affected by identifying only 4 clusters which implies that the majority of the messages fall

within the identified clusters. The performance of other algorithms is less with PAM at 0.923125, K-Means at 0.752302 and the DBSCAN algorithm is just above 0.6.

5.2.6 Threats to Validity

Although all experiments were tested on exactly the same machine and under the same experimental set-up, threats to *internal validity* might arise due to *intermediate variable* which might influence the cause-effect relationship between the independent and dependent variables. The experimental framework included the *n*-gram filtering as a necessary step to improve clustering, and avoid time complexities, however we cannot exclude that this filter could have been a *confounding factor*. We also list the most important threats to external validity, which might limit the generalisation of these findings to the protocols included in the experiment.

- **Representative Protocols:** Since we our study involved only four network protocols, they may not be representative of the entire family of network protocols. However, this threat is partially considered by selecting the possible types of network protocols (text & binary protocols).
- **Representative Traces:** Some of the collected network traces are relatively small in size and may not be representative of the protocol under study. The effect of some of the variables for the TFTP protocol vary from the rest of the protocols (DNS,SMB & HTTP), this is could be due to the fact that the gathered messages are not well trained to be representative of the protocol behaviour (lack of diversity of traffic seen in the trace).
- **Choice of the Response Variable:** As briefly mentioned in Section 5.2.2, initially we have experimented with four external validation measures: the *Rand Index* (RI),

Adjusted Rand Index (ARI), *Normalised Mutual Information* (NMI), and the *Fowlkes-Mallows Index* (FMI). However, we have chosen ARI after comparing the accuracy of each index on several data samples. The ARI and NMI external measures tend to produce relatively similar results, which are significantly more accurate than the RI and FMI. The performance of all external measures used in the pilot study is shown in the box plots attached to Appendix A. In this experiment, although four external validation measures have been investigated, however using other external measures may affect the results of the experiment.

- **Choice of the clustering algorithm:** The clustering algorithm applied through out the study is the agglomerative hierarchical clustering (AHC). The performance of the selected internal validation measures with regard to other clustering algorithms is still unknown. Therefore, we cannot state that these experimental results apply to other clustering algorithms as well.

5.3 Summary

In this chapter, we investigated the effect of four important variables on clustering accuracy and clustering time as part of reverse engineering protocols from network traces. We have quantified the effect of the sample size, length of the message, length of the n-gram, and choice of the distance measure on clustering accuracy, and clustering time. To support our investigation, we have extended the developed framework to produce arbitrary clustering configurations of protocol inferencing. We have used the framework on an experimental data sets from four widely used network protocols. Our experimental results indicate that:

- The choice of the distance measure and length of the message have the largest effect on clustering quality therefore, considered of paramount importance for clustering.

- The number of messages in the trace have negligible impact on clustering quality.
- The distance measure and sample size have the largest effect on clustering time.

The experimental results also show that it is possible to predict clustering configurations using internal validation measures. The results indicate that the *Ball-Hall* internal validity index along with the *Braun-Blanquet* similairty measure seem to give better results than other measures. Generally, certain intrinsic measures may perform better than others depending on the internal aspects of the data (e.g., shape, cluster density etc.), the criteria used for finding the right number of clusters, and the clustering algorithm. Therefore, we highly link the performance of the selected internal indices in our experiment to these aspects.

We have also discussed the threats that may affect the validity of our experiment, which (in general) can be mitigated by incorporating more diverse and balanced protocol samples.

Chapter 6

Segment-based Alignment and Message Structure Extraction

This chapter consists of two sections. The first section explains how we leverage segment-based alignment to align protocol messages. The second section explains how message structures are inferred from the aligned messages as well as a discussion on the expected quality of the inferred structures.

6.1 Segment-based Alignment

This thesis proposes the use of *segmental alignment* to work around the intrinsic limitations inherited by global alignment algorithms (e.g., Needleman Wunsch as discussed in Chapter 3, Section 3.2.2). Accordingly, this section reports on the necessary refinements that we had to apply. Also, a case study is used to demonstrate how the alignment approach works on a selected set of protocol messages.

In protocol reverse engineering, message alignment is often applied to reveal the structure of a protocol message. The alignment step is normally carried out after partitioning the

captured trace into homogeneous groups of messages as we discussed in Chapter 3, Section 3.1.1. Aligning messages of the same type reveals certain attributes of the underlying message structure (discussed in Chapter 2, Section 2.1.2), that is in terms of *static* and *dynamic* sections of the protocol message. Static sections within the aligned message indicate that these sections are protocol fields with fixed-length and they are either protocol *keywords* (reserved words that have special meaning to the protocol parser), or *delimiters* (field separators), or a hybrid of both. The dynamic sections, on the other hand, indicate that these sections represent protocol fields with variable length and their content tends to be volatile.

As explained in Chapter 2, Section 2.2.3, there are two types of alignments that can be applied to protocol messages: global and local alignments. Typically, global alignment algorithms (e.g., Needleman-Wunsch) are used when protocol messages are similar from the start to the end, and local alignment algorithms (e.g., Smith-Waterman [62]) are preferred when sequences share only isolated sections. The choice of which type to apply requires knowing (beforehand) the degree of similarity between the messages involved. When the dataset of the messages that we intend to align is small in size and short in length, then it is relatively easy to make a decision as to which type of alignment to choose. However, with large datasets and long sequences, it becomes very difficult to determine whether the sequences involved whether locally or globally related. Furthermore, traditional alignment methods rely on choosing an appropriate penalty for gaps introduced into the alignment which can significantly affect the alignment quality as discussed in Chapter 3, Section 3.2.

To work around the above problems, researchers in bioinformatics have developed segment-to-segment alignment approach which is an approach that we decided to apply within the protocol reverse engineering to overcome similar problems. Similar to gene sequences in bioinformatics, a protocol message is more than a sequence of arbitrary characters, the

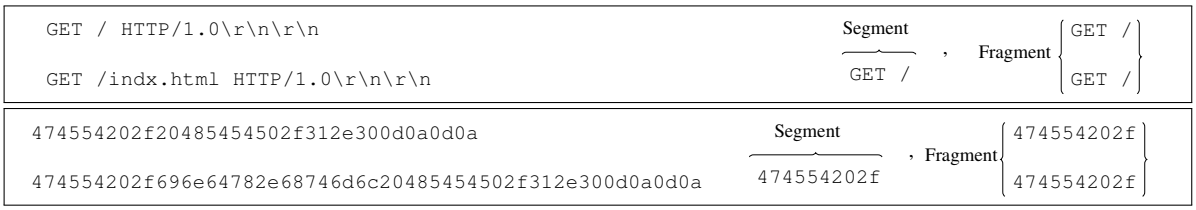


Figure 6.1 The concept of a segment and fragment applied on two basic HTTP messages. The figure shows two simple request messages using (GET method) encoded in both ASCII and hexadecimal formats, and a segment and a fragment of the same length extracted from both encoded messages (shown on the left)

.

occurrence of many parts of these messages normally follow a specific pattern, such as the length and order of these static and dynamic sections within the message.

A protocol message can be considered as sequence of static and non-static (dynamic) segments where static segments may be composed of keywords, delimiters etc. and dynamic segments represent variable content of protocol fields. A fragment is a pair of two segments from two different messages.

As we pointed out in Chapter 4, protocol messages are represented (re-encoded) as stream of hexadecimals that is to be able to print and align non-printable characters within text-based as well as binary-based protocols. Figure 6.1 shows a simple example of two HTTP messages in ASCII and their equivalent encoding in hexadecimal formats and the concept of a segment and fragment extracted from the pair of messages. Using the fragment concept, one may notice that a fragment may be larger than a protocol keyword (a combination of keywords), but it will not be smaller than one keyword, thus the fragment concept seems more abstract and closer to keyword-to-keyword comparison than byte-to-byte (character-by-character) comparison.

```
input  : Sequences, Prob_table // Fragment Probabilities
output : Aligned_Sequences
repeat
    for all pairs of sequences S1 and S2 do
        W ←ComputeWeights(S1,S2,Prob_table)
        L1 ←ComputePairwiseAlignment(W) /* Fragment Chaining */
        Sort(L1)
        foreach fragment ∈L1 do
            if consistent(fragment) then
                L2 ← Accept(fragment)
            end
        end
    end
until (no additional fragments found)
Insert gaps into sequences until selected fragments in L2 are matched
```

Algorithm 2. The applied procedure of segment-based alignment.

6.1.1 Applying the Algorithm

As described in the Chapter 2, Section 2.2.4 in the segment-to-segment alignment, pairwise and multiple alignments are constructed from pairwise local sequence similarities called *fragments*. A fragment is defined as an un-gapped pair of equal-length segments from two of the input sequences. Then based on statistical considerations, the program assigns a weight score to each possible fragment and attempts to find a consistent list of fragments with maximum total score. For pairwise alignment, a chain of fragments with maximum score can be identified. For multiple sequence sets, all possible pairwise alignments are performed and fragments contained in these pairwise alignments are integrated greedily into a resulting multiple alignment. The applied alignment steps are shown in Algorithm 2.

Segment-based alignment involves two parts, a scoring model that defines the similarity (or distance) between the sequences, and an algorithm that employs the scoring model to find an optimal (or near optimal) alignment between the sequences. Often researchers tweak both aspects of the alignment algorithm to achieve the best possible alignment. While the scoring

model is tweaked towards improving the scoring scheme (e.g., by enhancing the scoring matrices), the alignment algorithm is often improved towards efficiency in terms of space and time [75, 76].

To use segment-based alignment, it is essential that we modify the fragment scoring scheme to suit our application domain. The fragment scoring is an independent step form the alignment algorithm and does not require the modification of the fragment-chaining procedure [39].

Computing Fragment Weights

As explained in Chapter 2, Section 2.2.4, the segment-based approach computes the optimal score of an alignment by assembling a consistent set of fragments that yields the maximum aggregate sum of these fragment weights. Fragment weights are defined as the negative logarithms of their probabilities as shown in Equation 2.3

Applying the scoring scheme that is built into the standard segment-based alignment implementation (Dialign [70]) is unlikely to produce meaningful alignment results to our application. Segment-based alignment is an approach originally developed to align biological sequences. As such, there are about 22 "proteinomic" amino-acids, and the configuration of the distance-matrix that is used to compute alignment scores tends to be tailored to those amino acids, using carefully constructed distance matrices, such as the BLOSUM 62 substitution matrix [154]. Those substitution matrices are constantly tweaked to improve the scoring scheme and the quality of the alignment.

The scoring scheme in segment-based alignment is an independent step of the alignment algorithm (fragment chaining) [72]. This allows us to carry out the necessary changes to the current scoring scheme and replace it with a suitable one without affecting the alignment procedure. From Chapter 2, Section 2.2.4, we understand that the scoring scheme in segment-based alignment involves three steps: i) define a similarity measure between individual characters within each fragment, this is established using a similarity matrix. ii) calculating

a probability estimates for fragments which is derived from random experiments, and iii) assigning a weight based on the fragment probabilities. We have refined the three steps as follows:

Scoring Matrix

In order to apply segment-based alignment to network packet sequences, it was necessary to introduce a new similarity matrix that encompasses the similarity scores of all characters that we expect to observe in the network stream.

Since we are adopting the convention of converting protocol messages into hexadecimal format, this means every byte in a packet can be represented as a hexadecimal pair. There are 256 possible characters expected to appear in the network stream (0x00-0xff). Because each character consists of two hexadecimal numbers, this also entails that protocol messages will be aligned on half-byte basis.

For the similarity matrix, we provide a straightforward identity matrix instead. The similarity matrix defines scores between 16 possible characters (0-f) where two characters obtain a score of 1 if they are identical, and a score of 0 otherwise as shown in Figure 6.2.

Re-computing Probability Estimates

The quality of segment-based alignment depends first and foremost on the way weights of fragments are obtained. Recall from Chapter 2, Section 2.2.4 that the similarity score for a fragment is used to compute a weighting for the fragment, based on the probability that a random fragment of the same length would produce the same score. This relies on the prior computation of a probability table.

We computed this probability table by running 10^7 random experiments for all combinations of fragment lengths (1-40) and possible scores. The probability table is automatically calculated using the procedure explained in Chapter 2, Section 2.2.4 using the similarity

	0	1	2	3	4	5	6	7	8	9	*a*	*b*	*c*	*d*	*e*	*f*
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 6.2 A identity matrix contains all possible hexadecimal numbers expected in the trace and their similarity scores. The assigned score is 1 for a match and 0 for a mismatch.

matrix shown in Figure 6.2. In our experience, 10^7 random experiments are sufficient to generate highly accurate probability estimates since we observed that the alignment quality does not change beyond this threshold.

Overlap Weights

As explained in Chapter 2, Section 2.2.4, overlap weights are used to improve the alignment by placing more emphasis on motifs occurring in more than two sequences. The overlap weight reflects the fragment weight as well as the degree of overlap with other fragments.

Calculating the overlap weight for fragments is time-consuming step. For this reason, it is originally employed as an optional step aimed to improve the alignment quality when the number of sequences is relatively small (less than 35 sequences). However, with large datasets, this option can significantly increase the alignment time several times. Because our

	Parameter	Set Value
01	Maximum Iteration Number	03
02	Maximum Fragment Length	40
03	Minimum Fragment Length	02
04	Fragment Weight Threshold	0
05	Probability Threshold	10^{-9}

Table 6.1 Internal parameters of segment-based alignment.

protocol datasets normally involve large number of messages, this feature in the algorithm had to be deactivated.

Internal Parameters

In this subsection we discuss the values selected for a number of internal algorithmic parameters. These parameters are not user-dependent and tuned according to the best performance observed for a number of protocols as shown in Table 6.1.

- **Maximum Number of Iterations:** The maximum iteration parameter determines the number of times the multiple alignment procedure needs to be repeated to align (and re-align) all input sequences. Typically, the value for this parameter ranges from 1 to 3 [70] where 1 is the minimum number of iterations and 3 is the maximum number of iterations (that may be required) to identify all fragments and complete an alignment. The alignment may not require the 3 iterations, however, we set this parameter to 3 to achieve the best possible alignment.

- **Minimum/Maximum Fragment Length:** The maximum length for a fragment defines the maximum number of characters that can be included in a single fragment while the minimum length for a fragment is the minimum number of characters that a fragment should contain.

  Generally, the fragment-chaining procedure tends to compose alignments based on their weights not lengths. It makes no difference if we include a long fragment alignment

or if we split up the long fragment into several smaller ones, and then include all the fragments into the alignment - the resulting alignment is the same.

The effect of these parameters tends to depend on the weighting scheme. For example, if the fragment-chaining procedure tends to favour fragments from many shorter fragments than from a few longer ones, this indicates that the weighting scheme does not consider the length of the fragment in the weight calculation which is the case in the first version of Dialign-1 [39].

Because recent weighting schemes take into consideration the length of the fragment which gives relatively higher weights to fragments of significant similarity, changing the length of the maximum length of fragment did not seem to have any effect on the alignment quality. We set the maximum length for a fragment to *40* characters (as originally used in Dialign-2 [70]) and the minimum length to *2* characters. This is to make sure that the smallest fragment is equal to a byte's length (every pair of hexadecimals represent a byte).

- **Fragment Weight Threshold:** The selected set of fragments for the pairwise alignment can further be reduced to include only fragments that have a weight above certain threshold. Using a positive threshold to excluding fragments of lower significance and include only high scoring fragments improves alignment and reduces the alignment time. However, selecting a generic value that suits all sizes of samples is a difficult question. The default value for this parameter is to 0 (no fragments are excluded).

- **Probability Threshold:** As discussed in the previous chapter, this internal parameter is part of the Equation 2.2 and used to determine how the probability of the fragment is calculated. Although lower values (e.g., 10^{-8}) did not seem to have visible effect on the quality of alignment, this parameter is kept to 10^{-9}, as it was originally set in Dialign-2.

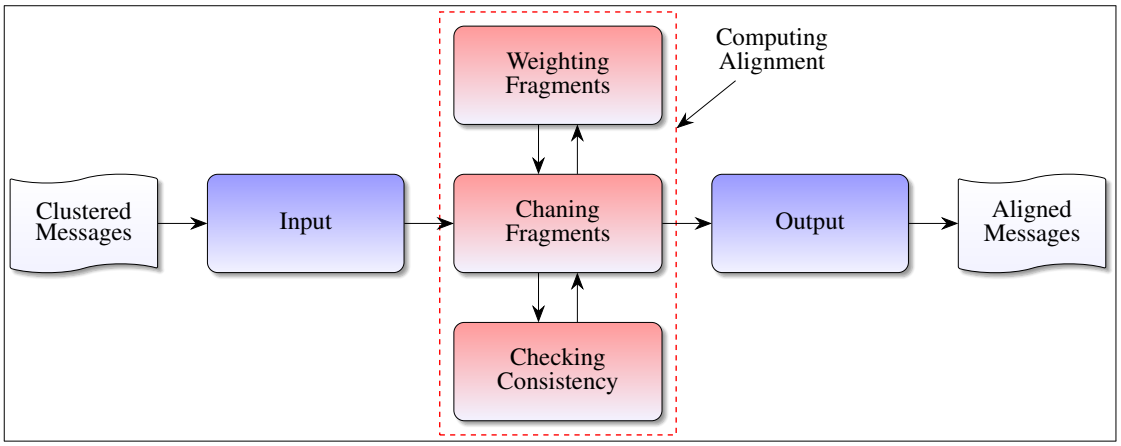


Figure 6.3 Architecture of the segment-based alignment tool (a reduced version of Dialign-2 [3]).

6.1.2 Re-engineering Dialign

As discussed in the previous section, segment-based alignment has been implemented in several projects. Dialign was first introduced in the *1996* and made available as a free software under the terms of the Free Software Foundation (GNU General Public License). The program has been written in *C* and has been revised over a number of publications where each version is aimed to introduce more features and improve the alignment procedure. Dialign is available on-line at the Gottingen Bioinformatics Computer Server (GOBICS) at [3].

The Dialign program provides a suitable platform for our own segment-based alignment step in the framework (discussed in Chapter 4) without re-writing the entire alignment procedure. However, Dialign is a mature program that is heavily tailored to bioinformatics with several functionalities and features specifically tuned to compute alignments of protein and nucleic sequences. Dialign-2.2 consists of 5000 lines of C code, nearly half of this code is dedicated to support features that are not essential part of the alignment algorithm.

For this reason, we re-engineer Dialign into a slim version that supports only the essential steps outlined in Algorithm 2. In re-engineering Dialign, while we have maintained the *direct*

Input Protocol Messages	Output Aligned Protocol Messages
> MSG-1 474554202f20485454502f312e300d0a0d0a > MSG-2 474554202f696e64782e68746d6c20485454502f312e300d0a0d0a > MSG-3 474554202f6b2e69636f20485454502f312e300d0a0d0a > MSG-4 474554202f696d672e706e6720485454502f312e300d0a0d0a > MSG-5 474554202f73742e63737320485454502f312e300d0a0d0a	> MSG-1 474554202f----------20485454502f312e300d0a0d0a > MSG-2 474554202f696e64782e68746d6c20485454502f312e300d0a0d0a > MSG-3 474554202f----6b--2e--69636f20485454502f312e300d0a0d0a > MSG-4 474554202f696d67--2e--706e6720485454502f312e300d0a0d0a > MSG-5 474554202f----73742e--63737320485454502f312e300d0a0d0a

Figure 6.4 Input/Output protocol messages in a FASTA file format.

greedy alignment approach implemented in the early versions of *Dialin-1* and *Dialign-2.2*[1], we have also integrated the latest fragment weighting formula introduced in *Dialign-T* and *Dialign-TX* as well as the modifications we discussed in the previous section to the weighting function.

The reduced version of the alignment module consists of about 2700 lines of *C* code. The main components of the implementation consists of: the *Input* module, the *Fragment Weighting* module, the *Fragment Chaining* module, the *Consistency Checking* module, and the *Output* module, as shown in Figure 6.3.

The Input module is responsible for reading protocol messages in a *FASTA* file format. A FASTA file is a simple text-based format where each message is preceded with its name and the sequence name should start with > character. Each message name and message body should be separated by a new line character as indicated in Figure 6.4. Also, the input module is hosting the scoring matrix described in Figure 6.2 as well as reading the re-computed probability estimates for the fragments as described in the previous section.

The pairwise alignment is computed using the fragment chaining module which is responsible of selecting optimal fragments by obtaining fragment scores from the fragment weighting module and checking the consistency of the positions of these fragments. The consistency

[1]Recent versions of Dialign (e.g., Dialign-TX) combines greedy as well as progressive approach to avoid spurious random similarities in fragments by considering the degree of similarity between the two sequences involved in the fragment [69].

Message	ASCII Encoding	Length
MSG_1	`GET / HTTP/1.0\r\n\r\n`	22
MSG_2	`GET /indx.html HTTP/1.0\r\n\r\n`	31
MSG_3	`GET /k.ico HTTP/1.0\r\n\r\n`	27
MSG_4	`GET /img.png HTTP/1.0\r\n\r\n`	29
MSG_5	`GET /st.css HTTP/1.0\r\n\r\n`	28
Message	Hexadicemal Encoding	Length
MSG_1	`474554202f20485454502f312e300d0a0d0a`	36
MSG_2	`474554202f696e64782e68746d6c20485454502f312e300d0a0d0a`	54
MSG_3	`474554202f6b2e69636f20485454502f312e300d0a0d0a`	46
MSG_4	`474554202f696d672e706e6720485454502f312e300d0a0d0a`	50
MSG_5	`474554202f73742e6373732048545450 2f312e300d0a0d0a`	48

Table 6.2 A set of five basic messages messages of the HTTP protocol in the text format (ASCII) and their equivalent Hexadecimal format .

checking procedure is carried out using the GABIOS-LIB module [80]. GABIOS-LIB is a library written in *C* and has been integrated into recent versions of Dialign project (Dialign-2 and above) to carry out "speedy" consistency tests during the multiple alignment procedure. The output module is employed to align the input sequences based on the positions of the selected fragments and inserting gaps (if necessary) into the sequences. This module is also used to print the aligned sequences in the FASTA file format as shown in Figure 6.4.

6.1.3 A Case Study

In this section we show how we use our segment-based alignment tool to perform pairwise as well as multiple alignment of protocol messages. The alignment tool has been tuned to the default values of the parameters shown in Table 6.1. In this example, we will refer back to the HTTP example used in the Chapter 3 (Section 3.1.1) and shown (again) in Table 6.2. For the pairwise alignment, we use the first and the second messages (MSG_1 & MSG_2) listed in the table, and for the multiple alignment, we use all five messages. In both cases (pairwise and multiple alignment), prior to alignment, the messages are encoded into hexadecimal format which is part of the *message preprocessing* step in the framework (see Chapter 4).

Pairwise Alignment

We have used the segment-based alignment tool to align MSG_1 and MSG_2. To align these two messages, the messages have to be in hexadecimal format as shown in Figure 6.5 (a), then the tool has identified two consistent fragments (F_1 & F_2) that produce an optimal pairwise alignment as illustrated in 6.5 (b). The overall procedure of pairwise alignment requires only one iteration.

According to the assigned weights, fragment F_2 needs to be inserted first into the final alignment list followed by F_1. Because both fragments are consistent, both fragments are used in the final alignment. In the last step, based on the coordinates of both fragments, the tool re-arranges the positions of two fragments within both messages, then inserts the gaps (in MSG_1) between F_1 and F_2 as indicated in Figure 6.5 (d). Sections in MSG_1 and MSG_2 which are not part of F_1 and F_2 are not considered aligned.

Multiple Alignment

For multiple alignment, we have added the three more messages to the pairs of messages selected in the previous example to become a set of five HTTP messages as illustrated in Figure 6.2. Similar to pairwise alignment, for each pairwise comparison, the tool needs to identify a consisted collection of fragments with maximum sum of weights.

The tool initially has identified a set of fragments that produce optimal pairwise alignments (from all pairwise comparisons between messages) which is *29* fragments, as shown in Figure 6.6 (b). Fragments that produce optimal pairwise alignments from aligning messages MSG_1 and MSG_2 are F_1 and F_2, and from aligning MSG_1 and MSG_3 are F_3 and F_4, and from aligning MSG_1 and MSG_4 are F_5 and F_6, and so forth. Seven of the "optimal" fragments are identified during the second iteration step. Also, out of the 29 fragments, there are four inconsistent fragments as shown in Figure 6.6 (b), which they had to to be rejected.

```
> MSG-1
474554202f20485454502f312e300d0a0d0a
> MSG-2
474554202f696e64782e68746d6c20485454502f312e300d0a0d0a
```

(a) A pair of HTTP messages in hexadecimal format.

Fragment	Messages	Start (Msg-1)	Start (Msg-2)	Length	Weight	Iteration	Status
F_1	1-2	1	1	10	28.61	1	cons
F_2	1-2	11	29	26	67.98	1	cons

(b) Optimal fragments obtained from pairwise alignment.

```
> MSG-1
474554202f------------------20485454502f312e300d0a0d0a
> MSG-2
474554202f696e64782e68746d6c20485454502f312e300d0a0d0a
```

(c) Resulting Alignment

Figure 6.5 Pairwise Alignment of two basic HTTP messages (in Hexadecimal) as constructed by our segment-based alignment tool. The tool has selected two fragments for the final alignment. Details of the selected fragments include: their start positions on both messages, fragment length, and the assigned weights to each fragment. The details also show on which iteration the fragment is identified.

```
> MSG_1
474554202f20485454502f312e300d0a0d0a
> MSG_2
474554202f696e64782e68746d6c20485454502f312e300d0a0d0a
> MSG_3
474554202f6b2e69636f20485454502f312e300d0a0d0a
> MSG_4
474554202f696d672e706e6720485454502f312e300d0a0d0a
> MSG_5
474554202f73742e63737320485454502f312e300d0a0d0a
```

(a) HTTP messages in hexadecimal format.

Fragment	Messages	Start (Msg-1)	Start (Msg-2)	Length	Weight	Iteration	Status
F_1	1-2	11	29	26	67.98	1	cons
F_2	1-2	01	01	10	28.61	1	cons
F_3	1-3	11	21	26	68.14	1	cons
F_4	1-3	01	01	10	28.77	1	cons
F_5	1-4	11	25	26	68.06	1	cons
F_6	1-4	01	01	10	28.69	1	cons
F_7	1-5	11	23	26	68.10	1	cons
F_8	1-5	01	01	10	28.73	1	cons
F_9	2-3	25	01	30	76.45	1	cons
F_{10}	2-3	17	01	16	40.88	1	incons
F_{11}	2-4	23	19	32	80.75	1	cons
F_{12}	2-4	19	17	02	1.67	1	cons
F_{13}	2-4	01	01	16	43.61	1	cons
F_{14}	2-5	29	23	26	67.69	1	cons
F_{15}	2-5	17	13	10	18.51	1	incons
F_{16}	2-5	01	01	12	29.51	1	incons
F_{17}	3-4	11	15	36	89.74	1	cons
F_{18}	3-4	01	01	10	28.44	1	cons
F_{19}	3-5	13	15	34	85.36	1	cons
F_{20}	3-5	01	01	12	29.67	1	cons
F_{21}	4-5	25	23	26	67.77	1	cons
F_{22}	4-5	10	10	13	20.75	1	incons
F_{23}	4-5	01	01	09	24.73	1	cons
F_{24}	2-3	19	13	02	1.74	2	cons
F_{25}	2-3	15	11	02	0.12	2	cons
F_{26}	2-3	01	01	10	28.37	2	cons
F_{27}	2-5	17	13	04	6.44	2	cons
F_{28}	2-5	01	01	10	28.32	2	cons
F_{29}	4-5	17	15	08	11.38	2	cons

(b) Optimal fragments obtained from pairwise alignments.

```
> MSG-1
474554202f------------------20485454502f312e300d0a0d0a
> MSG-2
474554202f696e64782e68746d6c20485454502f312e300d0a0d0a
> MSG-3
474554202f----6b--2e--69636f20485454502f312e300d0a0d0a
> MSG-4
474554202f696d67--2e--706e6720485454502f312e300d0a0d0a
> MSG-5
474554202f----73742e--63737320485454502f312e300d0a0d0a
```

(c) Resulting Alignment

Figure 6.6 Segmented-to-segment alignment of five basic HTTP messages as constructed by our segment-based alignment tool. The tool has selected two fragments for the final alignment. Details of the selected fragments include: their start positions on both messages, fragment length, and the assigned weights to each fragment. The details also show on which iteration the fragment is identified.

Therefore, the final list of fragments considered for the multiple alignment consists of only 25 fragments.

In a final step, the tool constructs an alignment from the final list of the fragments and arrange these fragments according to their shared coordinates and gaps are inserted between the fragments (when necessary) as illustrated in Figure 6.6 (d). The entire alignment procedure required two iterations to be completed.

6.2 Extracting Message Structure

The previous section described how segment-based alignment can be used to align protocol messages. This section explains how the produced alignment is used to reveal the message structure. The first part of this section explains how the resulting alignment is *generalised* to generate a *message pattern* (or a pattern) that serves as the final description of the inferred message structure. In the second part of this section, we provide a discussion on the expected quality of the produced pattern.

The inference of the message structure is the last step in the framework discussed in Chapter 4. The input to this step is the aligned protocol messages, and the output is the detected message structure, i.e., for each aligned cluster of messages, the message extraction step aims to infer the final message structure by determining the invariant fields in the message. It is worth mentioning that some of the previous state-of-the-art projects (see Table 3.1 in Chapter 2) consider message alignment is the final step in the inference process where the inference of the message structure is left unexplained or unautomated [102, 23].

Aligned protocol messages consist of static and variable fields. To clearly capture the pattern of these fields across all aligned messages, the resulting alignment needs to be generalised using a suitable generalisation method (explained below). Through this general pattern we should be able to distinguish between fixed (consistent) and variable (inconsistent) blocks

Alignment	474554202f----------20485454502f312e300d0a0d0a 474554202f696e64782e68746d6c20485454502f312e300d0a0d0a 474554202f----6b--2e--69636f20485454502f312e300d0a0d0a 474554202f696d67--2e--706e6720485454502f312e300d0a0d0a 474554202f----73742e--637373220485454502f312e300d0a0d0a
Generalisation	********** ********** 474554292f----------20485454502f312e300d0a0d0a

Table 6.3 Alignment generalisation of multiple HTTP messages in the hexadecimal format.

across the aligned messages. Also, through this pattern we should be able to determines the position and order of these fixed and non-fixed blocks. Therefore, our intention in this step is to generalise the produced alignment into a single description known as a *message pattern* through the pattern extraction step as illustrated in the framework (see Figure 4.1).

In the context of network packets, aligned regions tend to correspond to one or more protocol fields. For each cluster of aligned messages, the messages are listed in aligned form (similar to the example alignments shown in Table 6.3. This means that every aligned position can be referred to in terms of a column (e.g. column 1 refers to the first character in every sequence etc.). Normally the generalisation step is applied on the aligned messages in the hexadecimal format. For example, the multiple alignment produced by the segment-based alignment in the case study shown in Figure 6.6 (c) can be generalised as shown in Table 6.3. However, to simplify the concept of the alignment generalisation, we will refer to the aligned HTTP messages in the ASCII format as illustrated in Table 6.4.

Generalising the Alignment

The generalisation step is a separate step from the alignment. As explained in Chapter 2, Section 2.2.3, alignment generalisation is normally obtained by generating what is known (in bioinformatics) as the *consensus sequence* [66]. The information provided by the consensus sequence is then used to sketch a rough description of the message structure, and (commonly)

Alignment	GET /--------- HTTP/1.0\r\n\r\n GET /indx.html HTTP/1.0\r\n\r\n GET /---k.ic-o HTTP/1.0\r\n\r\n GET /i-mg.-png HTTP/1.0\r\n\r\n GET /--st.-css HTTP/1.0\r\n\r\n
Generalisation	***** ********** GET /--------- HTTP/1.0\r\n\r\n

Table 6.4 Alignment generalisation of five HTTP messages in their ASCII format.

presented in a form of a regular expression or a mark up language (e.g., XML language) which defines the rules of the extracted structure in human and machine readable form.

Table 6.4 shows that information is lost using this basic method of inferring the consensus sequence. Some characters at certain positions may be less frequent, but can still play an important role in defining the message structure. Take for example the "`.`" character which is part of the message URI in the listed example. The `.` character separates the two variable portions of the URI (the file name and its extension). If we only consider the characters that occurred in all messages, the "`.`" character will not be part of the final pattern definition and this information will be lost as shown in the Table 6.4.

The above problem might extend to a whole field level (not just a character) where aligned protocol messages may contain *optional* fields where these fields may not be consistent in their occurrences across all messages. This leads to the question: how often these fields should occur to be considered representative and included in the inferred pattern?

Position Weight Matrix. To extract a message pattern, we generate a *Position Weight Matrix* (PWM), also known as *Position Specific Scoring Matrix* (PSSM). A position weight matrix (PWM) is commonly used method to represent motifs and *sequence logos* in biological sequences [155] to enhance the inference of the consensus sequence. Typically, a PWM consists of one row for each character of the alphabet, and has one column for each position in the alignment. A PWM is constructed by counting the occurrences of each character observed

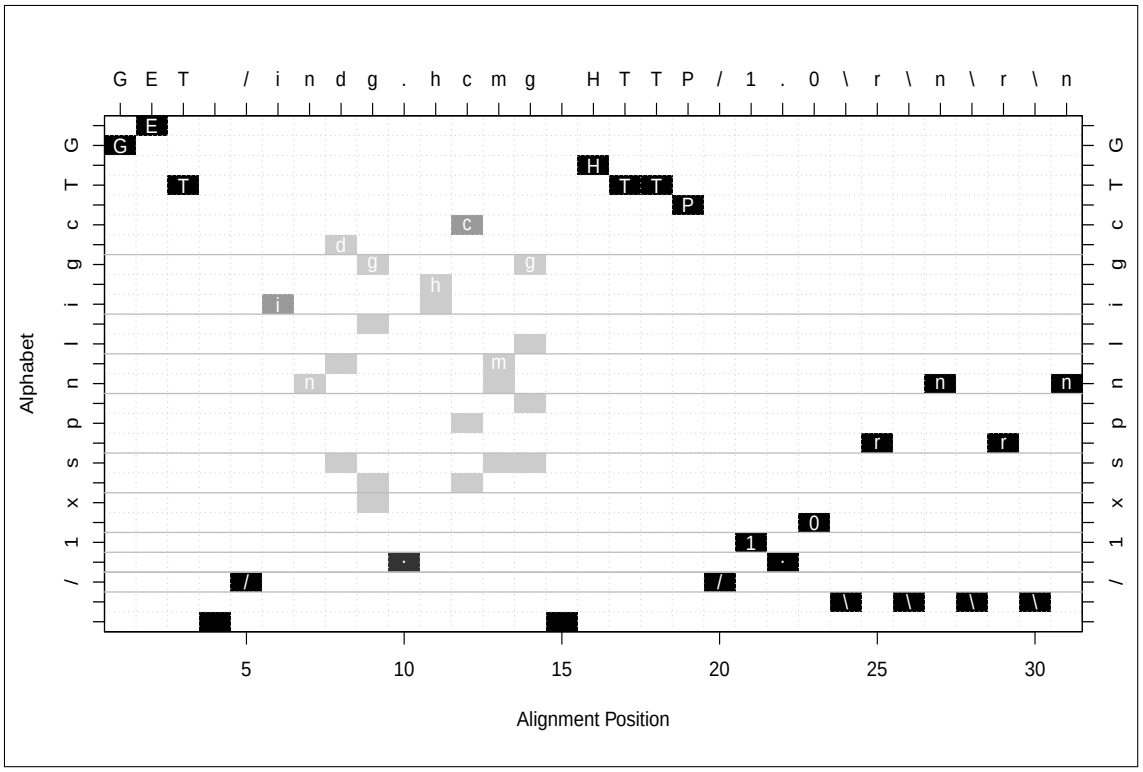
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
E	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	**0.4**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	**0.4**	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	**0.2**	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	**0.2**	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	0
s	0	0	0	0	0	0	0	**0.2**	0	0	0	0	**0.2**	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	**0.2**	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	**0.8**	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
/	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
\	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	**1**	0	**1**	0
	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.5 A Position Weight Matrix generated from the set of aligned HTTP messages introduced in the background (see Table 6.4). It shows on the top the number of columns (positions) and the alphabet as the matrix rows. To preserve space, the matrix only shows the used characters of the alphabet.

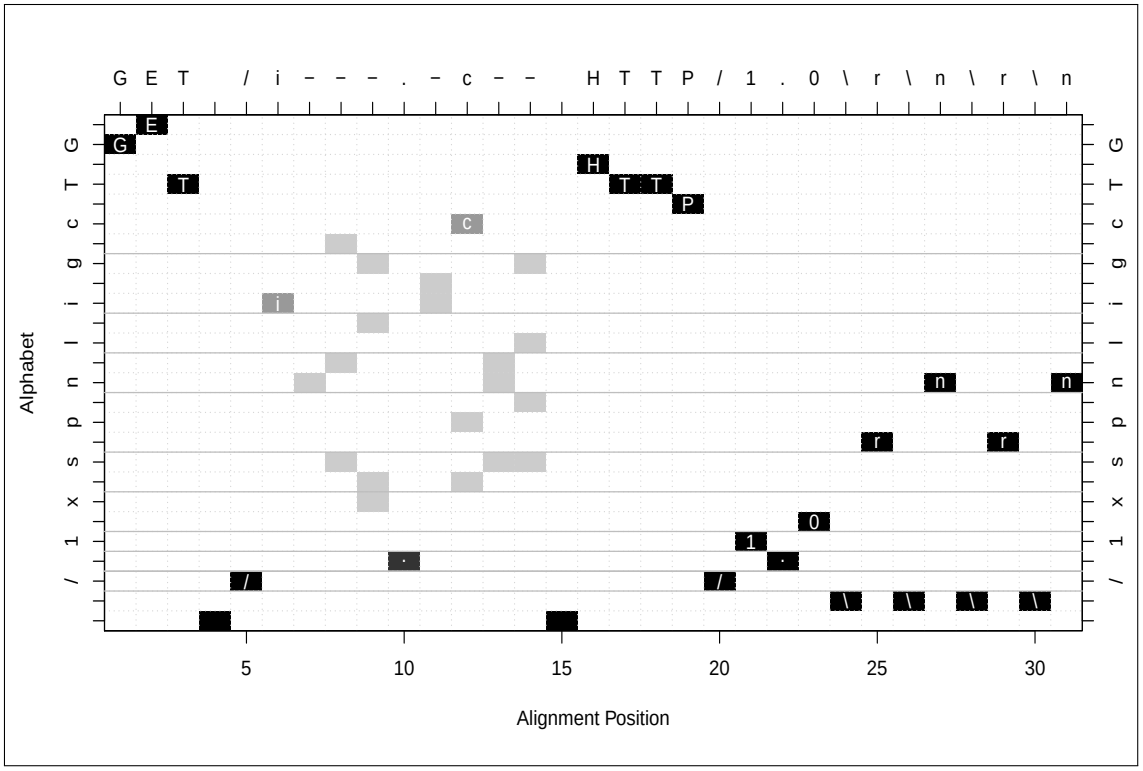
at each position (each coefficient in the matrix indicates the number of times that a given character occurred at a given position), we then normalise the frequency so that the occurrence rate of each character falls within the interval [0,1], where 0 indicates that particular character in that column (position) did not occur across all messages, and 1 indicates that character occurred in every single message at that particular position. The coefficients of the matrix can now be interpreted as probabilities of a given character occurring at a given position.

As an example, Table 6.5 shows the PWM that corresponds to the set of sequences that was shown in Table 6.4. Here, the character "`G`" has been observed across all aligned messages in position 1, therefore, assigned weight of 1, while the character "`i`" observed only twice at position 6, thus given the weight 0.4.

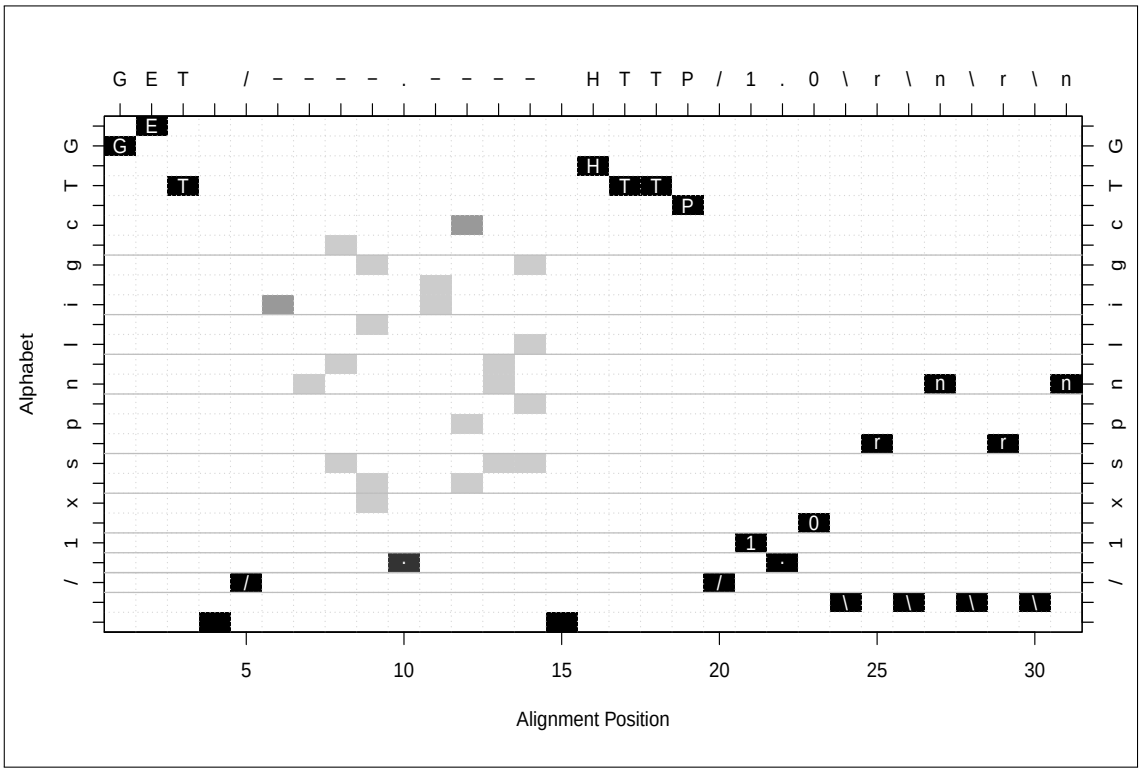
Generalisation Level. As discussed in the previous section, sometimes, we are only interested in the scores over a given threshold. After we generate the PWM matrix, we then


G E T / i n d g . h c m g H T T P / 1 . 0 \ r \ n \ r \ n
Alphabet
/ 1 x s p n l i g c T G
5
10
15
20
25
30
Alignment Position

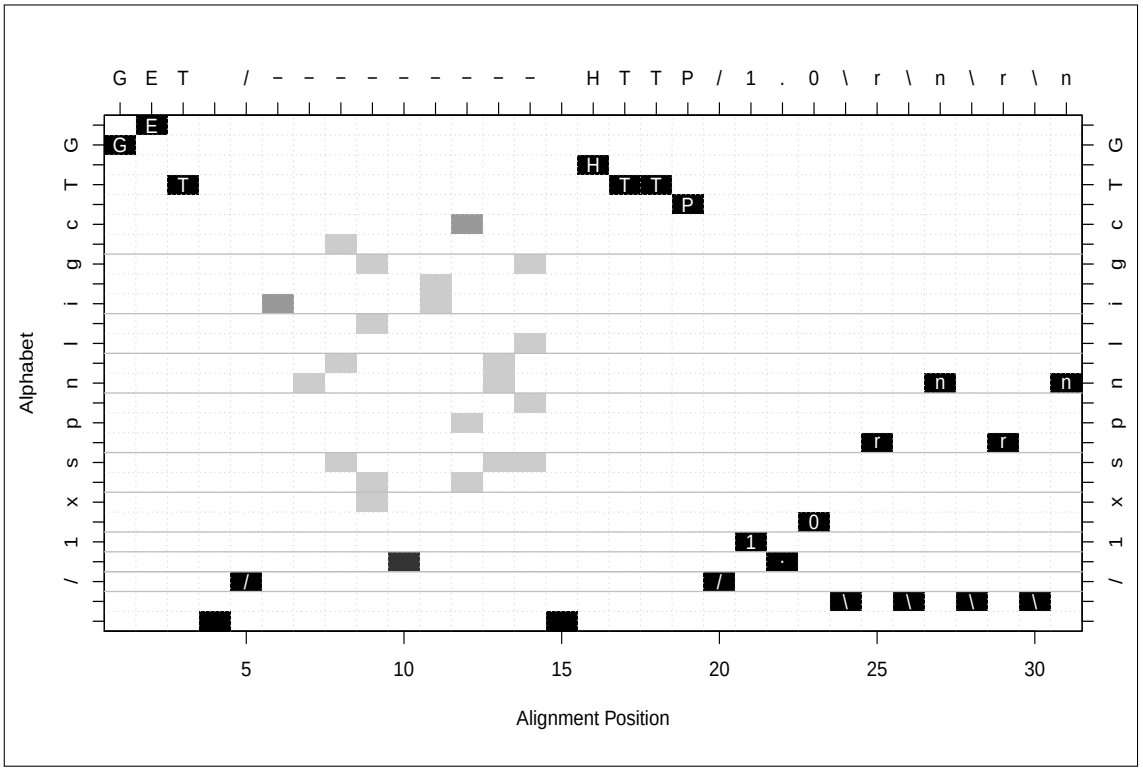
(a) $T = 0$


G E T / i – – – . – c – – H T T P / 1 . 0 \ r \ n \ r \ n
Alphabet
/ 1 x s p n l i g c T G
5
10
15
20
25
30
Alignment Position

(b) $T = 0.2$



(c) $T = 0.6$



(d) $T = 0.8$

Figure 6.7 The extracted message patterns from the correspondent PWM using different values for the generalisation threshold (T).

select a threshold value T between 0 and 1, which we use as a basis for controlling pattern generalisation to the level we desire. A value of 1 indicates that, for an alignment, we are only prepared to consider a character to be a part of an alignment if it occurred in every message at a given position. Lower values indicate that we are allowing a degree of error - for a proportion $(1-T)$ of the aligned messages not to contain that specific character.

For example, if $T = 0$, this means that we are interested in all aligned messages. If $T = 0.6$, this indicates that we are only interested in alignments that apply to at least 60% of the messages. The rest of the characters will be filled with gaps as illustrated in Figure 6.7 (a-d) choosing different values for T and the same PWM generated in the previous step. For each value chosen for T, the figure shows the extracted pattern shown on the top as well as the position and weight assigned to each character.

The inferred patterns are based on the alignment shown in Table 6.4 which is in the ASCII format. Appendix B shows how message patterns are inferred when the aligned messages are in the hexadecimal format. The example is based on the alignment shown in Table 6.3.

6.2.1 Expected Quality of the Message Pattern

Extracting a message structure entails several challenges. In this section, we discuss the quality of the produced patterns (shown in Figure 6.7) in relation to the documented HTTP specifications. The discussion revolves around a simple HTTP request message (GET) and how the generalisation threshold T affects the inferred structure.

The HTTP Request Message

In order to assess the quality of the inferred patterns shown in Figure 6.7, we need to refer the original format for the request message documented in the RFC for the HTTP protocol [45]. As discussed in Chapter 2, Section 2.1.2, protocol fields have different attributes such

```
<!-- Protocol = ''HTTP'' Version=''1.0''-->
<MsgFormat name=''Request''>
<String value= ''GET /'' type= ''Fixed'' Length= ''5''/>
<String value= ''Null'' type = ''Variable''/>
<String value= '' HTTP/1.0\r\n\r\n'' Type=''Fixed'' Length=''17''/>
</MsgFormat>
```

Figure 6.8 An XML message structure derived from the inferred message pattern shown in Figure 6.7 (d) when $T = 0.8$.

as, their length, type (text/binary), and purpose, i.e., a protocol field can be of a fixed-length (static), variable length field (dynamic), delimiters, keywords etc.

As discussed in the previous section, optionally, the inferred pattern can be presented in a form of an XML format or used for further fine-grained analysis (as explained in the future work - Chapter 8, Section 8.4). For example, the generated pattern shown in Figure 6.7 (d) can be defined using the XML format shown in Figure 6.8. The generated pattern can be split into three parts and from each part an information can be further extracted. The first part is `GET  /` of type string, and of fixed content with size 5 Bytes. The second part of type string, and variable content (gaps represent variable content here). The last part of the consensus sequence is of type string, and fixed content with length of 17 Bytes. The extracted information can be fed into a fuzzing framework (e.g., Peach Fuzzer [156]) to test the robustness of the target protocol.

The official documentation of the HTTP protocol (RFC 2616) specifies a request message issued by the client to the sever consists of four parts. The first part is known as the `Request-Line` which includes details about the request method. The second part is dedicated for the (optional) `Header Fields` which provides more information from the client to the server. Each header line must end with `CRLF` (a carriage return `CR` immediately followed by a line feed `LF`). The third part is the `CRLF` indicating the end of the request,

followed by the `message-body` (that is if any payload attached to the request message) as shown in in Figure 6.9 (lines 1-4).

The `Request-Line` part consists of a request `Method` field, followed by the `Request-URI`, and the `HTTP-version` of the protocol ending with `CRLF` token. The three fields are separated by the space (`SP`) character. The method field indicates the method to be applied on the URI. There are several request methods available such as, `GET`, `HEAD`, `POST`, etc. The `Request-URI` is a Uniform Resource Identifier used to identify the resource upon which to apply the request. The last field in the `Request-Line` is the protocol version used by the client for communications. A correct request message in the HTTP version 1.0 may include only the `Request-Line` part of the request as shown in our examples.

Choice of T. The information contained in the inferred pattern heavily depends on the value chosen for the generalisation threshold. This threshold influences the specificity of the pattern and the amount of data that needs to be kept to be considered representative of the true message structure. For example, when $T = 0.8$, the produced pattern consists of three parts with the URI field as one single field (as shown in Figure 6.7 (d)), but when the value of the threshold reduced to 0.6, the generated pattern segmented the URI field into two three parts with the `.` as a fixed-length field and in the middle and the other two parts as variable length fields (gaps). It may be difficult to find the optimal value for T to infer what is actually described in the RFC since practically we don't know the true format of the message. However, the choice of T is normally a trade-off between pattern specificity and sensitivity.

Field Semantics. For some applications, field semantics are occasionally inferred as well. In this thesis, we are inferring the message structure, but not the *semantics*. Typically, the inference of field semantics determines the meaning of certain values within the identified fields, i.e., an integer number could mean a port number, checksum value or a content of similar function etc. Although textual keywords may be indicative of the purpose of certain

```
1 Request= Request-Line;
2 ((Header Fields) CRLF);
3 CRLF;
4 [message Body]
```

```
Request-Line=Method SP Request-URI SP HTTP-version CRLF
```

Figure 6.9 Partial Specification of the HTTP Request Message Format (from RFC 2616).

fields. However, it is not always possible to capture semantics from network traces only [26]. Furthermore, field semantics are not always necessary for various forms of protocol testing.

6.3 Summary

In this chapter we have explained the necessary adjustments we had to carry out on the scoring scheme used in segment-based alignment. Along with the applied alterations, we have implemented the segment-based alignment module based on previous implementations of the Dialign project. We concluded this chapter with a case study demonstrated the procedure as well as the practicality of segment-based alignment to align protocol messages. Also, this his chapter has explained how a pattern is extracted from a set of aligned messages which serves as a concise representation of the message structure. This chapter has also explained a technique on controlling the pattern generalisation level as a mechanism to controlling the precision of the inferred pattern. The final part of this chapter highlighted some of the observation of the proposed approach and the expected quality of the generated pattern.

Chapter 7

Evaluation

This chapter comprises of two parts. The first part is an experiment that quantitatively evaluates the inferred structures by measuring their syntax and whether or not they are recognised by server implementations when synthesized and sent over a network. The second part is a comparative evaluation of the inferred structures against those inferred by the Protocol Informatics project [24]. The chapter concludes with a discussion on some of the key observations related to our approach.

7.1 Experimental Evaluation

In this section we evaluate the message structures inferred by our tool and explain how our evaluation approach deviates from current approaches.

In previous works [26, 102], packet structures were inferred from sets of messages from a known protocol, and the message structure was compared to a reference version (aka true formats) produced by an off-the-shelf network traffic analyser (e.g. Wireshark [144]). The true message formats are constructed from key fields in these messages (important fields). However, these key fields are not always easy to pin down. For example, Discoverer [26] have considered mandatory as well as optional fields to be part of the true formats, on the

other hand, Netzob [102] have considered only mandatory fields. Typically, the official documentations are consulted to determine mandatory fields from optional ones which is also subject to different interpretations since different versions of the same protocol may have different definitions of key fields. For example, `GET / HTTP/1.0` is a correct HTTP request for HTTP version 1.0 which consists of two mandatory fields: `GET` & `HTTP/1.0`. However, for HTTP version 1.1, the specifications stipulates that the header field `HOST` to be present for the message to be considered legal and correctly parsed by the server.

Typically, the reference format is composed of the mandatory fields in the message and do not include protocol delimiters. However, protocol delimiters are important element of the message format. For example, consider the HTTP server (version 1.1), if the inferred structure does not include the last delimiter in the message (`''\r\n\r\n''`), the server will fail to respond.

In our evaluation, we have avoided such issues by using the protocol implementation (sever) which can differentiate between mandatory and optional fields as well as understands the correct message format regardless of the protocol version. In this research, we are not just interested in whether the inferred packet patterns can be parsed, but we are also interested in potentially inferring packet structures that are still capable of eliciting responses from a server. Accordingly, we use our inferred message patterns to automatically synthesise messages that are sent to an implementation of the server in question, and monitor its responses. Specifically, in this part of our evaluation we seek to answer the following questions:

RQ1 Are inferred message patterns syntactically correct?

RQ2 Can inferred message patterns elicit valid server responses?

Protocol	Sample Size (Number of Messages)	Type
HTTP	4000	Text
SIP	5000	Text
TFTP	2300	Binary
SMB	5000	Binary

Table 7.1 Summary of network traces and the lengths of *n*-grams chosen for clustering.

7.1.1 Subject Protocols

We have chosen four data samples of open (documented) network protocols, which have been obtained from an on-line repository of network trace files [124]. Network protocols can either be in a textual or binary format. Depending on their format, they can pose completely different challenges from a clustering and alignment perspective as discussed in Chapter 5. Accordingly, we have selected two protocols from each family. A summary of the data samples is shown in Table 7.1. Their details are as follows:

- **HTTP & SIP**: HTTP [45] and SIP [157] are both text-based application-layer protocols. The messages for both protocols tend to be long and consists of several fields. Despite the fact that HTTP and SIP share similar message properties, both protocols are used for totally different purposes. The main purpose for the HTTP is to access and retrieve web documents, the SIP protocol is mainly used to manage (establish, modify, and terminate) communications sessions [157]. Furthermore, SIP is a stateful protocol, whereas HTTP is not.

- **TFTP & SMB/CISF**: TFTP [46] and SMB/CIFS [13, 158] are binary application-layer protocols. TFTP is a file transfer protocol with a simple message formats. TFTP is also extensively used to support remote booting of diskless devices and on some occasions for malicious purposes [32]. The SMB/CIFS is a network file sharing protocol with complex message formats. SMB/CIFS protocol consists of several flavours known as *dialects* where each dialect consists of a set of extra messages that

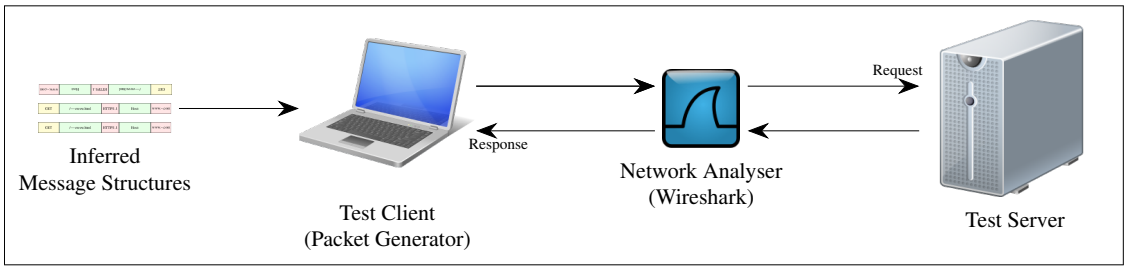


Figure 7.1 Evaluation methodology for the inferred request/response message patterns.

define a particular version [13, 158]. SMB is a proprietary protocol. The equivalent open-source implementation of SMB is widely known as the *Common Internet File System* (CIFS) [158].

Important difference between this evaluations and others is that, for various reasons (often performance related in the case of Needleman-Wunsch based approaches) evaluations of messages truncate messages, and only consider the front portion (which is often assumed to contain the most interesting fields) [99, 23]. In our case, we cluster messages using the suitable configurations for each protocol (see below), however we do not truncate the messages during the alignment, and always use the entire packet. This is to be able to capture the entire header of the protocol message particularly for protocols that tend to contain long variable fields within their message headers.

7.1.2 Experimental Set-up

Our experimental setup is illustrated in Figure 7.1. For each protocol we infer a set of protocol structures (one for each packet type as inferred by the clustering step). These are then provided to a simple *test client*. The test client synthesises a network packet from the inferred message structure and sends the packet to the *test server* – an off-the-shelf server implementation for the protocol in question. The test server accordingly sends responses, which we analyse to determine whether the server considered the messages to be valid or

not. Furthermore, we monitor all of the messages that are sent to the server with a network analyser (Wireshark in our case) to determine whether the messages are syntactically valid. The details of the individual components are elaborated below:

- **A Test Client**: A program that can package and send message patterns. For this task we have developed a packet generator that is able to synthesise and send different message patterns for the four selected protocols. The message synthesis is carried out in a naïve way: The message pattern (showed on the top of Figure 6.7 (a-d) , including any '-'s that fill gaps, is sent verbatim.

  For SMB this required a degree of tailoring; the SMB server can communicate with SMB clients through either raw TCP transport (port 445) or through NetBIOS Session (port 139). We have implemented SMB packet generator to send packets using the traditional NetBIOS session because the samples of traces we obtained from the online repository also used this type of transport. Accordingly, before we send SMB packets to the SMB server, it is first necessary to establish a NetBIOS session [158].

- **A Test Server**: An implementation of the target protocol. For each protocol, we have installed a suitable off-the-shelf server. For the HTTP protocol, we have set up *Lighttpd* server [159] (version 1.4.33) which is mature in development and with specific server responses. For SIP we have installed *Asterisk* [160, 161] (version 11.7.0). We have set up Asterisk to listen on the same port and transport protocol observed in the original captured traces (UDP transport, port 5060). For TFTP we installed *tftpd server* [162] (netkit-0.17). Finally, we chose *Samaba Server* [163] (version 4.3.3) as the dedicated server to test SMB extracted message patterns.

- **A Network Analyser**: We have chosen Wireshark [144] (more accurately, the command line version TShark) to automatically dissect and log data (will be explained ahead the type of data we are interested to log) of the generated patterns.

7.1.3 Methodology

Both research questions revolve around packets that we generate (automatically) from inferred message patterns. For every cluster (message type), we infer a set of message patterns that are slightly different from each other (explained below), and send them over the network. It is important to note that the number of packets that were synthesised varied for each protocol, depending on the number of message clusters that were generated (i.e. the number of message-types that were inferred). Whereas the clustering resulted in two clusters for HTTP and TFTP, it yielded five clusters for SIP and eight clusters for SMB.

RQ1: Are inferred message patterns syntactically correct?

To answer RQ1, we use Wireshark to automatically sniff the packets that we have generated and sent across the network. We employ a feature integrated into Wireshark known as *Expert Information* [164]. This feature provides more information on captured network packets and whether or not they are valid (according to their target protocol). The feature is provided to assist users and experts to understand the behaviour of protocol packets.

If network packets are erroneous, Expert Information consists of a specific *severity level* of the error, and a *group* to which these errors belong. In our experiment, packets are flagged as *invalid* when their expert information belong to the groups *protocol* (violate protocol specifications), *undecoded* (data could not be decoded for some unknown reason), and *malformed* as *syntactically invalid packets*. We also, consider *unrecognised packets* by Wireshark (as the intended application protocol) as syntactically invalid packets. Thus, the answer to RQ1 is obtained by logging the expert information for all sent packets (both inferred request & response packets), and counting the number of valid and invalid packets.

To assess the sensitivity to the threshold T, the experiment was repeated, increasing T from 0 to 1 in increments of 0.1.

RQ2: Can inferred message patterns elicit valid server responses?

To answer RQ2 we monitor the response codes to the synthesised request messages from the test servers. Depending on the protocols, their response is either a valid response, or indicates that the server has failed to properly parse the message. If there is no response at all, this is counted as an invalid response.

For HTTP and SIP the status-code element is a three-digit number indicating the result of the attempt to understand and satisfy the request. The first digit of the status code defines the class of the server response. For the HTTP protocol and SIP protocols, a "400 error" code signifies a bad request – i.e. a poorly formed message, so we count this as invalid. Other responses indicate that the message has been parsed correctly. These may (for both protocols) include a 402 error (URI / URL not found) – we count this as valid, because it is a response to a packet that probably has a valid structure, albeit with a nonsensical URL.

For TFTP protocol, we have observed three types of responses: 01 (File not found), 02 (Access violation), and 05 (Illegal TFTP operation). Server errors with codes 01 and 02 are not caused by invalid packet structures, and are thus treated as valid. The 05 error code is received due to malformed packets, and are thus considered invalid.

For Samba responses, the only responses we observed were either because a request was pointing towards a missing file, or because there was an invalid combination of parameters (even though the parameters had been correctly parsed). These were counted as valid responses. There were however many requests to the server that did not receive any response at all; these were all treated as invalid responses.

As above, the experiment was repeated for different values of the alignment threshold T, increasing from 0 to 1 in increments of 0.1.

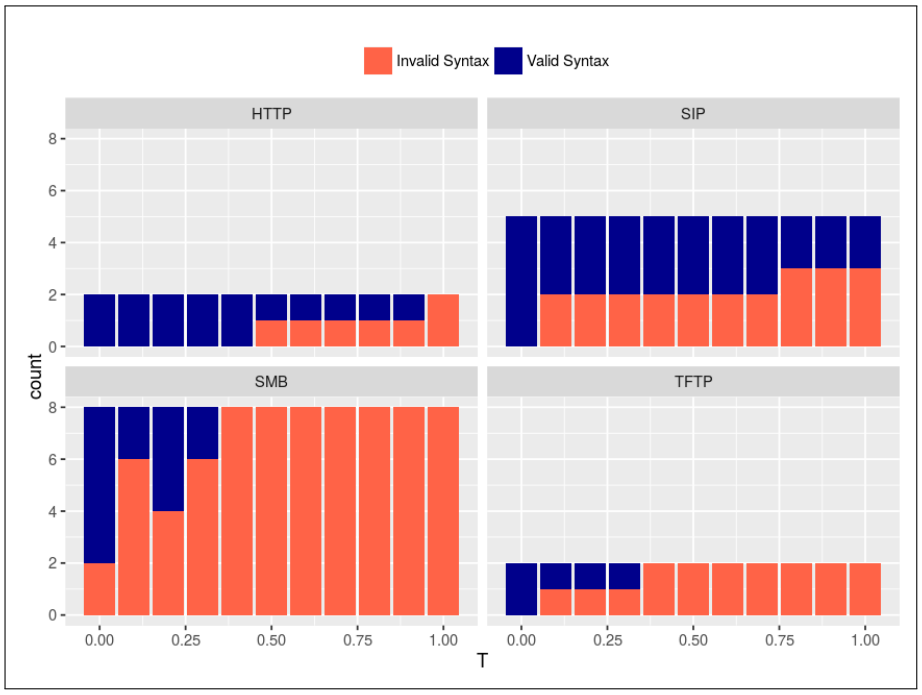


Figure 7.2 Number of syntactically valid/invalid message patterns - as indicated by Wireshark - in relation to the choice of the generalisation threshold (T).

7.1.4 Results

In this section we present the results of the experimental evaluation by answering the research questions introduced at the beginning of this Chapter.

RQ1: Are inferred message patterns syntactically correct?

The plots in Figure 7.2 contain the proportion of valid vs. invalid packets sent for each protocol. The plots are best viewed in colour, where valid and invalid packets are highlighted in blue and red respectively. For the HTTP and SIP protocols the results indicate that we were able to generate syntactically valid message patterns for most values of T. However, lower values of T tend to produce more synthetically valid packets than higher values. For HTTP all packets are valid for $T < 0.5$, and for SIP over half of the packets are valid for $T < 0.8$.

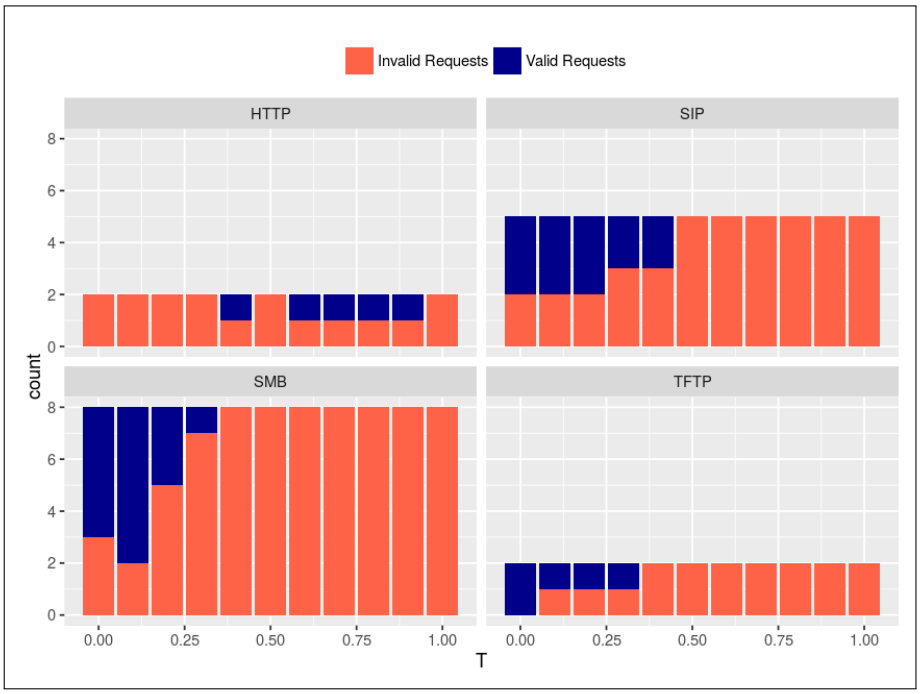


Figure 7.3 Number of valid/invalid message requests - returned by protocol servers - in relation to the choice of the generalisation threshold (T).

For the binary TFTP and SMB/CISF protocols, the validity of the packets is much more sensitive to the T parameter, and fails to yield any valid packets for $T > 0.3$ in both cases.

Conclusion: For all protocols, if we choose a suitable threshold value T, which tends to be $T \leq 0.3$ for all protocols, the inferred protocol structures are sufficiently accurate to enable the synthesis of syntactically valid messages.

RQ2: Can inferred message patterns elicit valid server replies?

The plots in Figure 7.3 show the indications from the server responses as to whether the synthesised messages were valid or not. The plots are best viewed in colour, where valid and invalid packets are highlighted in blue and red respectively. In contrast to the results for RQ1, synthesised packets for HTTP generated with low values of T failed to elicit valid server responses. Instead, this only occurred for T values in the range from 0.40 to 0.80.

For SIP, the successful responses were restricted to lower threshold values (for $T \geq 0.5$ all packets were treated as invalid). As with HTTP, even though Wireshark categorised several of the generated packets to be valid, they elicited an invalid response from the server.

With TFTP the responses from the server exactly matched the categorisations by Wireshark. Again, low values of T tended to lead to synthetic packets that elicited a valid response.

With SMB, the responses from Samba pretty much matched the syntactic validity assessments by Wireshark. Again, lower values of T tended to yield more valid packets. However, as was the case with HTTP, a lower value of T did not guarantee the best response. For SMB, $T = 0.1$ led to a majority of synthesised packets being parsed as valid by the server.

Conclusion: For all protocols, if we choose a suitable value for T, it is possible to synthesise packet structures from inferred protocol structures where at least half of the packets will be correctly parsed by the server. The selection of T is not as clear cut as with RQ1; values that led to packets that were deemed to be syntactically correct do not necessarily lead to packets that are recognised by the server in question. Finding a suitable value of T may therefore require a degree of experimentation. Since lower values of T tended to produce the best results (at least for SIP, SMB and TFTP), it would make sense to start off with lower values and work upwards.

7.1.5 Threats to Validity

With respect to external and internal validity, we consider the following threats:

- We have only applied it to four protocols. It is of course possible that there are different protocols for which the performance of the approach would be different.
- For each protocol, we downloaded samples of traffic from a repository. However, since this data was not collected in an active testing environment, it is generally not the case that every feasible behaviour of the protocols in question was covered in the traces.

- With respect to internal validity, our server-responses were from specific server implementations. It is possible that other implementations of the same protocol could have provided different responses. Sometimes, the extracted pattern is not complete. Parts of it may be missing or truncated due to bad sampling, clustering, or alignment. It may be syntactically almost (but not quite) correct, in which case will consist of a sequence of syntactically correct fields, or we may have picked up its beginning and missed the last part, or we may have missed the beginning and picked up somewhere in the middle. In each of the above cases, it is totally up to the parser in the protocol server to carry out the syntax analysis and its ability to detect syntax errors. Therefore, the choice of the protocol server is important as some protocol servers follow strict syntax parsing while others do not.

We did take care to mitigate these threats where possible. We selected protocols that were broadly diverse (all are widely used, spanning both text and binary families). The network data is intended to represent 'typical' network usage. It was not biased by the authors and is not (at least in its descriptions) associated with a specific purpose. For the network servers, all have been used in previous studies and considered to be accurate implementations of the original protocol specifications. As will be discussed in the Future Work, the intention is to address these threats further with a larger empirical study.

7.2 Comparative Alignment Evaluation

This section provides a comparative evaluation between two alignment approaches. First it presents a quantitative comparison between segment-based alignment (SBA) and a traditional alignment technique based on the Needleman-Wunsch (NW). The second part provides a qualitative comparison between both techniques. Specifically, we seek to answer the following research questions:

Protocol	Cluster	Message	# of Msgs	Msg Lentgh (char)			Content
				Min	Max	Avg	
HTTP	A	GET	300	58	1646	800	Diverse
	B	POST	100	334	1332	879	Similar
	C	HEAD	147	576	604	582	Similar
SIP	A	REGISTER	300	564	1392	611	Diverse
	B	SUBSCRIBE	300	1078	1406	1264	Similar
	C	OPTIONS	300	592	604	598	Similar

Table 7.2 Description of the selected clusters and their messages attributes.

RQ1 What is the accuracy of SBA in comparison to NW alignment?

RQ2 How can segment-based alignment improve protocol inference?

To answer the above questions we compare the performance of our alignment tool, which is based on the iterative segment-to-segment alignment approach, and the protocol informatics (PI) tool [24], which is based on the progressive alignment using the Needleman-Wunsch algorithm (as explained in Chapter 2, Section 2.2.3). To the best of our knowledge the *Protocol Informatics* (PI) [24] and the *Netzob* project [102] are the only publicly available implementations that use the Needleman-Wunsch to infer the message structure. However, Netzob is a complex project and its recent variants are based on a modified version of Neeldeman-Wusnch [102], therefore cannot be used as a baseline for the comparison.

For the comparative evaluation, we have selected data from two text protocols, the HTTP and SIP protocols, which are previously described in Section 7.1.1. From each protocol, we have chosen three clusters where each cluster contains messages of similar type that needs to be aligned and ultimately their alignments evaluated. To present different challenges to both alignment tools, the chosen messages vary in length and content. The details of the selected clusters and their messages are described in Table 7.2.

7.2.1 Methodology

In bioinformatics, there are several methods for evaluating the quality of a multiple sequence alignment [65], however these methods require a *baseline alignment* to be available, which in our application (and protocol reverse engineering in general) something we do not own. Generating our own reference alignment is not feasible because the length of these messages can stretch to thousands of characters (as described in Table 7.2). Therefore, we use different approaches to evaluate the quality of the produced alignments.

RQ1: What is the accuracy of SBA in comparison to NW alignment?

To answer this question, we evaluate the overall accuracy of the alignment by checking the number of keywords that are correctly aligned in the inferred pattern in relation to the overall number of keywords required by the protocol specifications (specified as mandatory fields and must appear in the message). We understand if the inferred pattern lacks certain mandatory keywords, the tool will make the wrong inference. Similar to previous approaches [26, 102], we do not consider *optional fields* in the definition of the general pattern structure, therefore they are excluded from the evaluation. The required keywords are identified from the formal documentations of the protocols, which are the *Request For Comments* (RFC) manuals for the HTTP [45] and SIP protocol [157]. Also, incomplete or fragmented keywords that partially appear within the pattern are not counted as valid keywords.

Similar to the evaluation carried out in Section 7.1, we set the generalisation threshold T to different levels (from 0 to 1), and each time we calculate the accuracy of the produced pattern. The keyword detection accuracy is defined as the number of correctly detected keywords in the pattern over all keywords required in the message format :

$$Accuracy = \frac{\#\ of\ Keywords\ Correctly\ Identified}{Total\ \#\ of\ Keywords} \tag{7.1}$$

No	Message
01	GET / HTTP/1.1\r\nHost: www.google.co.uk\r\nUser-Agent: Dillo/3.4\r\n\r\n
02	GET /myuri.html HTTP/1.1\r\nHost: www.abc.cba.net\r\n\r\n
03	GET /myveryverylonguril.html HTTP/1.1\r\nUser-Agent: Firefox\r\n\r\n
04	GET /verylongutiueyryhwpirrytbeyuujw.html HTTP/1.1\r\n\r\n
05	GET /shorteruri.png HTTP/1.1\r\nHost: www.bbc.com\r\nUser-Agent: Dillo/3.4\r\n\r\n
06	GET /abitlongeruri.html HTTP/1.1\r\nHost: www.abc.cba.net\r\n\r\n
07	GET /veryextsj.jpg HTTP/1.1\r\nUser-Agent: Firefox\r\n\r\n
08	GET /lklkjlkjlkwefrweriuw bjsdfcbjhjsdfkSDFJSGDFJHNBIUWEIWYUERuiyrweiu.html HTTP/1.1\r\n\r\n
09	GET / HTTP/1.1\r\nHost: www.le.ac.uk\r\nUser-Agent: wget/1.2\r\n\r\n
10	GET /thisislonglonglonglongrilieuiueiuelk.html HTTP/1.1\r\nHost: www.google.co.uk\r\nConnection: close\r\n\r\n

Table 7.3 Ten synthesised HTTP GET messages.

The method will give the highest score 1.0 to the alignment with all intact keywords within the pattern, and the lowest score 0 with no intact keywords.

RQ2: How can segment-based alignment improve protocol inference?

To answer this question, we conduct a qualitative and visual comparison of the generated alignments. To be able to observe the difference in the quality of alignments produced by both tools, we have generated a small set (a cluster) of similar HTTP request messages consisting of ten *GET* messages as shown in Table 7.3. Each message is composed of a *request-line* and various *header fields*. We opted not to use genuine trace messages because we need to keep them sufficiently short to fit into the page and be able to visually demonstrate the difference between two alignment techniques. The aim of this comparison is to show how our segment-based alignment fares against the traditional byte-wise alignment technique, and to explain some of the phenomena that were discussed in the previous research question.

In our comparison, we will focus on protocol fields that are correctly aligned by both tools. The protocol fields of interest here are keywords and delimiters, since these fields are hard-coded in protocol implementations and considered a prerequisite for correct parsing [45]. We consider the set of protocol keywords in these messages to be:

`GET, HTTP/1.1, Host, User-Agent,`

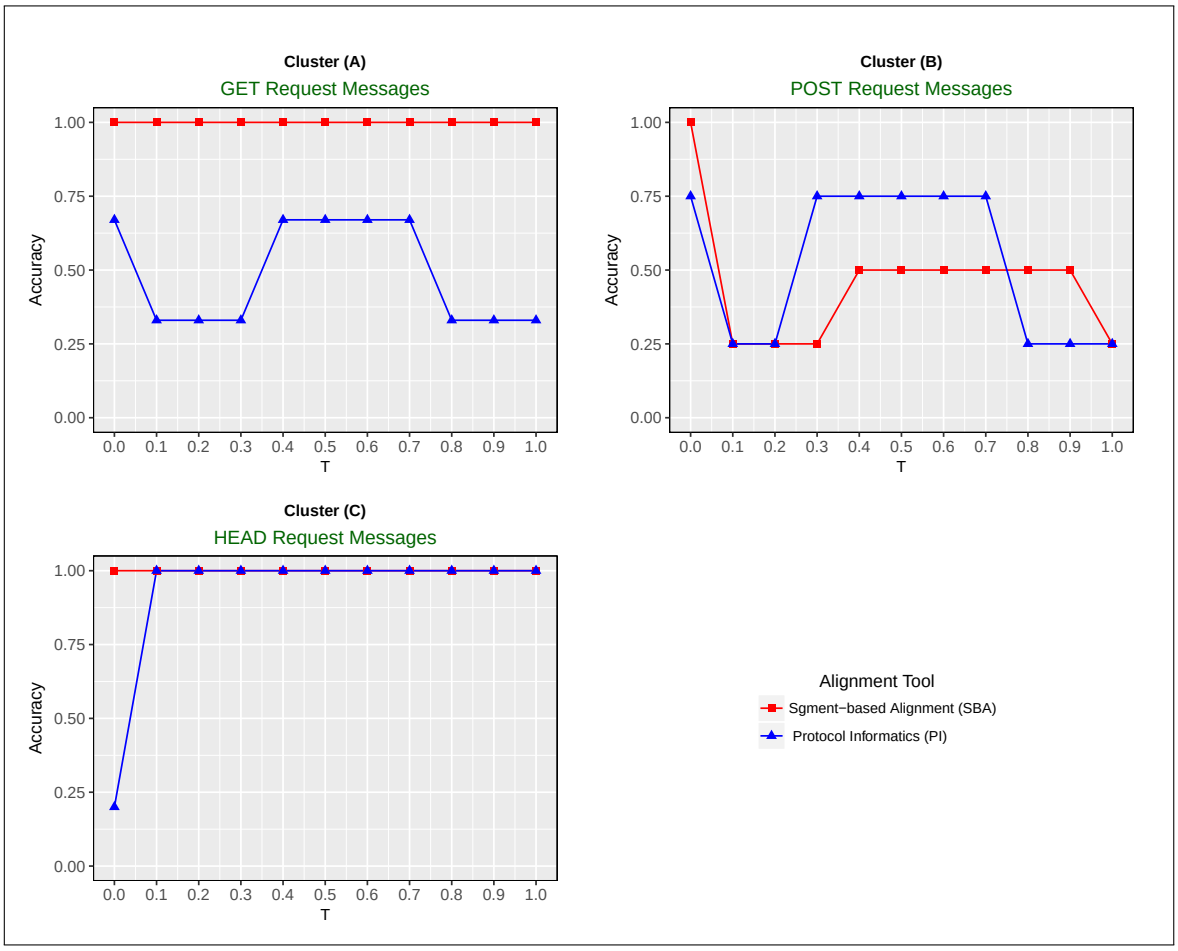


Figure 7.4 Accuracy of HTTP patterns inferred by Segment-based Alignment and the Protocol Informatics tool (PI) - in relation to the choice of the generalisation threshold (T).

```
Connection, close
```

We consider the set of delimiters to be:

```
/, [space], :, \r\n
```

7.2.2 Results

This section discusses the results of both research questions RQ1 and RQ2:

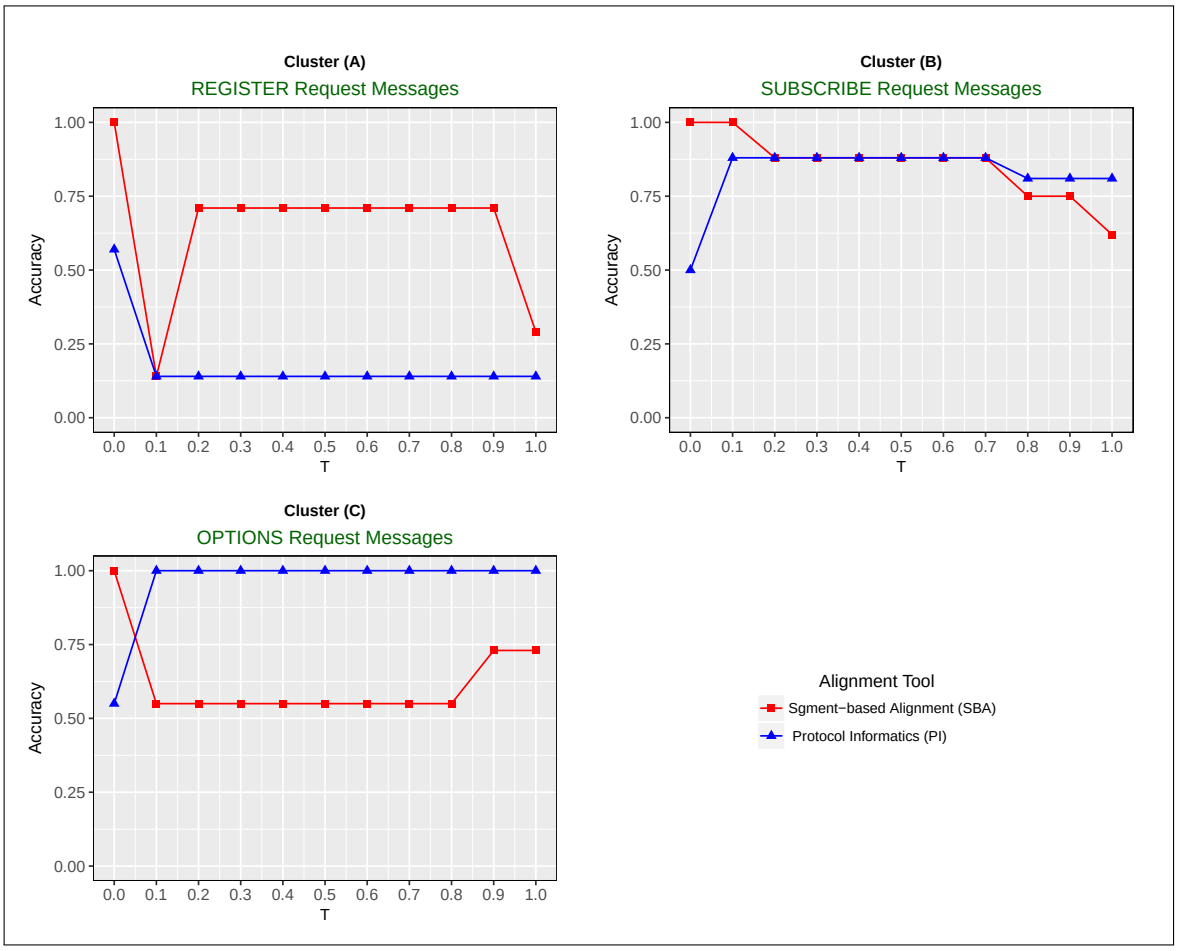


Figure 7.5 Accuracy of SIP patterns inferred by the Segment-based alignment tool and the Protocol Informatics (PI) - in relation to the choice of the generalisation threshold (T).

RQ1: What is the accuracy of SBA in comparison to NW alignment?

Generally the results indicate that segment-based alignment tends to generate highly accurate alignments with diverse messages (different field combinations) and lower values of T while the PI tool is relatively better when messages share strong similarity in content and length as illustrated in Figures 7.4 and 7.5.

- **HTTP Protocol**: As Figure 7.4 shows, for the "GET" messages (Cluster (A)), the segment-based alignment has correctly identified (aligned) all the keywords for all the choices of T while the accuracy of the PI tool is less than 0.75. For the "POST"

messages (Cluster B), the accuracy of the PI tool generally around 0.75 (or less) and the accuracy of segment-based alignment about 0.5, however, segment-based alignment succeeded in identifying all keywords when $T = 0$. As for the "HEAD" messages (Cluster (C)), the accuracy of the segment-based alignment is 1.0 and consistent for all the choices of T. The PI accuracy is similar except when $T = 0$ which is less than 0.25.

- **SIP Protocol**: For the SIP protocol, the accuracy of both tools fluctuates and depends on the overall similarity and length of the messages as illustrated in Figure 7.5. For the "REGISTER" message (Cluster A), the accuracy of the patterns produced by segment-based alignment is significantly higher than the PI tool. Again, segment-based alignment managed to detect all keywords when $T = 0$. For the "SUBSCRIBE" message (Cluster B), both tools tend to infer patterns with relatively higher accuracy than Cluster A. However, the performance of both tools declines when $T > 0.7$. In the "OPTIONS" message (Cluster C), PI generates patterns with significantly high accuracy (1.0). This is due the strong similarity and consistency of all protocol fields within these messages as well as the close similarity of the message lengths (as indicated in Table 7.2). In this cluster, the segment-based alignment managed to retrieve all keywords only when $T = 0$.

RQ2: How can segment-based alignment improve protocol inference?

The alignment produced by Protocol Informatics (using the default user parameters) is shown in Table 7.6. The corresponding segment-based alignment is shown in Table 7.7. The tables are best viewed in colour, where the keywords and delimiters are highlighted in red and blue respectively.

Both approaches successfully align the `GET` followed by the space and the '` /`', which begins all messages. They also successfully align the '`\r\n\r\n`' that is used to finish each

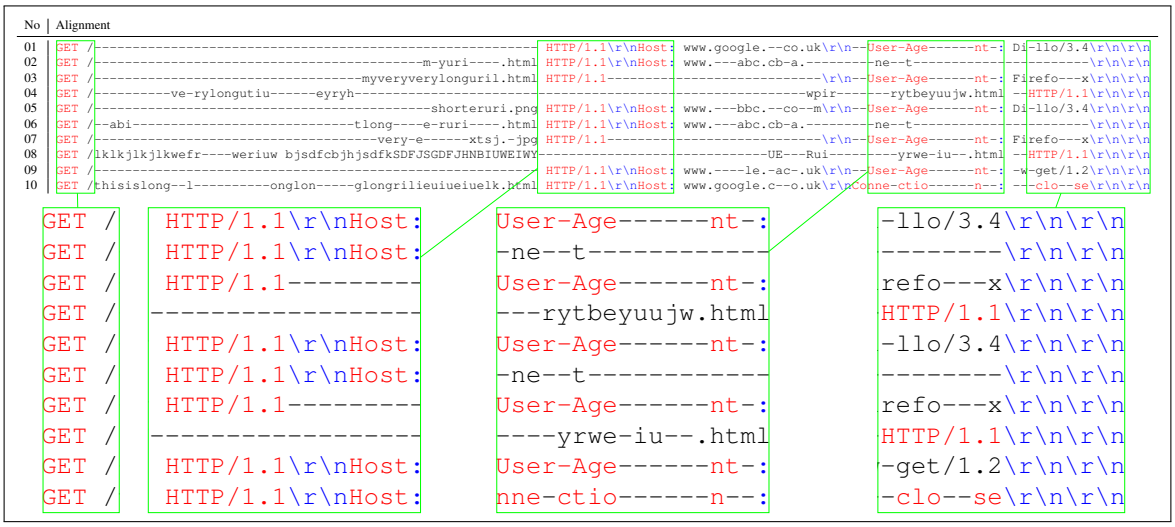


Figure 7.6 Alignment results produced by the Protocol Informatics project using the default user parameters (match=1,mismatch=0,gap=0).

message. However, for the keywords and delimiters that happen in between, the PI approach makes several misalignments, whereas they are all correctly aligned by the segment-based algorithm. The PI algorithm misaligns the `HTTP/1.1` segments for messages 4 and 8. This is because it misaligns the space before the keyword `HTTP/1.1`, forcing the `HTTP/1.1` to the end of the message. These are correctly aligned by the segment-based algorithm. The PI approach also fails to correctly align any of the `User-Agent` keywords, which are again correctly aligned by our approach.

One apparent reason for the improved quality of the segment-based alignments versus PI is that the latter focuses solely on the alignments of individual characters, whereas the former incentivises alignments of longer segments. This is why the segment-based alignments tend to successfully align the `User-Agent` keyword, whereas the same keyword disintegrates in the PI alignment. As shown in Table 7.7, segment-based alignment are able to align messages correctly even if they contain inconsistent fields (e.g., User-Agent). However, as discussed in the motivations (Chapter 4, Section 3.2.2), one of the major drawback of Needleman-Wunsch approaches is that they require these messages to be globally similar (from start to end) and

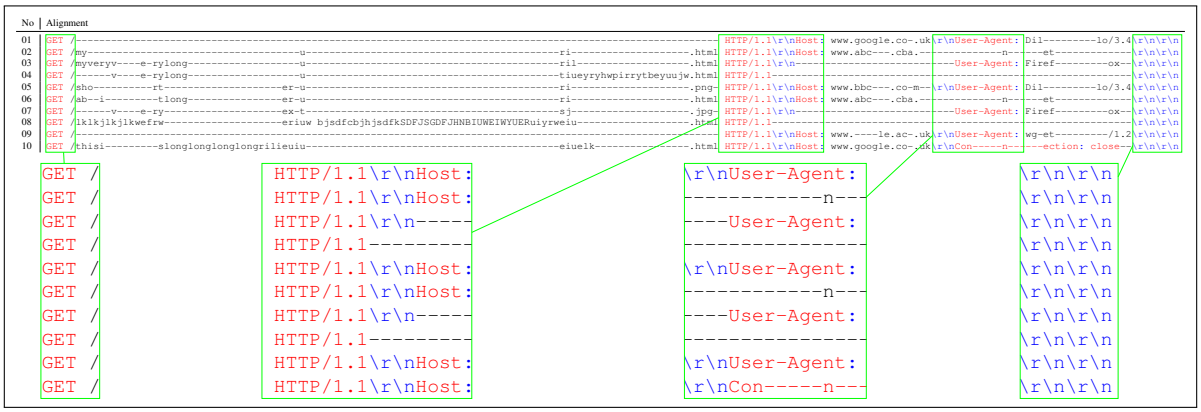


Figure 7.7 Alignment results produced by our segment-based alignment tool.

contain no inconsistent fields to generate accurate alignments. This means, to improve the alignment of the PI tool in our example, we need to split this cluster into two *sub-clusters* where messages in each cluster contain similar field compositions, i.e., messages 1, 3, 5, 7 and 9 can form one cluster, and the rest of the messages are grouped into a separate cluster. This is problem is often known as *over-clustering* where messages from the same type are partitioned into more than one cluster. Also, patterns inferred from these sub-clusters need to be *merged* to infer the final format, which is another step in the process. It is evident that the segment-based alignment approach is not affected by this challenge and does not require messages to be split into fine-grained clusters.

7.3 Discussion

In this section we discuss some of the aspects related to our proposed approach.

Selecting a suitable Threshold

Our experiments indicate that the inferred packet structures can be accurate enough to synthesise valid packets. However, the validity does depend on the choice of a threshold

parameter T. Choosing a high T means that an inferred packet will only contain those aligned symbols that occur in the majority of messages. If we choose a low value of T, we allow the alignment to encompass characters that occurred in a smaller proportion of the messages.

Although a low value of T has tended to produce the best results, it is generally a bad idea to set T too low (e.g. 0). If T is too low, the final alignment will end up containing many characters that are irrelevant to the packet structure - i.e. are not delimiters or keywords but instead belong to field data. However, these can still lead to combinations of characters that can lead to messages that are invalid, which is what appears to have happened with the HTTP traces, which contained various optional headers and variable-length fields.

The robustness of our approach depends on the choice of the generalisation parameter. However, the choice of T can be considered as a trade-off between pattern *correctness* and *completeness*, i.e., when we include only frequent messages (high values for T), we consider the pattern to be more generic, but lacks details (incomplete). Conversely, with less frequent messages, we expect the generated pattern to be complete, but over-specific and less accurate by containing variable and noisy fields. This trade-off for a range of values for T was clearly demonstrated from the evaluation.

Generally, with homogeneous clusters that contain similar messages, the choice of T does not seem to affect the inferred patten anyway. However, with heterogeneous clusters (when messages of the same type are composed of different optional fields), the choice of T, in fact is a favourable feature to mask some of the inconsistencies between messages caused by optional fields. Also, T in this situation can be used to recover from some of the errors caused by clustering and alignment algorithms.

Choosing Clustering Configurations

In the evaluation, we have used the clustering configurations described in Table 5.4 (using the entire samples) to find out the packet structures of four network protocols. As explained in

Chapter 5, Section 5.2.3, for undocumented protocols we rely on internal validation measures to infer suitable configurations for clustering. According to the experimental results, the procedure described in Section 5.2.3 can be further simplified to detect suitable clustering configurations as follows:

- **Sample Size**: As indicated by the experimental results, the sample size has a minimum impact on clustering accuracy, therefore we could use the entire sample for the protocol.
- **Message Length**: As for the message length, the results show that the length of the message has significant impact on clustering. The results generally indicate that shorter lengths tend to give better clustering results. Specifically, we fix this parameter to 16 bytes for the clustering step. For the alignment, the entire packet should be used.
- ***N*-gram Length**: Because the length of the *n*-gram is protocol dependent, we use the procedure explained in Section 5.2.3 to determine a suitable length for the *n*-gram. Specifically, we cluster at different points using different lengths for the *n*-grams (e.g., from 2 to 10), and then the Ball-Hall index is used to retrieve a suitable length for the *n*-gram.
- **Distance Measure**: The experimental results also show that binary-based distance measures (Jaccard, Dice, and Bruan-Blanquet) generate better clustering results than the Cosine and Euclidean measures. In this research, we use the Braun-Blanquet as the distance measure of choice since the Ball-Hall index tend to give better predictions with this distance measure as discussed in Chapter 5, Section 5.2.4.

Example. We have used the above procedure to predict a suitable length for the *n*-gram for the SIP protocol, which we have used in the evaluation. We have applied the agglomerative hierarchical clustering algorithm (using its default configurations described in Table 5.2) to cluster the protocol sample at various *n*-gram levels (from 2 to 10) , and then used the

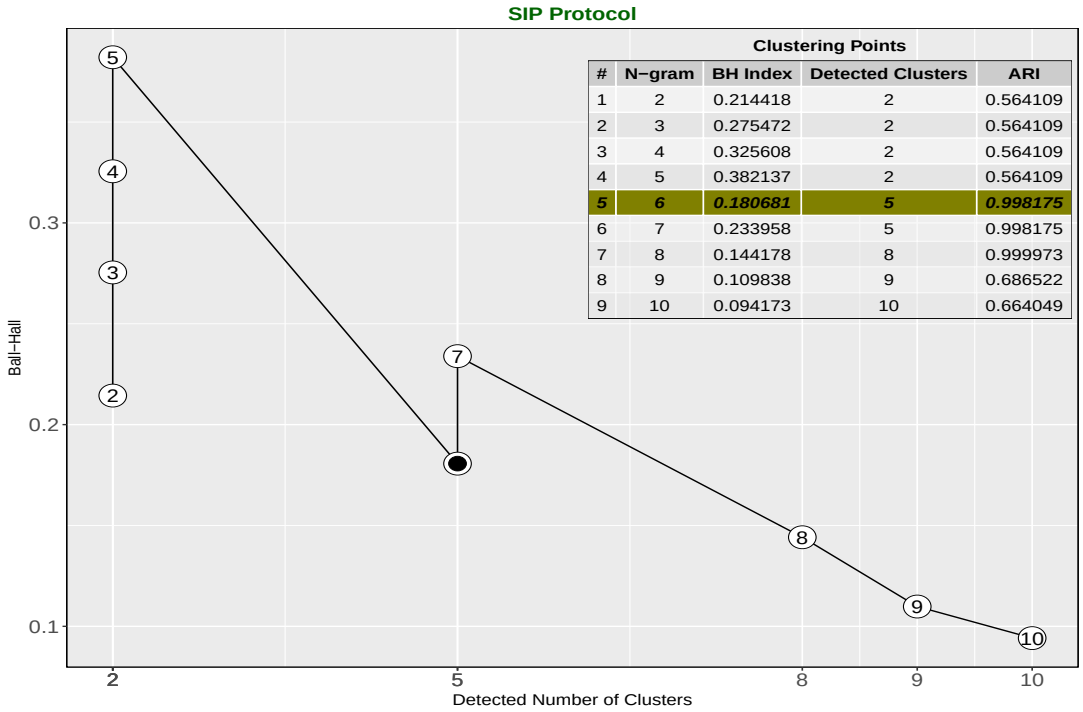


Figure 7.8 Choosing suitable length for the n-gram using the Ball-Hall index.

Ball-Hall index to predict a suitable length for the n-gram that corresponds to the predicted number of clusters. We have also used the Adjusted Rand Index (ARI) to extrinsically evaluate clustering results since the messages classes are known to us. As shown in Figure 7.8, the Ball-Hall measure managed to retrieve a length of $N = 6$, which matches high clustering score but not the correct number of clusters. It is evident that the best clustering score is achieved when $N = 8$, which matches the correct number of clusters. The Ball-Hall index did not predict the best clustering score due to the unbalanced sample (i.e., the majority of these messages belong to 5 clusters), which is considered a key factor that influences internal measures as discussed in Chapter 5, Section 5.2.4.

Chapter 8

Conclusions & Future Work

The purpose of this thesis is based on two motivations: 1) To investigate the impact of specific factors on message clustering, and whether the optimal configuration of these factors can be inferred. 2) To improve message alignment by mitigating the common draw-backs of traditional alignment algorithms (i.e., the Needleman-Wunsch algorithm). Accordingly, a modular framework has been developed and evaluated on four open network protocols.

8.1 Impact of Factors on Clustering

In this thesis, we have conducted an empirical study to investigate the impact of four factors on clustering quality, and time. Data samples of four open network protocols were used to determine the impact of the length of the *n*-gram, length of the message, size of the sample, and the choice of the distance measure on clustering. Generally, the results indicate the following:

- The choice of the distance measure, and the length of the message have significant effect on clustering quality and of paramount importance. Normally, long messages (longer than 16 bytes) tend to increase noisy features (irrelevant *n*-grams) and subsequently

reduce clustering quality. Generally, there should be an efficient feature selection mechanism to mitigate the impact of this factor. Also, distance measures which are based on binary similarity coefficients of the *Bruan-blanquet*, *Dice* and *Jaccard* tend to generate significantly better clustering results than other measures (the *Cosine*, &the *Euclidean* measures).

- The number of messages in the trace (sample size) have negligible impact on clustering accuracy. This is indicative that the frequency distribution of the selected features (*n*-grams) for clustering does not significantly change among the selected samples (sub-samples). On the other hand, clustering time primary depends on the number of messages in the trace.

8.2 Optimal Factor Configuration for Clustering

In this thesis, we have also investigated whether internal validation measures for clustering can be used to predict the best factor configuration for clustering. Internal validation measures are often used to predict the optimal number of clusters in a dataset when external measures can not be used (lack of ground truth labels). Based on this intuition, we have used the internal validation measures to retrieve the best clustering configuration that corresponds to the predicted number of clusters. Five intrinsic validation measures are used to evaluate the results of the agglomerative hierarchical clustering (AHC) algorithm on data samples collected from four different network protocols. The study indicates the following:

- Most of the selected internal measures tend to have certain limitations from different characteristics of the data samples, such as the shape of the clusters, noise, cluster sizes and the presence of of sub-clusters. The Ball-Hall index is the only measure that performed well on all data samples. The *Ball-Hall* is based on measuring the dispersion of data items within the clusters using the sum-of-square statistic. Unlike

other measures, Ball-Hall did not seem to be greatly affected by the presence of clusters with arbitrary shapes, sparse sub-clusters as well as clusters with significantly different sizes.

- The results of the experiment also indicate that the performance of the Ball-Hall index improves when binary similarity measures are used for clustering (e.g., Jaccard, Braun-Blanquet, Dice). Specifically, the Ball-Hall index produces more accurate predictions when it is combined with the Braun-Blanquet similarity coefficient.

Generally, the performance of intrinsic measures depends on the internal characteristics of the collected samples, such as the geometrical shapes of the clusters, presence of outliers (noise), density and cluster sizes. Therefore, there is no one single measure that can always be applied on all data samples. Therefore, it is paramount that datasets thoroughly explored before applying clustering to be able to choose a suitable measure. According to our datasets, classic intrinsic measures (e.g., Dunn index, silhouette measure, DB index) had their own limitations while variance based measures performed well. Also, the performance of internal measures is often coupled with the clustering algorithm. The reported results of our empirical study is based on the agglomerative hierarchical clustering, thus the performance of the internal measures may vary with other clustering algorithms.

8.3 Improving Message Alignment

This thesis has also demonstrated how segment-based alignment can be used to align network packets for the purpose of detecting packet structures. This is a departure from typical approaches, which have been based on variants of the Needleman-Wunsch algorithm, which is prone to inaccuracy when applied to network data. Specifically, this thesis have:

- Shown how segment-based alignment can be used to generate accurate alignments.

T	Message Pattern	Produced Grammar
0.8	`GET[SP]/[VAR][SP]HTTP/1.0\\r\\n\\r\\n`	0 → `GET1/[VAR]1HTTP/\1.\122` 1 → `[SP]` 2 → `\r\n`
0.6	`GET[SP]/[VAR].[VAR][SP]HTTP/1.0\\r\\n\\r\\n`	0 → `GET1/2.21HTTP/\1.\133` 1 → `[SP]` 2 → `[VAR]` 3 → `\r\n`

Table 8.1 An example of message patterns (shown on the left) generated with two different thresholds ($T = 0.8$ & $T = 0.6$), and the grammar (on the right) produced by SEQUITUR. In the original patterns, we have replaced gaps with the token [VAR] denoting a variable field. We have also replaced the space character with the [SP] token and escaped the character "\" due to their special meaning within the SEQUITUR program [1]

- Developed a proof of concept implementation.
- Experimentally shown that the packet structures identified by our alignments can be used to automatically synthesise new messages that can be recognised by a server.
- Show that the messages tend to be syntactically valid, albeit for a suitable choice of threshold parameter.
- In a comparative study, the results generally indicate that segmented-based alignment produces message structures with significantly higher accuracy than the Needleman-Wunsch based approaches especially with long and diverse protocol messages.

8.4 Future Work

The work of this thesis can be further extended in several directions, both with respect to the inference technique as well as its applications. Three areas which offer new avenues for future research are identified:

Detecting Message Hierarchical Structure

A protocol message often exhibits hierarchical structure. Normally, it consists of a repetitive phrases and sub-phrases of different fields. Despite the absence of field semantics [1], using an algorithm such as the *SEQUITUR* algorithm [165] can be a viable solution to the problem of detecting the hierarchical structure of a protocol message and without any prior assumptions about how the message is delimited.

SEQUITUR is generic algorithm that infers a hierarchical structure from a string made up of discrete characters. The basic idea is that sub-phrases which appear more than once within the sequence can be replaced with a grammatical rule that generates the phrase, and continuing this process recursively producing a hierarchical representation of the original sequence. Therefore, the resulted grammar rules offer further insights into the lexical structure of the sequence. The algorithm is also used as an effective compression tool for large texts.

As we briefly discussed in Chapter 6, Section 6.2.1, the inferred message pattern can be used for further analysis to produce a fine-grained representation of the identified message structure. To demonstrate the viability of the approach, consider the example shown in Figure 8.1. The example shows (on the left) two message patterns previously generated in Chapter6 (Figure 6.7 (d) & (c)). The patterns generated with two different thresholds ($T = 0.8$ and $T = 0.6$). In these patterns, we have replaced variable fields (gaps) with the token "`[VAR]`" to indicate the position of the variable length field. We have also replaced the space character with "`[SP]`" token and proceeded the character "\" with another "\" because SEQUITUR considers these characters as special characters in its implementation [1]. The Figure also shows (on the right) the grammar (for each pattern) generated by the SEQUITUR algorithm.

In the first pattern, the algorithm has generated three rules. The first rule represents the pattern, and the other two rules represent the two repeated sections within the pattern (space "`[SP]`" and the "`\r\n`"), and in the second pattern, the algorithm has generated four rules.

[1] Understating the meaning of a protocol field in terms of its purpose and relation to other fields.

Because the original pattern is slightly more detailed than the previous one, one extra rule is detected; i.e., in addition to the two delimiters (space "`[SP]`" & "`\r\n`"), the algorithm generated another rule for the variable field "`[VAR]`" reflecting the repetitive occurrence of these tokens as well.

The inferred hierarchical structure of the message ultimately extends the use of the inferred specifications to cover more applications that require more depth of detail such as writing protocol dissectors for generic traffic analysers [144].

Segment-based Alignment

The fragment-chaining procedure works independently from how fragment weights are calculated. This allows for further experimentation with different weighting schemes and enhance the weight calculations for the fragments. For example, the current version of the weighting function takes into account the length of the fragment and length of the input messages. Similarly, further messages features could be added to the weighting functions, such as the data type of the fragment (e.e., binary, ascii etc.) to assess whether characters within the fragment are related enough to form one consistent block and as a result given a higher score or it is of a combination of different and unrelated characters.

Simple Fuzzer

The structure of the inferred patterns could be used as a basis (a template) for an automated smart protocol fuzzing. Fuzz testing network protocols is the process of generating random or semi-random packets (malformed packets) and sending them to the protocol server and monitor its behaviour (i.e., for crashes, errors, etc.).

In the evaluation our tool, we have observed that extracting patterns with different levels of abstraction (using different values of T) has triggered a variety of server responses. This intuition could be leveraged to create a useful test cases for protocol fuzzing. With this

mechanism, malformed patterns are gradually created without suddenly breaking the overall structure of the packet. The appearance of partial parts of a keyword or a delimiter can expose potential errors. For example breaking up the multiple delimiter `\r\n\r\n` into `\r\n` (or just `\r`) may cause inaccurate assumptions about the length or place of that delimiter expected by the server. Variable protocol fields (denotaed as gaps in the pattern) could be easily to replaced with more sophisticated unusual characters that is proven to be problematic (e.g., null character, new-line character, semi-colon, etc.).

8.5 Final Notes

We conclude this thesis with some of the observations and lessons learned from reverse engineering network protocols from network traces.

- **Network Data.** One of the most important artefacts in the process of protocol reversing is the captured data. There are many on-line resources that offer readily captured protocol traces, but most of these traces are not properly trained, and often used toward specific application (e.g., Malware analysis). In many cases, collected samples may seem large in content when they contain only one or two messages categories. Also, the collected samples are often unbalanced where one message category contains significantly more messages than the other. Practically, even if we collect large sample consists of thousands of messages, we still can not guarantee that the dataset is balanced. That is because some protocols are more verbose than others and the actual size of the dataset may not reflect the messages types in the trace. Sometimes one request (e.g., *read* in the TFTP protocol) is encountered with thousands of *response* messages (e.g., data packets) from the server. As the experimental results indicate, most clustering algorithms and internal validation measures are affected by such issues.

As discussed in the thesis, there are several threats to validity that can emerge from the choice of a dataset. Although we have been careful in mitigating these threats, we believe a reference dataset is essential for such research. At the moment there is no a "standard" dataset or baseline samples that could be used for the evaluation purposes. Typically, a reference dataset should contain various protocols with diverse message types (i.e., balanced dataset) and with different complexities in their message structures. It is extremely vital for such research to have a standard protocol samples with precise description for benchmarking and performance evaluation.

- **Clustering.** The cornerstone of many protocol reverse engineering approaches is clustering. However, the quality of clustering depends on several algorithmic and non-algorithmic factors. In this research, in addition to the use of internal measures to predict suitable clustering configurations, we have been explored several routes to improve clustering quality, including *cluster ensembles* [166, 167], which deals with the problem of combining multiple clusterings into a single clustering solution (aka a consensus) - i.e., inferring consensus from the clusterings produced by different factor configurations in our case. Unfortunately, the results of this solution were not conclusive and seemed to fluctuate from one protocol sample to another. Also, it requires a significant amount of time for large protocol samples (1000+ messages).

  In this research, we have used basic feature selection mechanism, which is based on filtering infrequent *n*-grams. Clustering can be significantly improved if a robust feature selection algorithm is used, therefore investigating more feature selection algorithms should always be considered other than restricting our focus on improving the performance of clustering algorithms alone.

- **Alignment.** The comparative evaluation indicates that segment-based alignment is a technique that can produce highly accurate alignments particularly with long and diverse protocols messages, which is often the case in network protocols. Segment-

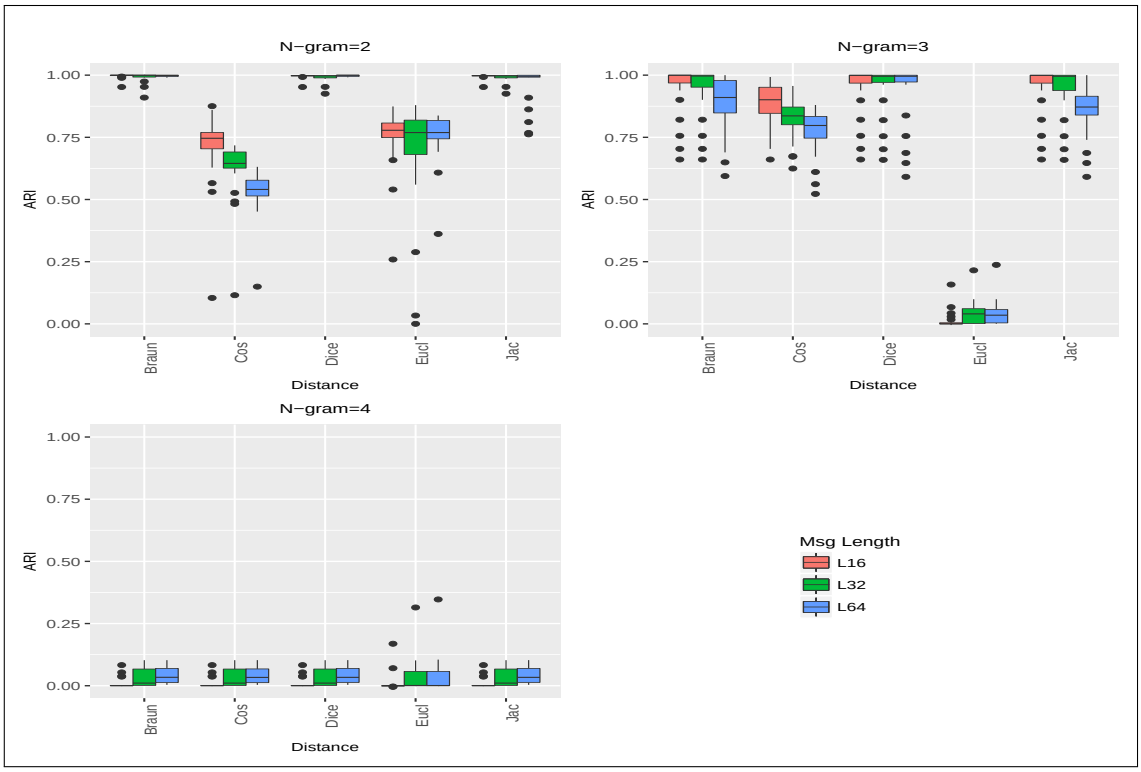
based alignment is also as a mechanism to avoid the over-clustering problem, which often encountered with Needleman-Wunsch based approaches where messages of the same type are clustered into several smaller sub-clusters - that share stronger similarities - to produce meaningful alignments. As explained in the background, the run time complexity of the segment-based alignment is considerably large. It generally depends on the size of the cluster (number of messages to be aligned) and the length of these messages, which ultimately depends on the generated fragments from these messages. However, this should not be a major limitation to the proposed technique since the alignment time can be accelerated by excluding fragments with low weights.

- **The Inference Approach.** Our inference approach is based on analysing data exchanged by the application protocol only, and avoided to include information contained in lower protocol layers in the protocol stack (e.g., Transport protocol, Network protocol etc.). While this design choice reduces the amount of data that needs to be processed, this direction has the disadvantage of not utilising the information contained in other layers to help improve the inference. For example, using the IP address (in the Network Layer) can be used to classify messages based on its direction (client-to-server/server-to-client) and improve the accuracy of clustering. Also, it is often that protocol fields contained in lower protocol layers exhibit high correlations with fields contained in the application layer. For example, the length of the overall packet provided by lower layers often correlates with the length of the payload contained in the application message, which is often difficult to determine from the application message alone. Therefore, utilising the protocol data embedded in other protocols can be invaluable information to infer more accurate protocol specifications.

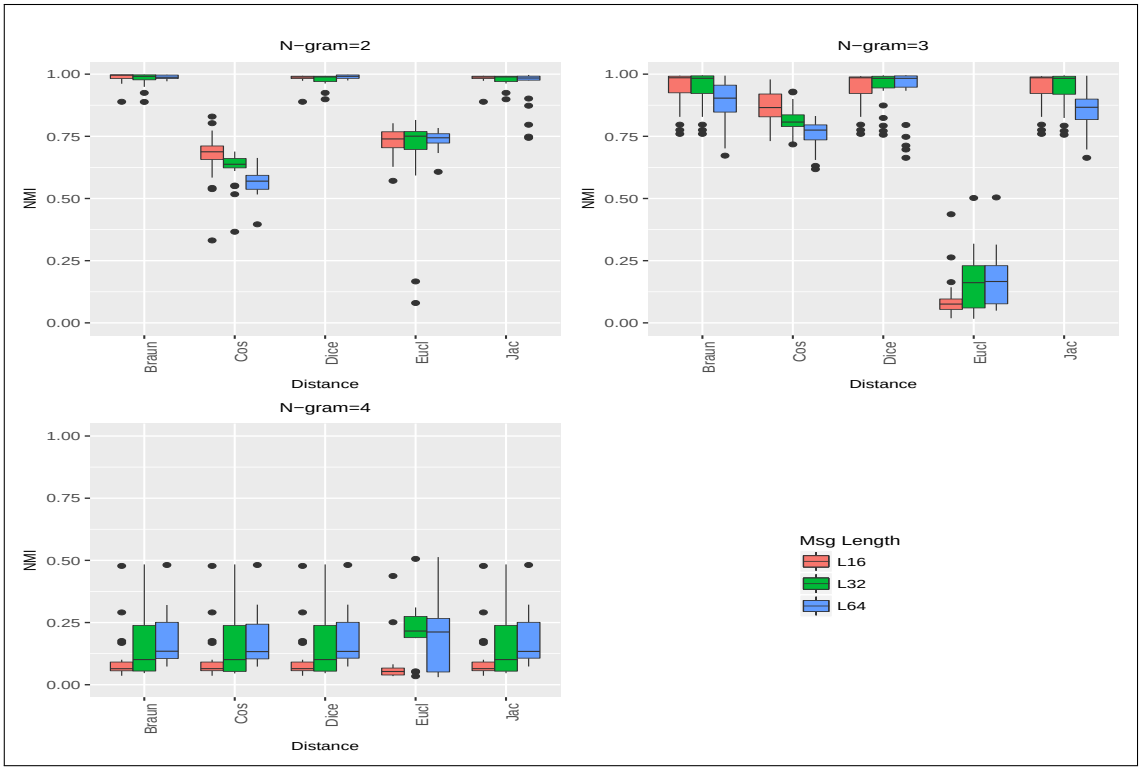
Appendices

A Performance of External Clustering Validation Measures

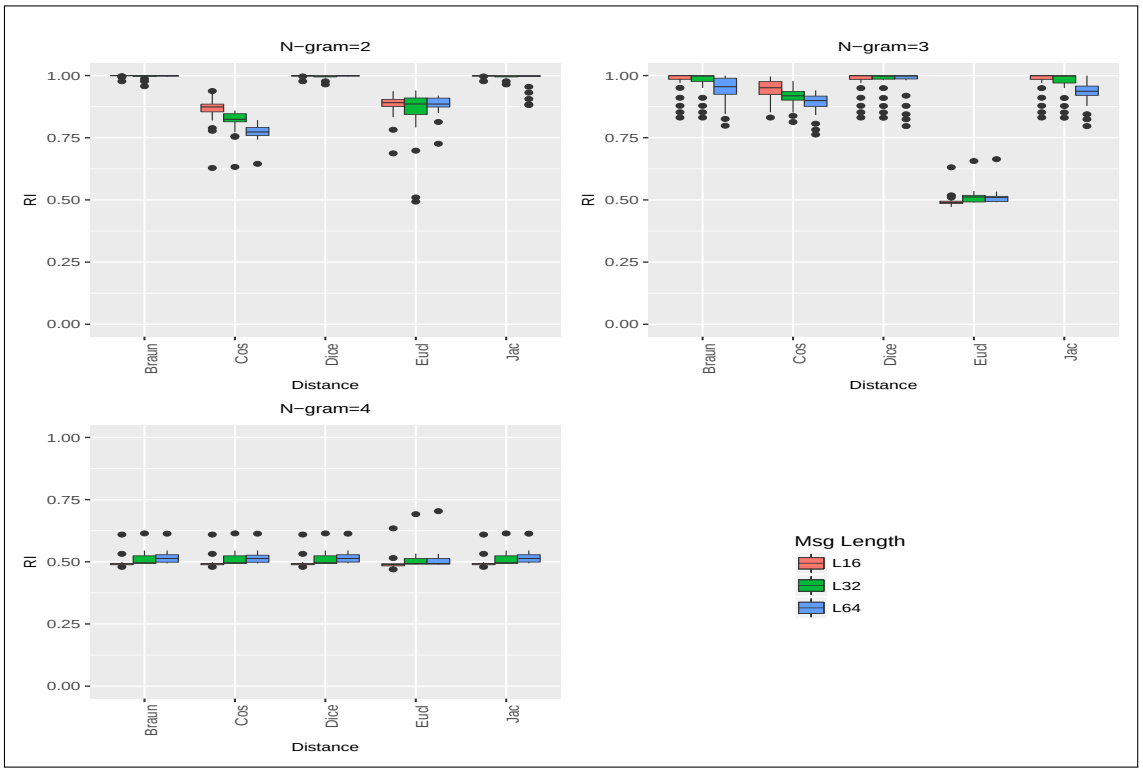
This appendix shows the performance of the external clustering validation measures used in the experiment explained in Chapter 5. The score of each index (ranges from 0 to 1) is shown against different factor configurations for each protocol sample.



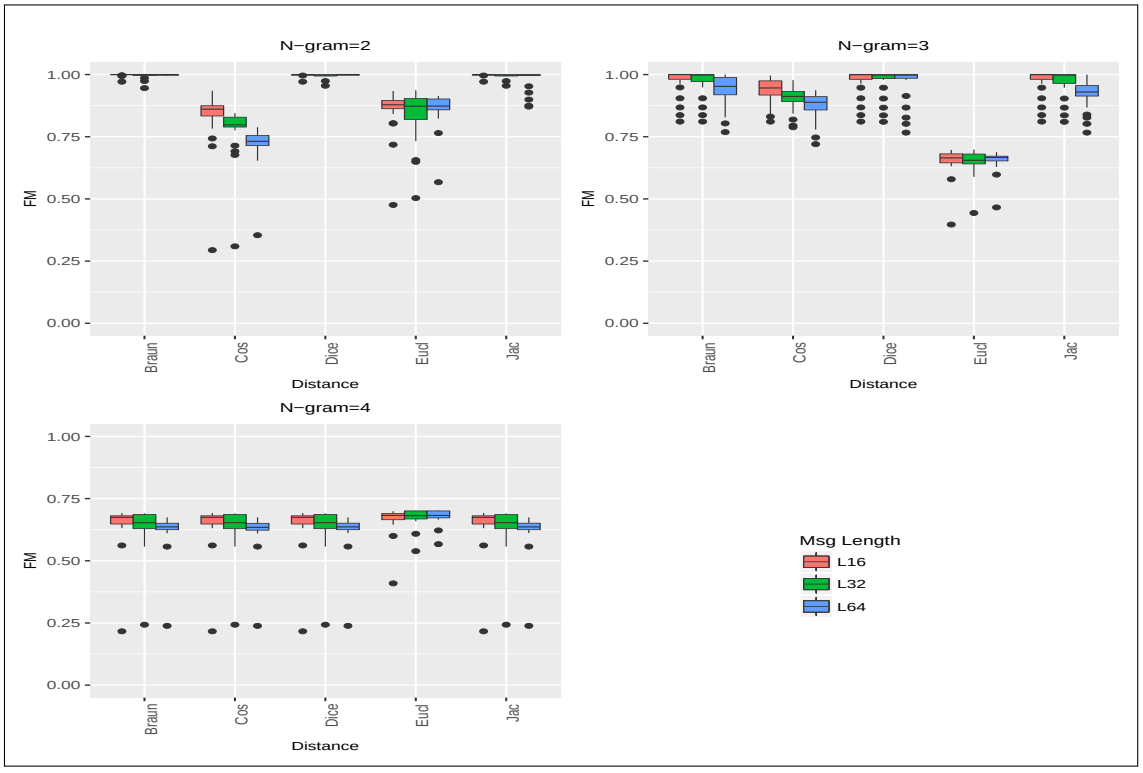
(a) Adjusted Rand Index (ARI)



(b) Normalised Mutual Information (NMI)

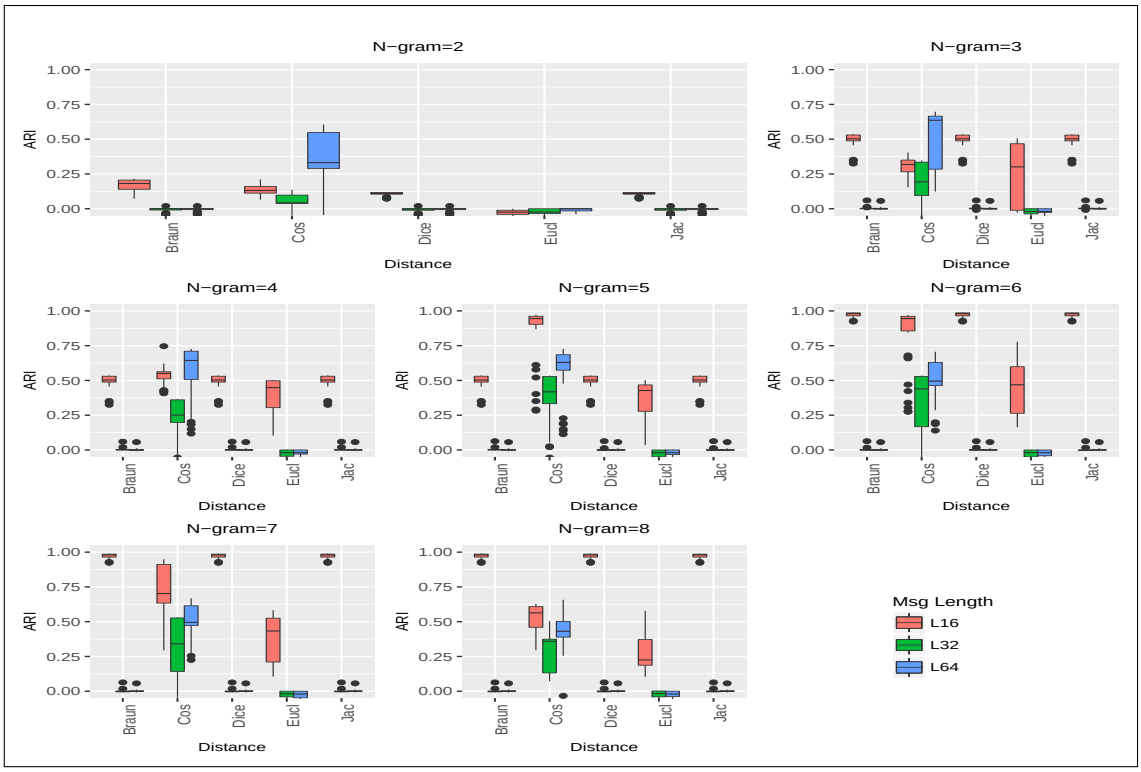


(c) Rand Index (RI)

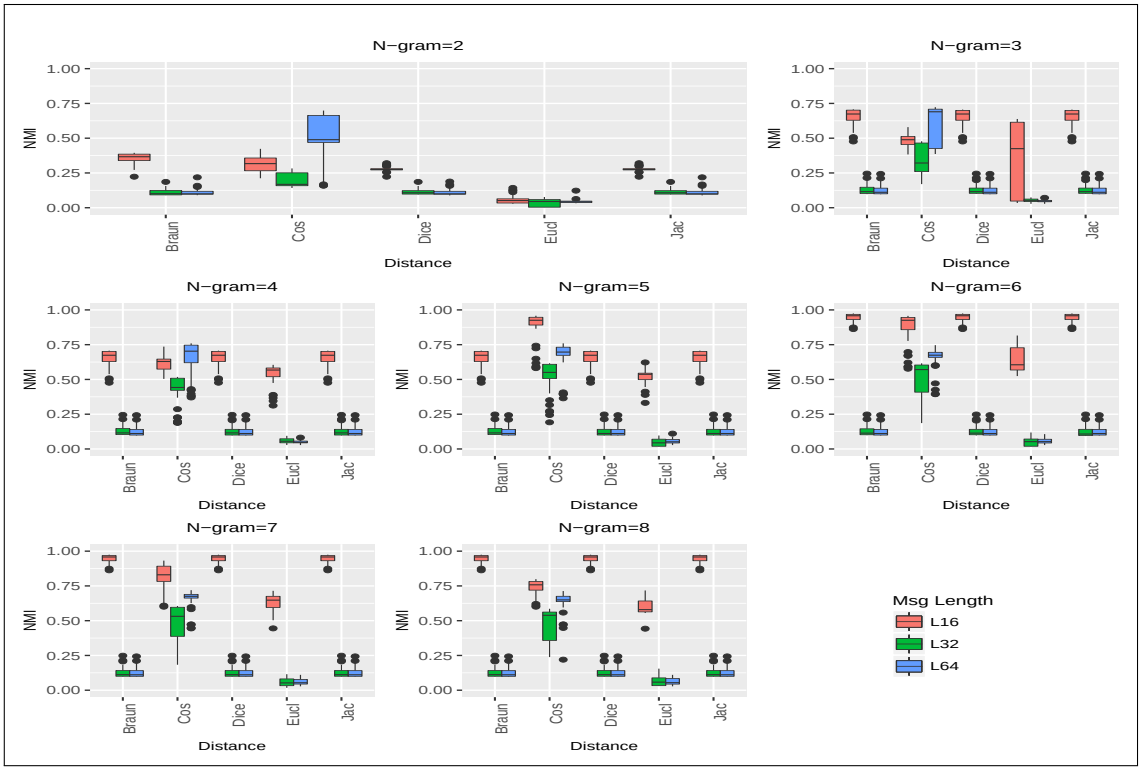


(d) Fowlkes-Mallows Index (FMI)

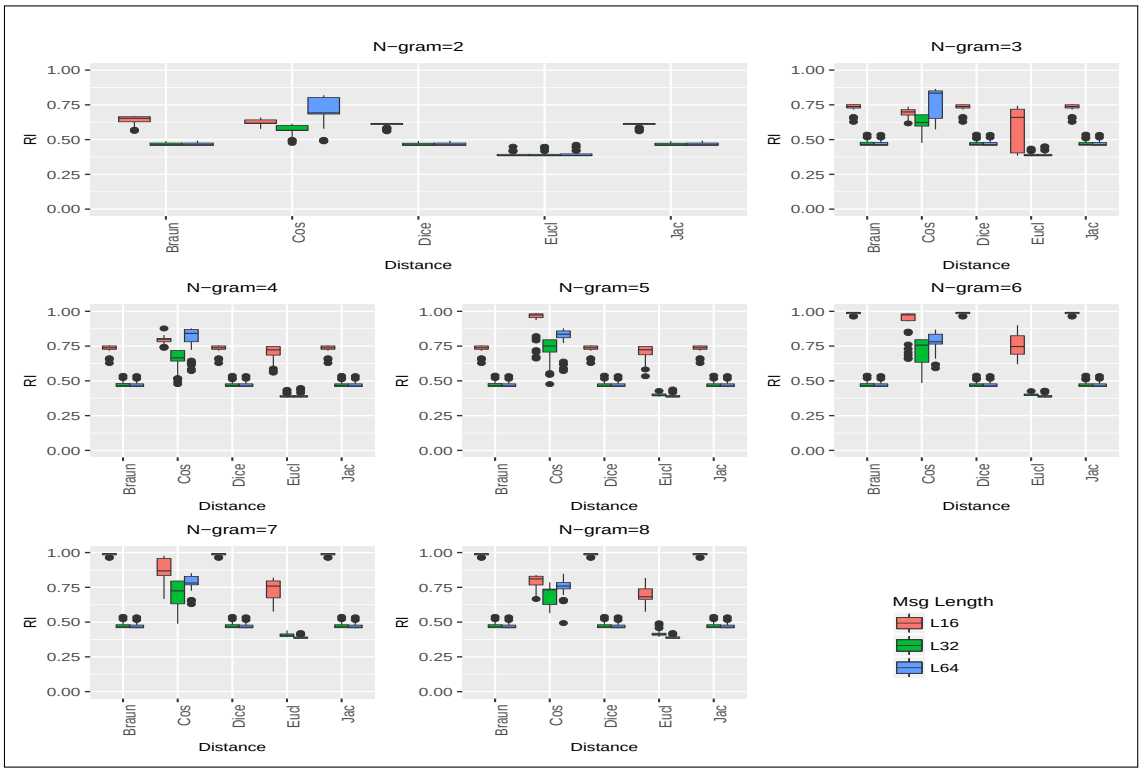
Figure A.1 Box plot comparison showing the performance of different external clustering validation measures for the TFTP protocol(a-d). Each plot shows the different configuration of factors and the correspondent clustering score of the chosen measure.



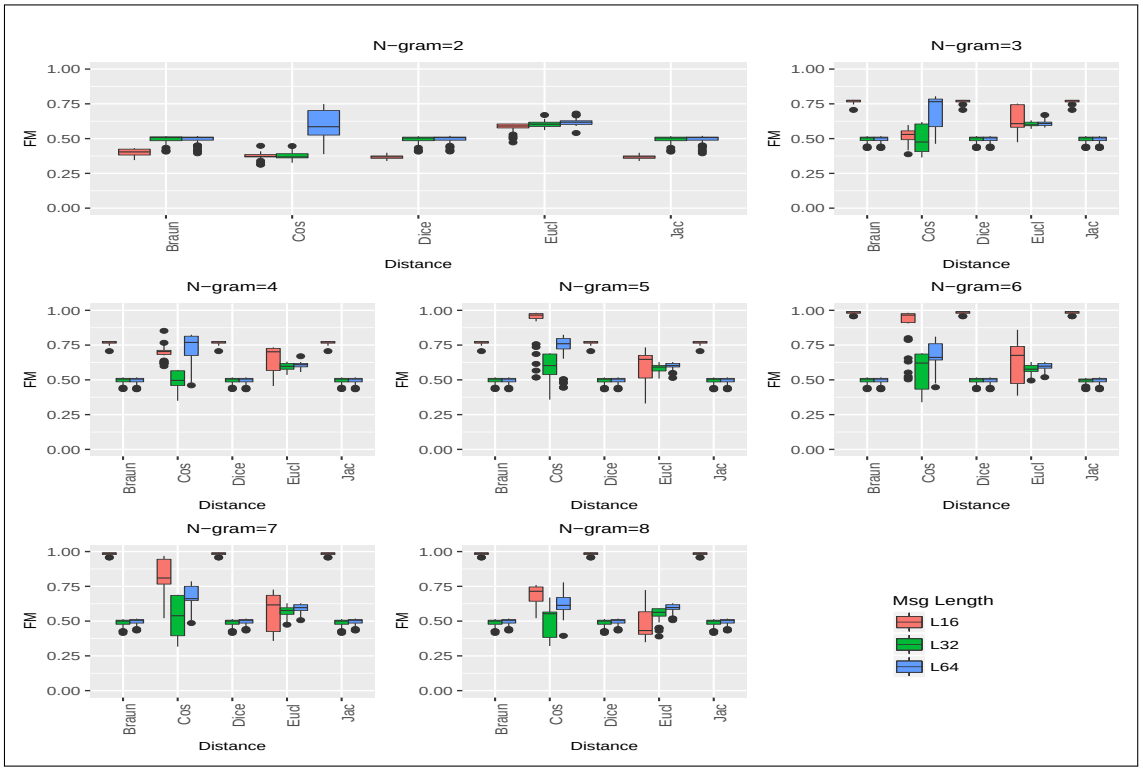
(a) Adjusted Rand Index (ARI)



(b) Normalised Mutual Information (NMI)

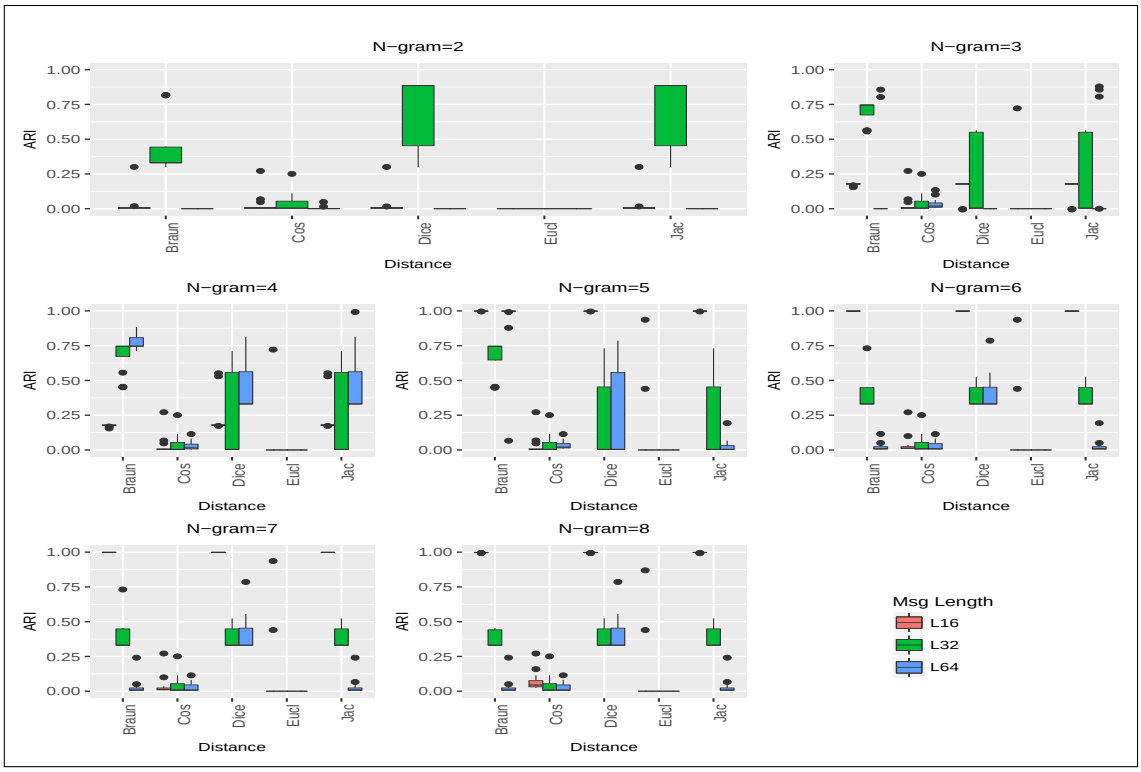


(c) Rand Index (RI)

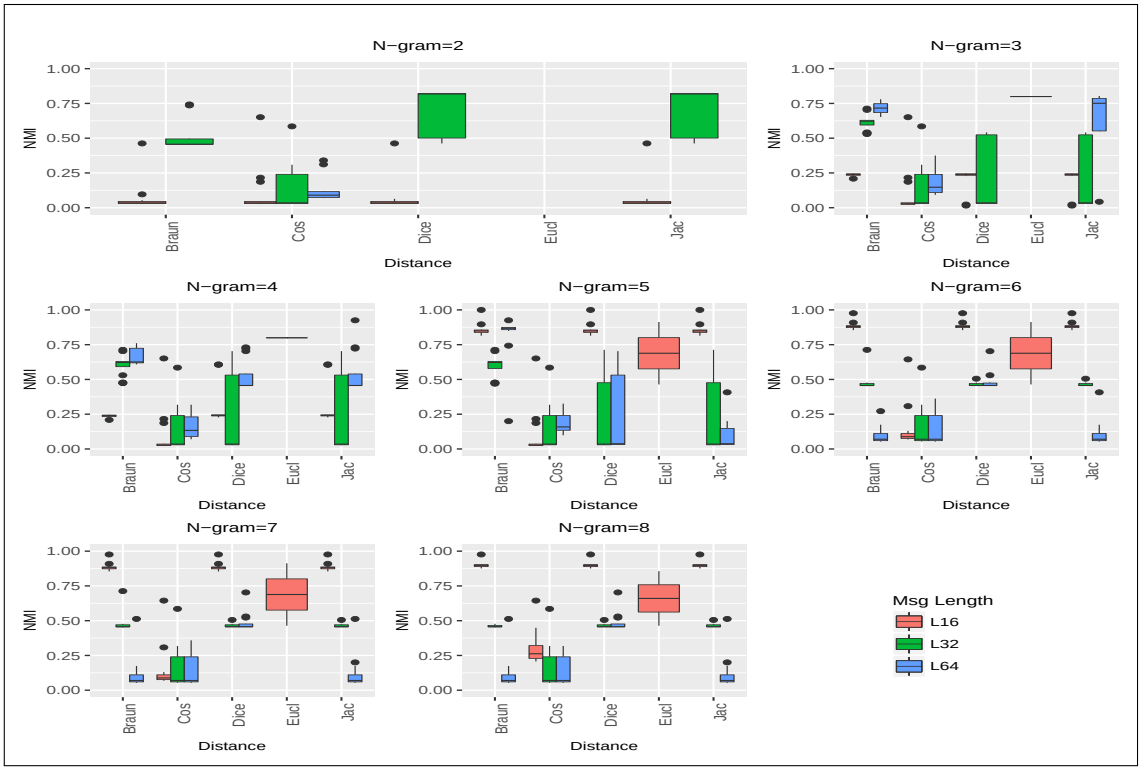


(d) Fowlkes-Mallows Index (FMI)

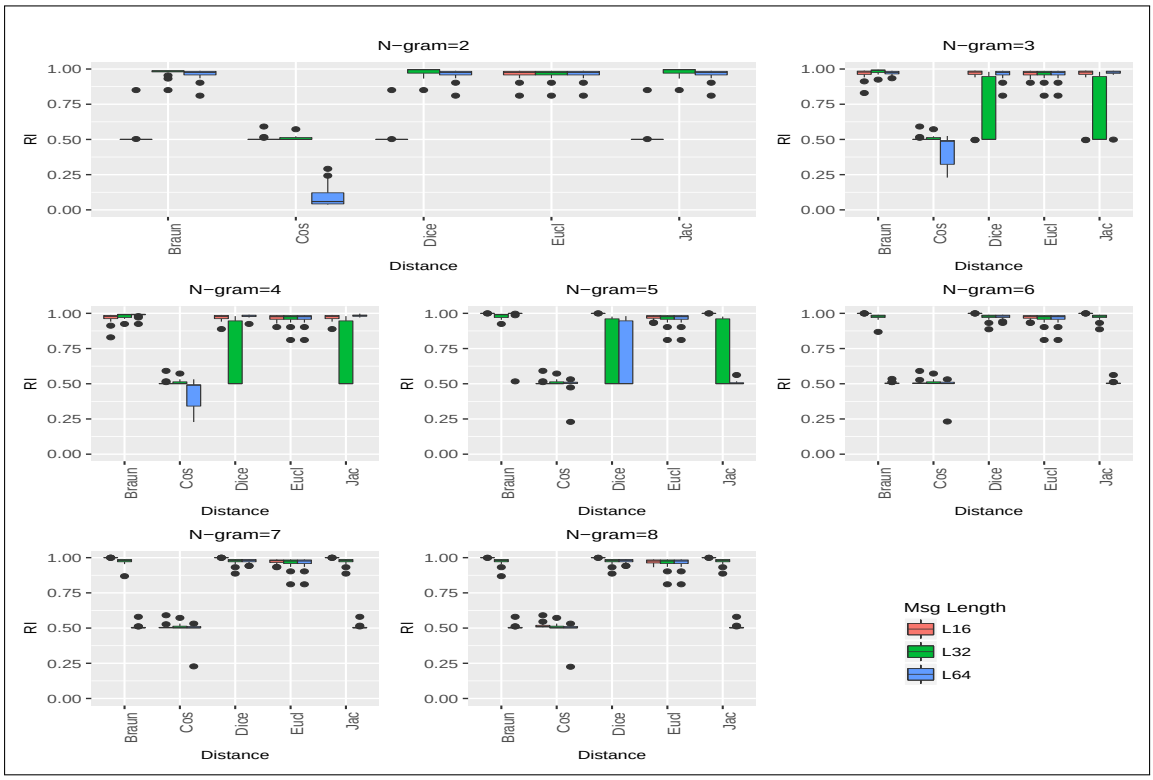
Figure A.2 Box plot comparison showing the performance of different external clustering validation measures for the DNS protocol (a-d). Each plot shows the different configuration of factors and the correspondent clustering score of the chosen measure.



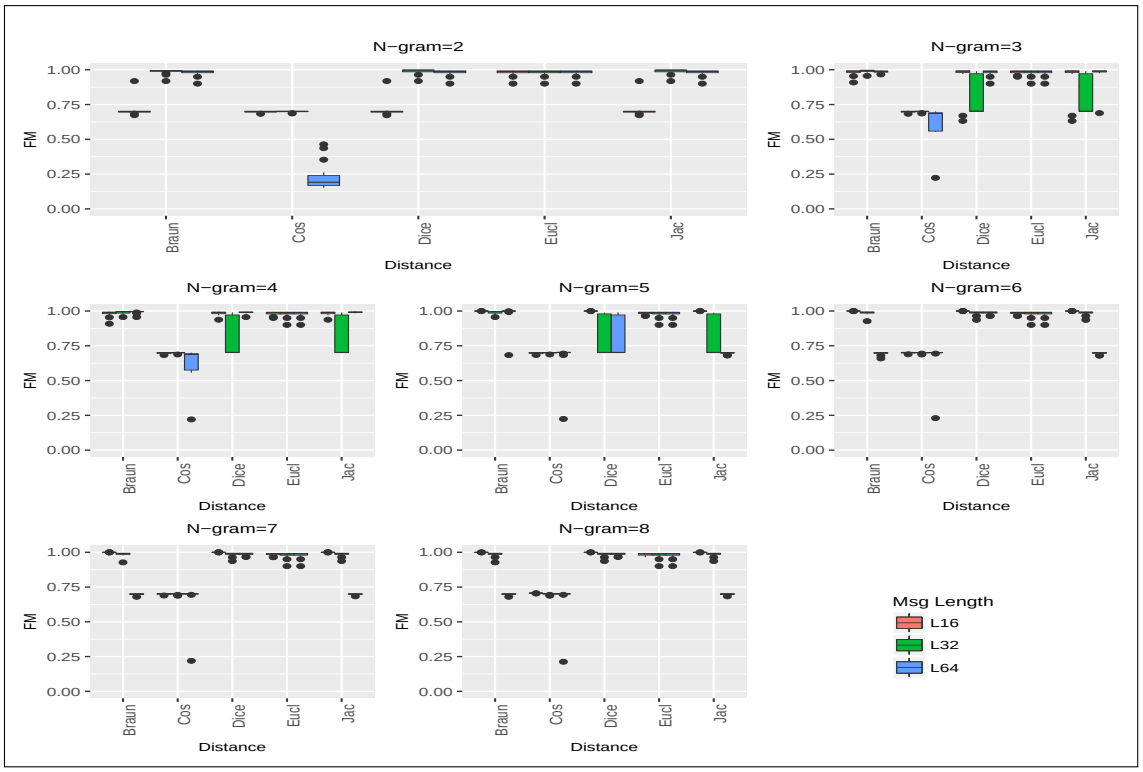
(a) Adjusted Rand Index (ARI)



(b) Normalised Mutual Information (NMI)

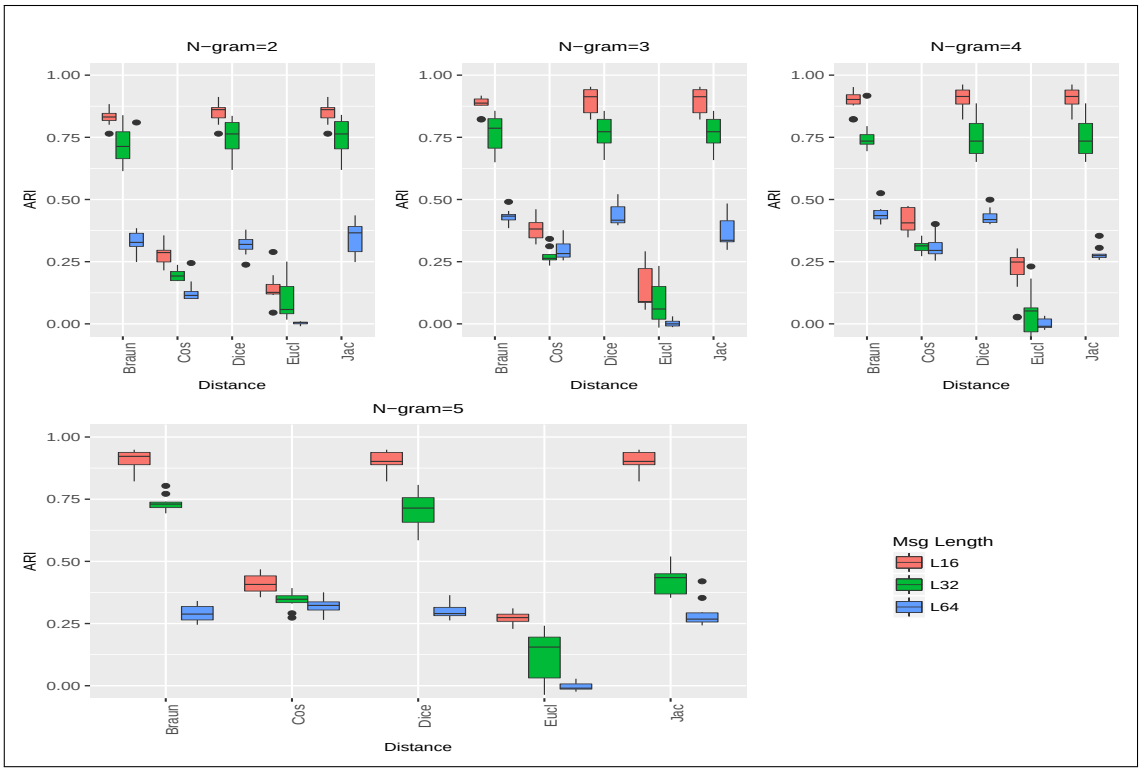


(c) Rand Index (RI)

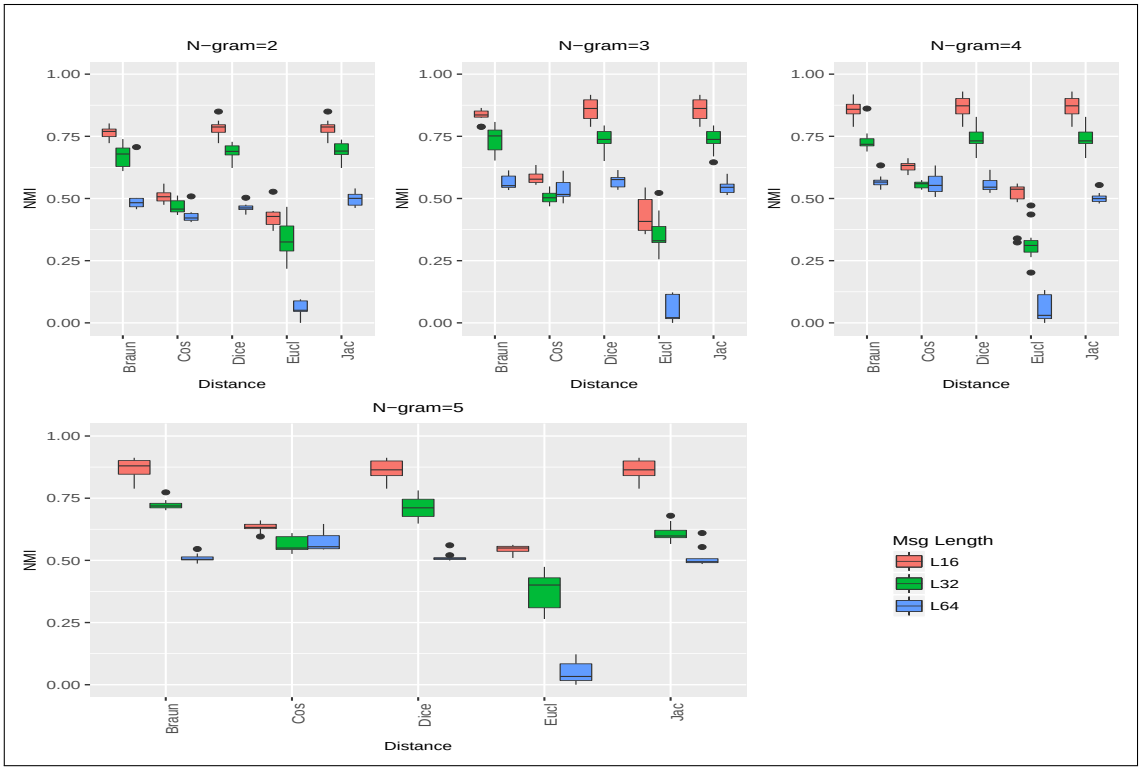


(d) Fowlkes-Mallows Index (FMI)

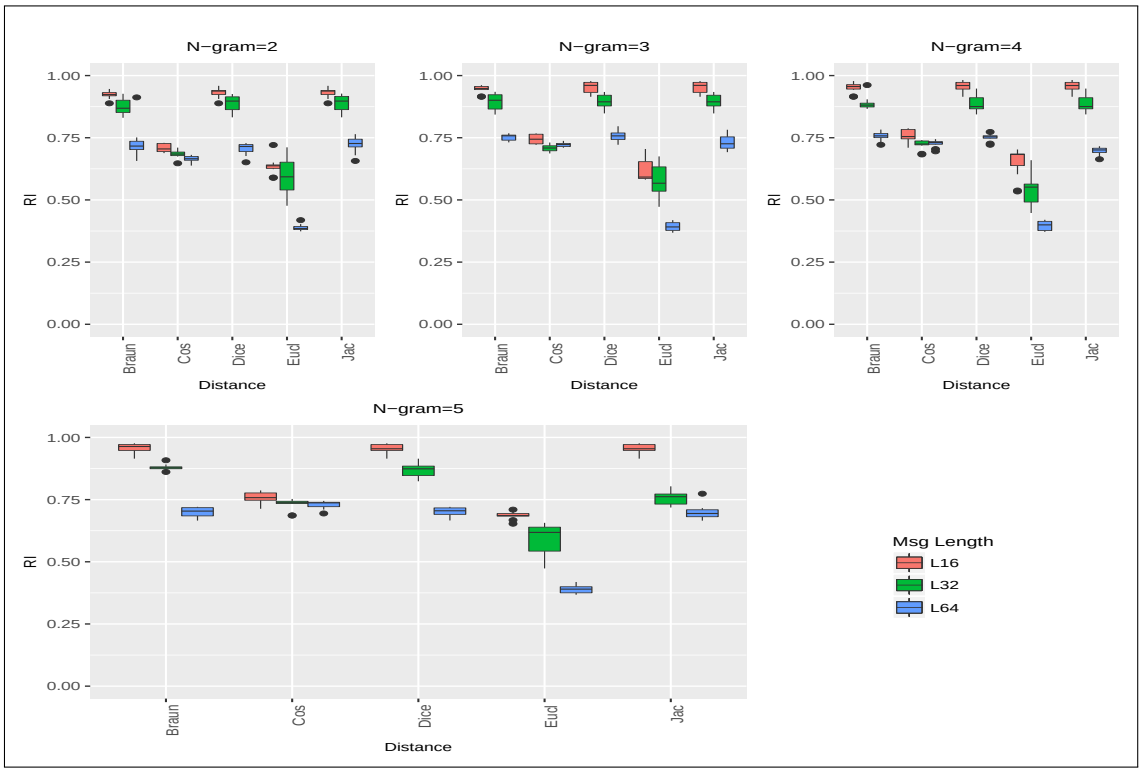
Figure A.3 Box plot comparison showing the performance of different external clustering validation measures for the SMB protocol (a-d). Each plot shows the different configuration of factors and the correspondent clustering score of the chosen measure.



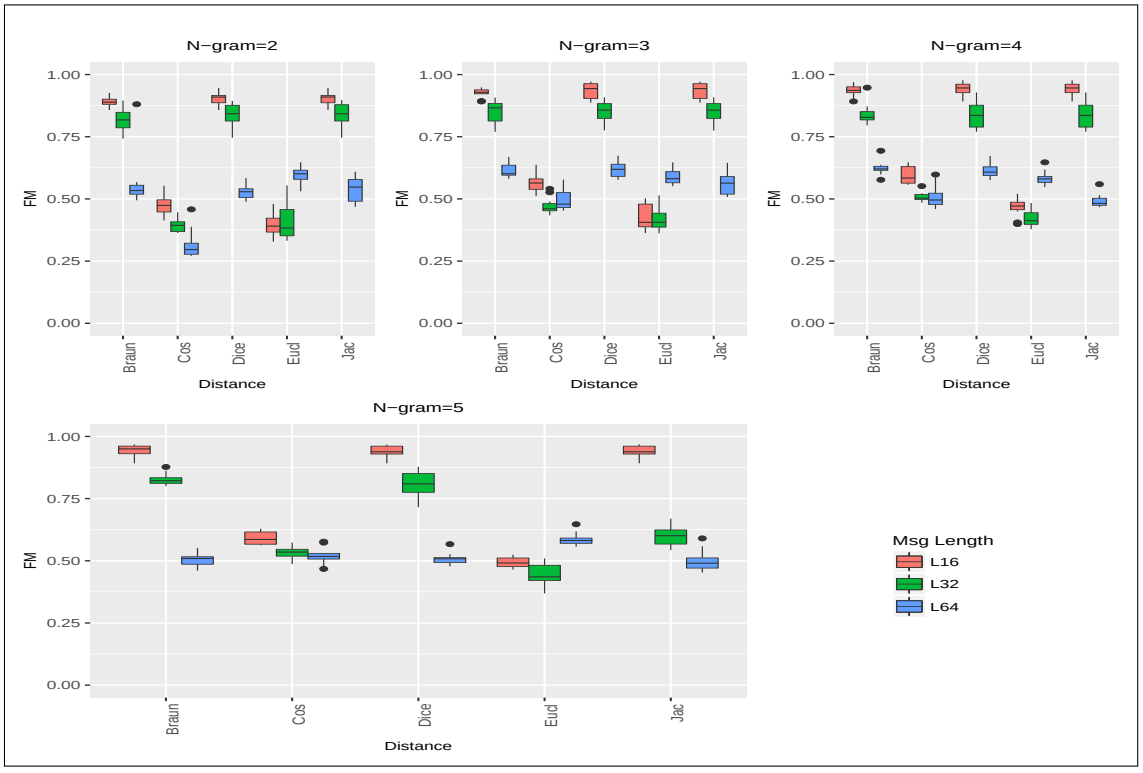
(a) Adjusted Rand Index (ARI)



(b) Normalised Mutual Information (NMI)



(c) Rand Index (RI)



(d) Fowlkes-Mallows Index (FMI)

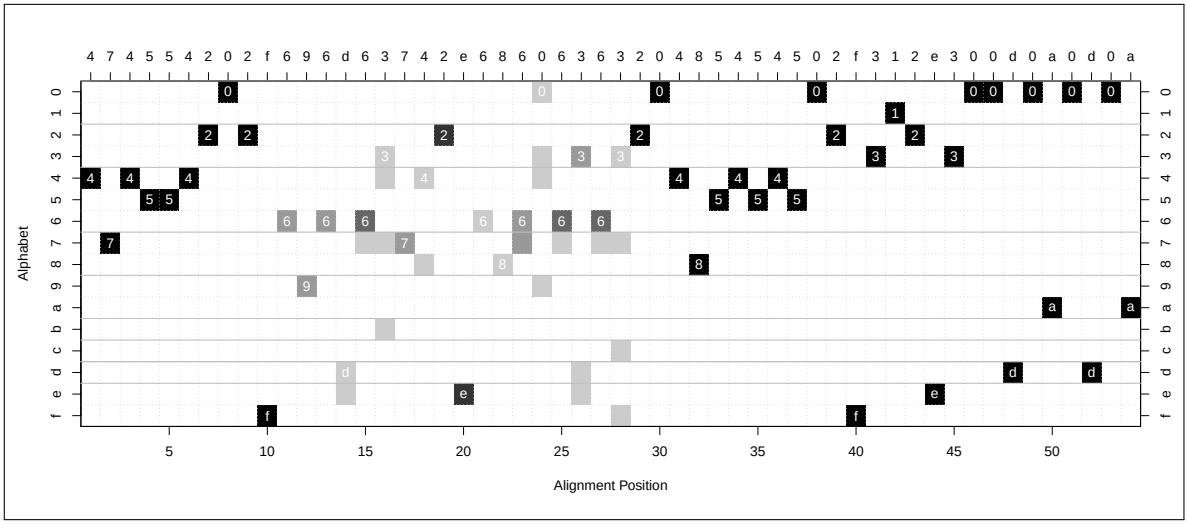
Figure A.4 Box plot comparison showing the performance of different external clustering validation measures for the HTTP protocol (a-d). Each plot shows the different configuration of factors and the correspondent clustering score of the chosen measure.

B Extracted message patterns in the hexadecimal format

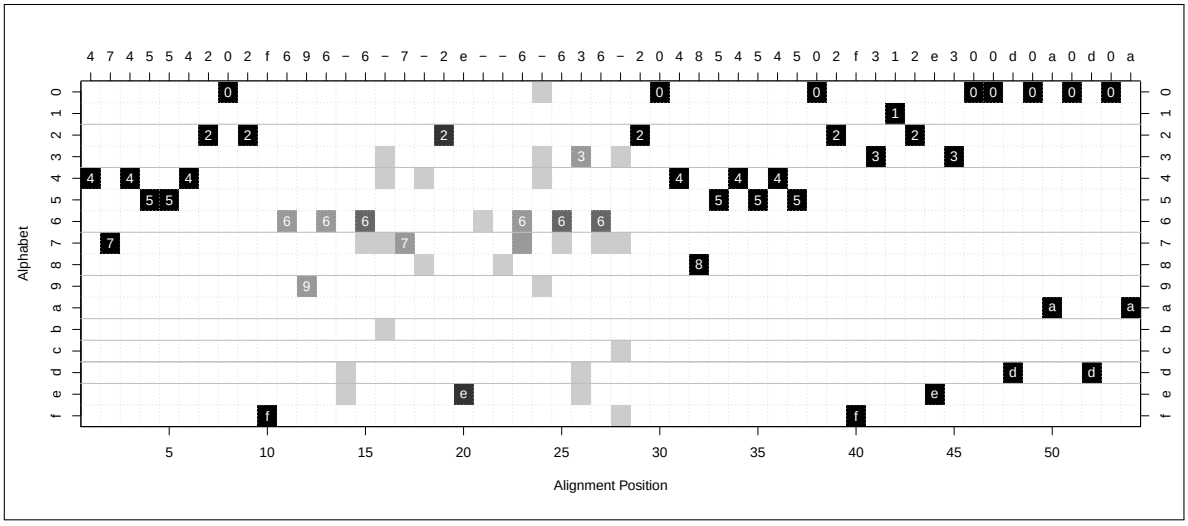
This appendix includes the Position Weighting Matrix (PWM) as well as the inferred message patterns from the alignment produced in the case study shown in Figure 6.6 (c). (see Chapter 6, Section 6.1.3). The example is also listed in Chapter 6, Table 6.3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	**1**	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	**1**	**1**	0	**1**	0	**1**	0	**1**	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	0	0	0	**0.8**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	1	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	**0.2**	0	**0.4**	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0
4	**1**	0	**1**	0	0	**1**	0	0	0	0	0	0	0	0	**0.2**	0	**0.2**	0	0	0	0	0	0	**0.2**	0	0	0	0	0	**1**	0	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	**0.4**	0	**0.4**	0	**0.6**	0	0	0	0	0	**0.2**	0	**0.4**	0	**0.6**	0	**0.6**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	**0.2**	**0.4**	0	0	0	0	0	**0.4**	0	**0.2**	0	**0.2**	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	**0.2**	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	**0.4**	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	**1**	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	**0.8**	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

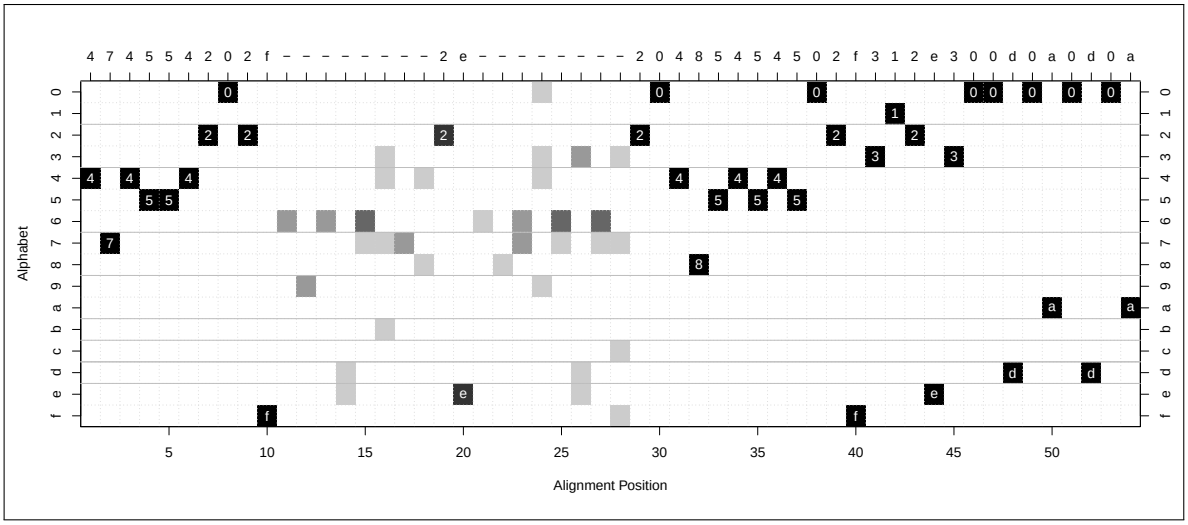
Table B.1 A Position Weight Matrix generated from the set of aligned HTTP messages in hexadecimal format introduced in chapter 5. It shows on the top the number of columns (positions) and the expected hexadecimal alphabet as the matrix rows.


Alphabet
Alignment Position

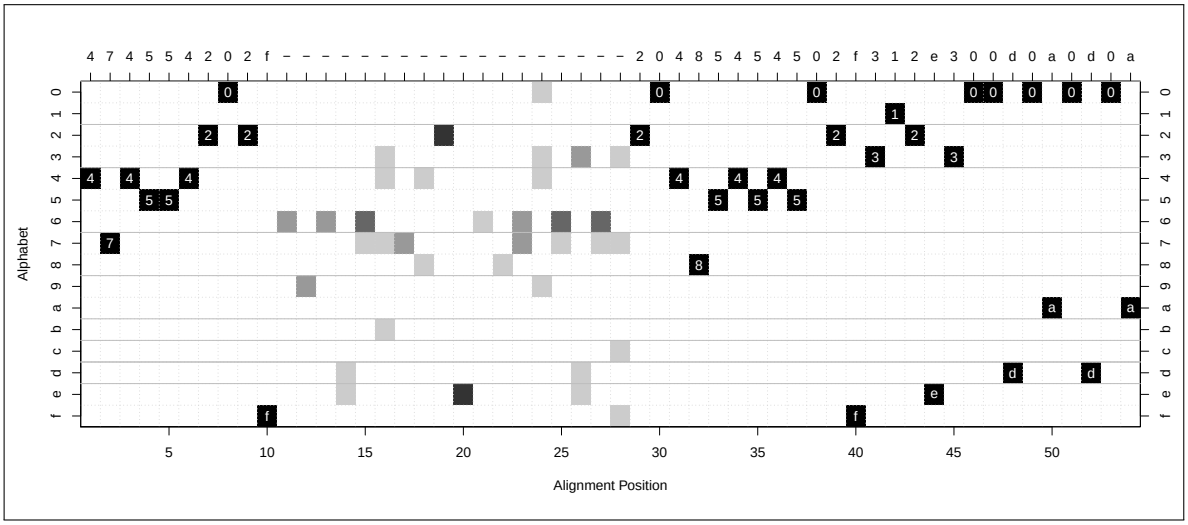
(a) $T = 0$


Alphabet
Alignment Position

(b) $T = 0.2$



(c) $T = 0.6$



(d) $T = 0.8$

Figure B.1 The extracted message patterns (in Hexadecimal) from the correspondent PWM using different values for the generalisation threshold (T).

Bibliography

[1] Sequitur: inferring hierarchies from sequences. http://www.sequitur.info/, 2017.

[2] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Advances in knowledge discovery and data mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[3] Dialign. http://dialign.gobics.de, 2017.

[4] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, December 1999.

[5] M. Sutton A. Greene and P. Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 2007.

[6] Oulu University Secure Programming Group. Protos: Security testing of protocol implementations. Technical report, University of Oulu, Electrical and Information Engineering, 1999.

[7] Dave Aitel. The advantages of block-based protocol analysis for security testing. Technical report, 2002.

[8] Juan Caballero, Min Gyung Kang, Shobha Venkataraman, Dawn Song, Pongsin Poosankam, and Avrim Blum. Fig: Automatic fingerprint generation. In *In 14th Annual Network and Distributed System Security Conference (NDSS*, 2007.

[9] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 313–326, New York, NY, USA, 2006. ACM.

[10] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoun Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 11:1–11:12, New York, NY, USA, 2008. ACM.

[11] David Brumley, Juan Caballero, Zhenkai Liang, James Newsome, and Dawn Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 15:1–15:16, Berkeley, CA, USA, 2007. USENIX Association.

[12] Ruoming Pang, Vern Paxson, Robin Sommer, and Larry Peterson. binpac: a yacc for writing application protocol parsers. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 289–300, New York, NY, USA, 2006. ACM.

[13] Microsoft. Microsoft smb protocol and cifs protocol overview. https://msdn.microsoft.com/en-gb/library/windows/desktop/aa365233(v=vs.85).aspx, 2016.

[14] Andrew Tridgell. How samba was written. http://www.samba.org/ftp/tridge/misc/french_cafe.txt, August 2003.

[15] File Transfer Protocol. RFC 959, October 1985.

[16] Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[17] Skype. https://www.skype.com, 2017.

[18] Icq. https://icq.com, 2017.

[19] D. Dagon, Guofei Gu, C.P. Lee, and Wenke Lee. A taxonomy of botnet structures. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 325–339, 2007.

[20] Gregory Fedynyshyn, Mooi Choo Chuah, and Gang Tan. Detection and classification of different botnet c&c channels. In *Proceedings of the 8th International Conference on Autonomic and Trusted Computing*, ATC'11, pages 228–242, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 621–634, New York, NY, USA, 2009. ACM.

[22] Juan Caballero and Dawn Song. Automatic protocol reverse-engineering: Message format extraction and field semantics inference. *Comput. Netw.*, 57(2):451–474, February 2013.

[23] Yipeng Wang, Xiao chun Yun, Muhammad Zubair Shafiq, Liyan Wang, Alex X. Liu, Zhibin Zhang, Danfeng Yao, Yongzheng Zhang 0002, and Li Guo. A semantics aware approach to automated reverse engineering unknown protocols. In *ICNP*, pages 1–10. IEEE, 2012.

[24] Marshall A. Beddoe. Network protocol analysis using bioinformatics algorithms. http://phreakocious.net/PI, 2004.

[25] M. Shevertalov and S. Mancoridis. A reverse engineering tool for extracting protocols of networked applications. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, pages 229–238, 2007.

[26] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. Discoverer: automatic protocol reverse engineering from network traces. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 14:1–14:14, Berkeley, CA, USA, 2007. USENIX Association.

[27] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 110–125, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, December 2001.

[29] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[30] Rui Fa, David J. Roberts, and Asoke K. Nandi. Smart: Unique splitting-while-merging framework for gene clustering. *PLoS ONE*, 9:e94141, 2014.

[31] Corrado Leita, Ken Mermoud, and Marc Dacier. Scriptgen: an automated script generation tool for honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 203–214, Washington, DC, USA, 2005. IEEE Computer Society.

[32] Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy Katz. Protocol independent adaptive replay of application dialog. Feburary 2006.

[33] Corrado Leita, Marc Dacier, and Frederic Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with scriptgen based honeypots. In *Proceedings of the 9th international conference on Recent Advances in Intrusion Detection*, RAID'06, pages 185–205, Berlin, Heidelberg, 2006. Springer-Verlag.

[34] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *Network and Distributed Systems Security Symposium (NDSS)*, 2008.

[35] Chia Yuan Cho, Domagoj Babi ć, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 426–439, New York, NY, USA, 2010. ACM.

[36] Yipeng Wang, Zhibin Zhang, Danfeng Daphne Yao, Buyun Qu, and Li Guo. Inferring protocol state machine from network traces: a probabilistic approach. In *Proceedings of the 9th international conference on Applied cryptography and network security*, ACNS'11, pages 1–18, Berlin, Heidelberg, 2011. Springer-Verlag.

[37] Tammo Krueger, Hugo Gascon, Nicole Krämer, and Konrad Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, AISec '12, pages 37–48, New York, NY, USA, 2012. ACM.

[38] Saul Ben Needleman and Christian Dennis Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[39] Burkhard Morgenstern. Multiple dna and protein sequence alignment based on segment-to-segment comparison. In *Proc. Natl. Acad. Sci. USA*, pages 12098–12103, 1996.

[40] Othman Esoul and Neil Walkinshaw. Using segment-based alignment to extract packet structures from network traces. In *2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017, Prague, Czech Republic, July 25-29, 2017*, pages 398–409, 2017.

[41] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 5th edition, 2011.

[42] Alok Sinha. Client-server computing. *Commun. ACM*, 35(7):77–98, July 1992.

[43] H. Zimmermann. Innovations in internetworking. chapter OSI Reference Model; The ISO Model of Architecture for Open Systems Interconnection, pages 2–9. Artech House, Inc., Norwood, MA, USA, 1988.

[44] Michelle S. Cotton, Lars Eggert, Dr. Joseph D. Touch, Magnus Westerlund, and Stuart Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335, August 2011.

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[46] The tftp protocol (revision 2). RFC 1350, March 2013.

[47] Domain names - implementation and specification. RFC 1035, November 1987.

[48] Huan Liu and Hiroshi Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[49] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[50] M. Dash, H. Liu, and J. Yao. Dimensionality reduction of unsupervised data. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pages 532–539, Nov 1997.

[51] M. Dash, K. Choi, P. Scheuermann, and Huan Liu. Feature selection for clustering - a filter solution. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 115–122, 2002.

[52] Jennifer G. Dy and Carla E. Brodley. Feature selection for unsupervised learning. *J. Mach. Learn. Res.*, 5:845–889, December 2004.

[53] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. In *Advances in Neural Information Processing Systems 18*, pages 507–514. MIT Press, Cambridge, MA, 2006.

[54] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, October 2002.

[55] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 73–84, New York, NY, USA, 1998. ACM.

[56] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[57] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[58] Konrad Rieck and Pavel Laskov. Linear-time computation of similarity measures for sequential data. *J. Mach. Learn. Res.*, 9:23–48, June 2008.

[59] J. C. Dunn. Well separated clusters and optimal fuzzy-partitions. *Journal of Cybernetics*, 4:95–104, 1974.

[60] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2):224–227, February 1979.

[61] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 911–916, Dec 2010.

[62] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[63] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011.

[64] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.

[65] Muhammad Tariq Pervez, Masroor Ellahi Babar, Asif Nadeem, Muhammad Aslam, Ali Raza Awan, Naeem Aslam, Tanveer Hussain, Nasir Naveed, Salman Qadri, Usman Waheed, and Muhammad Shoaib. Evaluating the accuracy and efficiency of multiple sequence alignment methods. *Evolutionary Bioinformatics*, 10:205–217, 12 2014.

[66] TD Schneider. Consensus sequence zen. *Applied Bioinformatics*, 1(3):111–119, 2002.

[67] Thomas D Schneider and R Michael Stephens. Sequence logos: a new way to display consensus sequences. *Nucleic acids research*, 18(20):6097–6100, 1990.

[68] Burkhard Morgenstern, William R Atchley, Klaus Hahn, and Andreas WM Dress. Segment-based scores for pairwise and multiple sequence alignments. In *Ismb*, pages 115–121, 1998.

[69] A. R. Subramanian, M. Kaufmann, and B. Morgenstern. Dialign-tx: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms for molecular biology : AMB*, 3(1):6+, May 2008.

[70] Burkhard Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, March 1999.

[71] Adrian J. Gibbs and George A. Mcintyre. The diagram, a method for comparing sequences. *European Journal of Biochemistry*, 16(1):1–11, 1970.

[72] B Morgenstern, K Frech, A Dress, and T Werner. Dialign: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290, 1998.

[73] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[74] Ian Korf, Mark Yandell, and Joseph Bedell. *BLAST*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[75] Burkhard Morgenstern. A space-efficient algorithm for aligning large genomic sequences. *Bioinformatics*, 16(10):948, 2000.

[76] B. Morgenstern. A simple and space-efficient fragment-chaining algorithm for alignment of dna and protein sequences. *Applied Mathematics Letters*, 15(1):11 – 16, 2002.

[77] Stephen F. Altschul and Bruce W. Erickson. A nonlinear measure of subalignment similarity and its significance levels. *Bulletin of Mathematical Biology*, 48(5):617 – 632, 1986.

[78] A. R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. Dialign-t: An improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6(1):66+, 2005.

[79] Samuel Karlin and Stephen F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences of the United States of America*, 87(6):pp. 2264–2268, 1990.

[80] Saïd Abdeddaïm and Burkhard Morgenstern. *Speeding Up the DIALIGN Multiple Alignment Program by Using the 'Greedy Alignment of BIOlogical Sequences LIBrary' (GABIOS-LIB)*, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[81] S. Boslaugh and P.A. Watters. *Statistics in a Nutshell: A Desktop Quick Reference*. In a Nutshell (O'Reilly). O'Reilly Media, 2008.

[82] Well A. D. Myers J. L. and Lourch R .F. *Research Design and Statistical Analysis*. Routledge, 3rd ed. edition, 2010.

[83] W.R. Shadish, T.D. Cook, and D.T. Campbell. *Experimental and Quasi-experimental Designs for Generalized Causal Inference*. Houghton Mifflin, 2002.

[84] Guido W. Imbens and Donald B. Rubin. *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction*. Cambridge University Press, New York, NY, USA, 2015.

[85] R.J. Grissom and J.J. Kim. *Effect Sizes for Research: A Broad Practical Approach*. Lawrence Erlbaum Associates, 2005.

[86] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. New York:Academic Press, 2nd ed. edition, 1988.

[87] J. Higgins M. Borenstein, L. Hedges and H. Rothstein. *Introduction to Meta-Analysis*. Wiley Online Library, March 2009.

[88] Gordon Fraser and Neil Walkinshaw. Assessing and generating test sets in terms of behavioural adequacy. *Softw. Test. Verif. Reliab.*, 25(8):749–780, December 2015.

[89] J. Cohen. Quantitative methods in psychology: A power primer. *Psychological Bulletin*, 112(1):155–159, 1992.

[90] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 317–329, New York, NY, USA, 2007. ACM.

[91] Juan Caballero and Dawn Song. Rosetta: Extracting protocol semantics using binary analysis with applications to protocol replay and nat rewriting. Technical Report CMU-CyLab-07-014, Cylab, Carnegie Mellon University, October 2007.

[92] Weidong Cui, Marcus Peinado, Karl Chen, Helen J. Wang, and Luis Irun-Briz. Tupni: automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 391–402, New York, NY, USA, 2008. ACM.

[93] Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, and Mike Grace. Reformat: automatic reverse engineering of encrypted messages. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 200–215, Berlin, Heidelberg, 2009. Springer-Verlag.

[94] Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *NDSS*, 2008.

[95] Yongjun He, Hui Shu, and Xiaobing Xiong. Protocol reverse engineering based on dynamorio. In *Proceedings of the 2009 International Conference on Information and Multimedia Technology*, ICIMT '09, pages 310–314, Washington, DC, USA, 2009. IEEE Computer Society.

[96] João Antunes and Nuno Neves. Automatically complementing protocol specifications from network traces. In *Proceedings of the 13th European Workshop on Dependable Computing*, EWDC '11, pages 87–92, New York, NY, USA, 2011. ACM.

[97] Joao Antunes, Nuno Neves, and Paulo Verissimo. Reverse engineering of protocols from network traces. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 169–178, Washington, DC, USA, 2011. IEEE Computer Society.

[98] Jayanthkumar Kannan, Jaeyeon Jung, Vern Paxson, and Can Emre Koksal. Semi-automated discovery of application session structure. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 119–132, New York, NY, USA, 2006. ACM.

[99] Yipeng Wang, Xingjian Li, Jiao Meng, Yong Zhao, Zhibin Zhang, and Li Guo. Biprominer: Automatic mining of binary protocol features. In *Proceedings of the 2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '11, pages 179–184, Washington, DC, USA, 2011. IEEE Computer Society.

[100] Antonio Trifilo, Stefan Burschka, and Ernst Biersack. Traffic to protocol reverse engineering. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 257–264, Piscataway, NJ, USA, 2009. IEEE Press.

[101] Tammo Krueger, Nicole Krämer, and Konrad Rieck. *ASAP: Automatic Semantics-Aware Analysis of Network Payloads*, pages 50–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[102] Georges Bossert, Frédéric Guihéry, and Guillaume Hiet. Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 51–62, New York, NY, USA, 2014. ACM.

[103] Patrice Godefroid, Michael Y. Levin, and David A Molnar. Automated whitebox fuzz testing. In *Network Distributed Security Symposium (NDSS)*. Internet Society, 2008.

[104] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 87–95, New York, NY, USA, 2000. ACM.

[105] James Newsome, David Brumley, Jason Franklin, and Dawn Song. Replayer: automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 311–321, New York, NY, USA, 2006. ACM.

[106] Nikita Borisov, David J. Brumley, and Helen J. Wang. A generic application-level protocol analyzer and its language. In *In 14th Annual Network & Distributed System Security Symposium*, 2007.

[107] XiangDong Li and Li Chen. A survey on methods of automatic protocol reverse engineering. In *Proceedings of the 2011 Seventh International Conference on Computational Intelligence and Security*, CIS '11, pages 685–689, Washington, DC, USA, 2011. IEEE Computer Society.

[108] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. A survey of automatic protocol reverse engineering tools. *ACM Comput. Surv.*, 48(3):40:1–40:26, December 2015.

[109] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

[110] Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum. Understanding data lifetime via whole system simulation. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 22–22, Berkeley, CA, USA, 2004. USENIX Association.

[111] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. Bitblaze a new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security.*, Hyderabad, India, Dec 2008.

[112] Nicholas Nethercote and Julian Seward. Valgrind a framework for heavyweight dynamic binary instrumentation. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '07, pages 89–100, New York, NY, USA, 2007. ACM.

[113] Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. Dynamic spyware analysis. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 18:1–18:14, Berkeley, CA, USA, 2007. USENIX Association.

[114] Ming-Ming Xiao, Shun-Zheng Yu, and Yu Wang. Automatic network protocol automaton extraction. In *Proceedings of the 2009 Third International Conference on Network and System Security*, NSS '09, pages 336–343, Washington, DC, USA, 2009. IEEE Computer Society.

[115] Michael D. Ernst. Static and dynamic analysis: Synergy and duality. In *WODA 2003: Workshop on Dynamic Analysis*, pages 24–27, Portland, Oregon, May 2003.

[116] Ignacio Bermudez, Alok Tongaonkar, Marios Iliofotou, Marco Mellia, and Maurizio M. Munafò. Towards automatic protocol field inference. *Comput. Commun.*, 84(C):40–51, June 2016.

[117] Tcpdump Public Repository. Tcpdump and libpcap. http://www.tcpdump.org/, 2012.

[118] Angela Orebaugh, Gilbert Ramirez, Josh Burke, and Larry Pesce. *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[119] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 121–134, New York, NY, USA, 2004. ACM.

[120] Christian Kreibich and Jon Crowcroft. Efficient sequence alignment of network traffic. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 307–312, New York, NY, USA, 2006. ACM.

[121] R Core Team. *R: A Language and Environment for Statistical Computing*. The R Foundation for Statistical Computing, Vienna, Austria, 2016.

[122] Steven McCanne and Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.

[123] Felix Fuentes and Dulal C. Kar. Ethereal vs. tcpdump: A comparative study on packet sniffing tools for educational purpose. *J. Comput. Sci. Coll.*, 20(4):169–176, Apr 2005.

[124] NETRESEC. Network forensics and network security monitoring. http://www.netresec.com/, 2017.

[125] WireShark. Libpcap file format. https://wiki.wireshark.org/Development/LibpcapFileFormat, 2004.

[126] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[127] Wei Lu, Goaletsa Rammidi, and Ali A. Ghorbani. Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, 34(3):502 – 514, 2011. Special Issue of Computer Communications on Information and Future Communication Security.

[128] Ingo Feinerer, Kurt Hornik, and David Meyer. Text mining infrastructure in r. *Journal of Statistical Software*, 25(5):1–54, March 2008.

[129] Ingo Feinerer, Kurt Hornik, and David Meyer. Text mining infrastructure in r. *Journal of Statistical Software, Articles*, 25(5):1–54, 2008.

[130] Kurt Hornik, Christian Buchta, and Achim Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232, 2009.

[131] David Meyer and Christian Buchta. *proxy: Distance and Similarity Measures*, 2015. R package version 0.4-15.

[132] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. ISBN 3-900051-07-0.

[133] Loic Pefferkorn. Ipdecap. https://loicpefferkorn.net/ipdecap, 2017.

[134] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1073–1080, New York, NY, USA, 2009. ACM.

[135] William M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, December 1971.

[136] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von mises-fisher distributions. *J. Mach. Learn. Res.*, 6:1345–1382, December 2005.

[137] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

[138] G. Ball and D. Hall. Isodata: A novel method of data analysis and pattern classification. Technical report, Stanford Research Institute, Menlo Park, 1965.

[139] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-Simulation and Computation*, 3(1):1–27, 1974.

[140] H.P. Friedman and J. Rubin. On some invariant criteria for grouping data. Journal of he American Statistical Associations, 62:1159-1178, 1967.

[141] M. Halkidi and M. Vazirgiannis. Clustering validity assessment: finding the optimal partitioning of a data set. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 187–194, 2001.

[142] Bernard Desgraupes. Clustercrit:clustering indices. Technical report, University of Paris Ouest - Lab Modal'X, 2016.

[143] Zhigang Li, Kunpeng Li, and Weijia Guo. On smart selection of clustering algorithms. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 8, pages 49–52, July 2010.

[144] WireShark. tshark:dump and analyze network traffic. https://www.wireshark.org/docs/man-pages/tshark.html, 2004.

[145] Richard G. Lawson and Peter C. Jurs. New index for clustering tendency and its application to chemical problems. *J. Chem. Inf. Comput. Sci.*, 30(1):36–41, January 1990.

[146] A. Banerjee and R. N. Dave. Validating clusters using the hopkins statistic. In *2004 IEEE International Conference on Fuzzy Systems (IEEE Cat. No.04CH37542)*, volume 1, pages 149–153 vol.1, July 2004.

[147] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[148] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, August 1999.

[149] Levent Ertoz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In Daniel Barbara and Chandrika Kamath, editors, *Proceedings of the Third SIAM International Conference on Data Mining (SDM 2003)*, volume 112 of *Proceedings in Applied Mathematics*. Society for Industrial and Applied Mathematics, 2003.

[150] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. Wiley, 1990.

[151] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[152] Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2018. R package version 2.0.7-1.

[153] Michael Hahsler and Matthew Piekenbrock. *dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms*, 2017. R package version 1.1-1.

[154] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[155] Xiujun Zhang. *Position Weight Matrices*, pages 1721–1722. Springer New York, New York, NY, 2013.

[156] Michael Eddington. Peach fuzzing platform. http://peachfuzzer.com, 2004.

[157] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol, 2002.

[158] Christopher Hertel. *Implementing CIFS: The Common Internet File System*. Prentice Hall Professional Technical Reference, 2003.

[159] Lighttpd. Lighttpd - fly light. https://www.lighttpd.net, 2017.

[160] Asterisk. Asterisk communications framework. http://www.asterisk.org, 2017.

[161] Jim Van Meggelen, Jared Smith, and Leif Madsen. *Asterisk™: The Future of Telephony, 2Nd Edition*. O'Reilly, second edition, 2007.

[162] TFTP. Ubuntu netkit-tftp package. https://launchpad.net/ubuntu/+source/netkit-tftp, 2017.

[163] Samba. Ubuntu samba package. https://packages.ubuntu.com/trusty/samba, 2017.

[164] Wireshark. Advanced topics: Expert information. https://www.wireshark.org/docs/wsug_html_chunked/ChAdvExpert.html, 2016.

[165] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Int. Res.*, 7(1):67–82, September 1997.

[166] Alexander Strehl and Joydeep Ghosh. Cluster ensembles: A knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, March 2003.

[167] A. Topchy, A. K. Jain, and W. Punch. Combining multiple weak clusterings. In *Third IEEE International Conference on Data Mining*, pages 331–338, Nov 2003.