Automatic Structural Inference of Binary Protocols

Fredrik Appelros Carl Ekerot

May 3, 2013

Abstract

Communication protocols are the foundation of everyday networking tasks but they are not always open in terms of available documentation. For reasons of interoperability or the collection of statistics some protocols need to be reverse engineered. This thesis presents a method based on density-based clustering and byte distribution classification for automatically inferring the structure of a protocol from a packet dump. We evaluate the precision of the method for finding different packet types and for finding fields, both globally across all packets and within the inferred packet types.

Contents

Chapter 1

Introduction

1.1 Background

Communication protocols are the foundation of everyday networking tasks. Each time a web page is served or an email is sent, a rigorous series of actions is taken to ensure that the requested piece of information is relayed. The information is packaged in a *message* and both the syntax and the semantics of these messages are defined by protocols.

Protocols may be open or closed in terms of available documentation. The reason for them being closed might be that they are proprietary or simply because no one invested in creating a documentation of the protocol in the first place. Despite the lack of a documentation there still sometimes exists a need to understand the inner workings of a certain protocol, e.g. when providing interoperability between services. One infamous example of this is Microsoft's proprietary SMB protocol which spawned the Samba project. This project was created in an attempt to provide access to certain Windows services for Unix-like operating systems. Tridgell (2003) explains that the Samba project reversed the SMB specification using techniques like *fuzzing*, a process where they inspect the response from a SMB server for a wide variety of generated requests. Using this approach, it took many years until most of the functionality of the SMB protocol had been implemented.

Another area where the need for inferring the structures of closed protocols is in *deep packet inspection* (DPI). Messages sent over a network are usually constructed hierarchically. At the bottom there is only raw binary data and at the top there is the application data, e.g. an email or a video. There in between exists several layers that contain metadata which is used to make sure that the data reaches its target and that it does not get corrupted during transmission. As a message is sent from one party to another it traverses a series of nodes in the network. These nodes only needs to look at the lower parts of the hierarchical structure of the message in order to relay it. If a node wants to conduct DPI however, the higher levels of the hierarchy is looked at as well. To be able to do this the structure of the protocol needs to be known.

There are many different reasons why one would want to perform DPI, one of which is to provide *quality of service* (QoS). QoS is needed in order to gather statistics for Internet service providers that manage the networks. One company

that delivers QoS solutions is Procera Networks on whose behalf we are writing this thesis.

In this thesis we have focused on finding the structure of binary protocols as opposed to textual ones. The reason for this being that textual protocols are intrinsically simple to read and understand and therefore do not require advanced methods of analysis. Some of the methods described here are however also applicable to textual protocols.

1.2 Related work

A popular approach to the problem of automatic inference of network protocols has been to draw a parallel to bioinformatics. Byte-streams have been processed using the same methodology as DNA sequence analysis in order to mine patterns in protocol messages. (Beddoe, 2005) explores the possibilities of using bioinformatics methods such as multiple sequence alignment in order to find patterns in both textual and binary protocols. These principles have been applied in e.g. Netzob[1], a tool for protocol analysis.

Using techniques from bioinformatics in protocol analysis seems natural since the analysis of messages, which may be viewed as the analysis of byte-streams, is easily converted to the problem of sequence analysis. Methods such as multiple sequence alignment does provide an insight to the structure of protocols, but further analysis is needed in order to find semantic meaning in segments of bytes.

Discoverer by Cui et al. is an approach which is specifically modelled after the behavior of network protocols. Discoverer tokenizes network dumps into textual and binary tokens. From the tokens Discoverer attempts to group messages based on their true message type, such as requests or responses. After the grouping of messages, Discoverer seeks semantic meaning in the tokens in common for each group of messages, and classifies the tokens into a set of predefined classes.

Caballero and Song (2012) proposes a method where protocol specifications are constructed by monitoring the execution state of the client or server software which utilizes the protocol. This approach has the obvious benefit of mapping data stored in memory at one state to the data subsequently present in a protocol message. This requires access to the actual software which generates the messages. In many cases this may limit the analysis to the messages sent by the client. When the server software is unavailable for execution state monitoring, one must resort to other methods in order to further analyse the protocol.

[1] http://www.netzob.org

Chapter 2

Theory

2.1 Features

In machine learning, a feature is a measurable property of the input data. An example of a feature when working with images could be the intensity of light for a certain pixel. Features do not need to directly correspond to a physically measurable property however, instead they can be constructed by transforming other features. These new high-level features can then be used just like any other feature in order to simplify a complex problem.

One common type of complex problem is working with high dimensional data. There are different ways to deal with such data; one of them being to use a solving method that scales linearly. Another way is to reduce the dimensionality of the problem itself.

2.1.1 Principal component analysis

One method to reduce the dimensionality of a problem is to use principal component analysis (PCA). PCA transforms the input data into another dataset where each dimension accounts for as much variance as possible while requiring it to maintain orthogonality to the other dimensions. The new dataset can be constructed to contain fewer dimensions than the original data and will then capture as much variance as possible in the given number of dimensions. PCA can be accomplished through singular value decomposition of a matrix.

2.2 Simple linear regression

In statistics, a response variable Y_i that is linearly dependent on an explanatory variable X_i can be modeled with a simple linear regression model. The relationship between the two variables are then

$$Y_i = \alpha + \beta X_i + \varepsilon_i \tag{2.1}$$

where α and β are the *regression coefficients* and ε_i is the error term. Given a set of data $\{Y_i, X_i\}_1^n$ the regression coefficients can be estimated through linear regression. This process is often called fitting as it can be seen as trying to find

Algorithm 1: RANSAC

Input: D, M, n, δ, γ, k
Output: P_b

$P_b \leftarrow \varnothing$
$\varepsilon_b \leftarrow \infty$
for k **do**
    $I \leftarrow n$ number of randomly chosen samples
    fit M to I
    $I_m \leftarrow$ samples where distance from $M \leq \delta$
    $I \leftarrow I \cup I_m$
    if $|I| \geq \gamma$ **then**
        fit M to I
        $\varepsilon \leftarrow$ deviation from M for all I
        if $\varepsilon < \varepsilon_b$ **then**
            $P_b \leftarrow$ parameters in M
            $\varepsilon_b \leftarrow \varepsilon$
return P_b

the model with the best fit for the data. There are different methods to do this as there is no consistent measure for what is the best fit.

2.2.1 Ordinary least squares

Ordinary least squares (OLS) is a method for estimating the regression coefficients in equation 2.1. It does this by minimizing the sum of squared residuals of the simple linear regression model. This can be formulated as minimizing the following cost function.

$$C(\alpha, \beta) = \sum_{i=1}^{n} (Y_i - \alpha - \beta X_i)^2 \tag{2.2}$$

The result of this is a model that minimizes the residuals globally across all samples. This intrinsically makes the method *non-robust*, i.e. sensitive to outliers.

2.2.2 RANSAC

Random Sample Consensus (RANSAC) (Fischler and Bolles, 1981) is a *robust* method for fitting a dataset to a model and was originally intended for image analysis. The robustness of the method allows it to avoid the influence of outliers in the model. Given a dataset D, a model M, a set of inliers I and two confidence parameters, δ and γ, it works the following way:

1. Fit M to I
2. Find the samples that are within δ distance from M and add those samples to I
3. Continue if $|I|$ is at least γ

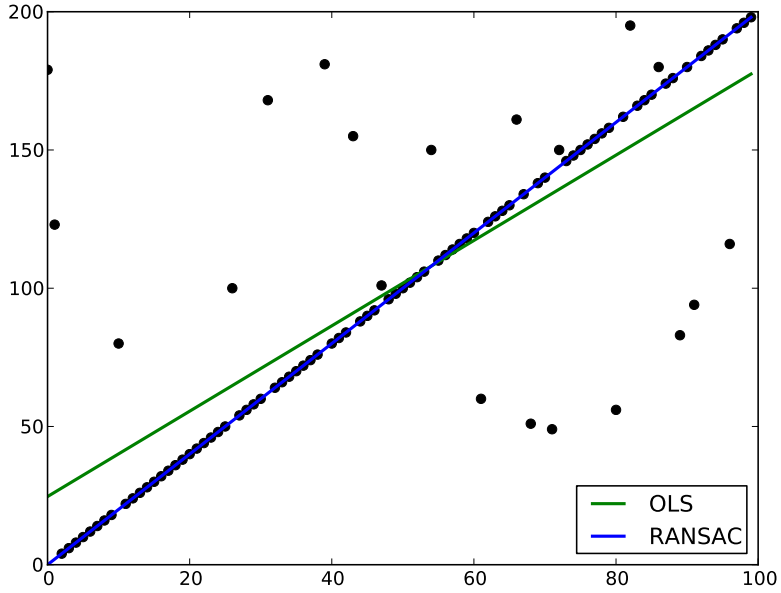


Figure 2.1: A comparison between OLS and RANSAC on a dataset containing outliers.

4. Fit M to I

5. Evaluate M from the residual of I

One problem with this approach is that an initial set of inliers is required. In most cases this is not available so we need to guess which samples might be inliers. A simple solution to this is to take n random samples and assume that they are inliers. This only works if the assumption is actually true and thus the resulting algorithm will be *non-deterministic*, meaning that it only produces a good fit with a certain probability. The algorithm is normally run k number of iterations until the probability of success is sufficiently large. Pseudocode for RANSAC can be seen in algorithm 1.

The model that RANSAC operates on can be any kind of mathematical model, therefore when handling two-dimensional linear data the simple linear regression model given in equation 2.1 can be used. This gives us a method that solves the problem of simple linear regression but that is also insensitive to outliers. A comparison between OLS and RANSAC on a dataset where some samples have been replaced by random noise can be seen in figure 2.1.

2.3 Clustering

A clustering is a grouping of samples based on similarity with respect to their features. The features are predefined to represent distinct properties of the samples. Similarities between samples are normally represented as distances in

an n-dimensional space, where n is the number of features. Distances may be calculated using any suitable norm.

In cluster analysis, the goal is normally to find a clustering from a set of samples such that the membership of a cluster represents some true relationship. In some clustering algorithms such as K-means (MacQueen et al., 1967), the number of clusters are predefined. When the number of clusters are unknown, other algorithms such as DBSCAN, OPTICS and hierarchical clustering methods may be used.

The clustering algorithm we use in our method is OPTICS, which is based on DBSCAN. In the rest of this section we will explain these algorithms and provide examples based on protocol data.

2.3.1 DBSCAN

DBSCAN is a *density-based* clustering algorithm, as opposed to partitioning clustering algorithms such as K-means. Density-based clustering algorithms provides the benefit of not having to provide an estimated number of clusters as a parameter to the algorithm.

DBSCAN was first presented by Ester et al. (1996). The algorithm takes a set of samples D and two parameters, $MinPts$ and ε. $MinPts$ decides how many samples that are needed in order to form a cluster.

The densities are defined by the distances between a set of points. The algorithm introduces the definition ε-*neighborhood* $N_\varepsilon(p)$ for a point p. The samples which are in $N_\varepsilon(p)$ are given by the following condition:

$$N_\varepsilon(p) = \{q \in D \mid dist(p, q) < \varepsilon\} \tag{2.3}$$

The samples in $N_\varepsilon(p)$ are defined as *directly density-reachable* from p. Given that $|N_\varepsilon(p)| \geq MinPts$, the samples in $N_\varepsilon(p)$ forms a cluster, and p is a *core point*.

A cluster is not limited to containing samples which are directly density-reachable. The DBSCAN algorithm also defines that samples which are not directly density-reachable may be *density-reachable*. A sample q is density-reachable from a sample p if there is a set of samples $S = \{s_1, .., s_n\}$ where $p = s_1$, $q = s_n$ and s_{i+1} is directly density-reachable from s_i for $0 < i < n$.

Since the density-reachable relationship is not symmetric, a looser relationship is introduced: *density-connected*. Two samples p and q are density-connected if there exists a third point o from which both p and q are density-reachable. The density-connected relationship is symmetric.

With these relationships, the algorithm defines cluster membership as follows:

Definition. *The following conditions needs to be satisfied for a point q to be a member of a cluster C, given that there is a sample $p \in C$:*

1. *q is density-reachable from p*
2. *q is density-connected to p*

One of the drawbacks of DBSCAN is that it can only find clusters with a density higher than the density decided by the ε-parameter. It is also hard to estimate the ε and $MinPts$-parameters for an unknown dataset. This makes

DBSCAN suitable for classifying data into known classes, but not as suitable for finding underlying structures in entirely unknown datasets. In figure 2.2 is an example of DBSCAN running on 5000 packets of DNS data. The parameters are manually selected to $\varepsilon = 0.085$ and $MinPts = 200$.

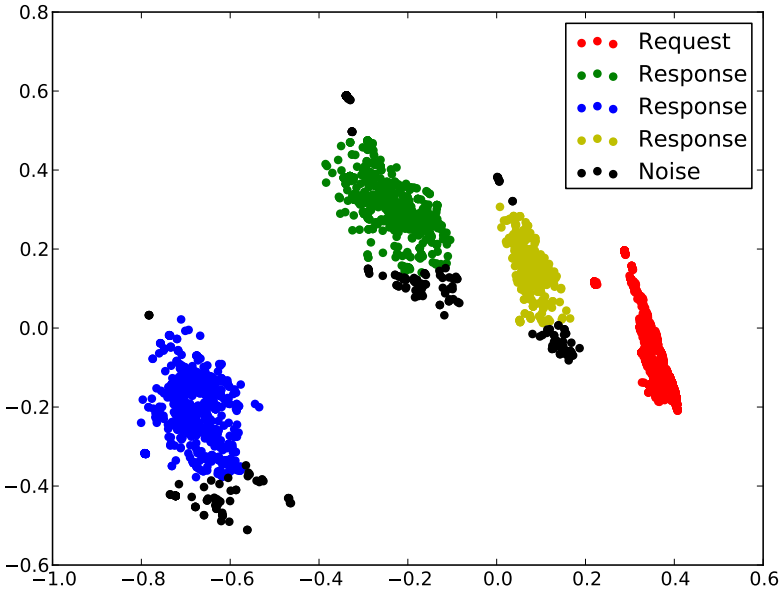


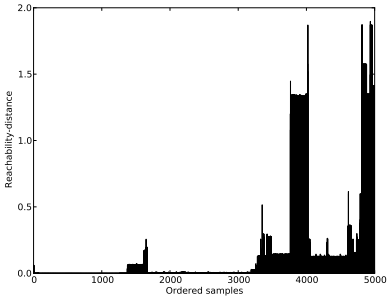
Figure 2.2: An example of clustering with DBSCAN running on samples generated from 5000 packets of DNS data. The samples has been projected down to two dimensions using PCA (see section 2.1.1).

2.3.2 OPTICS

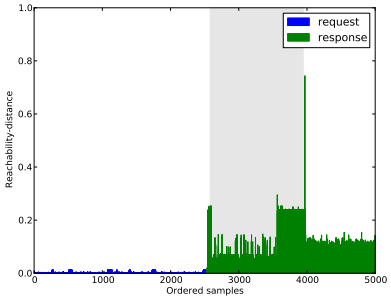
Addressing the difficulties in selecting parameters for DBSCAN, Ankerst et al. (1999) introduced the OPTICS algorithm, which is an extension of the concepts introduced in DBSCAN. OPTICS uses the definitions *directly density-reachable*, *density-reachable* and *density-connected* which were described in the DBSCAN algorithm, but eliminates the need for an explicit ε parameter. Instead the ε parameter is interpreted as the largest distance to consider when clustering. The results is an algorithm which is less sensitive to user-specified parameters, and is able to find clusters with varying densities.

One important aspect of OPTICS is that it does not generate actual clusters. Instead, it generates an *ordering* of samples and a set of corresponding *reachability-distances* which reveals density-based structure in the input data. The OPTICS algorithm extends DBSCAN with the definitions *core-distance* and *reachability-distance*.

Core-distance is a measurement of the distance ε which is required for a sample



(a) Reachability-plot for 5000 DHCP-packets with $MinPts = 200$.



(b) Hierarchical extraction for 5000 DNS-packets with $MinPts = 200$.

Figure 2.3: Reachability-plots generated from the ordering and reachability-distances given by OPTICS.

p to be a core point. That is, a core distance is the distance ε which satisfies $|N_\varepsilon(p)| = MinPts$.

Definition. *The core-distance for a sample p is defined as:*

$$\begin{cases} UNDEFINED, & \text{if } |N_\varepsilon(p)| < MinPts \\ Min\ \varepsilon\ \text{which satisfies } |N_\varepsilon(p)| = MinPts(p), & \text{otherwise} \end{cases}$$

The core-distance is $UNDEFINED$ when an upper limit on ε is given as a parameter to the algorithm. This parameter is not required, although it has an impact on the runtime of the algorithm.

The reachability-distance of a sample p is the smallest distance which is needed for p to be directly density-reachable to a core point q for some ε.

Definition. *The reachability-distance of a sample p with respect to some sample q is defined as:*

$$\begin{cases} UNDEFINED, & \text{if } |N_\varepsilon(q)| < MinPts \\ max(\text{core-distance}(q),\ \text{distance}(q,\ p)), & \text{otherwise} \end{cases}$$

where distance(q, p) is the smallest ε for which q is density-reachable from p.

From the ordering and reachability-distances, the cluster structure may be visualized in a *reachability-plot*. The reachability-plot provides an overview of the output from OPTICS as a bar-plot where the bars represent reachability-distances in the ordering generated by OPTICS.

An example of a reachability plot generated from 5000 DHCP-packets with $MinPts = 500$ is shown in figure 2.3a. Valleys in the reachability-plot represents samples which are close to each other. Spikes represents a large difference in distance between the samples to the left and right of the spike.

The pseudocode algorithm is given in algorithm 2.

Algorithm 2: OPTICS

Input: D, $MinPts$, ε
Output: *ordering*, *reachability-distances*

reachability-distances $\leftarrow \varnothing$
ordering $\leftarrow \varnothing$
seeds $\leftarrow 0, 1, ..., |D|$
$i \leftarrow 0$
while $|seeds| > 1$ **do**
  $p \leftarrow seeds.get(i)$
  $seeds.remove(i)$
  $ordering.add(p)$
  if *core-distance*$_{\varepsilon,MinPts}(p) \leq \varepsilon$ **then**
    $neighbors \leftarrow N_{\varepsilon}(p)$
    for *each neighbor sample* n **do**
      $rdist \leftarrow$ *reachabilty- distance*$_{\varepsilon,MinPts}(n, p)$
      reachability-distances$[n] \leftarrow rdist$
    $i \leftarrow$ seed with least reachability-distance
  else
    $i \leftarrow seeds.first$
`/* Proccess last remaining seed */`
$ordering.add(seeds.first)$
reachability-distances$[0] \leftarrow 0$
return *ordering, reachability-distances*

Cluster extraction

Ankerst et al. describes the principles of the OPTICS algorithm as running DBSCAN for an infinite number of distance parameters ε_i in the interval $0 \leq \varepsilon_i \leq \varepsilon$. They also explain that extracting clusters from the OPTICS ordering and reachability-distances using a static reachability-distance threshold ε_i gives a clustering which is roughly equivalent to the clustering obtained when running DBSCAN with the same $MinPts$ and $\varepsilon = \varepsilon_i$.

Sander et al. (2003) describes a hierarchical cluster extraction algorithm for OPTICS. As opposed to the DBSCAN-equivalent extraction approach, hierarchical cluster extraction retains many of the properties which comes with OPTICS, such as finding clusters with varying densities and subclusters nested in larger clusters.

The algorithm takes an ordering and reachability-distances from OPTICS and builds a hierarchical representation of the clustering in the form of a tree, where the leaves are clusters.

The pseudocode for hierarchical extraction is given in algorithm 3. Figure 2.3b is the result of the algorithm running on the same DNS dataset as figure 2.2 with $MinPts = 200$. The algorithm yields three clusters, each cluster distinguished by changing background color. The true types *request* and *response* are colored blue and green respectively in the reachability plot. Note that the algorithm is rather insensitive to the $MinPts$-parameter, and that the true types of DNS may be extended to different record types such as A/AAAA/CNAME-records.

	A	C	G	T
A	2	1	0	1
C	1	2	1	-1
G	0	1	2	1
T	1	-1	1	2

(a) The similarity matrix for the alphabet.

		A	A	T	G	C	A	G
	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
T	-2	1	3	3	2	1	0	-1
C	-3	0	2	2	4	4	3	2
G	-4	-1	1	3	4	5	4	5
G	-5	-2	0	2	5	5	5	6

(b) The alignment matrix for the two sequences AATGCAG and ATCGG.

Figure 2.4: An example of the Needleman-Wunsch algorithm run on two sequences constructed from the alphabet $\{A, C, G, T\}$ with a gap penalty $d = -1$. The similarity matrix used can be seen in (a) and the resulting alignments is given by the gray elements in (b).

2.4 Sequence alignment

The problem of aligning sequences with one another originally arose in the area of bioinformatics. There, a need to find similarities in long chains of amino acids existed. One of the first methods that solved this problem was the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970). The algorithm finds the maximal number of matching symbols between two sequences, allowing for gaps to be inserted into either sequence. This is done by tracing a path through an alignment matrix where each element represents a possible matching of symbols between the sequences.

Originally, the path was built only with respect to the number of matching identical symbols and not to the number of mismatches or gaps. This has since been improved upon to include pairwise scores for symbol matching and penalties for gaps. The different scores used when matching symbols is typically given in a similarity matrix. An example of such a matrix for a small alphabet of symbols is seen in figure 2.4a.

Given two sequences, S and T, of length m and n respectively, a scoring matrix C and a gap penalty d the algorithm does the following:

1. Construct an $(m+1) \times (n+1)$ alignment matrix A
2. Fill the first row of A with gap penalties: $a_{0,j} \leftarrow j \cdot d$ where $0 \leq j \leq n$
3. Fill the first column of A with gap penalties: $a_{i,0} \leftarrow i \cdot d$ where $0 \leq i \leq m$
4. Fill the rest of A by doing one of the following actions for each element $a_{i,j}$:
   - Match - $a_{i,j} \leftarrow a_{i-1,j-1} + C_{S_j,T_i}$
   - Delete - $a_{i,j} \leftarrow a_{i-1,j} + d$
   - Insert - $a_{i,j} \leftarrow a_{i,j-1} + d$

   The action that is chosen is the one that results in the largest score where, in the case of a tie, match is given priority. The priority of delete and insert can be chosen arbitrarily.

5. Backtrace through the matrix from $a_{m,n}$ to build the alignments in reverse order. Choose the path that was used to construct the score at the current element.

An example of applying the algorithm can be seen in figure 2.4 and the resulting alignments in figure 2.5.

```
AATGCAG
-AT-CGG
```

Figure 2.5: The resulting alignment of the sequences AATGCAG and ATCGG.

Algorithm 3: Hierarchical Cluster Extraction

Input: *ordering*, *reachability-distances*
Output: *clustering*

$R \leftarrow$ *reachability-distances* arranged according to *ordering*
$n \leftarrow$ number of samples
$L \leftarrow$ indices of local maximas in R
Sort L on $R[L_i]$
$R_{max} \leftarrow R[L.last]$
leaves $\leftarrow \varnothing$
function *cluster_tree(node, parent, L)*:
    if *L is empty* **then**
        leaves.add(*node*)
        return
    $s \leftarrow L.pop()$
    node.split $\leftarrow s$
    $sign\_thres \leftarrow significant\text{-}ratio * R[s]$
    Create two new nodes, $N1$ and $N2$
    $N1.samples \leftarrow$ samples left of s
    $N2.samples \leftarrow$ samples right of s
    $L1 \leftarrow$ local maximas left of s
    $L2 \leftarrow$ local maximas right of s
    if $N1$ *and* $N2$ *has average reachability* $<$ *sign_thres* **then**
        if $|N1| > MinPts$ **then**
            children.add($\{N1, L1\}$)
        if $|N2| > MinPts$ **then**
            children.add($\{N2, L2\}$)
        if *children is empty* **then**
            leaves.add(*node*)
            return
        if $R[s] \approx R[parent.split]$ **then**
            for $\{child, L\}$ *in children* **do**
                parent.children.add(*child*)
            parent.children.remove(*node*)
            $p \leftarrow parent$
        else
            for $\{child, L\}$ *in children* **do**
                node.children.add(*child*)
            $p \leftarrow node$
        for $\{child, L\}$ *in children* **do**
            cluster_tree(child, p, L)
    else
        cluster_tree(node, parent, L)

Create node *root* containing all samples
cluster_tree(root, null, L)

Chapter 3

Method

3.1 Approach

A protocol is generally made up of a predefined set of message types. A message type may define if the message is a request or response. If so, it is fairly easy to determine which parts of the message that defines if the message is a request or a response. One may simply look for values in a message that depends on the direction of the message. However, most protocols are not limited to being categorized into requests and responses. Protocols often define sets of control message types which changes the state of the protocol. Some protocols does not follow the request/response model, which is the case for many peer-to-peer protocols.

In order to thoroughly analyse a protocol, the first step is essentially to distinguish the different types of the protocol. This is a necessary first step which enables us to infer the different states of the protocol and allows for type-specific field analysis. The type is commonly specified by some flag or a combination of flags. A type flag may be a byte, a bit subset of a byte, or spread out over multiple bytes where each byte defines some subtype to a more general type.

Messages which shares type flags are likely to have a distinguishable structure which differs from messages with other type flags. Some message types may introduce fields which are not present in other message types, and the fields which are global for the protocol are more likely to be identical within a type. A message that is used for binary data transmission is often longer than a control message.

The first step in our approach is to group messages that are similar using cluster analysis. We then move on to analysing the grouped messages in an effort to discover the bytes which we rank the most probable to be responsible to the grouping we have found, and label these as the bytes which distinguishes a specific type. Once these bytes have been discovered, we may regroup the messages based on the value of that the byte takes.

From the new grouping, we are able to analyse the type specific properties of the messages. We focus on determining the field structure of each type, and make an effort to determine the semantic...

3.2 Type inference

Explain that our type inference step is actually two different clustering steps performed in sequence.

3.2.1 Initial clustering

Explain our choice of features and clustering algorithm.

3.2.2 Type distinguishers

Describe our FD clustering algorithm.

3.3 Field analysis

With our type inference complete we now have a number of clusters that contain similar messages. The next step in inferring the structure of the protocol is to identify field boundaries and their contents. This is a very complex problem, there can be as many fields as there are bits in the messages and the boundaries may lie in between any pair of adjacent bits. The number of possible combinations of fields that comprise the structure can easily become too large to handle. Fortunately, most protocols are not designed to utilize each and every bit and there are some common design principles that may be exploited.

First of all we restrict where boundaries may lie. We require fields to be a certain number of bytes in length and also that they must be aligned. We define alignment as the offset of where a field starts being a multiple of the length of the field. That means that a four byte field may be located at offset 0, 4 or 12 but not at offset 5 or 10. Most protocols follow this principle of bytes being used as the smallest component and aligning data...

Another principle that is often used is that the message starts with a header where most of the fields are fixed.

In order to accomodate for varying field lengths we start by aligning the data of the messages.

We align the messages by doing pairwise global sequence alignment of each message with the longest message in the cluster. We have also restricted the insertion of gaps so that the longest message will not be expanded. This was done by modifying the Needleman-Wunsch algorithm at a few key places.

The scoring matrix used is constructed...

Our assumption here is that header fields are of fixed length and that the messages in each cluster share the same structure.

Introduce the classes of fields that we have established as identifiable. Use plots of their byte distributions to motivate our decision.

3.3.1 Byte distribution

Explain how we reached the decision to use byte distributions in our attempts to find structures in protocols.

3.3.2 Constant fields

3.3.3 Flag fields

3.3.4 Uniform fields

3.3.5 Number fields

3.3.6 Incremental fields

3.3.7 Length fields

Explain how we use a linear model to model the relationship between length fields and message lengths.

3.4 Protocol state inference

Explain how we build a state machine from our clusters.

Chapter 4

Results

4.1 Data sets

Provide information about the data used to produce the results.

4.2 Metrics

Explain what metrics we use to measure our results.

4.3 Clustering performance

Display the results of applying our method to different data sets.

4.4 Field inference performance

Chapter 5

Discussion

5.1 Conclusions

Draw conclusions about our method and compare it to the related methods.

5.2 Limitations

Give a short introduction to the different limitations of our method.

5.2.1 Textual protocols

Explain why our method is not applicable to textual protocols.

5.2.2 Variable number of fields

Explain the problem with protocols that contain a variable number of fields and why our method does not accommodate for them. (Because we based our definition of protocols on a fixed number of fields?)

5.2.3 Non-aligned data

Explain why we only try to find aligned fields and the problem with complexity if we were to relax this requirement.

5.2.4 Bit precision

Talk a little bit about how some protocols do not use entire bytes as the sizes of their fields and that we will not find exact boundaries for them.

5.3 Future work

Mention the ideas that has come up during the thesis work that we have not had time to investigate further.

5.3.1 Correspondence analysis

Explain how correspondence analysis could give a better result than PCA.

5.3.2 Timestamp identification

Explain how timestamps could potentially be identified from their distinguished byte distribution pattern.

Chapter 6

Bibliography

Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Record*, 28(2):49–60, 1999.

Marshall A Beddoe. Network protocol analysis using bioinformatics algorithms, 2005.

Juan Caballero and Dawn Song. Automatic Protocol Reverse-Engineering: Message Format Extraction and Field Semantics Inference. *Computer Networks*, 2012.

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*, pages 1–14, 2007.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. University of Munich, München, Germany, 1996.

Martin A Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297. University of California, Los Angeles, USA, 1967.

Saul B. Needleman and Christian D. 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.

Jörg Sander, Xuejie Qin, Zhiyong Lu, Nan Niu, and Alex Kovarsky. Automatic Extraction of Clusters from Hierarchical Clustering Representations1. In *Advances in Knowledge Discovery and Data Mining: 7th Pacific-Asia Conference, PAKDD 2003. Seoul, Korea, April 30-May 2, 2003, Proceedings*, volume 7, page 75. Springer, 2003.

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