A Semantics Aware Approach to Automated Reverse Engineering Unknown Protocols

Yipeng Wang[*‡] Xiaochun Yun[§] M. Zubair Shafiq[†] Liyan Wang[†]
Alex X. Liu[†] Zhibin Zhang[*] Danfeng(Daphne) Yao[¶] Yongzheng Zhang[||] Li Guo[||]

[*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[†]Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, U.S.A.
[‡]Graduate School of Chinese Academy of Sciences, Beijing, China
[§]National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
[¶]Department of Computer Science, Virginia Tech, Blacksburg, VA, U.S.A.
[||]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
Email: yipeng.wang1@gmail.com, yunxiaochun@cert.org.cn, shafiqmu@cse.msu.edu, liyanwan@cse.msu.edu
alexliu@cse.msu.edu, zhangzhibin@ict.ac.cn, danfeng@cs.vt.edu, zhangyongzheng@iie.ac.cn, guoli@iie.ac.cn

***Abstract*—Extracting the protocol message format specifications of unknown applications from network traces is important for a variety of applications such as application protocol parsing, vulnerability discovery, and system integration. In this paper, we propose ProDecoder, a network trace based protocol message format inference system that exploits the semantics of protocol messages without the executable code of application protocols. ProDecoder is based on the key insight that the n-grams of protocol traces exhibit highly skewed frequency distribution that can be leveraged for accurate protocol message format inference. In ProDecoder, we first discover the latent relationship among n-grams by first grouping protocol messages with the same semantics and then inferring message formats by keyword based clustering and cluster sequence alignment. We implemented and evaluated ProDecoder to infer message format specifications of SMB (a binary protocol) and SMTP (a textual protocol). Our experimental results show that ProDecoder accurately parses and infers SMB protocol with 100% precision and recall. For SMTP, ProDecoder achieves approximately 95% precision and recall.**

I. Introduction

A. Motivation and Problem Statement

This paper concerns the automatic inference of protocol message format specifications from the network traces of unknown application protocols. This has many applications in networking and security. For instance, application protocol parsing requires protocol inference. Application protocol parsing, the translation of raw packet flows into higher level flows of semantic content, has a wide variety of current and future networking and security services such as semantics aware Intrusion Detection and Prevention Systems (IDSes/IPSes), network monitoring, network measurement, application-aware load balancing, application fingerprinting, tunnel detection, Quality-of-Service (QoS), and content-aware caching and routing. Take its application in IDSes/IPSes as an example. Traditional IDSes/IPSes treat packet payload as a sequence of bytes and match it against malware signatures represented as a set of regular expressions. This coarse grained signature checking is fundamentally limited due to its ignorance of the application protocol structure in the packet payload. Modern IDSes/IPSes become semantics aware by parsing packet payloads to get the value for each application protocol field based on application protocol message formats. Several application protocol parsers, such as FlowSifter [25], UltraPAC [20], binpac [28], and GAPA [4], have been proposed in prior literature.

All these application protocol parsers require protocol specifications in order to generate parsers for the corresponding protocol. However, many application protocols on the Internet are proprietary and have no publicly released specifications. According to the Internet2 NetFlow weekly reports on backbone traffic, more than 40% of Internet traffic belongs to unidentified application protocols [27]. Communication protocols used by malware and botnets do not have protocol specifications from their designers. To parse a flow of unknown application protocols, we first need to conduct protocol inference to get the protocol message format. Network monitoring tools such as Ethereal [1], NetDude [19], SNORT [31], and BRO [30] also require application protocol parsers to implement their functionalities.

Besides application protocol parsing, protocol inference is useful for many other applications such as vulnerability discovery and system integration. For vulnerability discovery, to detect vulnerability in a deployed application, people often perform penetration testing, which requires protocol specification for that application. For system integration, to develop applications that can work with proprietary protocols that have no publicly known specifications, protocol inference is needed to decode such protocols. For example, to develop an open source client program that works with the proprietary Yahoo Messenger protocol, one needs to first use protocol inference to decode the message format of this protocol. Furthermore, even for some application protocols with known specifications, protocol inference is also needed sometimes for identifying implementation bugs and for identifying implementation details that are not unambiguously specified.

Inferring protocol specification from executable code is extremely difficult. First, the executable code of these unidentified protocols, such as botnet command and control protocols, are often not available for reverse engineering. Second, even when such executable code is available, the reverse engineering process is labor intensive and error prone. For example, manually reverse engineering the Microsoft Server Message Block (SMB) protocol took 12 years in the open source SAMBA project [34].

For the protocol inference problem addressed in this paper, the input is a network trace of the target application protocol. Note that an application protocol typically have multiple types of messages where each message type has its own format. If the executable code of an application protocol is available, it can be run in a controlled environment to gather packet traces. Else, prior traffic classification schemes (such as [16]) can be used to separate network traffic of the target protocol from that of others. As traffic classification schemes often do not have a 100% accuracy, we do not assume that the input network trace contains only the packets of the target application protocol. The output of protocol inference are protocol message formats represented by regular expressions.

B. *Limitation of Prior Art*

Prior protocol message format inference methods fall into two categories: reverse engineering based methods [5]–[8], [21], [34], which infer protocol message format by reverse engineering the executable code of protocols, and network trace based methods [10], which infer protocol message format by analyzing network traces that contain the messages of a given protocol. Reverse engineering based methods are only applicable to protocols for which the executable code is available; however, the executable code of many unknown protocols are typically not available for reverse engineering.

The only prior network trace based application protocol message format inference method is Discoverer proposed by Cui *et al.* [10]. Discoverer first reassembles IP packets into application protocol messages; second, breaks up each message into a sequence of tokens based on a set of predefined delimiters such as space and tab; third, classifies messages into various clusters based on each message's token pattern; and finally merges similar message formats. Discoverer has three major limitations. First, Discoverer does not work for asynchronous application protocols. Furthermore, even for synchronous application protocols, it does not work with sampled network traces. This is because it requires assembling packets into application protocol messages. Discoverer treats each maximum sequence of consecutive packets as an application protocol message. This way of grouping packets into messages is inappropriate for asynchronous protocols because two end hosts may send packets to each other at the same time. Second, Discoverer assumes that the first constant number of bytes of a flow describe the complete structure of an application protocol. However, this assumption often does not hold in reality. For example, the Simple Mail Transfer Protocol (SMTP) indicates the end of the mail data by sending a line containing only a ".", where "." is an element of the message format and the email message can be of any length. Third, Discoverer assumes the existence of some delimiters for separating different fields in protocols. However, unknown protocols may not use delimiters and even if they use delimiters, such delimiters may not available to the public.

C. *Proposed Approach*

In this paper, we propose ProDecoder, a semantics aware approach that takes network traces as the input and automatically outputs the inferred protocol message format. ProDecoder does not assume prior knowledge of protocol specifications such as delimiters. It is applicable to both text and binary protocols. Our approach is based on the key insight that the n-grams of protocol traces exhibit highly skewed frequency distribution that can be leveraged for accurate protocol message format inference. ProDecoder has four major modules: n-gram generation, keyword identification, message clustering, and sequence alignment. We give an overview of each module below.

1) n-gram Generation: The input to this module is a set of packet traces that are of the same protocol. The process of classifying raw network traffic into flows of different protocols is called flow classification. The simplest flow classification method is to classify flows based on the transport layer port numbers. Of course, this simple method may misclassify traffic carried by tunnels. There are more advanced flow classification methods that have been proposed in prior literature [16], [18], [29]. The output of this module is protocol messages where each message is represented as a sequence of n-grams. An n-gram is a subsequence of n elements contained in a given sequence of at least n elements. For example, treating each character as an element, the 3-grams generated from message `MAIL FROM` are `MAI`, `AIL`, `IL_`, `L_F` `_FR`, `FRO` and `ROM`. Given many packets of the same protocol, ProDecoder first decomposes each message, denoted as a sequence of m bytes $b_1b_2\cdots b_m$, into a sequence of $m-n+1$ n-grams ($n \leq m$): $b_1b_2\cdots b_n$, $b_2b_3\cdots b_{n+1}, \cdots, b_{m-n+1}b_{m-n+2}\cdots b_m$.

2) Keyword Identification: This module uses a generative model from natural language processing to infer protocol keywords, which are used to define protocol message formats. We identify a protocol keyword as a set of n-grams that mostly show up together in protocol messages. For example, the set of 3-grams {`MAI`, `AIL`, `IL_`, `L_F`, `_FR`, `FRO`, `ROM`} can be a keyword because `MAIL FROM` often show up together in SMTP messages. A message can have multiple keywords. For example, an SMTP message `MAIL FROM: <alice@gmail.com> (RCPT TO: <bob@live.cn>)+ DATA` have three keywords that corresponds to `MAIL FROM`, `RCPT TO`, and `DATA`.

3) Message Clustering: This module clusters messages based on their keywords using machine learning techniques. Using the keywords associated with each message as features, we use the Information Bottleneck (IB) clustering algorithm to group similar messages into a cluster based on their semantics [33]. This module enables ProDecoder to distinguish among similar keywords belonging to different protocol messages.

4) Sequence Alignment: For the messages in each cluster, this module uses a well-known sequence alignment algorithm to find the common byte sequences among them. For example, given a set of SMTP messages, sequence alignment algorithms can identify `MAIL FROM` as a common byte sequence. These common byte sequences represent the stable part of protocol messages and therefore can be used to represent the message format of the protocol in the form of regular expressions.

D. Novelty and Advantages of Our Approach

The key novelty of ProDecoder lies in its exploitation of the semantic information in protocol messages. It distinguishes the different meanings of the same n-grams in different messages, which may have different semantics and therefore should be classified into different keywords. Consider the example SMTP message in Figure 1, where the 3-gram "250" represents different semantic meanings for different occurrences. In this example, we use numbers from 1 to 8 to indicate the order of the 8 messages, letter "S" to indicate the message from the email sender, and letter "R" to indicate the message from the email receiver. However, prior network trace based protocol message format inference methods cannot make such distinctions as they rely on counting the occurrences of strings, ignoring the context of each string. Furthermore, ProDecoder discovers the correlation among n-grams. In protocol messages, multiple n-grams together may form an element in the protocol message format. For example, in an SMTP message, the 3-grams, "250" and "_OK" together, denote a protocol element that is used to confirm the mail transaction. Using keyword identification, our approach can group correlated n-grams together to form a keyword. Keyword identification in ProDecoder is inspired from natural language processing literature, where a major research issue is to identify topics from a corpus of documents consisting of a vector of words.

```
1 S: MAIL FROM:<alice@USC-ISIE.ARPA:JQP@MIT-AI.ARPA>
2 R: 250 OK
3 S: RCPT TO:<joe@BBN-VAX.ARPA>
4 R: 250 OK
5 S: DATA
6 R: 354 Start mail input; end with <CRLF>.<CRLF>
7 S: Received: from MIT-AI.ARPA by USC-ISIE.ARPA ;
     2 Nov 81 22:40:10 UT
     Date: 2 Nov 81 22:33:44
     From: John Q. Public <JQP@MIT-AI.ARPA>
     Subject: The Next Meeting of the Board
     To: Jones@BBN-Vax.ARPA
     ...
8 R: 250 OK
```

Fig. 1. An example SMTP communication session

ProDecoder addresses the aforementioned three limitations of Discoverer. First, ProDecoder works with asynchronous application protocols and sampled network traces because it does not assemble IP packets into application-level messages. Second, ProDecoder does not assume that the first constant number of bytes of a flow describe the complete structure of an application protocol. Third, ProDecoder does not assume the existence of delimiters for separating different fields in protocols.

The rest of the paper proceeds as follows. In Section II, we review related work. We provide the technical details of ProDecoder in Section III. We present implementation details and experimental results for ProDecoder in Section IV. Finally, we conclude the paper in Section V.

II. Related Work

A. Reverse Engineering Based Methods

Such methods infer protocol message format by reverse engineering the executable code of protocols. Accurately reverse engineering protocols typically involves manual efforts, as described in [7] and [34]. There are several proposals about automating this process. Lim *et al.* proposed a method that automatically extracts the format from files and application data output functions, which may not be available [21]. Caballero *et al.* proposed a protocol reverse engineering method called Polyglot that uses dynamic analysis of program binaries [5]. The methods proposed in [8] and [6] infer protocol message formats by observing the dynamic execution of protocols. Lin *et al.* [22] and Wondracek *et al.* [36] proposed tools to reverse engineer network message formats based on observing how a program processes protocol messages. Cui *et al.* proposed Tupni, a protocol reverse engineering method for automatically identifying record sequences and record types in input formats [11]. In contrast to these methods, our approach does not require the binary code of protocols.

B. Other Related Work

Kannan *et al.* proposed a semi-automated method for extracting session structures of an application protocol based on the session logs between a pair of end hosts. Comparing Kannan's method with ProDecoder, first, the goals are different - Kannan's method aims at extracting session structures whereas ProDecoder aims at extracting protocol message formats. Note that session logs are typically unavailable for unknown application protocols. If we are given full logs of sessions between many pairs of end hosts, it is straightforward to extend ProDecoder to output session structures. Second, the level of automation is different - Kannan's method is semi-automated whereas ProDecoder is fully automated.

Haffner *et al.* proposed ACAS, a method for the automated construction of application signatures based on packet traces [17]. The goal of ACAS is different from ProDecoder: ACAS aims at obtaining application protocol signatures whereas ProDecoder aims at obtaining message formats. Application protocol message formats can be used as protocol signatures, but not vice versa. A major limitation of ACAS is that it assumes that the first 64 bytes of a flow completely describes the structure of the application protocol carried by the flow. This assumption often does not hold in reality especially for binary protocols. Ma *et al.* proposed another protocol identification method that uses statistical and structural content

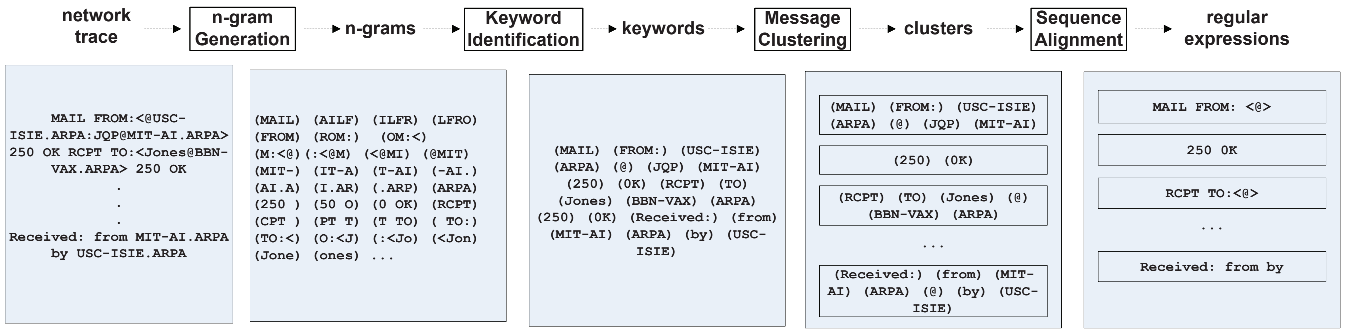

network trace
n-gram Generation
n-grams
Keyword Identification
keywords
Message Clustering
clusters
Sequence Alignment
regular expressions
MAIL FROM:<@USC-ISIE.ARPA:JQP@MIT-AI.ARPA> 250 OK RCPT TO:<Jones@BBN-VAX.ARPA> 250 OK
. . .
Received: from MIT-AI.ARPA by USC-ISIE.ARPA
(MAIL) (AILF) (ILFR) (LFRO) (FROM) (ROM:) (OM:<) (M:<@) (:<@M) (<@MI) (@MIT) (MIT-) (IT-A) (T-AI) (-AI.) (AI.A) (I.AR) (.ARP) (ARPA) (250) (50 O) (0 OK) (RCPT) (CPT) (PT T) (T TO) (TO:) (TO:<) (O:<J) (:<Jo) (<Jon) (Jone) (ones) ...
(MAIL) (FROM:) (USC-ISIE) (ARPA) (@) (JQP) (MIT-AI) (250) (OK) (RCPT) (TO) (Jones) (BBN-VAX) (ARPA) (250) (OK) (Received:) (from) (MIT-AI) (ARPA) (by) (USC-ISIE)
(MAIL) (FROM:) (USC-ISIE) (ARPA) (@) (JQP) (MIT-AI)
(250) (OK)
(RCPT) (TO) (Jones) (@) (BBN-VAX) (ARPA)
...
(Received:) (from) (MIT-AI) (ARPA) (@) (by) (USC-ISIE)
MAIL FROM: <@>
250 OK
RCPT TO:<@>
...
Received: from by

Fig. 2. Architecture of ProDecoder

models based on flow content [23]. Similarly, this method also assumes that a protocol is a distribution on sessions of length at most n, where n is 64 in [23].

Wang *et al.* proposed Veritas, a system for automatically inferring protocol state machines from network traces [35]. They use a Kolmogorov-Smirnov (K-S) test based method to extract application protocol signatures, which is needed in constructing protocol state machines. Veritas also differs from ProDecoder in terms of goals - Veritas aims at obtaining protocol state machines whereas ProDecoder aims at obtaining protocol message formats. Furthermore, Veritas only analyzes the first n bytes of each packet, where n is 12 in [35].

III. ProDecoder

In this section, we present details of ProDecoder, a semantics aware approach that takes network traces as the input and automatically outputs the inferred protocol message format. ProDecoder has four major modules: n-gram generation, keyword identification, message clustering, and sequence alignment. Figure 2 shows the architecture of ProDecoder. We next provide details of each component below.

A. *n-gram Generation*

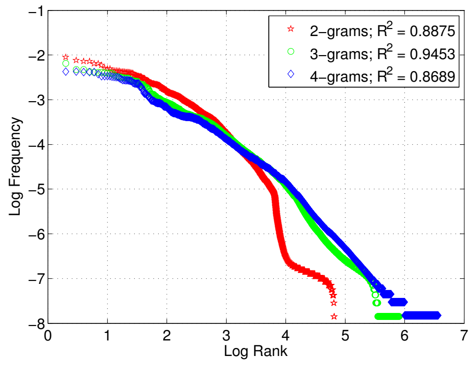
The input to ProDecoder is a set of packet traces that are of the same application protocol. The trace of one flow contains many packets. Each packet contains either a partial or complete *keyword*, or multiple *keywords* defined in the specification of the protocol. A keyword is essentially a byte sequence of arbitrary length. For example, the keywords used in the Simple Message Transfer Protocol (SMTP) include "MAIL FROM", "RCPT TO", "250", "OK", *etc.* In addition to text based protocols such as SMTP and HTTP, ProDecoder is also aimed to decode binary protocols such as SMB. Therefore, we need to further break down each keyword into constituent elements. For example, the keyword "250" can be decomposed into ”2”, "5", and "0". Note that we can aggregate consecutive elements to create up-scaled elements. For example, the three consecutive elements ”2”, "5", and "0" can be combined into the keyword "250". These elements are also known as tri-grams, where three consecutive elements can be joined together. Similarly, this process can be generalized to n-grams, where n denotes the number of consecutive elements that are joined together.

The key technical question in this module is what value should be used for n. We conducted a pilot study on the distribution of n-grams in two well-known protocols, SMTP and SMB, for varying values of n. Figure 3 shows the distributions of n-grams in both SMTP and SMB. As expected, the distribution of n-grams in both protocols is highly skewed. In Figure 3, x-axis denotes the rank of n-grams in terms of their frequency on y-axis and both axes are converted to logarithmic scale to emphasize the skewness in their distribution. We observe that the distribution of n-grams approximately follows a straight line on log-log scale, which is a characteristic of Zifp distribution [24]. It is well-known that for most natural languages, the word occurrences follow a Zipf distribution. Furthermore, we observe that the goodness-of-fit values improves for $n = 3$ compared to $n = 2$. However, we also observe that the distribution becomes highly sparse for values of $n > 4$. Using the these observations, we choose n to be 4 for our method. On a related note, some binary protocols may have keywords whose sizes are smaller than n. Such keywords are combined with adjacent bytes to from n-grams. Later in Section III-D, we show that ProDecoder can identify these keywords by using sequence alignment.

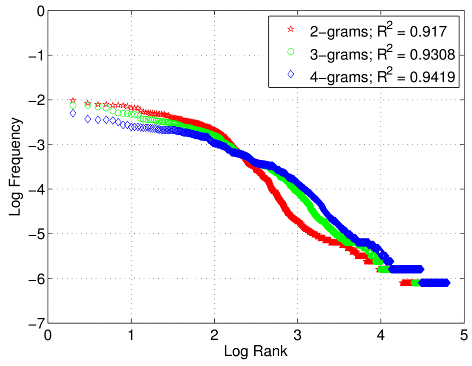
B. *Keyword Identification*

The purpose of this module is to identify the protocol keywords that appear in the given network trace of an application protocol. The input to this module is a sequence of n-grams generated by the previous module. The output of this module is a sequence of protocol keywords identified by ProDecoder, where each keyword is the concatenation of one or more n-grams. These keywords will be used in the next module to cluster messages.

We now present our Latent Dirichlet Allocation (LDA) based approach to keyword identification. Let m denote a message consisting of a vector of words $\vec{z}_m$, where each word z is a candidate protocol keyword and it consists of a vector of n-grams $\vec{w}_z$. In the context of this paper, here a message means a packet. To identify protocol keywords from a corpus of messages, we use a generative model called LDA, which has been widely used in natural language processing



(a) SMTP



(b) SMB

Fig. 3. n-gram distribution in SMTP and SMB protocols

[3], [9]. The counterpart of protocol keyword identification in natural language processing is topic identification from a corpus of documents, where each document consists of a vector of words. In our approach, each message is treated as a probability distribution of words, where each word is in turn a probability distribution over n-grams. Given a corpus of M messages, let $p(w)$ be the marginal probability that n-gram w appears in the corpus, let $p(z = z_k)$ be the marginal probability that word z_k appears in the corpus, and let $p(w|z = z_k)$ be the conditional probability that n-gram w appears in a message containing word z_k in the corpus. Therefore, for a corpus of M messages containing a total of K unique words, the marginal probability of n-gram w is defined as follows.

$$p(w) = \sum_{k=1}^{K} p(w|z = z_k)p(z = z_k) \qquad \text{subject to: } \sum_{k=1}^{K} p(z = z_k) = 1 \tag{1}$$

The task of LDA is to use the given corpus of M messages to estimate two types of distributions: (1) the n-gram distribution $p(w|z = z_k)$, denoted φ_k, for each word z_k, and (2) the word distribution $p(z)$, denoted ϑ_m, for each message m. We use two parameter sets, $\phi = \{\varphi_k\}_{k=1}^{K}$ and $\theta = \{\vartheta_m\}_{m=1}^{M}$, where each is a matrix, to represent these two types of distributions, respectively. Given ϕ and θ, the LDA model can generate a set of n-grams $w_{m,u}$, where u denotes the index of this n-gram with respect to message m. This generation process is dictated by two hyperparameters, α and β, where α is the Dirichlet prior parameter of per-message word distributions and β is the Dirichlet prior parameter of per-word n-gram distribution. Specifically, for each message m, a sample word distribution $\vec{\vartheta}_m \sim Dir(\alpha)$ is drawn, where $Dir(\alpha)$ is the Dirichlet distribution for parameter α. Based on $\vec{\vartheta}_m$, a word indicator $z_{m,u} \sim Mult(\vec{\vartheta}_m)$ is sampled for n-gram $w_{m,u}$, where $Mult(\vec{\vartheta}_m)$ is the multinomial distribution with parameter $\vec{\vartheta}_m$. For each word indicator $z_{m,u}$, the corresponding n-grams $w_{m,u} \sim Mult(\varphi_{z_{m,u}})$ are generated.

Our goal is to identify the set of keywords used in the application protocol message format. Towards this end, we need to identify the n-gram distribution for each word and the word distribution for each message in the given corpus. This is a classic Bayesian inference problem, where the target posterior distribution is defined as follows:

$$p(\vec{z}|\vec{w}) = \frac{p(\vec{z}, \vec{w})}{p(\vec{w})} = \frac{\prod_{i=1}^{W} p(z_i, w_i)}{\prod_{i=1}^{W} \sum_{k=1}^{K} p(w_i, z_i = k)} \tag{2}$$

Our target distribution $p(\vec{z}|\vec{w})$ represents word distributions for the given message corpus. The denominator $p(\vec{w})$ denotes the marginal probability of generating n-grams and the numerator $p(\vec{z}, \vec{w})$ denotes the joint probability of generating words and their corresponding n-grams. The solution to this problem gives us K keywords and their associated W n-grams for the given corpus. However, we cannot directly solve for the target distribution because denominator $p(\vec{w})$ involves a summation over K^W items and it does not factorize [3], [12]. Therefore, we cannot obtain a closed-form solution for the Bayesian inference problem described in Equation 2.

To obtain an approximate solution for the Bayesian inference problem, there are three candidate strategies: (1) variation Bayes, (2) expectation propagation, and (3) and Markov Chain Monte Carlo (MCMC). Among these three strategies, we choose MCMC because it is tolerant to local optima, requires little memory, and is competitive in speed [14]. Specifically, we use an MCMC algorithm called Gibbs sampling [15]. Gibbs sampling is an iterative algorithm, where in each iteration the value of each variable is updated by a value drawn from the target distribution of that variable conditioned on the rest of variables. For our problem, $p(\vec{z}|\vec{w})$ in Equation 2 is our target function from which we draw samples. Each sample z_i is replaced by a value drawn from the distribution $p(z_i|\vec{z}_{-i}, \vec{w})$, where z_i represents the ith component of $\vec{z}$ and $\vec{z}_{-i}$ represents z_j for any $j \neq i$. By further simplification, the conditional posterior distribution $p(z_i|\vec{z}_{-i}, \vec{w})$ can be derived from the following proportional relationship, where $n_k^{(t)}$ is the number of times that n-gram t is assigned to keyword k and $n_m^{(k)}$ is the number of times an n-gram from the message m

has been assigned to keyword k.

$$p(z_i = k|\vec{w}) \propto \frac{(n_k^{(t)} - 1 + \beta)(n_m^{(k)} - 1 + \alpha)}{(\sum_{i=1}^{W} n_k^{(t)} - 1 + W\beta)(\sum_{k=1}^{K} n_m^{(k)} - 1 + K\alpha)}, \tag{3}$$

After a sufficient number of iterations, MCMC converges and we obtain keywords $\vec{z}$, which are then used to estimate the parameter sets θ and ϕ according to the following equations:

$$\varphi_{k,t} = \frac{n_k^{(t)} + \beta}{\sum_{t=1}^{W} n_k^{(t)} + W\beta} \tag{4}$$

$$\vartheta_{m,k} = \frac{n_m^{(k)} + \alpha}{\sum_{k=1}^{K} n_m^{(k)} + K\alpha} \tag{5}$$

To ensure that the Gibbs sampling algorithm has converged and that the model with the estimated parameter sets θ and ϕ is generalizable, we use *perplexity* to quantify the quality of our estimation. Perplexity, which is defined as follows, is a well-known measure of the ability of a model to generalize to unseen data [2].

$$perplexity(D_{test}) = \exp\left\{-\frac{\sum_{m=1}^{M} \log p(\vec{w}_m)}{\sum_{m=1}^{M} N_m}\right\} \tag{6}$$

where N_m is the total number of n-grams in message m. We use all training data to compute the perplexity score. A lower perplexity score denotes better generalization performance in practice, so we prefer a lower perplexity score in ProDecoder. Perplexity also allows us to determine the right number of keywords for the given corpus of messages.

C. Message Clustering

An application protocol has many types of messages where each type of messages follow a particular format. To infer the different message formats used by an application protocol, we need to partition the given corpus of messages into multiple clusters where the messages in one cluster are of the same type following the same format. For example, given a corpus of four SMTP messages, `MAIL FROM: <alice@gmail.com>`, `MAIL FROM: <bob@live.cn>`, `RCPT TO:<smith@gmail.com>`, `RCPT TO:<john@gmail.com>`, we need to partition it into two clusters, one containing the first two messages and the other containing the last two messages. This message clustering module accomplishes this task.

In this module, for each message, we use the K keywords (and their corresponding probability) associated with the message as its K features. After keyword identification, each message m is labeled with K keywords where each keyword k is associated a probability $\vartheta_{m,k}$. Note that without the keyword identification module, we can use the n-grams generated from each message as its features; however, this naive solution has serious disadvantages compared to our method. First, keywords represent a high level abstraction of n-grams and incorporate the correlation among multiple n-grams. Directly using n-grams as features will lose such correlation information. Second, for a corpus of messages, the total number of keywords K is typically orders of magnitude smaller than the total number of n-grams; thus, using keywords as features significantly reduces the dimensionality of the clustering problem compared to directly using n-grams as features.

Using keywords and their corresponding probabilities as features, we apply the standard hierarchical clustering method. We use the Information Bottleneck (IB) [32] as the metric for cluster validation because of two main reasons. First, IB allows us to find a solution with suitable trade-off between the complexity of the model and its precision. Second, IB eliminates the need of defining similarity or distance measures for clustering in advance.

Given a set of feature vectors $X = x_1, x_2, ..., x_M$ (where M is the number of messages in the given corpus), IB allows us to partition the set into C clusters. Towards this end, we need to introduce another auxiliary random variable Y, which incorporates relevant features of X. The objective of IB is to cluster X into C clusters while preserving the relevant features Y as much as possible. Formally, IB optimizes the following expression:

$$\ell_{max} = I(C;Y) - \gamma I(C;X), \tag{7}$$

where γ works as a trade-off between $I(C;Y)$ and $I(C;X)$. Here $I(C;Y)$ denotes the mutual information between random variables C and Y. Let $p(C,Y)$, $P(C)$, and $P(Y)$ denote the joint distributions of C and Y, the marginal distribution of C, and the marginal distribution of Y, respectively. Thus, then mutual information is defined as:

$$I(C;Y) = \sum_{\forall c} \sum_{\forall y} p(c,y) \log \frac{p(c,y)}{p(c)p(y)}. \tag{8}$$

Mutual information I lies in the range $[0,1]$. The larger the value of mutual information is, the more the two random variables are dependent.

In ProDecoder, we heuristically set the number of clusters to 1.5 times the optimal number of keywords K, identified earlier. We also explored using Dunn index to select the suitable number of clusters; however, it resulted in degraded accuracy. After the number of clusters, denoted λ, is determined, we recursively merge clusters to minimize the merger cost ℓ_{max} after initially treating each message in the corpus as a distinct cluster. This recursive procedure continues till only λ clusters are left.

D. Sequence Alignment

For each cluster of messages, the sequence alignment module aims to infer the final protocol message formats for the cluster by finding the invariant fields among messages, which are in the form of regular expressions. For example, for the following cluster of three messages:

1) `MAIL FROM: <alice@microsoft.com>`
2) `MAIL FROM: <bob@berkeley.edu>`
3) `MAIL FROM: <carol@gmail.com>`

our sequence alignment module will output the regular expression

`MAIL FROM: <.*@.*..*>.`

In ProDecoder, we use the Needleman-Wunsch algorithm for sequence alignment [26]. The weight parameters of the Needleman-Wunsch algorithm used in this study are match $= 2$, mismatch $= -2$, and gap $= 1$. The basic Needleman-Wunsch algorithm can only deal with two sequences at a time. In ProDecoder, we extend it to handle multiple dimensions by lining up N sequences along N 1-dimensional edges of an N-dimensional hypercube. However, the computation of the scoring function in this scheme requires O($2^N L^N$) operations, where L represents the length of the final sequence after alignment. To improve efficiency, we use a well-known heuristic method called *progressive alignment* to perform multiple sequence alignment [13].

IV. Experimental Results

We evaluate the effectiveness of ProDecoder in inferring protocol formats of both textual and binary protocols. The input to ProDecoder is the real-world traffic trace containing packets of the target protocol and its output is the inferred corresponding protocol message formats. Below, we first describe the data set used for evaluating ProDecoder, then define the evaluation metrics, and finally present experimental results.

A. Data Set

We choose SMTP, which is used for email communication, as the target textual protocol, and SMB, which is used for file sharing, as the target binary protocol. For the traffic classification algorithm used for network trace collection, we simply use the TCP port numbers to filter traffic – port 25 for SMTP and port 445 for SMB. We collect both SMTP and SMB traces from a backbone router of a major ISP on the Internet. The payload of the SMTP (or SMB) packets in our trace will constitute the corpus of the SMTP (or SMB) messages. Our trace consists of 5,000 SMTP packets of a total of 0.34 MB, 5,000 SMB packets of a total of 0.87 MB, and 5,000 non-SMTP and non-SMB packets of a total of 1.21 MB. The overall size of our trace was limited due to computational complexity of keyword identification and message clustering modules. The average packet lengths are also small for both SMTP and SMB protocols because they mostly consist of command codes rather than payload data. We use ninety percent of the packet traces for training ProDecoder and the rest ten percent for measuring the precision and recall of ProDecoder.

B. Evaluation Metrics for Effectiveness

Given a packet trace of one application protocol, we first define the following three sets:

1) *True Positives*: the set of packets where each packet matches a regular expression generated by ProDecoder and indeed contains the application protocol fields that correspond to the regular expression.
2) *False Positives*: the set of packets where each packet matches a regular expression generated by ProDecoder but does not actually contain the application protocol fields that correspond to the regular expression.
3) *False Negatives*: the set of packets where each packet does not match any regular expression generated by ProDecoder but actually contains an application protocol field.

Next, we define the following two metrics that we use to quantitatively evaluate the effectiveness of ProDecoder:

$$\texttt{precision} = \frac{|\texttt{True Positives}|}{|\texttt{True Positives}| + |\texttt{False Positives}|} \tag{9}$$

$$\texttt{recall} = \frac{|\texttt{True Positives}|}{|\texttt{True Positives}| + |\texttt{False Negatives}|} \tag{10}$$

C. Effectiveness Results

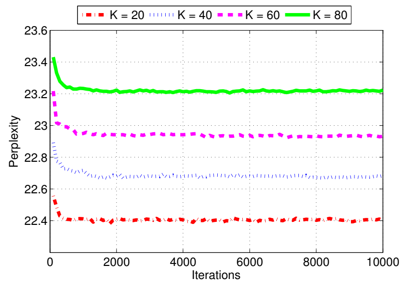
The keyword identification process of ProDecoder uses the following parameters

1) maximum iteration count L in Gibbs sampling algorithm
2) n-gram vocabulary size determined by P.
3) hyper-parameters α and β in LDA

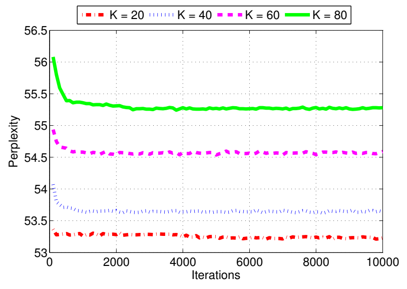
Next, we first discuss how to select a suitable value for L, and then present results for varying values of P, α, and β.

Recall that we use the Gibbs sampling algorithm to find correlations in n-grams to identify protocol keywords. We used perplexity as the metrics to ensure that the LDA model estimated using Gibbs sampling is generalizable. As Gibbs sampling is an iterative algorithm, it is important to select an appropriate maximum iteration count, denoted by L, for it to converge. To this end, we study how different values of L affect the perplexity values for both SMTP and SMB protocols. For both SMTP and SMB, we carry out experiments for $K = 20$, 40, 60, and 80 and $P = 40\%$, 60%, and 80%. Figure 4 shows the perplexity values for the above values of K and P for SMTP and SMB, respectively. For SMTP, we observe that the perplexity values typically converge by $8,000$ iterations. For SMB, we observe that the perplexity values typically converge by $1,000$ iterations. For the final evaluation of ProDecoder, we select conservative values of $L = 10,000$ for SMTP and $L = 2,000$ for SMB protocol to ensure that we achieve convergence. The keywords identified using Gibbs sampling algorithm are then used as features to cluster messages into groups. Figure 5 shows the dendrogram structure of hierarchical clustering for SMTP and SMB protocols. After clustering messages, we finally use sequence alignment to infer the final protocol message formats.

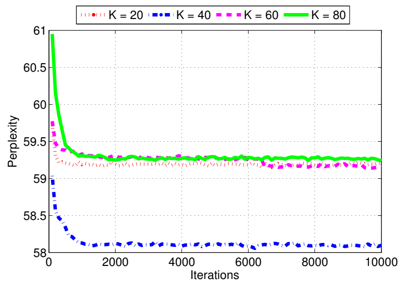
Next, we present the precision and recall results of ProDecoder for varying values of P, α, and β for both SMTP and SMB packet traces. In our evaluations, we vary the ranges of $\alpha \in \{0.1, 0.5, 0.9\}$, $\beta \in \{0.005, 0.01, 0.05, 0.1, 0.5\}$, and $P \in \{0.4, 0.6, 0.8\}$. Figures 6 and 7 show the plots of both precision and recall for varying values of α, β, and P for SMB



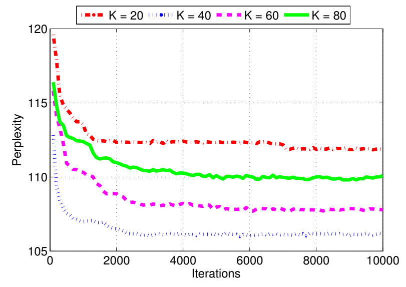
(a) SMB: P = 40%



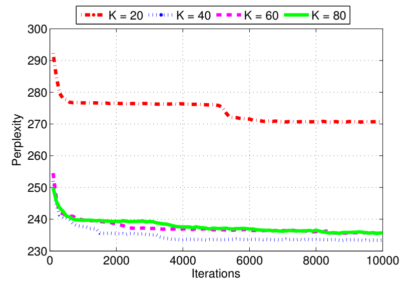
(b) SMB: P = 60%



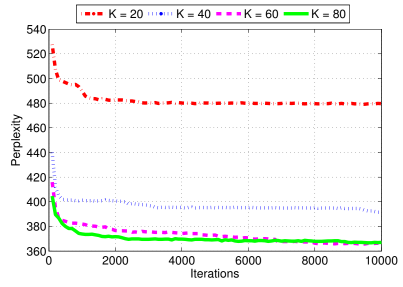
(c) SMB: P = 80%



(d) SMTP: P = 40%

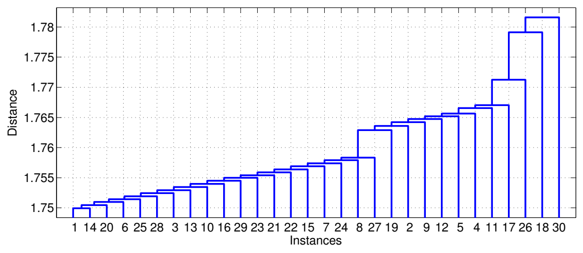


(e) SMTP: P = 60%

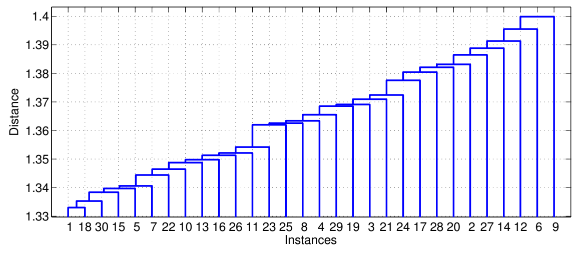


(f) SMTP: P = 80%

Fig. 4. Selection of L for SMTP and SMB protocols



(a) SMTP



(b) SMB

Fig. 5. Dendrograms of hierarchical clustering for SMTP and SMB protocols

TABLE I
SUMMARY OF RUNNING TIME OF PRODECODER'S MODULES

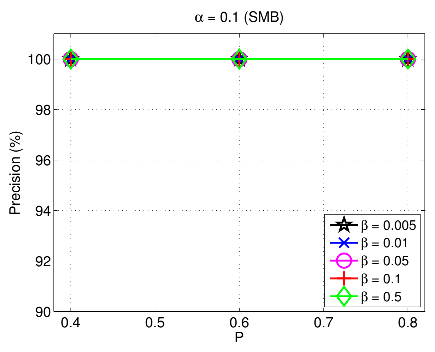
	n-gram Generation	Keyword Inference	Message Clustering	Sequence Alignment
SMTP	3 seconds	172 minutes	148 minutes	10 minutes
SMB	7 seconds	48 minutes	15 minutes	3 minutes

protocol. We observe that ProDecoder achieves 100% precision for all possible values of α, β, and P. Furthermore, the recall values of ProDecoder vary in the range of $60\% - 100\%$ for different parameter settings. The recall of ProDecoder degrades for higher values of α and β, and lower values of P. For SMB protocol, the optimal values of ProDecoder's parameters are $\alpha = 0.1$, $\beta = 0.005$, and $P = 0.8$ and the corresponding precision and recall values are both 100%.

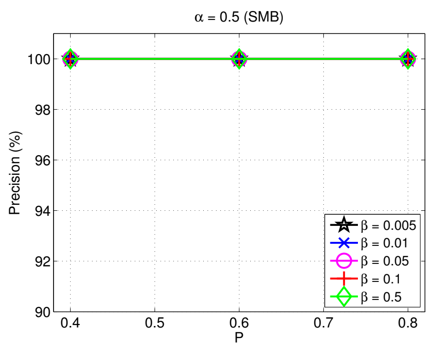
Figures 8 and 9 show the plots of both precision and recall for varying values of α, β, and P for SMTP protocol. For SMTP, we observe a different trend for the precision of ProDecoder. Specifically, the precision of ProDecoder decreases for higher values of P and lower values of β, whereas it is unrelated for α. The recall values of ProDecoder generally decrease for higher values of α and β, and lower values of P. Note that the trend observed for recall of ProDecoder is similar for both SMTP and SMB protocol. For SMTP, the optimal values ProDecoder's parameters are $\alpha = 0.1$, $\beta = 0.01$, and $P = 0.6$ and the corresponding precision and recall values are both approximately 95%.

D. Efficiency Results

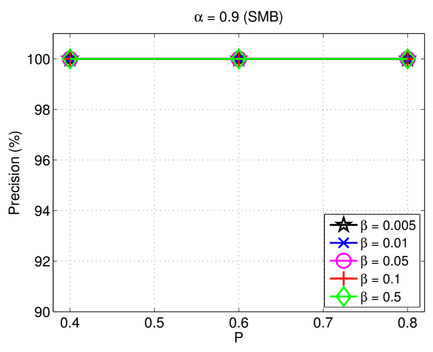
We evaluated the computational efficiency of difference modules of ProDecoder. The results are in Table I. Our experiments were executed on a cluster machine where each node had 4 quad-core Xeons processors running at 2.13GHz with 16GB RAM. We note that keyword identification and message clustering modules consume at least an order of magnitude more time than the n-gram generation module and the sequence alignment module. Note that ProDecoder runs offline for a given network trace.



(a) $\alpha = 0.1$

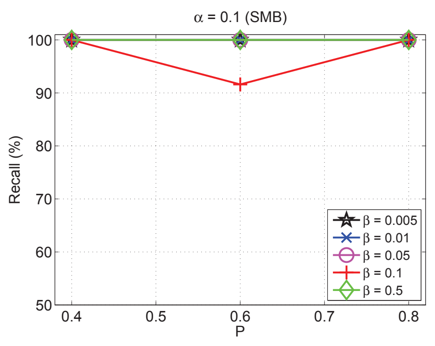


(b) $\alpha = 0.5$

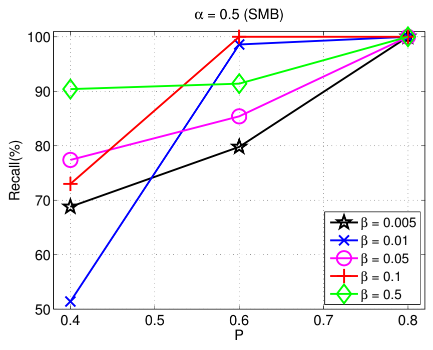


(c) $\alpha = 0.9$

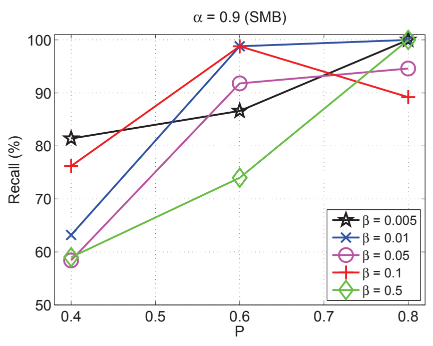
Fig. 6. Precision of ProDecoder for SMB protocol



(a) $\alpha = 0.1$

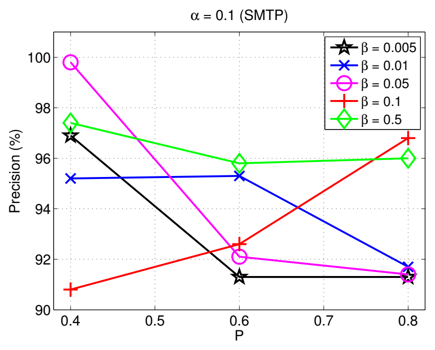


(b) $\alpha = 0.5$

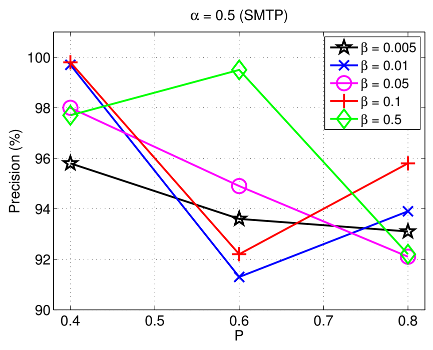


(c) $\alpha = 0.9$

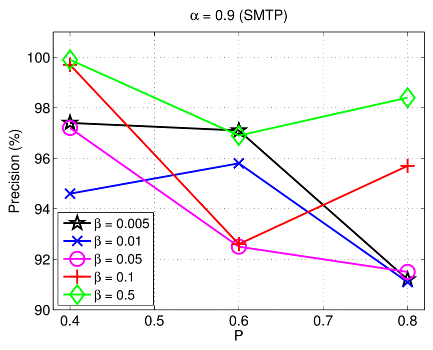
Fig. 7. Recall of ProDecoder for SMB protocol



(a) $\alpha = 0.1$

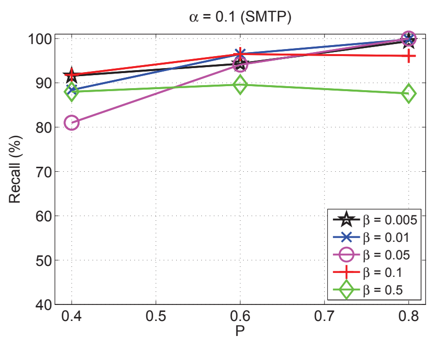


(b) $\alpha = 0.5$

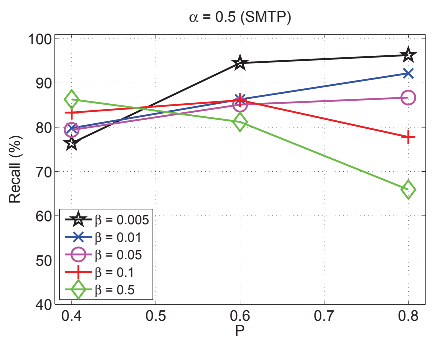


(c) $\alpha = 0.9$

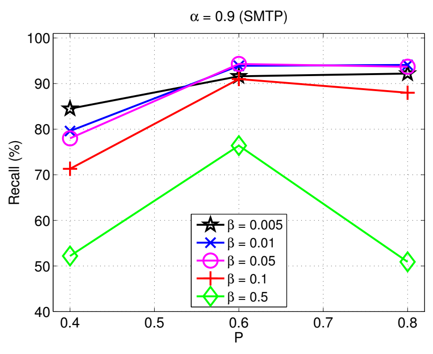
Fig. 8. Precision of ProDecoder for SMTP protocol



(a) $\alpha = 0.1$



(b) $\alpha = 0.5$



(c) $\alpha = 0.9$

Fig. 9. Recall of ProDecoder for SMTP protocol

V. Conclusions

This paper represents the first attempt that leverages the semantic information (such as the relationship among multiple common byte sequences) in protocol messages to infer their format specifications. ProDecoder is a novel multidisciplinary approach, which draws upon theories and techniques from natural language processing, machine learning, and bioinformatics literature. It is purely based on raw network packet traces and does not require protocol executable code. ProDecoder works with asynchronous application protocols and sampled network traces and does not assume any prior knowledge about protocol message formats (such as the delimiters used in protocol messages). Our evaluations on real-world network traces of two well-known textual and binary protocols showed that ProDecoder can accurately and efficiently infer protocol message format specifications.

References

[1] The Ethereal Network Analyzer. http://www.wireshark.org/.

[2] L. Azzopardi, M. Girolami, and K. van Risjbergen. Investigating the relationship between language model perplexity and IR precision-recall measures. In *Proceedings of the 26th International ACM SIGIR Conference*, pages 369–370, 2003.

[3] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, March 2003.

[4] N. Borisov, D. J. Brumley, and H. J. Wang. A generic application-level protocol analyzer and its language. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2007.

[5] J. Caballero, H. Yin, Z. Liang, and D. 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)*, 2007.

[6] C. Y. Cho, D. Babić, R. Shin, and D. Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM Conference on Computer and Communication Security (CCS)*, 2010.

[7] G. I. M. Client. http://gaim.sourceforge.net.

[8] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol specification extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (S&P)*, 2009.

[9] B. Croft. Language models for information retrieval. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, 2003.

[10] W. Cui, J. Kannan, and H. J. Wang. Discoverer: automatic protocol reverse engineering from network traces. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[11] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz. Tupni: Automatic reverse engineering of input formats. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*. 2008.

[12] J. M. Dickey. Multiple hypergeometric functions: Probabilistic interpretations and statistical uses. *Journal of the American Statistical Association*, 1983.

[13] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[14] D. Gamerman and H. F. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. 2006.

[15] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101:5228–5235, 2004.

[16] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. C. Claffy. GT: Picking up the truth from the ground for internet traffic. *SIGCOMM Computer Communication Review*, 39(5), 2009.

[17] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. Acas: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data*, 2005.

[18] A. R. Khakpour and A. X. Liu. Iustitia: An information theoretical approach to high-speed flow nature identification. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009.

[19] C. Kreibich. Design and implementation of netdude, a framework for packet trace manipulation. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2004.

[20] Z. Li, G. Xia, H. Gao, Y. Tang, Y. Chen, B. Liu, J. Jiang, and Y. Lv. NetShield: Massive semantics-based vulnerability signature matching for high-speed networks. In *Proceedings of AM SIGCOMM*, 2010.

[21] J. Lim, T. Reps, and B. Liblit. Extracting output formats from executables. In *Proceedings of the 13th Working Conference on Reverse Engineering*, 2006.

[22] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Proceedings of the 16th Network and Distributed System Security Symposium (NDSS)*, 2008.

[23] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference (IMC)*, 2006.

[24] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[25] C. Meiners, E. Norige, A. X. Liu, and E. Torng. Flowsifter: A counting automata approach to layer 7 field extraction for deep flow inspection. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[26] S. Needleman and C. 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.

[27] I. netflow statistics. http://netflow.internet2.edu.

[28] R. Pang, V. Paxson, R. Sommer, and L. Peterson. binpac: a yacc for writing application protocol parsers. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC)*, 2006.

[29] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[30] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[31] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Systems Administration Conference (LISA)*, pages 229–238, November 1999.

[32] N. Slonim and N. Tishby. Agglomerative Information Bottleneck. In *Proceedings of Neural Information Processing Systems (NIPS)*, pages 617–623, 1999.

[33] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

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

[35] Y. Wang, Z. Zhang, D. Yao, B. Qu, and L. Guo. Inferring protocol state machine from network traces: A probabilistic approach. In *Proceedings of the 9th International Conference Applied Cryptography and Network Security (ACNS)*. 2011.

[36] G. Wondracek, P. C. Milani, C. Kruegel, and E. Kirda. Automatic network protocol analysis. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.