Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 9161723, 14 pages
http://dx.doi.org/10.1155/2016/9161723

*Research Article*

# Analyzing Network Protocols of Application Layer Using Hidden Semi-Markov Model

**Jun Cai, Jian-Zhen Luo, and Fangyuan Lei**

*School of Electronics and Information, Guangdong Polytechnic Normal University, Guangzhou 510665, China*

Correspondence should be addressed to Jian-Zhen Luo; helu84@139.com

Received 13 November 2015; Accepted 22 March 2016

Academic Editor: Yakov Strelniker

With the rapid development of Internet, especially the mobile Internet, the new applications or network attacks emerge in a high rate in recent years. More and more traffic becomes unknown due to the lack of protocol specifications about the newly emerging applications. Automatic protocol reverse engineering is a promising solution for understanding this unknown traffic and recovering its protocol specification. One challenge of protocol reverse engineering is to determine the length of protocol keywords and message fields. Existing algorithms are designed to select the longest substrings as protocol keywords, which is an empirical way to decide the length of protocol keywords. In this paper, we propose a novel approach to determine the optimal length of protocol keywords and recover message formats of Internet protocols by maximizing the likelihood probability of message segmentation and keyword selection. A hidden semi-Markov model is presented to model the protocol message format. An affinity propagation mechanism based clustering technique is introduced to determine the message type. The proposed method is applied to identify network traffic and compare the results with existing algorithm.

## 1. Introduction

Network protocol specifications, describing the structure of protocol messages and regulating the behaviors of communication entities on the Internet, play an important role in addressing numbers of security or management oriented issues in several domains of computer and networking. For example, intrusion detection systems and firewall systems require protocol specifications to perform deep packet inspection. Security experts spy and understand the specification of command & control (C&C) protocols [1] to detect and defend the botnets. Network management administrators build up application signatures based on protocol specifications to identify protocols and tunnels in monitored network traffic. Fuzz tests [2] make use of protocol specifications to reduce the number of fault-inserted files while still maintaining the maximum test case coverage. The protocol specifications are also powerful tools to enable the interoperation between multiple systems based on incompatible protocols [3–5].

A complete specification is referred to as both protocol message format and protocol state machine. The former reveals the protocol syntax which conducts the process of constructing different types of messages to be exchanged between communication entities, while the latter formulates the behaviors of protocol entities during the whole process of communication, such as the order in which different types of messages should be sent or received. For open protocols, like HTTP and FTP, their specifications can be obtained by means of accessing to the published documents. However, for proprietary protocols used by enterprises or hackers, their specifications would not be unpublished for commercial or security reasons. To date, more and more new protocols and mobile applications emerge every day due to the rapid development of mobile Internet and unprecedented popularity of smart phones [6]; network management administrators need to know about the specifications of these protocols or applications to monitor the network traffic. However, there is no public documentation about their specifications. Over the past few years, researchers deem that the only available option to spy the specification of proprietary protocol or new emerging mobile applications is protocol reverse engineering.

Traditionally, protocol reverse engineering is performed by manual analysis, which is time-consuming and error-prone. For example, the Samba project has taken over 12 years to manually recover the specification of SMB/CIFS [3]. In the Pidgin project [4], the Pidgin plug-ins have to be patched when the target protocol is changed and the delay between the protocol changes and working patches could be months, caused by reverse engineering. In order to address these problems, automatic protocol reverse engineering has been proposed over the last decade and has become a heat topic in research field of network traffic analysis.

Automatic protocol reverse engineering is a process of recovering protocol message formats and inferring protocol state machine without access to the specification of target protocol. Generally, automatic protocol reverse engineering can be divided into network trace based approach and binary analysis based approach. The network trace based approach takes captured network trace as input and reconstructs message formats by identifying basic components, such as message fields or protocol keywords, using techniques introduced from the fields of data mining, bioinformatics, nature language processing, and so on. The binary analysis based approach operates by observing how the executable binary software implementing the target protocol makes use of the memory and registers during the runtime to process the received messages or construct the sent message. The former approach is easy to deploy and relies only on the network trace generated by the target protocol, while the latter approach is useful for the scenarios where executable binary software is available and can be run in a control environment.

In this paper, we focus on recovering the message formats from network trace using the network trace based approach. Our goal is to identify the location of message fields and determine the length of protocol keywords. The message format is comprised of message fields. Some fields (called keyword fields) contain the protocol keywords. The protocol keywords are some constants or commands used by network protocol. For example, "GET", "HTTP", and "POST" are some protocol keywords used by HTTP protocol.

The first challenge in our research is to determine the length of protocol keywords. Previous works [7–12] which are based on longest common subsequence (LCS) criteria select longest frequent substrings to be protocol keywords. For example, if "G", "E", "T", "GE", "ET", and "GET" are frequent substrings, "GET" will be chosen as the protocol keyword, since it is the longest substring. However, if the frequency threshold is low enough, "GET abc" ("abc" is a string that follows "GET") will become a frequent string, so "GET abc" will be chosen as protocol keyword, while the true keyword "GET" would be dropped. Therefore, it is not rational to simply choose the longest frequent substrings as protocol keywords.

The second challenge is to deal with binary protocols. It is easy to define and understand the protocol keywords that bound the message fields in text protocols which restrict their content to printable ASCII characters. However, for binary protocols, fields are predefined by the protocol specifications to represent specific meanings instead of using the protocol keywords as the preambles. Messages containing only fixed-length fields are not difficult to recover. However, the complexity will increase dramatically when the fields are variable in length.

The third challenge is to determine the location relationship of message fields. The relationship of fields varies from sequence to juxtaposition. For example, in the request message of HTTP, the request method field "GET" and the HTTP version field "HTTP/1.1" are of sequential relation, which means that "GET" must occur in some location before the position of "HTTP/1.1" and the location of the two fields can not be exchanged, while some other fields, such as the "Host" field and the "Server" field, are of juxtapositional relation, which means that their locations can be exchanged with each other.

In this paper, we apply a probabilistic model, hidden semi-Markov model (HsMM) [13], to address the challenges of our work. On the one hand, one can find out the optimal length of the protocol keyword with maximal likelihood probability based on the HsMM. Obviously, the length of keyword based on maximal likelihood probability is much more reasonable and rigorous than those empiristic decisions of choosing the longest frequent substrings. On the other hand, the HsMM model is a probabilistic directed graph (lattice). Each node in the lattice represents a state that can emit various observations. The states in the same longitude are of sequential relation, while states in the same latitude are of juxtapositional relation. Therefore, it is natural to use HsMM to model the sequential and juxtapositional relation of fields.

The organization of this paper is as follows. In Section 2, related work about protocol reverse engineering is studied. In Section 3, a brief review of the concept and definition about HsMM is illustrated. In Section 4, the proposed method of modeling message format using HsMM is presented in detail. In Section 5, the system architecture is presented and some implementation issues are discussed. In Section 6, the proposed method is evaluated and the experiment results are shown. Finally, a conclusion is made in Section 7.

## 2. Related Work

Over the past few years, automatic protocol reverse engineering has attracted tremendous research interest in both research and industry field of computer and networking application. Numbers of works have been published to discuss and address many issues about the heat topic. Beddoe [7] proposes to make use of algorithms widely used in the field of bioinformatics, that is, the sequence alignment algorithms and phylogeny construction algorithm, to determine the location and size of field in each individual packet. Beddoe presents his effort in the protocol informatics project and implements his approach in Python to extract the longest common subsequence (LCS) as message fields with constant value. Kreibich and Crowcroft [8] introduce a novel variant of the Jacobson-Vo algorithm [14] to compute the LCSs of input strings and employ a flexible gap-minimising algorithm to improve the efficiency and effectiveness of network traffic alignment. The authors show that their method outperforms the commonly used Smith-Waterman approach on a wide range of network protocols. Both Beddoe [7] and Kreibich and Crowcroft [8] aim to mine the commonalities of messages as the basic components of message formats based on LCS, while our approach is to infer the location and length of message fields based on the maximal likelihood probability.

Cui et al. present Discoverer [15] to recursively cluster and align the token patterns of messages to infer protocol message format idioms. Although Discoverer is practicable to recover the protocol message formats of three selected protocols, that is, HTTP, RPC, and SMB/CIFS, there are still about 10% of the message formats that could not be correctly inferred due to some inaccurate parsing. Discoverer firstly tokenizes the protocol messages and initially clusters messages according to the token patterns. Thus, the lengths of fields are factitiously forced to be consistent with the size of tokens and the boundaries of message fields in the text protocols are restricted to some separators (such as space) specified by the authors. Moreover, the relationship of fields in message formats inferred by Discoverer is sequential. In our approach, we do not make any assumption about the separators and aim to infer the optimal length of fields by maximizing the likelihood probability of message segmentation. Meanwhile, we capture the location relationship of fields, such as sequential and juxtapositional relation, by learning a probabilistic directed lattice graph.

Wang et al. [16] present a framework to infer message formats by improving the Aho-Corasick (AC) algorithm [17] to identify frequent sequences and mining the association rules among the frequent sequences. They evaluate the framework in wireless environment and show that the framework can identify ARP and ICMP packets in high accuracy. However, their framework only searches for association rules of some frequent fields in protocol messages, while the aim of our scheme is to infer the whole format of message by inferring all of the message fields.

Wang et al. propose Biprominer [18] to extract binary protocol message formats based on the statistical nature of message formats. Firstly, the Biprominer recursively learns and labels frequent patterns in the message based on the frequency of blocks (comprised of several bytes). Then, the messages with labeled blocks are converted into a transition probability model. Antunes and Neves [19] present building an automaton based on sequence alignment algorithm for recovering message formats from network trace. They firstly extend the partial order alignment algorithm to generate an initial automaton from messages, then apply sequence alignment techniques to find out the optimal alignment between the automaton and the new coming messages, and finally use the alignment results to further extend the automaton. These researches focus on modeling the transition probability of message blocks or finding out the acceptable paths of bytes in the automatons, while our work aims to identify message fields with variable length as well as model the location relationship of fields.

Some works leverage the semantics analysis of message fields to infer message formats. The so-called semantics analysis is to identify the keyword sequences, each of which indicates a specific intention of the protocol message. Krueger et al. [20] present a semantics-aware tool for network payloads analysis to automatically extract semantics-aware components from captured network trace. They map protocol messages to a vector space based on tokens or words and identify communication templates corresponding to the base directions in the vector space. Wang et al. propose ProDecoder [21] to reconstruct the message formats based on semantics-aware approach. ProDecoder first identifies keywords using Latent Dirichlet Allocation (LDA) models taken from natural language processing. Protocol messages are then clustered according to their semantics (different combination of keywords) using the Information Bottleneck clustering algorithm. Finally, messages in each cluster are aligned to find out the common parts among them using well-known sequence alignment algorithms. These methods aim to reveal the semantics characteristics of protocol messages under specific communication motivations, so the message formats are expected to be affected by the user intentions. However, our method captures the general structures of messages of the target protocol.

As an alternative approach to understand the unknown or proprietary protocols, binary analysis based techniques also draw much research attention in the field of network security. For example, Polyglot [22], Tupni [23], AutoFormat [24], Prospex [25], and Dispatcher [26] are all systems based on binary analysis techniques. They are workable and applicable in the scenarios where the binary software is available and can be run in a controlled environment. In addition, binary analysis techniques can not work when the binary clients apply some interference techniques, such as obfuscation, to protect themselves from being detected and reverse-engineered. In this paper, we narrow our research into the application scene that only the network trace of target protocols is available. Hence, we do not discuss these binary analysis based techniques but focus on those approaches based on network trace.

## 3. Hidden Semi-Markov Models

A hidden semi-Markov model (HsMM) as shown in Figure 1 is an extension of hidden Markov model (HMM) by allowing the underlying process to be a semi-Markov chain with a variable duration time for each state [13, 27].

The basic elements of HsMM include the hidden state set

$$\mathcal{S} = \{1, 2, \ldots, M\}, \tag{1}$$

the state duration set

$$\mathcal{D} = \{1, 2, \ldots, D\}, \tag{2}$$

and the observation set

$$\mathcal{V} = \{v_1, v_2, \ldots, v_K\}. \tag{3}$$

The hidden state of underlying process at time $t$ is donated as $s_t \in \mathcal{S}$. The symbols $i$ and $j$ are used to

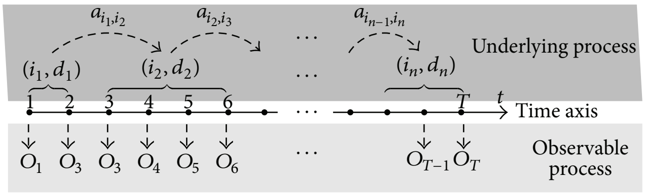


Figure 1: Hidden semi-Markov model.

represent substantive values of state variable $s$. For simplicity of notation, we denote the following:

(i) $s_{t_1}^{t_2} = i$ means $s_{t_1} = s_{t_1+1} = \cdots = s_{t_2} = i$; however, the previous state $s_{t_1-1}$ and the next state $s_{t_2+1}$ may or may not be $i$.

(ii) $s_{[t_1}^{t_2]} = i$ means $s_{t_1} = s_{t_1+1} = \cdots = s_{t_2} = i$; however, neither $s_{t_1-1}$ nor $s_{t_2+1}$ is $i$.

(iii) $s_{t_1}^{t_2]} = i$ means $s_{t_1} = s_{t_1+1} = \cdots = s_{t_2} = i$ and $s_{t_2+1} \neq i$; however, the previous state $s_{t_1-1}$ may or may not be $i$.

(iv) $s_{[t_1}^{t_2} = i$ means $s_{t_1} = s_{t_1+1} = \cdots = s_{t_2} = i$ and $s_{t_1-1} \neq i$; however, the next state $s_{t_2+1}$ may or may not be $i$.

As shown in Figure 1, the observation sequence $O_1^T$ is the observable process, while the state sequence $s_1^T$ and the state transitions $(i_m, d_m) \rightarrow (i_{m+1}, d_{m+1})$, $m = 1, 2, \ldots, n-1$, are underlying process that cannot be observed. For each pair $(i_m, d_m)$ in the underlying process, $d_m$ is the time duration of state $i_m$.

Formally, a HsMM can be represented by

$$\lambda = (A, B, P, \pi), \tag{4}$$

where $A$ is the state transition probability matrix, $B$ is the emission probability matrix, $P$ is the distribution of state durations, and $\pi$ is the initial distribution of states.

The state transition probability matrix is defined as

$$A = \left\{a_{i,j} \mid \forall i, j \in \mathcal{S}\right\}, \tag{5}$$

where $a_{i,j} = P(s_{[t+1} = j \mid s_{t]} = i)$, subject to $\sum_{j \in \mathcal{S} \setminus i} a_{i,j} = 1$ and zero self-transition probabilities $a_{i,i} = 0$, for all $i, j \in \mathcal{S}$.

The emission probability matrix $B$ is defined as

$$B = \left\{b_i(v) \mid \forall i \in \mathcal{S}, v \in \mathcal{V}\right\}, \tag{6}$$

where $b_i(v) = P(o_t = v \mid s_t = i)$ means that $v$ is observed at $t$ in state $i$.

The distribution of the state duration is

$$P = \left\{p_i(d)\right\}, \quad i \in \mathcal{S}, \ d \in \mathcal{D}. \tag{7}$$

The initial distribution of states indicates the probability of the initial state before time $t = 1$; that is,

$$\pi(i) = P\left(s_t = i\right), \quad t \le 1, \ i \in \mathcal{S}. \tag{8}$$

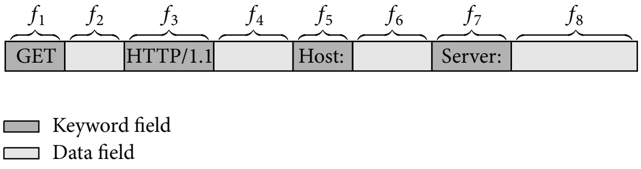


Figure 2: Message format.

## 4. Protocol Modeling

*4.1. Modeling Network Protocol Using HsMM.* A network protocol is a set of rules for regulating the exchange of messages in the Internet. The specification of network protocol describes the strict syntactical format for valid message and defines the strict procedure rules of data exchange. The alphabet of valid messages is the set of all possible values of a single byte; that is,

$$\Sigma = \{0x00, 0x01, 0x02, \ldots, 0xFF\}. \tag{9}$$

A string $\omega$ over $\Sigma$ is defined as a finite sequence of letters in $\Sigma$; that is, $\omega = \overline{a_1, a_2, \ldots, a_n}$, $(a_1, a_2, \ldots, a_n \in \Sigma)$. The set of all finite strings over alphabet $\Sigma$ is represented as $\Sigma^*$.

The protocol message, denoted as $m$, is defined as the basic data unit exchanged between different communicating entities of application-layer protocol. A message consists of a set of message fields, including keyword fields and data fields, as shown in Figure 2. The message fields, denoted as $f$, are strings over $\Sigma$; that is, $f \in \Sigma^*$.

The valid messages exchanged by communicating entities are constructed according to the protocol message format. The relationship of field location in the message format is varying from sequential to juxtapositional. For example, according to the HTTP specification, message fields $f_1$, $f_2$, and $f_3$ in Figure 2 are of sequential relation; that is, the location of $f_2$ must go after $f_1$ but preceded $f_3$. However, the relation of fields $f_5$ and $f_7$ is juxtapositional that means the location of $f_5$ and $f_7$ can be exchanged with each other.

In order to model message format using HsMM, protocol message is treated as an observation sequence representing the observable process. Each field is a block of observations associated with a specific hidden state with the length of this field as the corresponding state duration. For example in Figure 3, $f_1$ is the block of observations from $t = 1$ to 3 associated with state $i_1$ and duration $d_1 = 3$. In this model, the emission probability matrix $B$ implies the relationship between observations and hidden states, while the state transition probability matrix $A$ implies the relationship of field location.

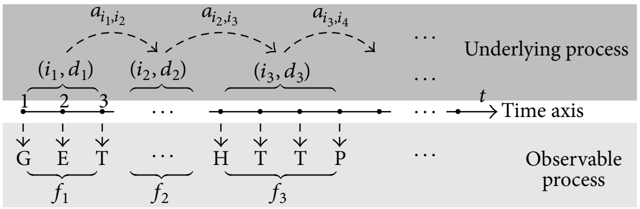


FIGURE 3: Illustration of modeling HTTP based on hidden semi-Markov model.

Let $O$ be an observation sequence and let $\Omega$ be the set of frequent strings that occurred in $O$. Given $\omega', \omega \in \Omega$, we denote that $\omega' \subset \omega$, if $\omega'$ is the substring of $\omega$. The string $\omega$ is closed in $\Omega$, if there does not exist a string $\omega'' \in \Omega$ to satisfy $\omega \subset \omega''$. The set of closed frequent strings in $\Omega$ is denoted as $\mathscr{L}$.

Each closed string in $\mathscr{L}$ is associated with different hidden states; thus, the number of hidden states for closed string in $\mathscr{L}$ is $N = \|\mathscr{L}\|$. Suppose that $\omega_i \in \mathscr{L}$ is associated with state $i$; then all characters in $\omega_i$ are observations of state $i$.

Additionally, we define other $M - N$ special states ($i = N+1, N+2, \ldots, M$) which are associated with any characters in $\Sigma$.

*4.2. Parameters Reestimation.* In this section, we discuss an iterative procedure for reestimating the parameters of $\lambda = (A, B, P, \pi)$, based on the Baum-Welch method [28]. At the beginning, a random initialization of $A$ and $\pi$ is selected, while the initialization of $B$ and $P$ is processed as follows.

For $i \in \{1, 2, \ldots, N\}$, $b_i(c) = \exp(-\|\omega_i\|/10)$, if $c \in \Sigma$, where $\omega_i \in \mathscr{L}$ is the closed frequent string associated with $i$. Otherwise, $b_i(c) = 0$, if $c$ does not occur in $\omega_i$.

For $i \in \{N+1, N+2, \ldots, M\}$, the emission probability of letter $c$ in state $i$ is $b_i(c) = \exp(-20)$.

For $i \in \mathscr{S}$ and $d \in \mathscr{D}$,

$$p_i(d) = \frac{d^2}{\sum_{k=1}^{D} k^2}. \tag{10}$$

In the forward-backward procedure, the forward variable is defined as

$$\alpha_t(j, d) \equiv P\left((s_t, \tau_t) = (j, d), O_1^t\right), \tag{11}$$

where $\tau_t$ is the remaining time of the current state $s_t$.

Initially, $\alpha_1(i, d) = \pi(i) b_i(O_1) p_i(d)$.

The inductive solution for $\alpha_t(i, d)$ when $t > 1$ is given as follows:

$$\begin{aligned} \alpha_t(i, d) = {} & \alpha_{t-1}(i, d+1)\, b_i(o_t) \\ & + \left(\sum_{j \neq i} \alpha_{t-1}(j, 1)\, a_{ji}\right) b_i(o_t)\, p_i(d), \\ & \qquad d \geq 1. \end{aligned} \tag{12}$$

The backward variable is defined as

$$\beta_t(i, d) \equiv P\left(O_{t+1}^T \mid (s_t, \tau_t) = (i, d)\right). \tag{13}$$

Initially, $\beta_T(i, d) = 1$.

The inductive solution for $\beta_t(i, d)$ when $1 \leq t < T$ is given as follows:

$$\begin{aligned} \beta_t(i, d) &= b_i(o_{t+1})\, \beta_{t+1}(i, d-1), \quad d > 1, \\ \beta_t(i, 1) &= \sum_{j \neq i} a_{ij} b_j(o_{t+1}) \left(\sum_{d \geq 1} p_j(d)\, \beta_{t+1}(j, d)\right). \end{aligned} \tag{14}$$

We define the probability that the state $i$ ends at time $t$, while the state $j$ starts at time $t+1$, given the entire observation sequence $O_1^T$, as follows:

$$\begin{aligned} \xi_t(i, j) &\equiv P\left(O_1^T, s_{t-1} = i, s_t = j\right) \\ &= \alpha_{t-1}(i, 1)\, a_{ij} b_j(o_t) \left(\sum_{d \geq 1} p_j(d)\, \beta_t(j, d)\right). \end{aligned} \tag{15}$$

The probability that the state $j$ ends at time $t$ with its duration being $d$, given the entire observation sequence $O_1^T$, is defined as

$$\begin{aligned} \eta_t(i, d) &\equiv P\left(s_{t-1} \neq i, s_t = i, \tau_t = d, O_1^T\right) \\ &= \left(\sum_{j \neq i} \alpha_{t-1}(j, 1)\, a_{ji}\right) b_i(o_t)\, p_i(d)\, \beta_t(i, d). \end{aligned} \tag{16}$$

The probability that the state at time $t$ is $i$, given the entire observation sequence $O_1^T$, is defined as

$$\gamma_t(i) \equiv P\left(s_t = i, O_1^T\right). \tag{17}$$

In order to solve for $\gamma_t(i)$, we consider the following identities:

$$\begin{aligned} P\left(s_t^{t+1} = j, O_1^T\right) &= P\left(s_t = j, O_1^T\right) - P\left(s_{t]} = j, O_1^T\right), \\ P\left(s_t^{t+1} = j, O_1^T\right) &= P\left(s_{t+1} = j, O_1^T\right) \\ &\quad - P\left(s_{[t+1} = j, O_1^T\right). \end{aligned} \tag{18}$$

Thus, we have a recursive formula for $\gamma_t(i)$ as follows:

$$\gamma_t(i) = \gamma_{t+1}(i) + \sum_{j \neq i} \left(\xi_{t+1}(i, j) - \xi_{t+1}(j, i)\right). \tag{19}$$

In the phase of recursively computing $\gamma_t(i)$, the initial condition is given as follows:

$$\gamma_T(i) = \sum_{d \geq 1} \alpha_T(i, d). \tag{20}$$

With these notations, the parameters of $\lambda$ can be updated and improved by the following equations:

$$\begin{aligned}
\widehat{\pi}_i &= \frac{\gamma_1(i)}{\sum_i \gamma_1(i)}, \\
\widehat{a}_{ij} &= \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{j\neq i}\sum_t \xi_t(i,j)}, \\
\widehat{p}_i(d) &= \frac{\sum_{t=1}^{T} \eta_t(i,d)}{\sum_{d=1}^{D}\sum_{t=1}^{T} \eta_t(i,d)}, \\
\widehat{b}_i(v_k) &= \frac{\sum_{t=1}^{T} \gamma_t(i)\,\mathbf{I}(o_t == v_k)}{\sum_k \sum_{t=1}^{T} \gamma_t(i)\,\mathbf{I}(o_t == v_k)}.
\end{aligned} \tag{21}$$

Note that $\mathbf{I}(\text{expression}) = 1$, if expression is true. Otherwise $\mathbf{I}(\text{expression}) = 0$, if expression is not true.

*4.3. Inferring Protocol Keywords.* Given the reestimated HsMM $\widehat{\lambda} = (\widehat{A}, \widehat{B}, \widehat{P}, \widehat{\pi})$ and an observation sequence $O$, the forward and backward variables can be computed based on forward-backward algorithm. Then, the variable $\eta_t(j,d)$ can be computed using (16). In what follows, we can infer the state sequence with maximal likelihood probability based on the Viterbi algorithm [29]. The inference procedure is given as follows:

$$\begin{aligned}
(i_1, d_1) &= \arg\max_{j,d} \left(\eta_T(j,d)\right), \\
(i_2, d_2) &= \arg\max_{j,d} \left(\eta_{T-d_1}(j,d)\right), \\
(i_3, d_3) &= \arg\max_{j,d} \left(\eta_{T-d_1-d_2}(j,d)\right), \\
&\ \ \vdots \\
(i_n, d_n) &= \arg\max_{j,d} \left(\eta_{T-d_1-d_2-\cdots-d_{n-1}-d_n}(j,d)\right).
\end{aligned} \tag{22}$$

The iteration proceeds until $d_1 + d_2 + \cdots + d_n = T$. Thus, the observation $O$ is divided into a sequence of fields with the $k$th field to be $\omega_k = O_{T-d_1-\cdots-d_{k-1}-d_k+1}^{T-d_1-\cdots-d_{k-1}}$. $i_k$ is referred to as the state of $\omega_k$. If $1 \le i_k \le N$, $\omega_k$ is a protocol keyword with the corresponding field as keyword field. If $N < i_k \le M$, then $\omega_k$ is a data string and the corresponding field is a data field.

*4.4. Inferring Message Type.* In this section, we present an algorithm to determine the type of protocol messages. The messages which belong to the same type have similar formats with each other. Thus, the type of protocol messages can be determined using clustering method according to the similarities between their message formats.

In this paper, we apply an unsupervised clustering algorithm proposed by Frey and Dueck [30] to solve the problem. The algorithm based on the affinity propagation mechanism takes the similarity matrix of data points as input and recursively selects representative exemplars for each point. Each of the selected exemplars represents a data type, while the type of other data points is determined by the exemplars they select. The number of clusters need not be specified beforehand. The similarity metric need not be defined strictly in a continuous space and does not have to satisfy the symmetric and the triangle inequality. Therefore, we can define the similarity in any reasonable way.

Before the further discussion about the message clustering algorithm, we define some basic notations. Suppose the universal set of protocol keywords is denoted as $K$ and the set of protocol keywords that occurred in message $m_j$ is denoted as $K_j$. Given a protocol keyword $k$ of message $m_i$, the cost of encoding $k$ in $m_i$ using the keyword set of message $m_j$ using $K_j$ as the code book is defined as

$$c_{i,j}(k) = \begin{cases} \log_2 \left|K_j\right|, & k \in K_j, \\ \log_2 |K|, & \text{otherwise}. \end{cases} \tag{23}$$

The similarity of $m_i$ to $m_j$ is defined as the minus summation of cost of encoding all keywords in $m_i$ using $K_j$ as code book is defined as

$$\text{Sim}(i,j) = -\sum_{\forall k \in \theta_i} c_{i,j}(k). \tag{24}$$

The affinity propagation algorithm exchanges two kinds of information between data points during the clustering process: responsibility ($r(i,k)$) and availability ($a(i,k)$). The "responsibility" $r(i,k)$, sent from an ordinary data point $i$ to the candidate exemplar point $k$, reflects the accumulated evidence for how well-suited point $k$ is to serve as the exemplar for point $i$, taking into account other potential exemplars for point $i$. The "availability" $a(i,k)$, sent from candidate exemplar point $k$ to point $i$, reflects the accumulated evidence for how appropriate it would be for point $i$ to choose point $k$ as its exemplar, taking into account the support from other points that point $k$ should be an exemplar.

In this paper, we treat each message as a data point, and the responsibility and availability are updated according to the following equations:

$$\begin{aligned}
r(i,k) &\longleftarrow \text{Sim}(i,k) - \max_{k' \neq k} \left\{a\left(i,k'\right) + \text{Sim}\left(i,k'\right)\right\}, \\
a(i,k) & \\
&\longleftarrow \min\left\{0, r(k,k) + \sum_{i' \neq i,k} \max\left\{0, r\left(i',k\right)\right\}\right\}.
\end{aligned} \tag{25}$$

Specially, $a(k,k)$ is updated by

$$a(k,k) \longleftarrow \sum_{\forall i' \neq k} \max\left\{0, r\left(i',k\right)\right\}. \tag{26}$$

The affinity propagation algorithm clusters messages into subclusters, each of which represents a type of messages. The results of message type inference are important for constructing protocol state machine which will be discussed in our future work.

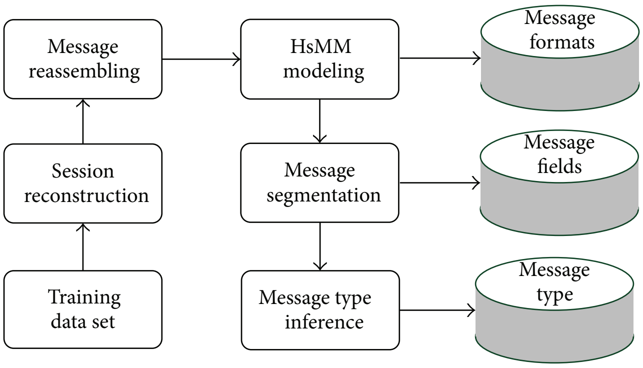


Figure 4: Overview of system architecture.

## 5. System Implementation

In this section, we will illustrate an overview of our system architecture and discuss some implementation issues which have to be addressed when one implements the proposed approach.

*5.1. System Overview.* A brief view of our system architecture is shown in Figure 4. Training data set is raw traffic captured from real world using a well-known network traffic analysis tool called tshark [31].

Since well-known protocols, such as HTTP, are well studied and described in public documents, almost all of pop analyzer tools of network traffic embed and identify well these protocols, so the true ground of well-known protocols is easy to be obtained. As a result, we consider some well-known protocols to validate and evaluate our approach in this paper and assume that the training data set is generated by only one protocol.

In the session reconstruction phase, we reconstruct the sessions according to the 5-tuple, that is, transport protocol, source transport number, destination transport number, source IP address, and destination IP address. For TCP-based protocol, a session starts at the packet with the SYN flag in TCP header and finishes when the FIN flag is acknowledged. For UDP protocol, a session is defined as the packets shared the same 5-tuple.

In the message reassembling phase, messages of TCP-based protocols are reassembled from packets according to the TCP sequence number and acknowledgement number while the messages of UDP-based protocols are reassembled according to the arrival time stamp of packets and the transmission direction of packets.

In the HsMM modeling step, an algorithm based on the Baum-Welch method is performed to reestimate the parameters of the HsMM-based protocol model. The reestimated HsMM model produced by this step implies the message format.

In the message segmentation phase, the reestimated HsMM model is applied to determine the optimal length of protocol keywords and divide message into field sequence.

In the step of message type inference, protocol messages are clustered using the affinity propagation mechanism and each cluster represents a type of messages.

```
(1)  **Input**: observation O, frequency threshold $\Gamma$
(2)  **Output**: closed frequent string set $\mathscr{L}$
(3)  # Find out the frequent strings
(4)  **Initialization**: frequent candidate set $C_1 = \Sigma$, $i = 1$
(5)  **while** $C_i \neq \phi$ **do**
(6)      # Check frequency of strings in $C_i$
(7)      **for** $\omega \in C_i$ **do**
(8)          # Freq($\omega$) is the frequency of $\omega$ in $O$
(9)          **if** Freq($\omega$) $< \Gamma$ **then**
(10)             Delete $\omega$ from $C_i$
(11)         **end if**
(12)     **end for**
(13)     # Generate new candidate set $C_{i+1}$
(14)     **for** $\omega_1, \omega_2 \in C_i$ **do**
(15)         **if** $\omega_1[1 : i-1] = \omega_2[2 : i]$ **then**
(16)             Create a new string $\omega'[1 : i+1]$
(17)             Let $\omega'[1 : i] = \omega_2[1 : i]$, $\omega'[i+1] = \omega_1[i]$
(18)             Add $\omega'$ to $C_{i+1}$
(19)         **end if**
(20)     **end for**
(21)     $i = i + 1$;
(22) **end while**
(23) # Find out the closed frequent strings
(24) **Initialization**: $i = 1$
(25) **while** $C_{i+1} \neq \phi$ **do**
(26)     **for** $\omega_1 \in C_i$ **do**
(27)         **for** $\omega_2 \in C_{i+1}$ **do**
(28)             # delete the substrings
(29)             **if** $\omega_1 \subset \omega_2$ **then**
(30)                 Delete $\omega_1$ from $C_i$
(31)                 Break
(32)             **end if**
(33)         **end for**
(34)     **end for**
(35)     Update $\mathscr{L} = \mathscr{L} \cup C_i$
(36)     $i = i + 1$
(37) **end while**
(38) Update $\mathscr{L} = \mathscr{L} \cup C_i$
```

Algorithm 1: Closed frequent string algorithm.

*5.2. Extracting Closed Frequent Strings.* Suppose that $\mathscr{L}$ is a frequent string set. If $\omega \in \mathscr{L}$ and there do not exist $\omega' \in \mathscr{L}$ satisfying that $\omega$ is the substring of $\omega'$, then $\omega$ is a closed frequent string in $\mathscr{L}$. In this section, the Apriori algorithm [32] widely used in data mining field is introduced and modified to address the problem of mining closed frequent strings as shown in Algorithm 1.

The frequent string candidate set $C_i$ is initialized as $C_1 = \Sigma = \{0, 1, \ldots, 255\}$, each element in which represents a one-byte character (line (4)). Note that the length of each element in $C_i$ is $i$. The frequencies of elements in $C_i$ are checked and the ones whose frequencies are less than the frequency threshold $\Gamma$ would be deleted from $C_i$ (lines (6)~(12)). The candidates of frequent strings with length of $i+1$ are generated in lines (14)~(20), where the notation $\omega[1 : i]$ represents the byte sequence from the first byte to $i$th byte in $\omega$. If $\omega_1, \omega_2 \in C_i$ and the first $i-1$ characters of $\omega_1$ are equal to the last $i-1$ characters of $\omega_2$, then the two strings can be combined into

Table 1: Results of keyword extraction for text-based protocols.

| System | Protocol | True keyword | Inferred keyword | True positive | Precision (%) | Recall (%) |
|---|---|---|---|---|---|---|
| HsMM | HTTP | 36 | 80 | 33 | 41.25 | 91.67 |
| | SSDP | 24 | 54 | 24 | 44.44 | 100 |
| Discv | HTTP | 36 | 859 | 25 | 2.33 | 69.44 |
| | SSDP | 24 | 94 | 22 | 23.40 | 91.67 |
| PI | HTTP | 22 | 1 | 1 | 100 | 4.55 |
| | SSDP | 20 | 12 | 4 | 33.33 | 20.00 |

a new string $\omega'$ by merging their overlap; that is, $\omega'[1 : i] = \omega_2[1 : i]$, and $\omega'[i+1] = \omega_1[i]$. Lines (24)~(38) aim to find out the closed frequent strings by deleting any strings in $C_i$ if and only if they are the substrings of some elements in $C_{i+1}$.

*5.3. Underflow Problem.* The joint probabilities of observation sequence often decay exponentially as the sequence length increases, which leads to a severe underflow problem when the forward-backward algorithms are implemented in a real computer. To the best of our knowledge, there are three approaches to solve this problem.

Firstly, one can implement the forward-backward algorithm in the logarithmic domain to avoid the underflow problem [33].

Secondly, one can also refine the forward-backward algorithm based on the notion of posterior probabilities to make the HsMM robust against the underflow problem. The refined forward-backward algorithms replace the joint probabilities with conditional ones and automatically avoid the underflow problem without increasing the computational complexity. More information about the posterior probabilities and refined HsMM based on conditional joint probabilities can be found in the work by Yu [13].

Thirdly, the forward-backward probabilities are adjusted by multiplying a scaling factor whenever an underflow is likely to occur [27, 34, 35]. In this paper, we tackle the underflow problem of HsMM based on this scaling method. In each $t$, we first compute $\alpha_t(i)$ based on the procedure of (12) and then compute the scaling factor in time $t$, denoted as $c_t$, as follows:

$$c_t = \frac{1}{\sum_{i=1}^{M} \alpha_t(i)}, \tag{27}$$

where $M$ is the number of states in the HsMM.

For the $\beta_t(i)$ term in the backward algorithm, we use the same scaling factors for each time $t$ as we used for $\alpha$ in the forward algorithm; that is,

$$\overline{\beta}_t(i) = c_t \beta_t(i). \tag{28}$$

As stated by Rabiner [27], the scaling factors will not affect the transition variable $A$, initial state probability distribution $\pi$, and the observation matrix $B$. However, the procedure for computing $P(O \mid \widehat{\lambda})$ is changed as follows:

$$P(O \mid \widehat{\lambda}) = \frac{1}{\prod_{t=1}^{T} c_t}. \tag{29}$$

In order to avoid the underflow problem, we prefer to calculate the logarithmic form of $P(O \mid \widehat{\lambda})$:

$$\log P(O \mid \widehat{\lambda}) = -\sum_{t=1}^{T} \log c_t. \tag{30}$$

## 6. Evaluation

In this section, we evaluate the proposed approach on data sets captured from the Internet entrance of our department on 23 December 2013. The data set contains network trace generated by six protocols, including two text-based protocols (HTTP and SSDP) and four binary-based protocols (BitTorrent, QQ, DNS, and NetBIOS).

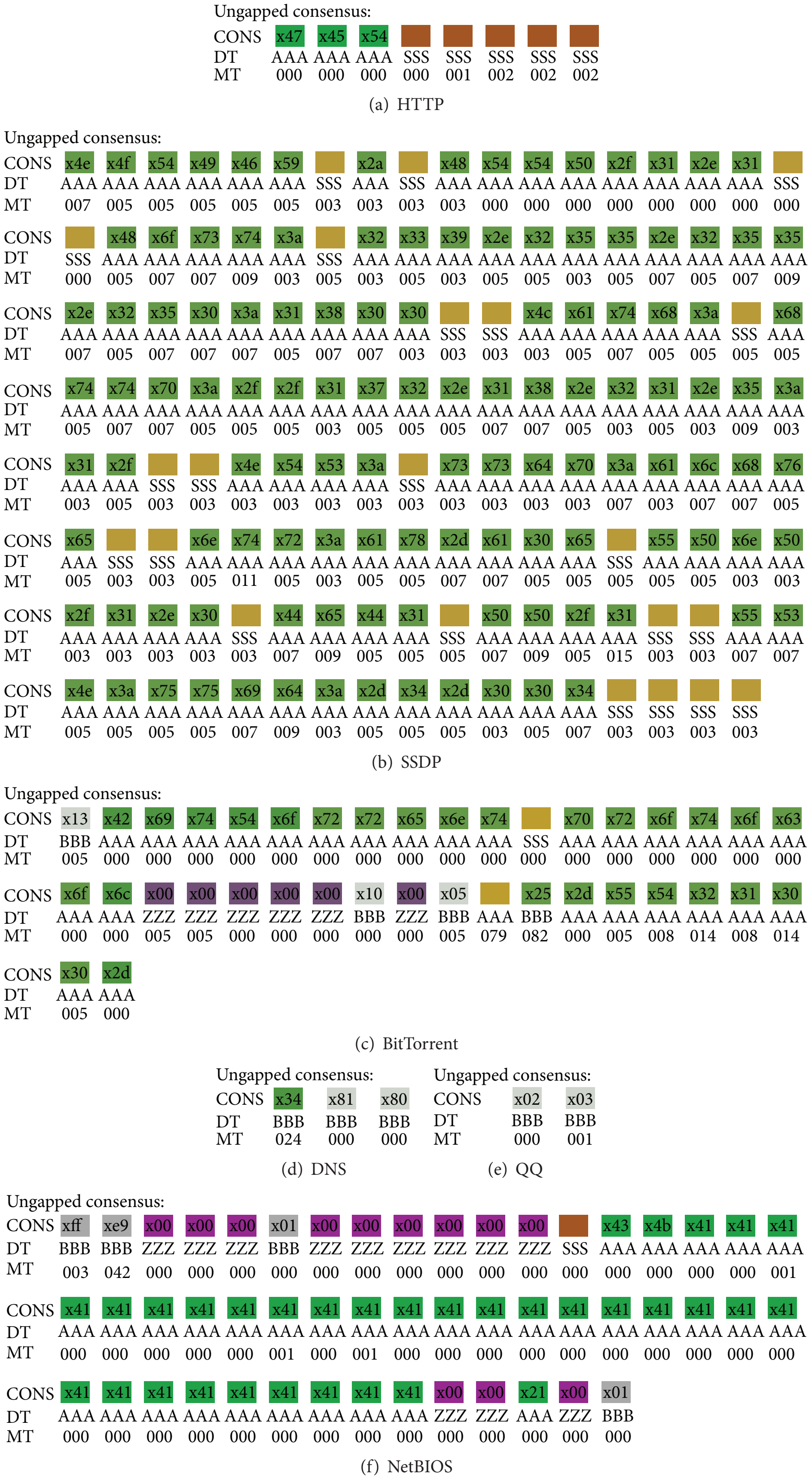
Existing algorithms such as PI (protocol informatics) and Discoverer are also applied to analyze the same data set. The PI project has released an open source Python code for researchers in the project home page [7], so we apply the code and perform it to analyze the data set. The Discoverer system is implemented according to the work presented by Cui et al. and the parameters are set as reported in their previous work [15].

*6.1. Protocol Keyword Extraction.* Since there is no information about protocol keywords of binary protocols in published protocol specifications, we only evaluate protocol keyword extraction for text-based protocols (i.e., HTTP and SSDP) in this section. We use the metrics of recall and precision to evaluate the quality of keyword extraction. The definition of these metrics is presented in the following:

(i) Recall: the recall rate is defined as the ratio from the number of true positives of inferred keywords to the total number of keywords in the data set.

(ii) Precision: the precision rate is defined as the ratio from the number of true positives of inferred keywords to the total number of inferred keywords.

We randomly select 100 connections of each protocol and only consider the first 1460 bytes (it is long enough to contain the headers of protocol messages) of each message. The results of protocol keyword extraction are shown in Table 1, where "Discv" represents Discoverer system and "PI" represents PI project. The column of "true keyword" records the true number of protocol keywords that occurred in the trace, while the column of "inferred keyword" records the number of inferred keywords. Compared with Discoverer



```
Ungapped consensus:
CONS x47 x45 x54 ■   ■   ■   ■   ■
DT   AAA AAA AAA SSS SSS SSS SSS SSS
MT   000 000 000 000 001 002 002 002
```

(a) HTTP

```
Ungapped consensus:
CONS x4e x4f x54 x49 x46 x59 ■   x2a ■   x48 x54 x54 x50 x2f x31 x2e x31 ■
DT   AAA AAA AAA AAA AAA AAA SSS AAA SSS AAA AAA AAA AAA AAA AAA AAA AAA SSS
MT   007 005 005 005 005 005 003 003 003 003 000 000 000 000 000 000 000 000

CONS ■   x48 x6f x73 x74 x3a ■   x32 x33 x39 x2e x32 x35 x35 x2e x32 x35 x35
DT   SSS AAA AAA AAA AAA AAA SSS AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA
MT   000 005 007 007 009 003 005 003 005 003 005 005 003 005 007 005 007 009

CONS x2e x32 x35 x30 x3a x31 x38 x30 x30 ■   ■   x4c x61 x74 x68 x3a ■   x68
DT   AAA AAA AAA AAA AAA AAA AAA AAA AAA SSS SSS AAA AAA AAA AAA AAA SSS AAA
MT   007 005 007 007 007 005 007 007 003 003 003 003 005 007 005 005 005 005

CONS x74 x74 x70 x3a x2f x2f x31 x37 x32 x2e x31 x38 x2e x32 x31 x2e x35 x3a
DT   AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA
MT   005 007 007 005 005 005 003 005 005 005 007 007 005 003 005 003 009 003

CONS x31 x2f ■   ■   x4e x54 x53 x3a ■   x73 x73 x64 x70 x3a x61 x6c x68 x76
DT   AAA AAA SSS SSS AAA AAA AAA AAA SSS AAA AAA AAA AAA AAA AAA AAA AAA AAA
MT   003 005 003 003 003 003 003 003 003 003 003 003 003 007 003 007 007 005

CONS x65 ■   ■   x6e x74 x72 x3a x61 x78 x2d x61 x30 x65 ■   x55 x50 x6e x50
DT   AAA SSS SSS AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA SSS AAA AAA AAA AAA
MT   005 003 003 005 011 005 003 005 005 007 007 005 005 005 005 005 003 003

CONS x2f x31 x2e x30 ■   x44 x65 x44 x31 ■   x50 x50 x2f x31 ■   ■   x55 x53
DT   AAA AAA AAA AAA SSS AAA AAA AAA AAA SSS AAA AAA AAA AAA SSS SSS AAA AAA
MT   003 003 003 003 003 007 009 005 005 005 007 009 005 015 003 003 007 007

CONS x4e x3a x75 x75 x69 x64 x3a x2d x34 x2d x30 x30 x34 ■   ■   ■   ■
DT   AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA SSS SSS SSS SSS
MT   005 005 005 005 007 009 003 005 005 005 003 005 007 003 003 003 003
```

(b) SSDP

```
Ungapped consensus:
CONS x13 x42 x69 x74 x54 x6f x72 x72 x65 x6e x74 ■   x70 x72 x6f x74 x6f x63
DT   BBB AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA SSS AAA AAA AAA AAA AAA AAA
MT   005 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

CONS x6f x6c x00 x00 x00 x00 x00 x10 x00 x05 ■   x25 x2d x55 x54 x32 x31 x30
DT   AAA AAA ZZZ ZZZ ZZZ ZZZ ZZZ BBB ZZZ BBB AAA BBB AAA AAA AAA AAA AAA AAA
MT   000 000 005 005 000 000 000 000 000 005 079 082 000 005 008 014 008 014

CONS x30 x2d
DT   AAA AAA
MT   005 000
```

(c) BitTorrent

```
Ungapped consensus:
CONS x34 x81 x80
DT   BBB BBB BBB
MT   024 000 000
```

(d) DNS

```
Ungapped consensus:
CONS x02 x03
DT   BBB BBB
MT   000 001
```

(e) QQ

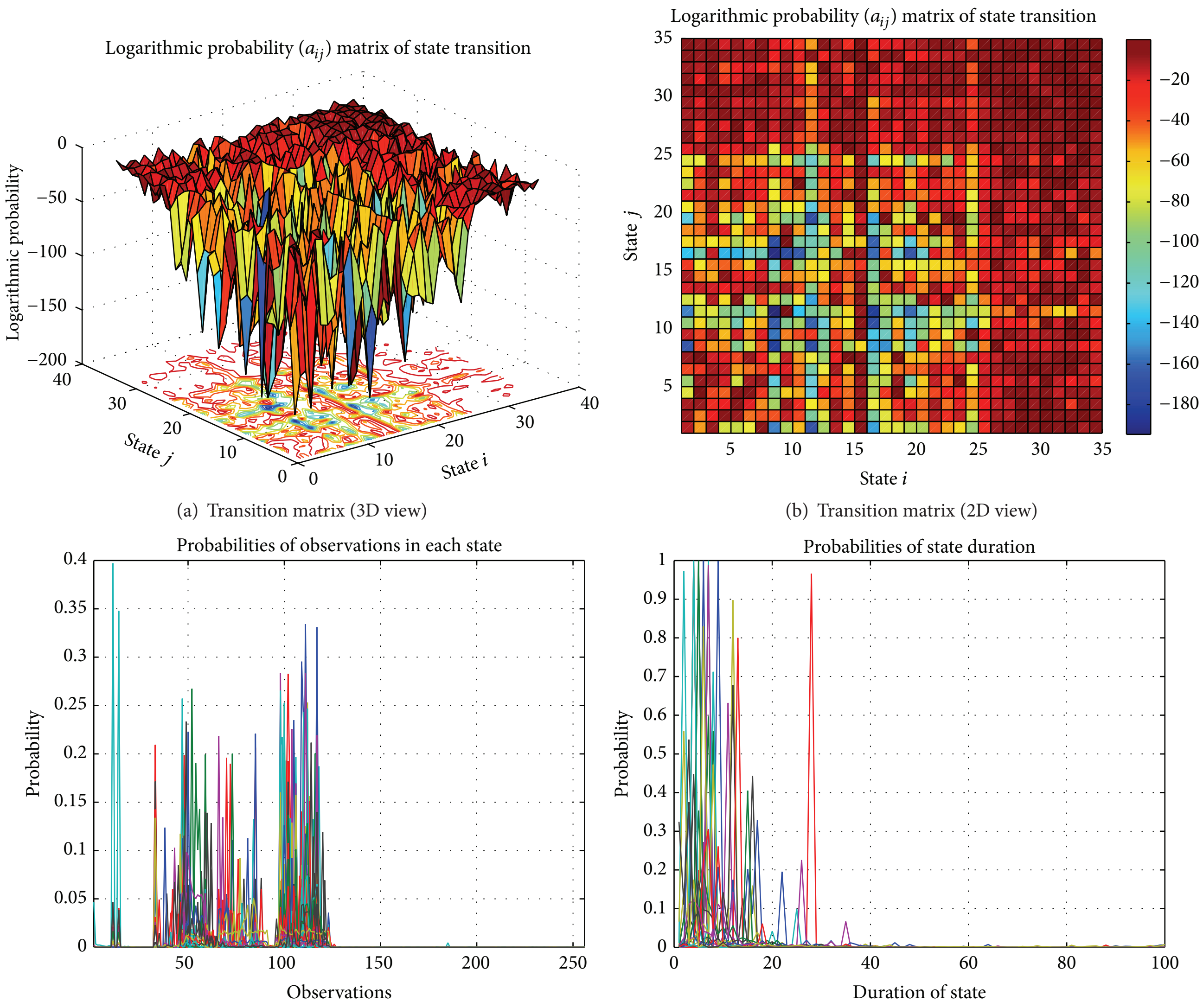
```
Ungapped consensus:
CONS xff xe9 x00 x00 x00 x01 x00 x00 x00 x00 x00 x00 ■   x43 x4b x41 x41 x41
DT   BBB BBB ZZZ ZZZ ZZZ BBB ZZZ ZZZ ZZZ ZZZ ZZZ ZZZ SSS AAA AAA AAA AAA AAA
MT   003 042 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 001

CONS x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41 x41
DT   AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA
MT   000 000 000 000 000 001 000 001 000 000 000 000 000 000 000 000 000 000

CONS x41 x41 x41 x41 x41 x41 x41 x41 x41 x00 x00 x21 x00 x01
DT   AAA AAA AAA AAA AAA AAA AAA AAA AAA ZZZ ZZZ AAA ZZZ BBB
MT   000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

(f) NetBIOS

Figure 5: The results output by PI.



(a) Transition matrix (3D view)

(b) Transition matrix (2D view)

(c) Observation probability distribution

(d) State duration distribution

FIGURE 6: The HsMM-based models for message format. (a) and (b) indicate the transition probability of states. (c) shows the probabilities of observations for each state; each line is corresponding to a state. (d) illustrates the state duration distribution; each line is corresponding to a state.

and PI project, HsMM-based method has a higher true positive, precision, and recall rate. We found that Discoverer infers too many keywords, while PI project infers too little.

Actually, there are far more protocol keywords inferred by our approach than the true keywords. Most of them are frequent and indispensable in the protocol messages, such as some parameters used frequently. So, all of these strings are also treated as protocol keywords and they play important role in inferring message formats and analyzing protocol state machine.

We also note that it has been found that the proposed HsMM-based approach can not only extract frequent keywords but also extract some keywords whose occurrence frequency is low.

*6.2. Protocol Message Format Inference.* We illustrate the results analyzed by PI in Figure 5. The message formats are inferred as the longest common substrings of protocol messages. As shown in Figure 5, only a few protocol keywords (such as "GET") and fields are inferred by PI, so PI does not seem to be expert in generating effective message formats.

As shown in Tables 2–4, we present the results of HTTP protocol for Discoverer, PI, and HsMM in a similar form to make it more clear for the readers. Discoverer infers message format based on token sequence and determines the attribute of token, such as constant token or variable token. Far more protocol keywords (such as "HTTP/1.1" and "Host:") are inferred by Discoverer than PI. However, some frequent strings (e.g., "ocspd" and "x86_64") which are not protocol keywords are also inferred as keywords.

Table 2: HTTP message format inferred by Discoverer. $c(t, \text{“XXX”})$ means a constant field, “$c$” means constant, “$t$” means text, and “XXX” is the value of the field; $v(t)$ means a variable field.

| Field ID | Token | Field ID | Token | Field ID | Token |
|---|---|---|---|---|---|
| 1 | $c(t, \text{“GET”})$ | 8 | $c(t, \text{“User-Agent:”})$ | 15 | $v(t)$ |
| 2 | $v(t)$ | 9 | $v(t)$ | 16 | $c(t, \text{“(x86\_64)”})$ |
| 3 | $c(t, \text{“HTTP/1.1”})$ | 10 | $c(t, \text{“ocspd”})$ | 17 | $v(t)$ |
| 4 | $c(t, \text{“Host:”})$ | 11 | $v(t)$ | 18 | $c(t, \text{“Connection:”})$ |
| 5 | $v(t)$ | 12 | $c(t, \text{“(unknown”})$ | 19 | $v(t)$ |
| 6 | $c(t, \text{“.com”})$ | 13 | $v(t)$ | 20 | $\cdots$ |
| 7 | $v(t)$ | 14 | $c(t, \text{“Darwin”})$ | | |

Table 3: HTTP message format inferred by PI.

| Field ID | Token | Field ID | Token |
|---|---|---|---|
| 1 | $c(t, \text{“GET”})$ | 2 | $v(t)$ |

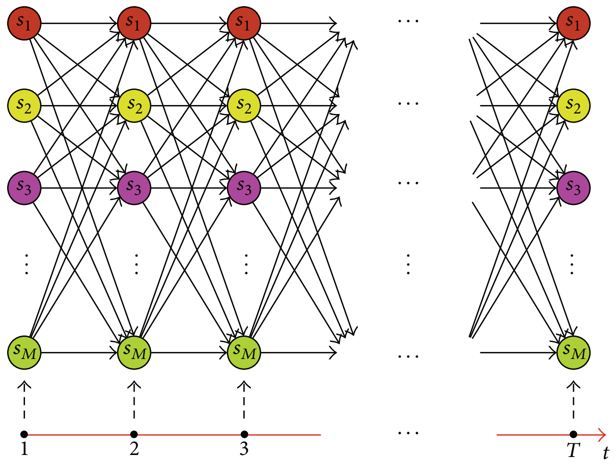


Figure 7: The lattice structure for computing $\alpha_t(i)$ based on forwarding algorithm. $s_1, s_2, \ldots, s_M \in \mathcal{S}$ represent the state of observation at time $t$.

In this paper, the proposed approach embeds the message formats into a HsMM-based protocol model. For each protocol, we train a HsMM by recursively reestimating the model parameters, including initial state probability, state transition probability matrix, and observation probability matrix. Using the HsMM, the optimal lengths of protocol keywords are determined and optimal segmentation of protocol message is inferred based on the maximal likelihood probability.

The parameter of a trained HsMM-based HTTP model is shown in Figure 6. In this model, the number of states is assigned to 35. As shown in Figure 6(c), the observations are mainly distributed in the area of $[10, 127]$ for each state, while the probabilities of observations located in $[128, 256]$ are much smaller. We can also find that the duration of each state is mainly distributed between 1 and 30, which means that the lengths of most fields in the message vary from 1 to 30.

When the HsMM is used to analyze an observation sequence (such as protocol message or network flow), a lattice, as shown in Figure 7, is constructed to compute the forward variable $\alpha_t(i)$ based on the model parameters. The state at each time $t$ may be in one of $M$ states in $\mathcal{S}$, while each state may emit multiple observations (characters varying from 0 to 255) with different probabilities, so HsMM could reveal the characteristics of both sequential fields and juxtapositional fields and such lattice implies the message formats. When forward variable computing is finished, the Viterbi algorithm can be applied to infer an optimal path which leads to a message field sequence with maximal likelihood probability. An example of inferring message fields based on our approach is illustrated in Figure 8. In this illustration, protocol keywords are labeled with states less than 26, while other fields are labeled with states between 26 and 35. Some *key*-*value* pairs are found in the message, such as the *key*-fields labeled with state 28 (in green) and *value*-fields labeled with state 33 (in light blue). IP address and some number sequence are labeled with state 33. We also found that the carriage return line feed (i.e., “0D0A” in hex) is labeled with the state of 32.

*6.3. Message Type Inference.* The results of message type inference are important for the future work of constructing protocol state machine. In our experiment, one type of messages will be clustered into serval types. However, we can treat them as serval different types since each inferred type represents a cluster of messages which share the same characteristic and have similar message format.

In order to compute the accuracy of message type inference, we label each cluster as the true type which dominates the cluster. The accuracy of message type inference is shown in Table 5.

*6.4. Traffic Identification.* The proposed technique can be used for network traffic identification in the application of network management or network monitoring. Suppose $\Lambda = \{\widehat{\lambda}_1, \widehat{\lambda}_2, \ldots, \widehat{\lambda}_n\}$ is the set of learned HsMM-based protocol models, where $\widehat{\lambda}_i$ $(i = 1, 2, \ldots, n)$ is the HsMM-based model of protocol $i$. For each session $\mathbf{o} = O_1^T$, the class of $\mathbf{o} = O_1^T$, denoted as $C(\mathbf{o})$, can be inferred as follows:

$$C(\mathbf{o}) = \arg\max_{i=1,2,\ldots,n} \log\left(P\left(\mathbf{o} \mid \widehat{\lambda}_i\right)\right), \tag{31}$$

where $P(\mathbf{o} \mid \widehat{\lambda}_i)$ is the likelihood of $\mathbf{o}$ given $\widehat{\lambda}_i$. $P(\mathbf{o} \mid \widehat{\lambda}_i)$ can be computed by (30).

This scenario is related to previous work by Ma et al. [36] in the fields of traffic identification based on application-layer

Table 4: HTTP message format inferred by HsMM.

| Field ID | Token | Field ID | Token | Field ID | Token |
|---|---|---|---|---|---|
| 1 | $c(t, \text{“GET”})$ | 6 | $c(t, \text{“User-Agent:”})$ | 11 | $v(t)$ |
| 2 | $v(t)$ | 7 | $v(t)$ | 12 | $c(t, \text{“Referer:”})$ |
| 3 | $c(t, \text{“HTTP/1.1”})$ | 8 | $c(t, \text{“Accept-Language:”})$ | 13 | $v(t)$ |
| 4 | $c(t, \text{“Host:”})$ | 9 | $v(t)$ | 14 | $c(t, \text{“Connection:”})$ |
| 5 | $v(t)$ | 10 | $c(t, \text{“Accept:”})$ | ⋯ | ⋯ |

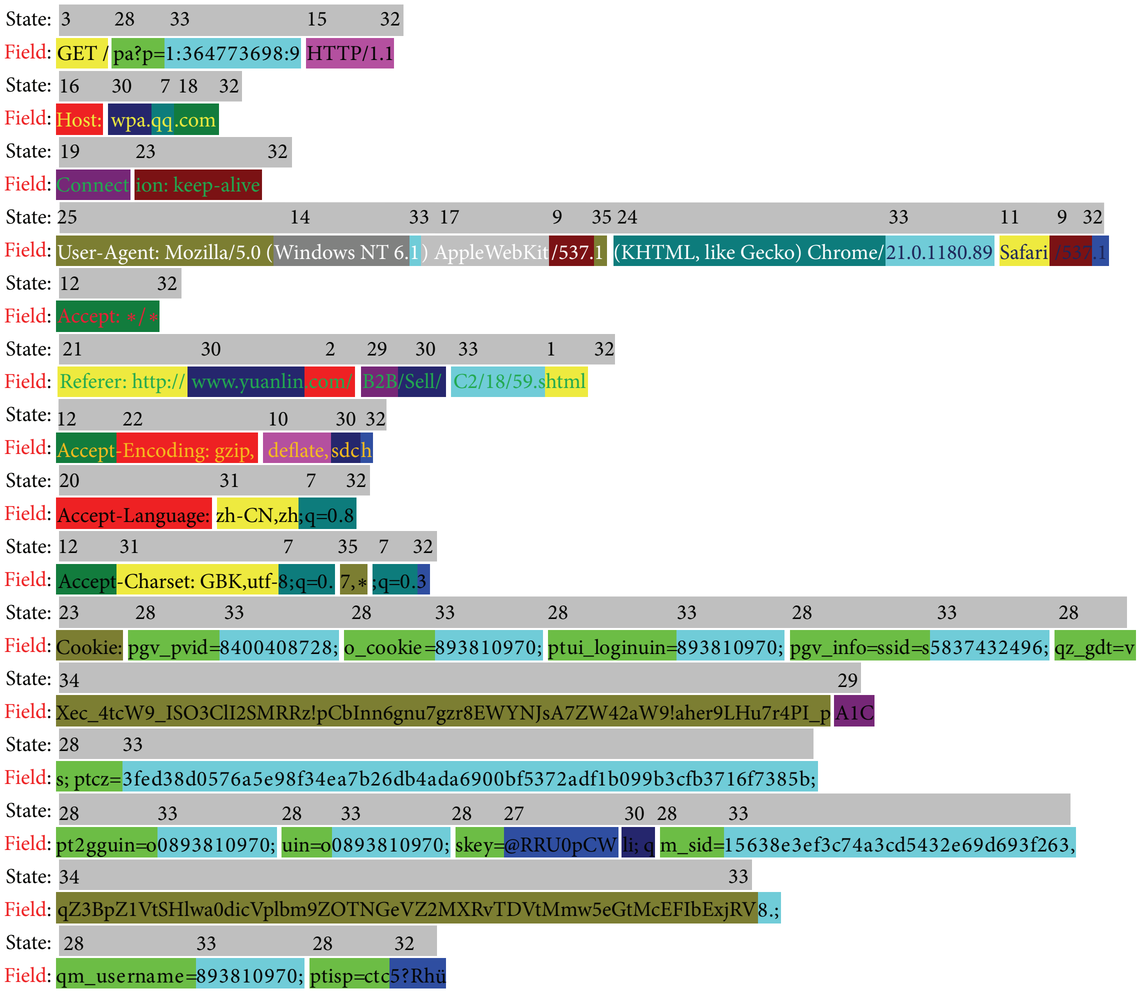

State: 3 28 33 15 32
Field: GET / pa?p=1:364773698:9 HTTP/1.1
State: 16 30 7 18 32
Field: Host: wpa.qq.com
State: 19 23 32
Field: Connect ion: keep-alive
State: 25 14 33 17 9 35 24 33 11 9 32
Field: User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.89 Safari/537.1
State: 12 32
Field: Accept: */*
State: 21 30 2 29 30 33 1 32
Field: Referer: http:// www.yuanlin.com/ B2B/Sell/ C2/18/59.shtml
State: 12 22 10 30 32
Field: Accept-Encoding: gzip, deflate, sdch
State: 20 31 7 32
Field: Accept-Language: zh-CN,zh;q=0.8
State: 12 31 7 35 7 32
Field: Accept-Charset: GBK,utf-8;q=0. 7,* ;q=0.3
State: 23 28 33 28 33 28 33 28 33 28
Field: Cookie: pgv_pvid=8400408728; o_cookie=893810970; ptui_loginuin=893810970; pgv_info=ssid=s5837432496; qz_gdt=v
State: 34 29
Field: Xec_4tcW9_ISO3ClI2SMRRz!pCbInn6gnu7gzr8EWYNJsA7ZW42aW9!aher9LHu7r4PI_p A1C
State: 28 33
Field: s; ptcz=3fed38d0576a5e98f34ea7b26db4ada6900bf5372adf1b099b3cfb3716f7385b;
State: 28 33 28 33 28 27 30 28 33
Field: pt2gguin=o0893810970; uin=o0893810970; skey=@RRU0pCW li; q m_sid=15638e3ef3c74a3cd5432e69d693f263,
State: 34 33
Field: qZ3BpZ1VtSHlwa0dicVplbm9ZOTNGeVZ2MXRvTDVtMmw5eGtMcEFIbExjRV8.;
State: 28 33 28 32
Field: qm_username=893810970; ptisp=ctc5?Rhü

Figure 8: Illustration of field segmentation for HTTP message.

payload. Ma et al. build Markov models from the application-layer payload and apply them to identifying network traffic. In this paper, we also implement the Markov model as stated in [36] and compare their results with ours, as shown in Figure 9. The results show that the proposed method outperforms the Markov based method in the field of traffic identification.

## 7. Conclusion

The protocol keywords and message fields are inferred based on hidden semi-Markov model by maximizing the likelihood probability of message segmentation. The segmentation of messages reveals some semantic information about the field, such as keyword, IP address, and *key-value* pair. The proposed technique is shown to be applied to the field of network traffic identification and outperforms existing algorithm.

The proposed HsMM-based protocol message format can be applied to field of intrusion detection or anomaly detection. One can use the HsMM-based message format of normal traffic to calculate the average likelihood probability of the new coming traffic and check whether the average likelihood probability is deviated from a normal level. Our

Table 5: The accuracy of message type inference.

| Protocol | HTTP | SSDP | BitTorrent | QQ | DNS | NetBIOS |
|---|---|---|---|---|---|---|
| Accuracy (%) | 96.63 | 97.46 | 100 | 95.16 | 96.18 | 100 |

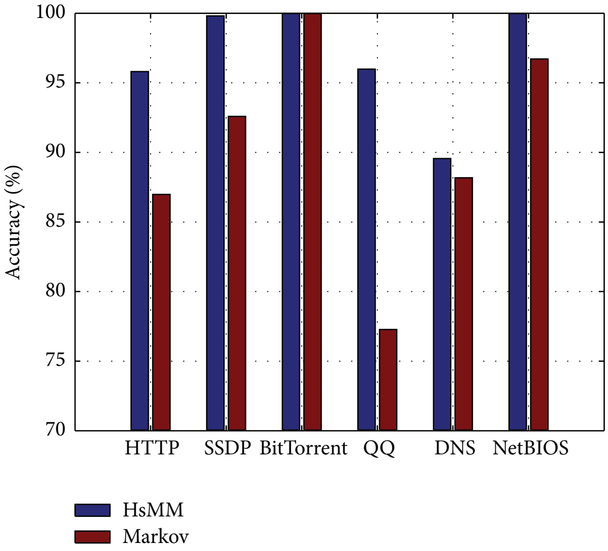


Figure 9: The accuracy of traffic identification. "HsMM" represents the HsMM-based method proposed in this paper, while the "Markov" represents the Markov model based method presented by previous work.

method can also be applicable for traffic identification, fuzz test, vulnerability discovery, and so on.

## Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

## Acknowledgments

This work was supported by The National Natural Science Foundation of China (61571141); Guangdong Natural Science Foundation (2014A030313637); Guangdong Provincial Department of Education Innovation Project (2014KTSCX149); The Excellent Young Teachers in Universities in Guangdong (YQ2015105); Guangdong Provincial Application-Oriented Technical Research and Development Special Fund Project (2015B010131017).

## References

[1] C. Y. Cho, D. Babić, C. E. 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 Communications Security (CCS '10)*, pp. 426–439, ACM, Chicago, Ill, USA, October 2010.

[2] H. C. Kim, Y. H. Choi, and D. H. Lee, "Efficient file fuzz testing using automated analysis of binary file format," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 259–268, 2011.

[3] A. Tridgell, *How Samba Was Written*, 2003, http://www.samba.org/ftp/tridge/misc/french_cafe.txt.

[4] Pidgin, Pidgin, 2014, http://www.pidgin.im/.

[5] rdesktop, *rdesktop: A Remote Desktop Protocol Client*, 2014, http://www.rdesktop.org/.

[6] G. Maier, F. Schneider, and A. Feldmann, "A first look at mobile hand-held device traffic," in *Passive and Active Measurement: 11th International Conference, PAM 2010, Zurich, Switzerland, April 7–9, 2010. Proceedings*, A. Krishnamurthy and B. Plattner, Eds., vol. 6032 of *Lecture Notes in Computer Science*, pp. 161–170, Springer, Berlin, Germany, 2010.

[7] M. A. Beddoe, "Network protocol analysis using bioinformatics algorithms," 2004, http://www.4tphi.net/~awalters/PI/PI.html.

[8] C. Kreibich and J. Crowcroft, "Efficient sequence alignment of network traffic," in *Proceedings of the 6th ACM SIGCOMM on Internet Measurement Conference (IMC '06)*, pp. 307–312, ACM, October 2006.

[9] B.-C. Park, Y. Won, M.-S. Kim, and J. Hong, "Towards automated application signature generation for traffic identification," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '08)*, pp. 160–167, Salvador, Brazil, April 2008.

[10] M. Ye, K. Xu, J. Wu, and H. Po, "Autosig-automatically generating signatures for applications," in *Proceedings of the 9th IEEE International Conference on Computer and Information Technology (CIT '09)*, vol. 2, pp. 104–109, Xiamen, China, October 2009.

[11] J.-Z. Luo and S.-Z. Yu, "Position-based automatic reverse engineering of network protocols," *Journal of Network and Computer Applications*, vol. 36, no. 3, pp. 1070–1077, 2013.

[12] J.-Z. Luo, S.-Z. Yu, and J. Cai, "Capturing uncertainty information and categorical characteristics for network payload grouping in protocol reverse engineering," *Mathematical Problems in Engineering*, vol. 2015, Article ID 962974, 9 pages, 2015.

[13] S.-Z. Yu, "Hidden semi-Markov models," *Artificial Intelligence*, vol. 174, no. 2, pp. 215–243, 2010.

[14] G. Jacobson and K.-P. Vo, "Heaviest increasing/common subsequence problems," in *Combinatorial Pattern Matching*, A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, Eds., vol. 644 of *Lecture Notes in Computer Science*, pp. 52–66, Springer, Berlin, Germany, 1992.

[15] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: automatic protocol reverse engineering from network traces," in *Proceedings of the 16th USENIX Security Symposium on USENIX Security Symposium*, pp. 1–14, USENIX Association, Boston, Mass, USA, August 2007.

[16] Y. Wang, N. Zhang, Y.-M. Wu, B.-B. Su, and Y.-J. Liao, "Protocol formats reverse engineering based on association rules in wireless environment," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '13)*, pp. 134–141, Melbourne, Australia, July 2013.

[17] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the Association for Computing Machinery*, vol. 18, no. 6, pp. 333–340, 1975.

[18] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, and L. Guo, "Biprominer: automatic mining of binary protocol features," in *Proceedings of the 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '11)*, pp. 179–184, IEEE, Gwangju, The Republic of Korea, October 2011.

[19] J. Antunes and N. Neves, *Building an Automaton Towards Reverse Protocol Engineering*, 2009, http://www.di.fc.ul.pt/~nuno/PAPERS/INFORUM09.pdf.

[20] T. Krueger, N. Krmer, and K. Rieck, "Asap: automatic semantics-aware analysis of network payloads," in *Privacy and Security Issues in Data Mining and Machine Learning: International ECML/PKDD Workshop, PSDML 2010, Barcelona, Spain, September 24, 2010. Revised Selected Papers*, vol. 6549 of *Lecture Notes in Computer Science*, pp. 50–63, Springer, Berlin, Germany, 2011.

[21] Y. Wang, X. Yun, M. Z. Shafiq et al., "A semantics aware approach to automated reverse engineering unknown protocols," in *Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP '12)*, pp. 1–10, IEEE, Austin, Tex, USA, November 2012.

[22] 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 '07)*, pp. 317–329, ACM, November 2007.

[23] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: automatic reverse engineering of input formats," in *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS '08)*, pp. 391–402, ACM, Alexandria, Va, USA, October 2008.

[24] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution," in *Proceedings of the 15th Symposium on Network and Distributed System Security (NDSS '08)*, The Internet Society, San Diego, Calif, USA, February 2008.

[25] 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*, pp. 110–125, Berkeley, Calif, USA, May 2009.

[26] J. Caballero, P. Poosankam, C. Kreibich, and D. 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)*, pp. 621–634, ACM, Chicago, Ill, USA, November 2009.

[27] L. R. Rabiner, "Tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[28] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[29] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[30] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[31] Wireshark, Network protocol analyzer, 2012, http://www.wireshark.org.

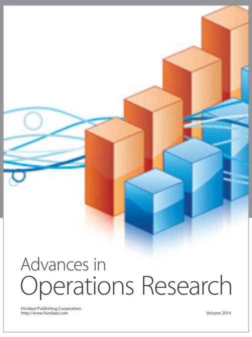
[32] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, pp. 487–499, Santiago de Chile, Chile, September 1994.

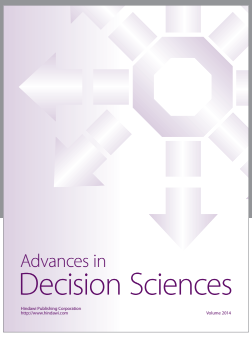
[33] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51, no. 2, pp. 175–185, 2003.

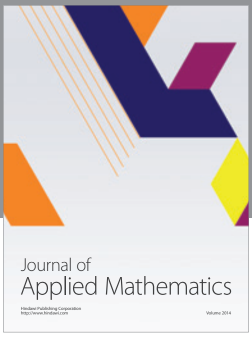
[34] Y. Cohen, A. Erell, and Y. Bistritz, "Enhancement of connected words in an extremely noisy environment," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 2, pp. 141–148, 1997.

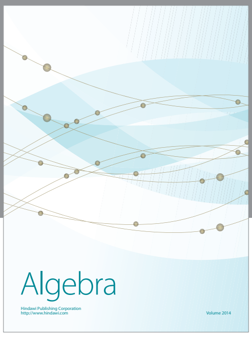
[35] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *The Bell System Technical Journal*, vol. 62, no. 4, pp. 1035–1074, 1983.

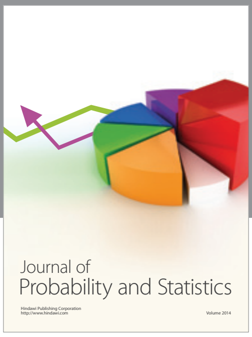
[36] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference," in *Proceedings of the 6th ACM SIGCOMM on Internet Measurement Conference (IMC '06)*, pp. 313–326, ACM, October 2006.


Advances in
Operations Research

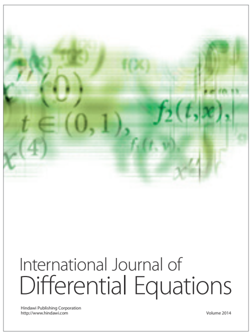

Advances in
Decision Sciences


Journal of
Applied Mathematics

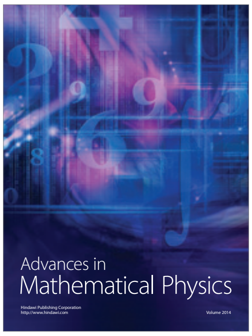

Algebra

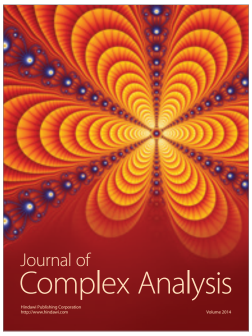

Journal of
Probability and Statistics

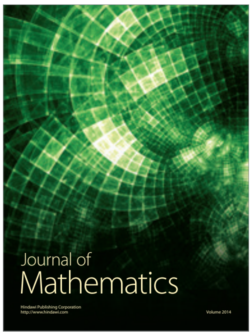
The Scientific
World Journal

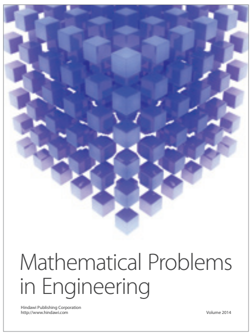

International Journal of
Differential Equations

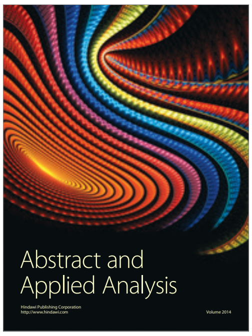
International Journal of
Combinatorics

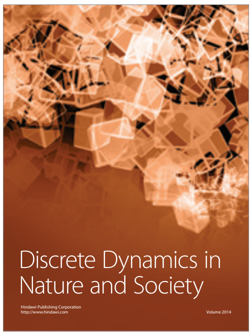

Advances in
Mathematical Physics

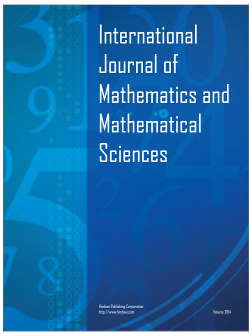

Journal of
Complex Analysis


Journal of
Mathematics


Mathematical Problems
in Engineering

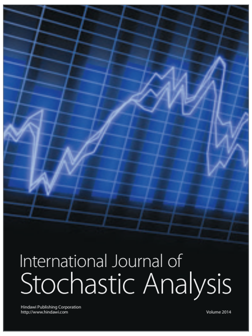

Abstract and
Applied Analysis


Discrete Dynamics in
Nature and Society


International
Journal of
Mathematics and
Mathematical
Sciences

Journal of
Discrete Mathematics

Journal of
Function Spaces


International Journal of
Stochastic Analysis

Journal of
Optimization