A Large-Scale Hidden Semi-Markov Model for Anomaly Detection on User Browsing Behaviors

Yi Xie and Shun-Zheng Yu, Member, IEEE

Abstract—Many methods designed to create defenses against distributed denial of service (DDoS) attacks are focused on the IP and TCP layers instead of the high layer. They are not suitable for handling the new type of attack which is based on the application layer. In this paper, we introduce a new scheme to achieve early attack detection and filtering for the application-layer-based DDoS attack. An extended hidden semi-Markov model is proposed to describe the browsing behaviors of web surfers. In order to reduce the computational amount introduced by the model’s large state space, a novel forward algorithm is derived for the online implementation of the model based on the M-algorithm. Entropy of the user’s HTTP request sequence fitting to the model is used as a criterion to measure the user’s normality. Finally, experiments are conducted to validate our model and algorithm.

Index Terms—Anomaly detection, browsing behaviors, DDoS, hidden semi-Markov Model, M-algorithm.

I. INTRODUCTION

DISTRIBUTED denial of service (DDoS) attacks constitute one of the major threats and are among the hardest security problem facing today’s Internet [1]. Because of the seriousness of this problem, many defense mechanisms, based on statistical approaches [2]–[11], have been proposed to combat these attacks.

In statistical approaches to defend against DDoS attacks, the statistics of packet attributes in the headers of IP packets, such as IP address, time-to-live (TTL), protocol type, etc., are measured and the packets deemed most likely to be attack, s, are dropped based on these measurements. Such approaches often assume that there are some traffic characteristics that inherently distinguish the normal packets from the attack ones. Therefore, “abnormal” traffic can be detected based on those traffic characteristics during a DDoS attack.

Although the approaches based on statistics attributes of TCP or IP layers are valuable for certain DDoS attacks (e.g., SYN/ACK flooding), they are not always workable for some special DDoS attacks which work on the application layer. This has been witnessed on the Internet in 2004, when a worm virus named “Mydoom” [12] used pseudo HTTP requests to attack victim servers by simulating the behavior of browsers.

Because the DDoS defense mechanisms that are based on the statistical attributes of lower layer information could not distinguish the abnormal HTTP requests from the normal ones, the victim’s server soon collapsed. In this paper, we call such application-layer DDoS attacks “App-DDoS” attacks.

The challenge of detecting App-DDoS attacks is due to the following aspects. (i) The App-DDoS attacks utilize high layer protocols to pass through most of the current anomaly detection systems designed for low layers and arrive at the victim webserver. (ii) App-DDoS attacks usually depend on successful TCP connections, which makes the general defense schemes based on detection of spoofed IP address useless. (iii) Flooding is not the unique way for the App-DDoS attack. There are many other forms, such as consuming the resources of the server (CPU, memory, hard disk, or database), arranging the malicious traffic to mimic the average request rate of the normal users or utilizing the large-scale botnet to produce the low rate attack flows. This makes detection more difficult.

From the literature, few studies can be found that focus on the detection of App-DDoS attacks. This paper develops a novel model to capture the browsing patterns of web users and to detect the App-DDoS attacks. Our contributions are in three aspects: (i) based on the hidden semi-Markov model (HsMM) [13], [14], a new model is introduced for describing the browsing behavior of web users and detection of the App-DDoS attacks; (ii) a new effective algorithm is proposed for implementation of the forward process of HsMM and on-line detection of the App-DDoS attacks; and (iii) experiments based on real traffic are conducted to validate our detection method.

This paper is organized as follows. In Section II, we describe the related works of our research. In Section III, we present the assumptions made in our scheme and establish a new model for our scheme in Section IV. Then, in Section V, we propose a novel detection algorithm based on HsMM. The whole scheme is validated in Section VI by conducting an experiment using real data of traffic. Finally, in Section VII, we discuss some important issues on the scheme and conclude our work in Section VIII.

II. RELATED WORK

Most current DDoS-related studies focus on the IP layer or TCP layer instead of the application layer. Mirkovic et al. [2] devised a defense system called D-WARD installed in edge routers to monitor the asymmetry of two-way packet rates and to identify attacks. IP addresses [3] (which assume that attack traffic uses randomly spoofed addresses) and TTL values [4] were also used to detect the DDoS attacks. Statistical abnormalities of ICMP, UDP, and TCP packets were mapped...
to specific DDoS attacks based on the Management Information Base (MIB) [5]. Limwiwatkul [6] discovered the DDoS attacking signature by analyzing the TCP packet header against the well-defined rules and conditions. Noh [7] attempted to detect attacks by computing the ratio of TCP flags to TCP packets received at a webserver and considering the relative proportion of arriving packets devoted to TCP, UDP, and ICMP traffic. Basu [8] proposed techniques to extract features from connection attempts and classify attempts as suspicious or not. The rate of suspicious attempts over a day helped to expose stealthy DoS attacks, which attempt to maintain effectiveness while avoiding detection.

Little existing work has been done on the detection of App-DDoS attacks. Sanjan et al. [9] used statistical methods to detect characteristics of HTTP sessions and employed rate-limiting as the primary defense mechanism. Besides the statistical methods, some researchers combatted the App-DDoS attacks by “puzzle”. Kandula [10] designed a system to protect a web cluster from DDoS attacks by (i) designing a probabilistic authentication mechanism using CAPTCHAs (acronym for “ Completely Automated Public Turing test to tell Computers and Humans Apart”) and (ii) designing a framework that optimally divides the time spent in authenticating new clients and serving authenticated clients.

However, statistical methods can hardly distinguish the vicious HTTP requests from the normal ones. Hence, such methods can merely be used for anomaly monitoring but not for filtering. Defenses based on puzzle, requiring each client to return the solution before accessing the site, are not effective because (i) CAPTCHAs might annoy users and introduce additional service delays for legitimate clients; (ii) puzzles also have the effect of denying web crawlers access to the site and disabling search engines which index the content. Furthermore, new techniques may render the puzzles solvable using automated methods [16]. In [17], Jung et al. proposed methods to filter the HTTP-level offending DoS traffic. They provided topological clustering heuristics for a web server to distinguish denial of service attacks from flash crowd behavior according to the source IP address. However, IP addresses are subject to spoofing and using a white-list of source addresses of legitimate clients/peers is difficult, since many hosts may have dynamic IP addresses due to the use of NAT, DHCP, and mobile-IP.

Web user behavior mining has been well studied. The existing works can be summarized as the following four types. The first is based on probabilistic model, e.g., [18], which use a Zipf-like distribution to model the page jump-probability, a double Pareto distribution for the link-choice, and a log-normal distribution for the revisiting, etc. The second is based on click-streams and web content, e.g., [19], which uses data mining to capture a web user’s usage patterns from the click-streams dataset and page content. The third is based on the Markov model, e.g., [20], which uses Markov chains to model the URL access patterns that are observed in navigation logs based on the previous state. The fourth applies the user behavior to implement anomaly detection, e.g., [21], which uses system-call data sets generated by programs to detect the anomaly access of UNIX system based on data mining.

The main disadvantage of the first type of methods is that they do not take into account the user’s series of operations information (e.g., which page will be requested in the next step). Therefore, they cannot describe the browsing behavior of a user because the next page the user will browse is primarily determined by the current page he/she is browsing. The second type of methods need intensive computation for page content processing and data mining, and hence they are not very suitable for on-line detection. On the other hand, the click-streams data sets required for the algorithms are collected by additional means (e.g., cookies or JAVA applets), which require the support of the clients. The third type of methods omit the dwell time that the user stays on a page while reading and they do not consider the cases that a user may not follow the hyperlinks provided by the current page and some user’s HTTP requests responded by proxies cannot arrive at the server. Besides, when the total number of pages (i.e., the state space of Markovian) in the website is very large, the algorithm would be too complex to be implemented in online detection. The last type of methods merely focus on system calls of a UNIX program, instead of web user browsing behavior. Obviously, system calls can be recorded completely by the log, but the observations of web users’ requests are incomplete because many HTTP requests are responded by proxies or caches, which cannot arrive at the server and be recorded in the log. Thus, a new method designed for detection of App-DDoS attacks has to consider these issues.

III. MODEL PRELIMINARIES AND ASSUMPTIONS

The proposed scheme for App-DDoS attack detection and filtering is based on two results obtained by web mining [22]: (i) about 10% of the webpages of a website may draw 90% of the access; (ii) web user browsing behavior can be abstracted and profiled by users’ request sequences. Thus, we can use a universal model to profile the short-term web browsing behavior, and we only need the logs of webserver to build the model without any additional support from outside of the webserver.

Another assumption of the proposed scheme is that it is difficult for an attacker or virus routine to launch an App-DDoS attack by completely mimicking the normal web user access behavior. This assumption is based on the following considerations. Usually, browsing behavior can be described by three elements: HTTP request rate, page viewing time, and requested sequence (i.e., the requested objects and their order). In order to launch stealthy attacks, attackers can mimic the normal access behavior by simulating the HTTP request rate, the viewing time of webpage, or the HTTP request sequence. The first two simulations can be carried out by HTTP simulation tools (e.g., [23]). The HTTP attack traffic generated this way has similar statistical characteristics as the normal ones, but it cannot simulate the process of the user browsing behavior because it is a pre-designed routine and cannot capture the dynamics of the users and the networks. Besides, the low rate, stealthy, attack requires more infected computers, which needs the help of a large-scale botnet. Another option for attack is simulating the HTTP request sequence. Four options can be used to carry out this intention, including stochastic generation, presetting, interception with replaying, and remote control. Using the first method, attackers (routines) generate the HTTP request objects randomly,
which makes the attackers’ requested contents and orders different from the normal users’. By the second method, an attack sponsor has to embed a HTTP request list into the routine before the worm virus is propagated. Thus, when the infected computers launch attacks through the presetting HTTP request list, one visible characteristic is that many users browse/request the same web objects periodically and repeatedly. Using the third method, the worm virus launches attack s by intercepting the request sequences sent by the browser of an infected computer and replaying them to the victim webserver. However, because the users of infected computers may not access the victim website, it is not easy for this method to be successful. The last way is that the sponsor of attacks establishes a center control node and communicates with the attack routines residing in the infected computers periodically. By this means, the attacker can control the attack behavior of each zombie completely, e.g., distribute the new HTTP attack sequences or vary the attack mode to conceal its baleful behaviors. However, though the attacker may be able to control its botnet without being detected by anti-virus software on the client-side, we can suppose that the attack controller is unable to obtain historical access records (i.e., logs) from the victim webserver or stay in front of the victim to intercept all of the incoming HTTP requests. This makes it difficult to launch an attack that simulates the dynamics of normal user behaviors, even though it is able to capture a few of the users’ request sequences and replay them repeatedly. Therefore, the attacker can be detected by server-side intrusion detection systems using the user’s behavior profiles.

Among the existing attack modes, the most simple and effective way of App-DDoS attacks is utilizing the HTTP “GET /” to launch attacks by requesting the homepage of the victim website repeatedly. Because such attack merely needs to know the domain name of the victim website without specifying the URL of each object, attacker s can launch the offensive easily. Furthermore, the homepage usually is the most frequently accessed webpage and has a lot of embedded objects, thus, attacking through homepage is effective and not easily to be detected.

IV. MODEL AND ALGORITHM

A. Basic Ideas for Modeling Web Browsing Behaviors

The browsing behavior of a web user is related to two factors: the structure of a website, which comprises of a huge number of web documents, hyperlinks, and the way the user accesses the webpages. Usually, a typical webpage contains a number of links to other embedded objects, which are referred to as in-line objects. A website can be characterized by the hyperlinks among the webpages and the number of in-line objects in each page. When users click a hyperlink pointing to a page, the browser will send out a number of requests for the page and its in-line objects. These requests may arrive at the webserver in a short time interval, except those responded by proxies or caches. Which and how many requests of the clicked page can arrive at the webserver are not determinist. Because the inline objects are overlapped among pages, the real clicked pages of the user cannot be observed directly affecting the server-side’s observations on the user behavior. These proxies and caches may respond to the HTTP requests emitted by the browser, and thus some of the user’s requests may not arrive at the webserver. This leads to different logs of requests for the same page being accessed by different users at different times or through different paths.

In considering the factors described above, we introduce our web browsing model, as shown in Fig. 2. When a user clicks a hyperlink pointing to a page, the browser will send out a number of requests for the page and its in-line objects. These requests may arrive at the webserver in a short time interval, except those responded by proxies or caches. Which and how many requests of the clicked page can arrive at the webserver are not determinist. Because the inline objects are overlapped among pages, the real clicked pages of the user cannot be observed directly on the server-side, which can only be estimated from the received request sequence. In this paper, the estimation is based on data clustering. Because the HTTP OFF period is usually much longer than the HTTP ON, the requested objects in the observation sequence can be grouped into different clusters according to this characteristic. Each group presents a webpage. Then, the user’s request sequence (e.g., $r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}$ in Fig. 2) is transformed into the corresponding group sequence (e.g., $1\ 2\ 3$ in Fig. 2). The order or transition of consecutive groups implies the user’s click behavior when browsing from one page to another.

In Sections IV-B and IV-C, we will introduce a new HsMM to describe the dynamic web access behavior and implement the anomaly detection and filtering for the App-DDoS attack.

Fig. 1. Browsing behavior.

Fig. 2. Web browsing model.
B. Hidden Semi-Markov Model

HsMM is an extension of the hidden Markov model (HMM) with explicit state duration. It is a stochastic finite state machine, specified by \((S, \pi, A, P)\) where:
- \(S\) is a discrete set of hidden states with cardinality \(N\), i.e., \(S = \{1, \ldots, N\}\);
- \(\pi\) is the probability distribution for the initial state \(\pi_m \equiv \Pr(s_1 = m)\). \(s_t\) denotes the state that the system takes at time \(t\) and \(m \in S\). The initial state probability distribution satisfies \(\sum_m \pi_m = 1\);
- \(A\) is the state transition matrix with probabilities: \(a_{mn} \equiv \Pr[s_t = n | s_{t-1} = m]\), \(m, n \in S\), and the state transition coefficients satisfy \(\sum_n a_{mn} = 1\);
- \(P\) is the state duration matrix with probabilities: \(p_{dn}(d) \equiv \Pr[\tau_t = d | s_t = m]\). \(\tau_t\) denotes the remaining (or residual) time of the current state \(s_t\), \(m \in S, d \in \{1, \ldots, D\}\), \(D\) is the maximum interval between any two consecutive state transitions, and the state duration coefficients satisfy \(\sum_d p_{dn}(d) = 1\).

Then, if the pair process \((s_t, \tau_t)\) takes on value \((m_0, d_0)\), the semi-Markov chain will remain in the current state \(m_0\) until time \(t + d_0 - 1\) and transits to another state at time \(t + d_0\), where \(d_0 \geq 1\). The states themselves are not observable. The accessible information consists of symbols from the alphabet of observations \(O = \{o_1, \ldots, o_T\}\) where \(o_t\) denotes the observable output at time \(t\) and \(T\) is the number of samples in the observed sequences. For every state an output distribution is given as \(b_m(k) \equiv \Pr[o_t = k | s_t = m]\) where \(m \in S\) and \(k \in V = \{1, \ldots, K\}\). \(K\) is the size of the observable output set. The output probabilities satisfy \(\sum_k b_m(k) = 1\). \(b_m(o_k) = \prod_{k=1}^{T} b_m(o_k)\) when the “conditional independence” of outputs is assumed, where \(o_k \equiv \{o_t : a \leq t \leq b\}\) represents the observation sequence from time \(a\) to time \(b\). Thus, the set of HsMM parameters \(\lambda\) consists of initial state distributions, the state transition probabilities, the output probabilities and state duration probabilities. For brevity of notation, they can be denoted as \(\Lambda = \{(\pi_m), (a_{mn}), (b_m(k)), (p_{dn}(d))\}\). The algorithms of HsMM can be found in [13] and [14].

C. Problem Formulation

We use the Markov state space to describe the webpage set of the victim website. Each underlying semi-Markov state is used to present a unique webpage clicked by a web user. Thus, the state transition probability matrix \(A\) presents the hyperlink relation between different webpages. The duration of a state presents the number of HTTP requests received by the webserver when a user clicks the corresponding page. The output symbol sequence of each state throughout its duration presents those requests of the clicked page which pass through all proxies/caches and finally arrive at the webserver. We use a simple example to explain these relations by Fig. 2. The unseen clicked page sequence is \((\text{page 1, page 2, page 3})\). Except those responded by the proxies or caches, the corresponding HTTP request sequence received by the webserver is \((r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11})\). When the observed request sequence is inputted to the HsMM, the algorithm may group them into three clusters \((r_1, r_2, r_3, r_4), (r_5, r_6, r_7), (r_8, r_9, r_{10}, r_{11})\) and denote them as a state sequence \(\{\{1, 2, 3\}\}\). The state transition probability \(a_{12}\) represents the probability that page 2 may be accessed by the user after accessing the current page 1. The duration of the first state \(1\) is \(d = 4\), which means 4 HTTP requests of page 1 arrived at the webserver, i.e., \((r_1, r_2, r_3, r_4)\).

The web browsing behavior can be formulated in terms of the HsMM as follows. We use a random vector \(o_t = (q_t, r_t)\) to denote the observation at the \(t\)th request in the request sequence, where \(r_t \in V = \{1, \ldots, K\}\) is the index of the object that the \(t\)th request intends to get from the original web server, such as an HTML page or an embedded image, and \(q_t \in C = \{1, 2, \ldots\}\) is the interval between \(r_{t-1}\) and \(r_t\). \(K\) is the size of all accessible object sets. \(o_t\) forms the observation vector sequence \(\{o_t : t = 1, \ldots, T\}\); \(T\) is the total length of the observation sequence. We use a underlying semi-Markov process \(s_t : t = 1, \ldots, T\) to describe the user’s click behavior, where \(s_t \in S\) is the index of the webpage browsed by the user at the \(t\)th request. \(S\) is the Markov state space which presents webpage set of the website. We use \(N\) to denote the size of \(S\). Transition from \(s_{t-1}\) to \(s_t\) is considered as the user’s browsing behavior from one webpage to another, following a hyperlink between the two pages. The state transition probability \(\pi_m = 1\) means the probability of the user browsing from page \(s_{t-1}\) to page \(s_t\). If \(s_{t-1} = s_t\), then we assume the \((t-1)\)th request \(r_{t-1}\) and the \(t\)th request \(r_t\) are made for the same page.

In considering that the user may jump from one page to another page by typing URLs or opening multiple windows without following the hyperlinks on the webpages in browsing the website, we introduce a Null page which is denoted as \(\Lambda\) to describe these crawling behaviors. That is, if page \(n\) has no direct link to page \(m\) (i.e., the transition probability \(a_{nm} = 0\)), and the user jumps from \(n\) to \(m\) directly in browsing the website, we then assume that the user transits from page \(n\) to the Null page \(\Lambda\) at first, and then transits from the Null page \(\Lambda\) to page \(m\) with the transition probabilities \(a_{\Lambda m}\) and \(a_{\Lambda m}\), respectively, as shown in Fig. 3. Correspondingly, the state space \(S\) is extended as \(S = \{1, \ldots, N, \Lambda\}\). In addition, we use another random variable \(\tau_t\) to denote the remaining (or residual) number of requests of the current webpage \(s_t\).

Then the browsing behaviors of web users can be described by HsMM, and the state sequence \(\{s_t\}\) and the model parameters can be estimated from the observed sequence \(\{o_t\}\) of the web users’ requests.
In order to handle the newly introduced Null state and reduce the computations, we extend the re-estimation algorithm proposed in [13] to the new one for web browsing behavior. The improved algorithm is presented as follows.

We first define the forward and backward variables and then, according to the state transition diagram shown in Fig. 3, derive the forward and backward formulas as follows.

The forward variables and the forward formulas:

\[ \alpha_t(m, d) \triangleq \Pr[\sigma_{t|T}, (s_t, \tau_t) = (m, d)] = \alpha_{t-1}(m, d + 1) b_m(\alpha_t) \]
\[ + \left( \sum_{n \in S, \Omega_t \neq m, \Omega_t \neq 0} \alpha_{t-1}(n, 1) a_{nm} + \alpha_{t-1}(\Lambda) a_{n\Lambda} \cdot b_m(\alpha_t) p_n(d) / \beta_t(d, m) \right), \]
\[ \alpha_t(\Lambda) \triangleq \Pr[\sigma_{t|T}, s_t = \Lambda] = \sum_{n \in S} \alpha_t(n, 1) a_{n\Lambda}, \]
\[ \alpha_1(m, d) = a_{\Lambda m} b_m(\alpha_t) p_t(d). \tag{3} \]

The backward variables and the backward formulas:

\[ \beta_t(m, d) \triangleq \Pr[\sigma_{t|T}, (s_t, \tau_t) = (m, d)] = b_m(\alpha_{t+1}) / \beta_{t+1}(m, d - 1), \text{ for } d > 1, \]
\[ \beta_t(\Lambda) \triangleq \Pr[\sigma_{t|T}, s_t = \Lambda] = \sum_{n \in S} \alpha_{t+1}(n, 1) a_{nm}, \]
\[ \beta_t(m, 1) = \sum_{n \in S} a_{nm} a_{n\Lambda} \sum_{d \geq 1} p_n(d) \beta_{t+1}(n, d), \]
\[ \beta_t(\Lambda, m, d) = 1, \text{ for } d > 1, \tag{7} \]
where \( s_{t+1} \) is the Null state between \( t \) and \( t+1 \) when \( a_{n\Omega_{t+1}} = 0, m \in S, d \in \{1, \ldots, D\}, t = 1, \ldots, T, \) and \( \Lambda \) is the Null state. Using the forward and backward variables defined above, we derive several other variables as follows:

\[ \xi_t(m, n) \triangleq \Pr[\sigma_{t|T}, \text{ state } m \text{ directly transits to } n] = \alpha_{t-1}(m, 1) a_{nm} b_n(\alpha_t) \sum_{d \geq 1} p_n(d) / \beta_t(n, d), \]
\[ \text{for } a_{nm} \neq 0 \text{ and } m \neq n, \tag{8} \]
\[ \xi_t(m, \Lambda) \triangleq \Pr[\sigma_{t|T}, s_t = \Lambda, n_t = n, \Omega_t = \Lambda] = \alpha_{t-1}(m, 1) a_{nm} a_{n\Lambda} b_n(\alpha_t), \]
\[ \cdot \sum_{d \geq 1} p_n(d) / \beta_t(n, d), \text{ for } m \neq n, \tag{9} \]
\[ \xi_t(\Lambda, n) \triangleq \Pr[\sigma_{t|T}, s_t = \Lambda, \Omega_t = \Lambda] = \alpha_{t-1}(\Lambda) a_{\Lambda n} b_n(\alpha_t) \cdot \sum_{d \geq 1} p_n(d) / \beta_t(n, d), \tag{10} \]
\[ \eta_t(m, d) \triangleq \Pr[\sigma_{t|T}, \text{ state } m \text{ starts at } t, \tau_t = d] = \left( \sum_{n \in S} \alpha_{t-1}(n, 1) \left( a_{nm} + a_{n\Lambda} a_{\Lambda m} \right) \cdot b_m(\alpha_t) p_n(d) / \beta_t(d, m) \right), \]
\[ \text{where we have implicitly considered that } a_{nn} = 0, \text{ and } \gamma_t(m) \triangleq \Pr[\sigma_{t|T}, s_t = m] = \Pr[\sigma_{t|T}, s_t \neq m, s_{t+1} = m] + \Pr[\sigma_{t|T}, s_t = m, s_{t+1} \neq m] = \gamma_{t+1}(m) + \sum_{n \in S, n \neq m} \left( \xi_t^+(m, n) - \xi_t^-(n, m) \right), \tag{13} \]
where \( m, n \in S, d \in \{1, \ldots, D\}, t = 1, \ldots, T, \) and \( \Lambda \) is the Null State.

Now we yield the estimation formulas based on these variables. Considering the observation variables \( r \) and \( q \) in \( b_m(q, r) \) are independent of each other, then we have \( b_m(q, r) = b_m(q) b_m(r) \). Furthermore, because our anomaly detection method will have multiple observation sequences on multiple user behaviors, we derive the re-estimation algorithm of the HSMM for multiple observation sequences by the frequency in this paper. Assuming there are \( L \) observation sequences with different lengths \( T^l (l = 1, \ldots, L) \), we obtain the extended re-estimation formulas:

\[ \hat{a}_{nm} = \sum_{l=1}^{L} \sum_{t=1}^{T^l} \xi_t^l(m, n) / R_{t^l}(m), \]
\[ \text{for } a_{nm} \neq 0 \text{ and } m \neq n, \tag{14} \]
\[ \hat{a}_{n\Lambda} = \sum_{l=1}^{L} \sum_{t=1}^{T^l} \xi_t^l(n, \Lambda) / R_{t^l}(m), \tag{15} \]
\[ \hat{a}_{\Lambda n} = \sum_{l=1}^{L} \sum_{t=1}^{T^l} \xi_t^l(\Lambda, n) / R_{t^l}(\Lambda), \tag{16} \]
\[ \hat{b}_t(r) = \sum_{l=1}^{L} \sum_{t=1}^{T^l} \gamma_t^l(m) \delta(d_t - q) / R_{t^l}(r), \tag{17} \]
\[ \hat{b}_t(q) = \sum_{l=1}^{L} \sum_{t=1}^{T^l} \gamma_t^l(m) \delta(d_t - q) / R_{t^l}(q), \tag{18} \]
\[ \hat{p}_t(d) = \sum_{l=1}^{L} \sum_{t=1}^{T^l} \eta_t^l(m, d) / R_{t^l}(m), \tag{19} \]
\[ \Pr[\sigma_{t|T}, (s_t, \tau_t) = (m, d) | \Lambda] = \sum_{l=1}^{L} \Pr[\sigma_{t|T}, (s_t, \tau_t) = (m, d) | \Lambda], \tag{20} \]
where \( m, n \in S, r \in V, q \in C, d \in \{1, \ldots, D\}, l = 1, \ldots, L, \) and \( \hat{a}_{\Lambda \Lambda} = 0, \hat{a}_{n\Lambda} = 0, \) for all \( m, n \in S, \) the variables whose superscript are \( l \) belong to the \( l \)th user, and \( R_{t^l}(m), R_{t^l}(q), R_{t^l}(d), \) and \( R_{t^l}(m) \) are the normalization factors, such that \( \sum_{n \in S} \hat{a}_{nm} + \hat{a}_{n\Lambda} = 1, \sum_{n \in S} \hat{a}_{\Lambda n} = 1. \)
\[ \sum_{v \in V} \hat{b}_m(\tau) = 1, \quad \sum_{q \in C} \hat{b}_m(q) = 1, \quad \text{and} \quad \sum_d \hat{p}_m(d) = 1, \quad \text{respectively.} \]

We define the average logarithmic likelihood per observed symbol, \( \ln \left( \Pr \left[ \frac{d}{T_\tau} \middle| \lambda \right] \right) / T_\tau \), as the measure of normality of the observation sequence, and call it “entropy” in the rest of this paper.

V. ANOMALY DETECTION

A. Normality Detection and Filter Policy

Using the above re-estimation algorithm, we establish the new HsMM to describe normal web users’ browsing behaviors by training the model from a set of request sequences made by a lot of normal users. We define the deviation from the mean entropy of training data as the abnormality of an observed request sequence made by a user. The smaller the deviation, the higher the normality of the observation sequence.

Fig. 4 shows the detector and the filter that are based on the behavior model. The filter, residing between the Internet and the victim, takes in a HTTP request and decides whether to accept or reject (drop) it. If a request is accepted, it can pass through the filter and reach the victim. In Section VI, we will use the false negative ratio (FNR) to evaluate the performance of the filter.

We have designed a self-adaptive algorithm for on-line update of the HsMM in our previous work [26]. Due to the constraint in computing power, the detector/filter is unable to adapt its policy rapidly. Because the web access behavior is short-term stable [22], the filter policy must be fixed for only a short period of time, which will be called a slot in the rest of this paper. Define \( T_\delta \) as the length of the request sequence for anomaly detection. For a given HTTP request sequence of the \( t \)th user, we compute the average entropy of the user’s request sequences over the period of \( T_\delta \). Then we calculate the deviation of the average entropy from the mean entropy of the model. If the deviation is larger than a predefined threshold, the user is regarded as an abnormal one, and the request sequence will be discarded by the filter when the resource (e.g., memory and bandwidth) is scarce. Otherwise, the user’s HTTP request can pass through the filter and arrive at the victim smoothly. Then, when the given slot is timed out, the model can implement the on-line update by the self-adaptive algorithm proposed in [26]. This enables the model to achieve a long-term automatic running. We will discuss this issue in detail in Section VII.

B. On-Line Algorithm for the Computation of Normality

The total number \( N \) of states is very large, which requires a lot of computations for anomaly detection. To implement the algorithm online, we introduce a new algorithm based on the M-algorithm [15] to solve this issue.

The M-algorithm is being widely adopted in decoding digital communications because it requires far fewer computations than the Viterbi algorithm. The aim of the M-algorithm is to find a path with distortion or likelihood metrics as good as possible (i.e., minimize the distortion criterion between the symbols associated to the path and the input sequence). However, contrary to the Viterbi algorithm, only the best \( M \) paths are retained at every instant in the M-algorithm. It works as follows. At time \( t \), only the best \( M \) paths are retained for extension. Associated to each path is a value called the path metric, which is the accumulation of transition metrics and acts as a distortion measure of the path. The transition metric is the distortion introduced by the distance between the symbols associated to a trellis transition and the input symbol. The path metric is the criterion to select the best \( M \) paths. Then the M-algorithm moves forward to the next time instant \( t+1 \) by extending the \( M \) paths it has retained to generate \( N \cdot M \) new paths. All the terminal branches are compared to the input data corresponding to this depth metric, and the \((N-1) \cdot M\) poorest paths are deleted. This process is repeated until all the input sequences have been processed.

We apply the M-algorithm into our on-line anomaly detection based on the following considerations. (i) We have mentioned in Section III that there are only a small number of pages that are referenced frequently. Since we use the Markov states to describe the clicked webpages, this means many rarely visited states (or, say, pages) are redundant. (ii) Based on (20), we see that the entropy of an observation sequence can be computed by the forward variables \( \alpha_T^T(m, d) \). Thus, we only need to optimize the forward process of HsMM, which is very suitable for using the M-algorithm to reduce the computations.

Hence, in this paper, we limit the number of states survived in computing the entropy of the browsing path of a user. We define \( S_t \) as the set of states survived in the \( t \)th forward step and \( S_t^+ \) the set of states that can transit directly from one of the states in \( S_t \). Thus, \( S_{t+1} \) is selected from \( S_t^+ \) according to the M-algorithm. The M-algorithm for computing the entropy of the \( t \)th user sequence is as follows.

Step 1. Select an initial subset \( S_0 \) containing \( M \) states, such that for each \( m \in S_0 \), \( a_{m\lambda} \geq a_{\lambda m} \) for all \( n \in S - S_0 \) and let \( S_0^+ = \{ n : a_{mn} \neq 0, m \in S_0, n \in S \} \) and \( t = 1 \), where \( M \) is a given number.

Step 2. Compute \( \alpha_T^T(m, d) \) for all \( m \in S_{t-1}^+ \) by (1)-(3).

Step 3. Select \( M \)-states subset \( S_t \subseteq S_t^+ \), such that for each \( m \in S_t \), \( \sum_d \alpha_T^T(m, d) \geq \sum_d \alpha_T^T(n, d) \) for all \( n \in S_{t-1}^+ - S_t \) and let \( S_0^+ = \{ n : a_{mn} \neq 0, m \in S_t, n \in S \} \).

Step 4. Set \( t = t+1 \) and repeat step 2 and step 3 until \( t = T \) where \( T \) is the length of \( t \)th user’s sequence.

Step 5. Use (20) to compute the entropy of the user’s observation sequence.

Figs. 5 and 6 show the pseudocodes of the proposed scheme which includes model training and anomaly detection.

C. Computational Complexity

In the training phase, our algorithm needs to estimate about \((N + 1)^2 + N(1 + K + D)\) parameters. For example, in our experiment \( N \) is about 1500 (pages), \( K \) is about 4000 (objects).
During our experiment, we find that most elements of the model parameters, i.e., $a_{mn}$, $b_m(r)$, $b_n(q)$, $p_m(d)$ are zeros. So we can apply a sparse matrix to express these variables. This method can save memory and improve the computational efficiency in the practical implementation. In the anomaly detection phase, the most important factor is the computational complexity in computing the user’s entropy. By using the new algorithm introduced in the last subsection, only a subset of the forward variables $\alpha(t)$, i.e., formulae (1)-(3), are needed to be computed. Let $\lambda = \{a(t, d), b_m(r), b_n(q)\}$, the computational complexity in computing the user’s entropy is $O(N N' D)$, where $N < N' < N$. Besides, $(N + 1)^2 + N(K + D)$ memory units are needed for the model parameters and $N' D$ for the forward variables. We do not need to store the observations. Therefore, we can compute the entropies for all observation sequences efficiently and implement the on-line anomaly detection.

### VI. Experiment Results and Analysis

In this section, we used the busiest 6-hour trace from an educational website to validate our anomaly detection algorithms. Among the selected data, we randomly used two thirds of the user request sequences (about 4700 sequences) within the first two hours to implement the HsMM training. The remainder data are used for testing. The model was performed on a computer with an Intel Pentium IV 2.80 GHz CPU, 1 Gbyte of RAM.

#### A. Initialization of Model Parameters

In our HsMM, a state denotes a page, transition among states denotes hyperlinks among different pages, and emissions of a state are objects forming the corresponding page. Therefore, the state transition matrix is determined by the website’s structure of hyperlinks and the emission/observation matrix is determined by the objects embedded in the pages. In the model training phase, we only needed to determine the state transition probabilities that describe how frequently the users browse from one page to another, and the emission/observation probabilities for each state that describe how often the requests for the objects forming a page can arrive at the original webserver. Hereby, we extract the website’s basic information, e.g., hyperlink structure and each page’s objects, to initialize the model.

Errors exist for dynamic pages whose hyperlinks or in-line objects are dynamically produced and determined by the scripts (e.g., Java script or VB script) because we cannot explore all possible transitions among dynamic pages. Therefore, we limit a state to denote a static page and use a macro state to represent all dynamic pages.

#### B. Characteristics of Web Browsing Behaviors

We first did some statistics for the collected traffic data. As shown in Fig. 7, about 100 pages (states) draw 98% of web access in the website. This result is similar to most previous works on web mining. Therefore, the most frequently appearing states are limited to about 100. This implies that the M-algorithm is suitable for reducing the computational amount in anomaly detection online. We then used the HsMM to estimate the state sequences of users from the collected data. Fig. 8 shows the distribution of the number of states (or pages) experienced in browsing the website by each user. There are several peaks in the plot and the biggest one is at about 30 pages. Most request sequences’ length of our experiment data is within $[20, 40]$. This implies that the M-algorithm is suitable for reducing the computational amount in anomaly detection online.
browsing behavior. Fig. 9 shows the distribution of the browsing time for each page. It resembles Pareto distribution.

C. App-DDoS Attacks Detection

We simulate the App-DDoS attack based on [24] and [25]. We assume the attack coverage is about 0.025, which is defined as the ratio of attack clients to whole clients. Each node attacks the webserver by sending GET requests with random objects. We insert the App-DDoS attack requests into the normal traffic shown in Fig. 10(a). As shown in Fig. 10(b), the App-DDoS attacks persist about two hours from 8725 s to 16250 s. In order to generate a stealthy attack which is not easily detected by the traditional methods, we shape each attack node’s output traffic to approximate the average request rate of normal user. The App-DDoS attack aggregated from the low-rate malicious traffic as shown in Fig. 10(b), the histogram of arrival rate distribution of normal users and attack clients is shown in Fig. 11(a), and the QQplot of the histogram is shown in Fig. 11(b). Obviously, the statistics of these characteristics for both normal clients and attackers are similar. Thus, the frequency analysis and the average arrival rate analysis are not suitable for distinguishing the low-rate attackers from the normal users. We then apply the proposed scheme to the detection of App-DDoS attacks. Fig. 12 shows the histogram of entropies of the normal users (curve $a$) and the abnormal users (curve $b$). There are significant differences in the entropy distributions between these two groups: the entropy for most of the normal users is larger than $-5$, but for the attack nodes, it is less than $-5$ and is mainly between $(-8, -5)$. Therefore, the model can distinguish the attackers from normal users by their entropies.
TABLE I

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average($\mu$)</td>
<td>-2.846759</td>
<td>-6.830405</td>
</tr>
<tr>
<td>Standard Deviation($\sigma$)</td>
<td>1.069519</td>
<td>0.460759</td>
</tr>
</tbody>
</table>

Fig. 13. Cumulate distribution of average entropies.

Fig. 14. ROC curve.

The statistical results of the entropy of normal training data and emulated App-DDoS attacks are given in Table I. Fig. 13 shows that if we take $-6.0$ for the threshold value of normal web traffic’s average entropy, the false negative ratio (FNR) is about 1%, and the detection rate is about 98%. Fig. 14 is the receiver operating characteristics (ROC) curves showing the performance of our anomaly detection algorithms on App-DDoS attacks. In our scheme, user’s normality depends on the entropy of his/her HTTP sequence fitting to the model. Thus, we need a threshold of the sequence’s length to decide whether the sequence is normal or not. We call this threshold “decision length” which includes two issues: (i) the amount of the request in the sequence, “sequence decision length”; (ii) the time of the HTTP sequence, “time decision length”. Decision length may affect the real-time response time and precision of our detection system. The shorter the decision length, the better the real-time response, with less reliability. Fig. 15 shows the entropies varying with two types of decision length. Fig. 15(a) and (c) present the entropy versus sequence length for normal users and attack clients, respectively. We can see the entropies converge with the increase of sequences’ length. When the length is larger than 150, the normal users’ entropies centralize to $[-4, -2]$ while those of attack sequences centralize to $[-8, -6]$. Thus, the best sequence decision length for this trace is about 150. Fig. 15(b) and (d) present the entropy versus time for normal users and attack clients. Similarly, when the duration of continuous HTTP request sequence is larger than 100 seconds, entropies of normal users centralize to $[-4, -2]$ while the attack sequences’ entropies centralize to $[-8, -6]$. Hence, the best response time used to detect an abnormal user is about 100 seconds.

D. Efficiency of the Detection Algorithm

In the experiment, there are 1493 pages (i.e., the size of $S$) and 4130 accessed objects (i.e., the size of $V$). We assume $D$ is 15 and $M$ for the M-algorithm changes from 25 to 1493 with increment of 25. As shown in Fig. 16, when $M$ is 500 (about 1/3 of the total states), the average entropy is about -2.803, and the difference in the average entropy between the classical algorithm and the new algorithm is very small (about 0.002). This shows that we can use the new algorithm with small $M$ to compute the user’s entropy, which can reduce the computational complexity and memory requirement significantly in the on-line implementation. As for the training phase, Fig. 17 shows that the algorithm can converge at a point after about 10 iterations. The timer shows the execution time of model parameter estimation is less than 20 minutes in our experiment.

VII. DISCUSSION

Several related issues of the proposed scheme are discussed in this section.

A. Expansion of the Model

In our experiment, we only consider the attack manner whose request rate and length are similar with those of normal users’. For other special attacks, e.g., the length of attack sequence less than 150 and/or arranging each vicious sequence to be not more than 100 seconds, the above scheme should be extended. We show its main idea based on the experiment results. Fig. 8 shows that the length distribution of requested page sequences has several peaks in the histogram and the biggest one is located at the point of about 30 pages, which imply the browsing behaviors of the users can be classified into five groups (i.e., $P = 5$). Thus in this case, in order to obtain more accurate detection results, five HsMMs are required. Then, the detection is implemented according to Fig. 18. This way the incoming sequences of any length (including the short attacks issuing 10 requests for pages) can also be detected. The expansion method includes three aspects, as follows.

First, we cluster the web users’ sequences (i.e., training data) into $P$ groups ($g_{i, i = 1, \ldots, P}$) by analyzing the distribution of their request sequence length. The groups are indexed by their length distribution, i.e., $g_1$ denotes the group in which the average length of users’ request sequences is shortest, $g_2$ denotes the group with second shortest average length of request sequences and so on. $g_P$ is the last group with longest average length of request sequences.

Second, model $M_i (i = 1, \ldots, P)$ is trained by the group $g_i$ and is used to describe the access behavior characteristics of $g_i$.

Then, an increasing length of incoming request sequence is examined by $M_1$, then $M_2$, $M_3$, and so forth, as shown in Fig. 18. In any stage, the system can issue an alarm for a suspicious client if the deviation of her/his access behavior from the model is beyond the acceptable threshold.
Fig. 15. Varieties of users’ likelihood. (a) Entropy versus index of request for normal users. (b) Entropy versus time for normal users. (c) Entropy versus index of request for attack sequences. (d) Entropy versus time for attack sequences.

Fig. 16. Performance of the new algorithm. (a) log(likelihood) of different $M$. (b) log(likelihood) difference of different $M$ between the new algorithm and the classical algorithm.

Fig. 17. Convergence of model training.

B. About the Model

HsMM is applied to describe the web user behavior. It has been successfully applied in many fields, such as machine recognition, sequences clustering, mobility tracking in wireless networks, and inference for structured video sequences. The most notable characteristic of HsMM is that it can be used to describe most physical signals without many hypotheses. Furthermore, the non-stationary and the non-Markovian properties of HsMM can best describe the self-similarity or long-range dependence of network traffic that has been proved by vast observations on the Internet [14], which quite benefits our study. On the other hand, the structure of HsMM is also very suitable to describe the web browsing behavior, e.g., the relation between webpage and in-line objects, the effect caused by the middle proxies and caches. Last, as we have discussed in Section V, based on the forward-backward algorithm and the proposed M-algorithm for HsMM, both computational complexity and memory requirement of the improved model are lower than the data mining, thus it can satisfy the requirement of on-line detection.

C. Deviation of Model

Central to the notion of our scheme is the idea of being able to detect “outliers” or abnormal browsing behavior by computing entropy of the HTTP sequence against the model. If the threshold that is used to measure entropy as abnormal is not properly set, too many false alarms will be issued. Here, we present a method to determine the threshold. In the experiments of this paper, the DR is 98% and the FPR is 1.5% when the threshold of entropy is set to 6.0. However, when the system is put into real application, we cannot preset this threshold. What we know is just the entropy distribution of the normal request sequences and the FPR. We suggest a general method to determine the threshold. The second method is
Based on the Central Limit Theorem. That is, given a distribution with a mean and variance, the sampling distribution approaches a Gaussian distribution. Thus, we can describe entropy distribution of the training data using the following Gaussian distribution:

\[
G(\text{entropy}, \mu, \sigma) = \exp \left\{ -\frac{(\text{entropy} - \mu)^2}{2\sigma^2} \right\} \left( \frac{1}{\sqrt{2\pi}\sigma} \right)
\]

where \(\mu\) is the mean and \(\sigma\) is the variance. In considering that \(3\sigma\) error level could give us a confidence interval of 99.7%, it should be good enough for even high precision detection scenarios. Table II lists the detection threshold setting and their corresponding FPR and DR of our experiments (the mean \(\mu\) and variance \(\sigma\) have been given in Section VI). As indicated in this table, the detection level could be safely selected to be \(\mu \pm 3\sigma\) and this choice ensures us with FPR smaller than 2.0% and DR larger than 97%.

**VIII. CONCLUSIONS AND FUTURE WORK**

In this paper, we focused on the detection of App-DDoS attacks and presented a new model to describe browsing behavior of web users based on a large-scale Hidden semi-Markov Model. A new on-line algorithm based on the M-algorithm was designed for the anomaly detection. A set of real traffic data collected from an educational website and a generated App-DDoS attack traffic were used to validate our model. The experiment results showed that when the detection threshold of entropy is set as \(\mu \pm 3\sigma\), the DR is 98% and the FPR is 1.5%, which demonstrated that the model could be used effectively to describe the browsing behavior of normal web users and detect the App-DDoS attacks. We also showed that the new algorithm can reduce the memory requirement and improve the computational efficiency.

Several issues will need further research: 1) coping with the attacks launched by dynamic webpage (e.g., script); 2) detecting the App-DDoS attacks mimicking behavior of proxy server instead of individual web surfers; and 3) applying the proposed scheme to detect other application-layer based attacks, such as FTP attacks.

**ACKNOWLEDGMENT**

The authors would like to thank the anonymous referees for their helpful comments and suggestions to improve this work. They also thank the Network Center of Sun Yat-Sen University for their help with the access logs.

**REFERENCES**


Yi Xie received the B.S. and M.S. degrees from Sun Yat-Sen University, Guangzhou, China. From July 1996 to July 2004, he was working with the Civil Aviation Administration of China (CAAC). He returned to academia in 2004. He is currently working toward the Ph.D. degree at Sun Yat-Sen University. He was a visiting scholar at George Mason University during 2007 to 2008. His research interests are in network security and web user behavior model algorithms.

Shun-Zheng Yu (M’08) received the B.Sc. degree from Peking University, Beijing, China, and the M.Sc. and Ph.D. degrees from the Beijing University of Posts and Telecommunications, China. He was a visiting scholar at Princeton University and IBM Thomas J. Watson Research Center during 1999 to 2002. He is currently a Professor at the School of Information Science and Technology, Sun Yat-Sen University, Guangzhou, China. His recent research interests include networking, traffic modeling, anomaly detection and algorithms for hidden semi-Markov models.