SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2014; **7**:1849–1859
Published online 10 December 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.898

RESEARCH ARTICLE

Behavior-based botnet detection in parallel

Kuochen Wang[1], Chun-Ying Huang[2]*, Li-Yang Tsai[1] and Ying-Dar Lin[1]

[1] Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
[2] Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan

ABSTRACT

Botnet has become one major Internet security issue in recent years. Although signature-based solutions are accurate, it is not possible to detect bot variants in real-time. In this paper, we propose behavior-based botnet detection in parallel (BBDP). BBDP adopts a fuzzy pattern recognition approach to detect bots. It detects a bot based on anomaly behavior in domain name service (DNS) queries and transmission control protocol (TCP) requests. With the design objectives of being efficient and accurate, a bot is detected using the proposed five-stage process, including: (i) traffic reduction, which shrinks an input trace by deleting unnecessary packets; (ii) feature extraction, which extracts features from a shrunk trace; (iii) data partitioning, which divides features into smaller pieces; (iv) DNS detection phase, which detects bots based on DNS features; and (v) TCP detection phase, which detects bots based on TCP features. The detection phases, which consume approximately 90% of the total detection time, can be dispatched to multiple servers in parallel and make detection in real-time. The large scale experiments with the Windows Azure cloud service show that BBDP achieves a high true positive rate (95%+) and a low false positive rate (∼3%). Meanwhile, experiments also show that the performance of BBDP can scale up linearly with the number of servers used to detect bots. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS

anomaly detection; behavior-based; botnet detection; cloud computing; fuzzy pattern recognition; parallel process

***Correspondence**

Chun-Ying Huang, Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan.
E-mail: chuang@ntou.edu.tw

1. INTRODUCTION

Botnet has become one major threat to Internet users in recent years. A botnet is comprised of a large number of bots, which are networked computers compromised by malicious attackers. With a botnet, an attacker controls the bots to launch various types of attacks, such as phishing and spamming, and thereby receives huge economic benefits. Consequently, detecting bots' activities and preventing users from being infected are critical to security experts and researchers.

Most commodity solutions detects bot activities based on predefined patterns and signatures retrieved from well-known bots [1–6]. Although signature-based solutions are able to detect bots accurately, it has two major drawbacks. First, a bot is able to evade signature-based detection by using a code obfuscation technique. For example, the Mariposa bot adopts such a technique to prevent it from being detected [7]. Second, patterns or signatures used to detect bots are retrieved from known bots. This means that there would be no protection for a new bot before its patterns or signatures are identified. Therefore, we believe that a behavior-based (or anomaly-based) solution would be a good alternative to detect bots. With well-tuned parameters, behavior-based solutions are able to achieve high detection rates and low error rates. In addition, it is able to detect bot variants and even unknown bots.

This study presents a behavior-based botnet detection technique that is capable of detecting bots in parallel. On the basis of our previous work [8], we show that a behavior-based botnet detector not only detects bots effectively but also efficiently. The proposed solution also leverages a fuzzy pattern recognition approach and detects bots in two phases. The domain name service (DNS) phase analyzes DNS queries requested by clients and investigates possibly malicious queries sent from bots. In contrast, the transmission control protocol (TCP) phase examines TCP request packets and response packets and identifies anomaly access patterns in terms of packet counts and packet sizes. We revise the previously proposed algorithm to have more sophisticated membership functions in order to achieve higher detection (true positive) rates and lower error (false positive) rates. In addition, a parallel process design is proposed as well to improve the performance of

the detection system. By adopting modern infrastructure as a service cloud computing services, the required detection time of the proposed system can be shortened linearly to the number of allocated detecting servers.

The remaining of this paper is organized as follows. Section 2 briefly introduces the behavior of botnet and reviews several previous works related to the proposed solution. Section 3 explains the proposed approach in detail. Section 4 presents the experiment results for the proposed solution using commodity cloud computing services and real-world botnet traces. Finally, Section 5 provides the conclusion.

2. BACKGROUND AND RELATED WORK

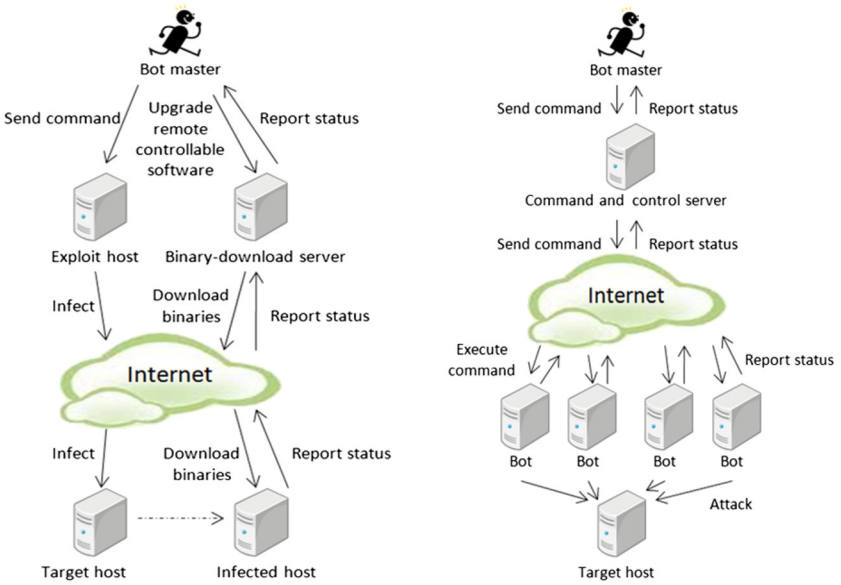
2.1. Overview of botnet behavior

Figure 1 shows the two common phases of a bot's behavior, that is, the infection phase and the attack phase. In the infection phase, a bot master attempts to intrude in a victim and then turn the victim into a bot. There are many techniques to intrude in a host such as remote exploits and drive-by downloads. Once the bot master has successfully intruded in a victim, remote controllable software (bot software) is downloaded and installed into the victim. The infection phase then finished after the bot software has been successfully installed and configured. Bot software is usually configured to launch automatically when the system boots. The attack phase then starts as well. In the attack phase, the bot software is responsible for reporting the status of the infected host to the bot master, receiving attack commands from the bot master, and launching commanded tasks. Possible commands include, but not limited to, launching distributed denial of services, setting up phishing sites, relaying malicious traffic, and sending spam mails. Behavior-based solutions can detect bots in the infection phase, the attack phase, or both, depending on how the algorithms are designed.

2.2. Related work

Park *et al.* [9] proposed an automated approach to generate semantic patterns for bot detection. They proposed to identify one pattern that represents the important behavior of an entire class of bots, rather than of individual instances. They adopted static analysis techniques to characterize bot behavior and proposed to use hierarchical clustering of the resulting semantic patterns from a set of bot programs. The major contribution of this work is that patterns and signatures are generated automatically. However, because it is based on static analysis from assembly source codes, the effectiveness is therefore limited when code obfuscation techniques are applied. In addition, signature-based detection limits its ability to detect bot variants and unknown bots.

Yu *et al.* [10] proposed online botnet detection based on an incremental discrete Fourier transform (DFT) approach. They first defined the concept of "feature streams" to describe raw network traffic. They then compared feature streams originated from different hosts and detect suspicious bot activities if similar feature streams are identified. It is innovated to represent network traffic as feature streams and detecting bots using incremental DFT.



(a) The infection phase. **(b)** The attack phase.

Figure 1. Two common phases of a bot's behavior: (a) the infection phase and (b) the attack phase.

However, the proposed solution has two issues. First, represent network traffic as feature streams and detect using DFT is a computation-intensive work. It could be inefficient to detect bot activities. Second, according to the experiment data provided by the authors, the error rates (false positive rates) is not low enough.

A number of works attempted to detect and discuss specific type of bots. Perdisci *et al.* [11] proposed to detect bots that leverage hypertext transfer protocol (HTTP) channels to communicate. Hsu *et al.* [12] and Lin *et al.* [13] proposed to detect a specific type of bots called fast-flux bots, which attempts to extend the life time of malicious web or Internet services using dynamic domain names with shorter time to live. Shirley *et al.* [14] discussed the possibilities that a bot could evade detection if a bot detection mechanism did not associate bots' communications with the corresponding hosts. Huang [15] proposed to detect bots based on failures generated from bots. If a bot never generate a failure, it could be missed.

Wang *et al.* [8] proposed to detect bots based on a fuzzy pattern recognition approach. They proposed a traffic reduction algorithm to reduce the amount of network traffic that the solution needs to process. To work with fuzzy pattern recognition techniques, they then designed several membership functions to compute the probability of being bot activities from aggregated DNS and TCP traffic. Although the simple functions adopted by this work are able to perform well on detection bots, the detection accuracy can be improved with more sophisticated membership functions. Some network traces generated by regular network activities such as checking new software updates may lead to false positives. With more sophisticated membership functions, it is possible to reduce the error (false positive) rates caused by regular network activities.

The proposed solution differs from previous researches in several aspects. First, it does not examine the binary codes or source codes of bot software. Detection is made on the basis of external behaviors. Therefore, code obfuscation does not prevent a bot from being detected. Second, because it is a behavior-based detector, bot variants and event unknown bots can be detected. The proposed solution extends Wang's *et al.* work. Similar to their work, the proposed solution adopts a fuzzy pattern recognition approach. However, it extends the previous work in two directions. First, the proposed solution adopts even more sophisticated membership functions to detect bots. It hence improves the overall detection (true positive) rates and reduces the error (false positive) rates. Second, the proposed solution pays more attention on detection bot activities in large scale networks. Therefore, the workload of detection tasks can be dispatched to multiple servers and detect bots in parallel. The efficiency of the detector can be improved linearly when the number of allocated detection server increases.

3. THE PROPOSED SOLUTION

3.1. Design objective

Given a network packet trace, the goal of the proposed behavior-based botnet detection in parallel (BBDP) is to detect bot activities from the trace. Two objectives of the proposed solution are accuracy and efficiency. The proposed solution should be able to detect as many bots as possible in a reasonable detection time. To achieve the goal, BBDP splits the process of the trace into five stages, as shown in Figure 2. The five stages (in the order) are traffic reduction, feature extraction, data partitioning, DNS detection, and TCP detection. The first two stages are used to reduce the amount of data that has to be processed by the system. To be more efficient, BBDP attempts to parallelize the detection process by dispatching workloads to multiple detectors in the third stage. Finally, BBDP detects bots in two phases, which focus on DNS queries and TCP requests generated by bots. Although the two phases are similar to our previous work [8], we revised the detection algorithms and adopted more detection policies to improve the overall detection accuracy. The details of each stage are discussed later in this section. In addition to detect bot activities, BBDP retrieves bot relevant domain names and Internet protocol (IP) addresses from

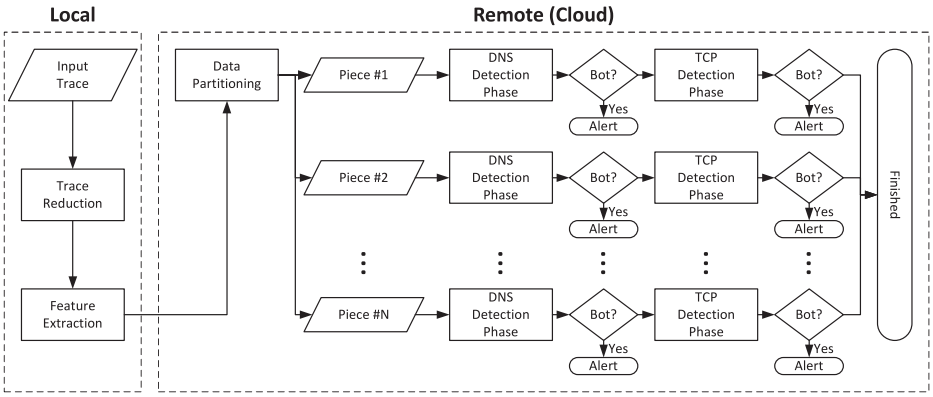


Figure 2. The five stages of the proposed behavior-based botnet detection in parallel.

the detected activities and sends the information to firewalls and/or intrusion detection systems to prevent other hosts from being attacked.

3.2. Traffic reduction

To improve the system efficiency, it would be better if the system only process required packets. Therefore, BBDP adopts a packet filtering process to reduce the number of packets that the system has to examine. Figure 3 shows the packet filtering process. On the basis of our observations, we found that a bot's activities often start from DNS lookups. This is because a bot often has to obtain new commands from the bot master or the command and control (C&C) servers, which are usually hard-coded as a list of domain names in bot software. With the bot master's or the C&C servers' IP addresses, the bot then attempts to interact with each of the returned IP addresses from DNS queries. We also found that most bots interact with the bot master or the C&C servers using TCP connections. Therefore, the proposed solution currently focused only on examining TCP packets. The filter process discards a packet if it is neither a DNS request/response packet nor a packet with known source/destination IP addresses. If a packet is not discarded, it is passed to the next stage and is used for detecting bot activities.

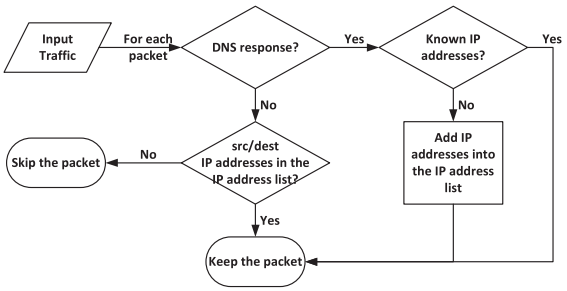


Figure 3. The packet filtering process to reduce the number of processed packets for the proposed system.

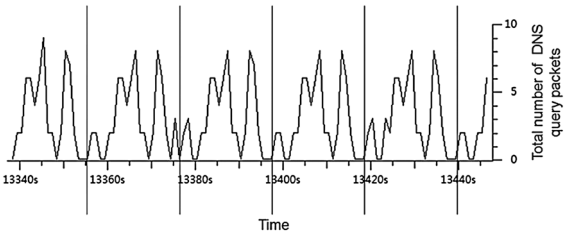


Figure 4. The number of domain name service (DNS) query packets sent by an observed live bot. (*x*-axis: time in seconds; *y*-axis: total number of DNS query packets.)

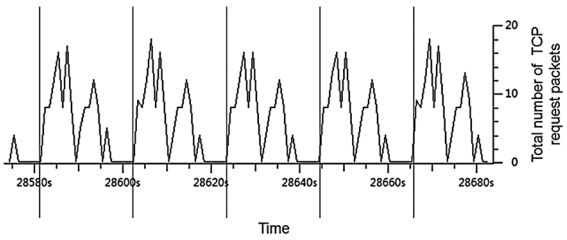


Figure 5. The number of transmission control protocol (TCP) request packets sent by an observed live bot. (*x*-axis: time in seconds; *y*-axis: total number of TCP request packets.)

3.3. Feature extraction

Since bot activities often start with DNS queries and then followed by interactions using TCP flows, BBDP retrieves several features relevant to DNS queries and TCP flows for botnet detection. For DNS features, we observed that a bot often sends DNS queries regularly in a period. Figure 4 shows an example of DNS query packets sent from a bot. BBDP collects DNS packets for a period and then retrieves relevant DNS features including the queries round trip times, queries intervals, and queries frequencies.

For TCP features, we also observed that a bot would setup regular network flows to the bot master or the C&C server, as shown in Figure 5. BBDP collects TCP packets for a period as well and then retrieves relevant TCP features including packet count per second, byte count per packet, requests intervals, and request frequencies. Both the retrieved DNS and TCP features are passed to the next stage for botnet detection.

3.4. Data partitioning

Behavior-based botnet detection in parallel aims to be an accurate and efficient bot detection system. It is straightforward to improve system efficiency by splitting the whole workloads into smaller pieces and then dispatching pieces to multiple servers. However, if workloads are not split properly, the detection accuracy could be degraded. Therefore, the system has to consider how input features are split so that it can achieve high efficiency without losing its accuracy. BBDP splits features by following a similar manner to traffic reduction. Since a bot's activities start with DNS queries and then followed by a number of relevant TCP flows, the DNS queries and the incurred TCP flows should not be split into different pieces. For example, suppose a host H made a DNS lookup to a domain name D and receives a list of n corresponding IP addresses IP_1, IP_2, …, IP_n. The DNS features involved with D and the TCP features involved with $\mathrm{IP}_1, \mathrm{IP}_2, \ldots, \mathrm{IP}_n$ should not be split into different pieces. For the ease of processing, we use H's IP address as the key to split traffic. Consequently, both DNS features and relevant TCP features for H would be placed in the same piece. Because BBDP detect bots for an entire network, features collected for different hosts can be split into different pieces and therefore improves the parallelism.

3.5. Behavior-based botnet detection in parallel

Behavior-based botnet detection in parallel adopts a fuzzy pattern recognition approach to identify bots. The bot detection process contains two phases—the DNS detection phase and the TCP detection phase. The two phases detect bots using features retrieved from DNS packets and TCP packets, respectively. BBDP attempts to detect a bot in the DNS detection phase first. If a bot is detected in the DNS detection phase, BBDP ignores TCP features associated with the DNS features and reports the detection result. In contrast, if no bot is detected, the TCP detection phase is then applied to detect a bot. Both the DNS detection phase and the TCP detection phase detect bots using the max membership principle to determine whether a retrieved feature is more close to bot activities or benign activities. A number of fuzzy membership functions are defined to determine a retrieved feature set from host activities is a member of bot features or benign features. Given a feature set retrieved from a networked host, if membership functions for bots output higher values, the host is detected as a bot. Similarly, if membership functions for benign hosts output higher values, the host is detected as a benign host. The basic concept of the max membership principle is shown in Figure 6. The detection is made by finding the maximal values from the defined membership functions. We summarize host behaviors inspected by the proposed solution in Table I. The details of how the membership functions are defined are discussed later in this subsection.

3.5.1. Domain name service detection phase.

Given a predefined observation period and a trace file, we defined a packet feature vector $x = (\alpha, \beta, \gamma, \lambda)$ for the DNS detection phase. α is a set of time intervals measured between a pair of a DNS query and the corresponding DNS response. Suppose n DNS queries are observed from a host, each measured time interval in α is notated as α_i, where $1 \leq i \leq n$. β is a set of counters, which count the number of concurrent DNS queries within the period defined by α. The cardinality of β ($|\beta|$) should be equivalent to that of α ($|\alpha|$) because the observation is made for exact the same trace file. Each counter in β is notated as β_i, where $1 \leq i \leq n$. γ is another set of counters, which count the number of total times that a domain name or an IP address has been queried by a host. Suppose a monitored host has queried N distinct remote IP addresses, the cardinality of γ ($|\gamma|$) would be N. A counter $\gamma_i \in \gamma$, $1 \leq i \leq N$, maintains the number of times that a domain names or IP addresses found in the trace. λ is also a set of counters, which count the number of DNS queries in each second within the observation period. Suppose a trace has been monitored for M seconds, there would be M counters in λ. In the DNS detection phase, we defined four states and proposed the corresponding membership functions, as described in the following.

(i) *Normalized variance of number of concurrent DNS queries*

A bot often generates concurrent DNS queries to shorten its online time. A higher variance of number of concurrent DNS queries indicates a host is possibly a bot. Therefore, we defined a

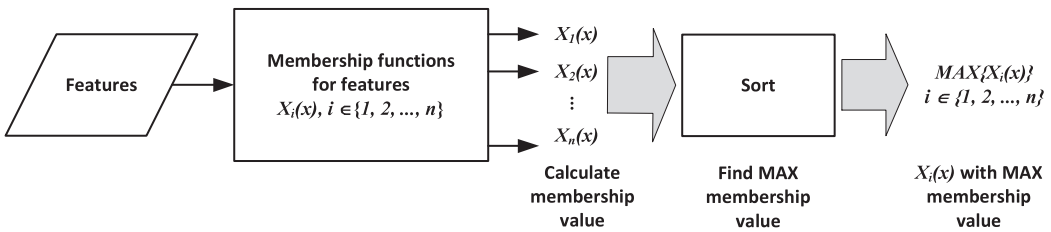


Figure 6. The max membership principle.

Table I. Summary of inspected host behavior.

Type	Equation	Description
Domain name service	X_1	Number of concurrent queries
	X_2	Cumulated queries to each distinct domain
	X_3	Query frequency
Transmission control protocol	X_1	Average packet rate (per connection)
	X_2	Average packet size (per connection)
	X_3	Packet count variance (per connection)
	X_4	Packet size variance (per connection)
	X_5	Average packet rate (per host)

membership function X_1 for calculating the normalized variance of number of concurrent DNS queries as follows.

$$X_1(x) = \begin{cases} 0, & \text{if all } \left(\beta_i - \overline{\beta}\right)^2 < T_{x_1} \\ \dfrac{\max\left(\left(\beta_i - \overline{\beta}\right)^2\right)}{\sum\left(\left(\beta_i - \overline{\beta}\right)^2\right)}, & \text{otherwise;} \end{cases} \tag{1}$$

for $i \in \{1, 2, \ldots, n\}$, where n is the total number of DNS queries and T_{x_1} is the variance threshold of being abnormal.

(ii) *Normalized total times that a node queried the same domain name or IP address*

A bot may query specific domain names or IP addresses many times when it is activated. Therefore, we calculated the number of queries to a domain name or an IP address to identify abnormal hosts. We defined a membership function X_2 for calculating normalized total times that a node queried the same domain name or IP address.

$$X_2(x) = \begin{cases} 0, & \text{if all } \gamma_i < T_{x_2} \\ \dfrac{\max(\gamma_i)}{\sum(\gamma_i)}, & \text{otherwise;} \end{cases} \tag{2}$$

for $i \in \{1, 2, \ldots, N\}$, where N is the number of domain names and IP addresses that a host has queried and T_{x_2} is the threshold of the abnormal contact counts for a domain name and an IP address.

(iii) *Normalized total number of DNS query and response packets per second*

A bot sends DNS queries several times when it is activated. Therefore, we calculate the total number of DNS queries per second to identify anomalies. We defined a membership function X_3 for calculating normalized total number of DNS query and response packets per second as follows.

$$X_3(x) = \begin{cases} 0, & \text{if all } \lambda_i < T_{x_3} \\ \dfrac{\max(\lambda_i)}{\sum(\lambda_i)}, & \text{otherwise;} \end{cases} \tag{3}$$

for $i \in \{1, 2, \ldots, M\}$, where M is the duration of an input trace in seconds and T_{x_3} is the threshold for the total number of DNS query and response packets per second.

The first three membership functions define bots' DNS activities. We also defined a membership function X_4 for calculating the probability of being a normal DNS activity.

$$X_4(x) = 1 - \max(X_1(x), X_2(x), X_3(x)) \tag{4}$$

3.5.2. Transmission control protocol detection phase.

We defined a packet feature vector $x = (\alpha, \beta, \gamma, \lambda)$ as well for the TCP detection phase. α is a set of time intervals measured between a pair of a TCP request and the corresponding response. Suppose n TCP requests are observed, each measured time interval in α is notated as α_i, where $1 \leq i \leq n$. β is a set of counters, which count the number of TCP request packets involved in each TCP request. The cardinality of β ($|\beta|$) should be equivalent to that of α ($|\alpha|$) because the observation is made for exact the same trace file. Each counter in β is notated as β_i, where $1 \leq i \leq n$. γ is a set of counters, which count the number of payload bytes that are sent in a TCP request. Similarly, the cardinality of γ ($|\gamma|$) should be equivalent to that of α ($|\alpha|$). Each counter in γ is notated as γ_i, where $1 \leq i \leq n$. λ is another set of counters, which count the average number of TCP request and response packets sent per second. In the TCP detection phase, we defined six states and proposed the corresponding membership functions, as described in the following.

(i) *Normalized packet counts per second*

It is abnormal for a TCP connection to send a large number of request packets in a second. On the basis of the assumption, we defined a membership function X_1 for calculating the normalized packet counts per second as follows.

$$X_1(x) = \begin{cases} \dfrac{\beta_t/\alpha_t}{T_{x_1}} - 1, & 1 < \dfrac{\beta_t/\alpha_t}{T_{x_1}} < 2 \\ 1, & \dfrac{\beta_t/\alpha_t}{T_{x_1}} >= 2 \\ 0, & \text{otherwise;} \end{cases} \tag{5}$$

where β_t is the total number of TCP packets in an input trace, α_t is the duration of an input trace in seconds, and T_{x_1} is the threshold for abnormal packet count per second.

(ii) *Normalized byte counts per packet*

If a bot master attempts to send commands to its controlled bots, the byte count per TCP packet may reflect the anomaly. We defined a membership function X_2 for calculating the normalized byte count per packet as follows.

$$X_2(x) = \begin{cases} \dfrac{\gamma_t/\beta_t}{T_{x_2}} - 1, & 1 < \dfrac{\gamma_t/\beta_t}{T_{x_2}} < 2 \\ 1, & \dfrac{\gamma_t/\beta_t}{T_{x_2}} >= 2 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where γ_t is the total number of bytes in an input trace, β_t is the total number of TCP packets in an input trace, and T_{x_2} is the threshold for abnormal byte counts per packet.

(iii) *Normalized variance of total number of TCP packets in each request*

A bot often sends a large number of request packets in a short time. The burst can then be found through the high variance of requested TCP packets. Therefore, we defined a membership function X_3 for calculating the normalized variance of total number of TCP packets in each request as follows.

$$X_3(x) = \begin{cases} 0, & \text{if all } \left(\beta_i - \overline{\beta}\right)^2 < T_{x_3} \\ \dfrac{max\left(\left(\beta_i - \overline{\beta}\right)^2\right)}{\sum\left(\left(\beta_i - \overline{\beta}\right)^2\right)}, & \text{otherwise;} \end{cases} \quad (7)$$

for $i \in \{1, 2, \ldots, n\}$, where n is the total number of TCP requests and T_{x_3} is the variance threshold of being abnormal.

(iv) *Normalized variance of the total number of payload bytes in each request*

In addition to send packets in a burst, we observed that the sizes of the payloads carried by the packets from a bot are large. Therefore, we defined a membership function X_4 for calculating the normalized variance of the total number of payload bytes in each request as follows.

$$X_4(x) = \begin{cases} 0, & \text{if all } (\gamma_i - \overline{\gamma})^2 < T_{x_4} \\ \dfrac{\max\left((\gamma_i - \overline{\gamma})^2\right)}{\sum\left((\gamma_i - \overline{\gamma})^2\right)}, & \text{otherwise;} \end{cases} \quad (8)$$

for $i \in \{1, 2, \ldots, n\}$, where n is the total number of TCP requests and T_{x_4} is the variance threshold of being abnormal.

(v) *Normalized total number of TCP request and response packets per second*

A bot may send TCP request and response packets many times when it is activated. Therefore, we calculated the total number of TCP request and response packets per second to identify anomalies. We defined a membership function X_5 for calculating the normalized total number of TCP request and response packets per second as follows.

$$X_5(x) = \begin{cases} 0, & \text{if all } \lambda_i < T_{x_5} \\ \dfrac{\max(\lambda_i)}{\sum(\lambda_i)}, & \text{otherwise;} \end{cases} \quad (9)$$

for $i \in \{1, 2, \ldots, M\}$, where M is the duration of an input trace in seconds, λ_i is the total number of TCP request and response packets in the i^{th} second, and T_{x_5} is the threshold of the total number of TCP packets per second.

The first five membership functions define bots' TCP activities. We also defined a membership function X_6 for calculating the probability of being a normal TCP activity.

$$X_6(x) = 1 - \max(X_1(x), X_2(x), X_3(x), X_4(x), X_5(x)) \quad (10)$$

4. PERFORMANCE EVALUATION

4.1. Trace collection

We collected a large number of bot traces from 250 real bot samples and iteratively launched each of them in a controlled environment, as shown in Figure 7. Each bot was launched in a virtual machine running an unpatched Windows XP service pack 3 operating system. Each virtual machine moves a bot from the share folder to its local disk, launches the bot for 2 h, and then restores itself to a clean state. The virtual machines repeatedly launch bots from the share folder until all bots have been examined. The traces generated from the virtual machines were then captured by an external sniffer. In the experiments, 240 out of the 250 bots had generated network traces. Both packet headers and the complete packet payloads were stored. We called the collected malicious traces "data set M."

In addition to bot traces, we collected benign traces to evaluate the proposed system. Benign traces were collected from two different sources. One was collected from the campus dormitory network [16], and the other was collected from our lab. We collected traces generated from 695 hosts in the dormitory network and from five hosts in

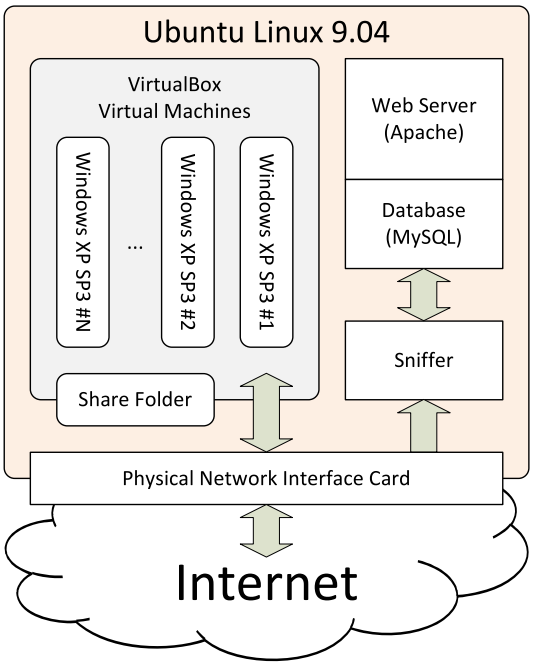


Figure 7. The experimental environment for botnet traces collection.

our lab. Each host was collected for 2 h as well. These traces contained many types of benign applications including Internet Relay Chat (IRC), HTTP, and peer-to-peer traffic. We called the traces collected from dormitory network "data set B1" and the traces collected from our lab "data set B2."

4.2. Feature evaluation

Behavior-based botnet detection in parallel detects bots based on counting membership values for the selected features. Therefore, we have to know whether the selected features are able to differentiate bots from benign hosts before making experiments. We randomly selected 25 bot traces and 25 normal traces to evaluate the selected features. Figure 8 shows the scatter-plots for the selected features. The scatter-plots are plotted from six selected features including the variance of packet interarrival time, the variance of bytes per packet, the number of packets between request and response packets, and the contact counts per host, the average number of packets per second, and the average number of bytes per packet. Therefore, there are total C_2^6 plots. We found that bots and normal hosts can be roughly differentiated on the basis of the combinations of the selected features. In addition, we also found that some features are especially useful for detecting certain types of bots. For example, "the number of packets between request and response packets" and "the contact counts per host" are better for detection IRC bots. In contrast, "the average number of packets per second"

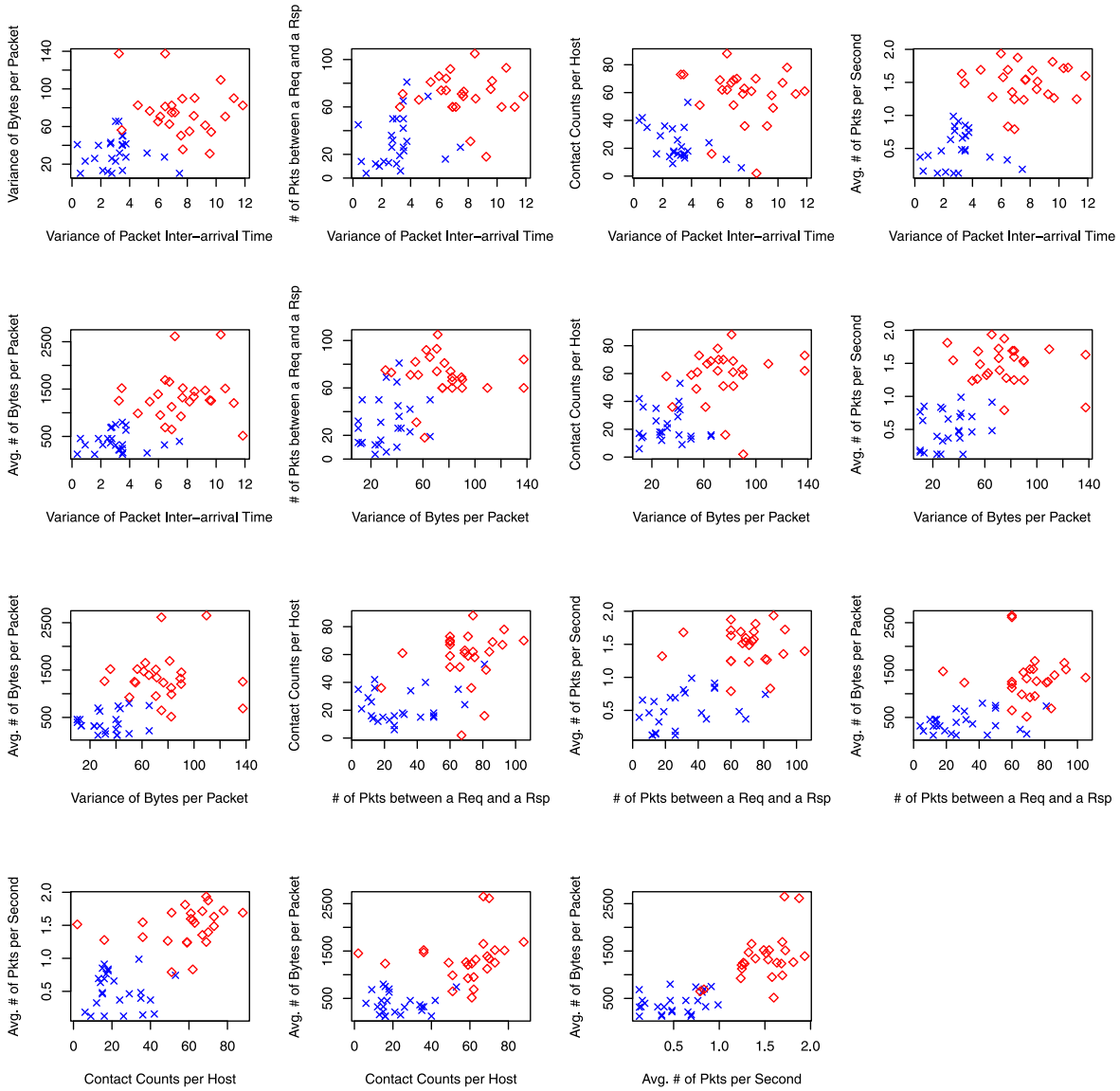


Figure 8. Scatter-plots for combinations of selected features. Blue-cross for normal traces and red-diamond for bot traces.

and "the average number of bytes per packet" are better for detection HTTP bots.

4.3. Detection threshold

To work with the proposed solution, we have to decide the thresholds used by the membership functions. We obtained the thresholds by making statistics to the collected benign and bot traces. The thresholds used for detection are listed as follows:

(1) T_{x_1} for Equation (1): 4
(2) T_{x_2} for Equation (2): 45
(3) T_{x_3} for Equation (3): 1.25
(4) T_{x_1} for Equation (5): 1.5
(5) T_{x_2} for Equation (6): 75
(6) T_{x_3} for Equation (7): 4
(7) T_{x_4} for Equation (8): 45
(8) T_{x_5} for Equation (9): 1.25

4.4. Detection accuracy

We used the collected traces to evaluate BBDP. A summarization of the detection accuracy is provided in Table II. In addition to high traffic reduction rates, BBDP has a low false negative rate (4.17%) and low false positive rates (3.45% for data set B1 and 0% for data set B2). Among the total 10 false negatives, we found that six instances are IRC bots and four instances are HTTP bots. In contrast, there are total 24 false positives. We further investigated the origin of false positives. Figure 9 shows the origin of false positives in the DNS detection phase and the TCP detection phase. The statistics also show that false positives are distributed evenly across all the membership functions.

4.5. Detection efficiency

We investigated the distribution of execution time in each stage of BBDP. The experiments show that the traffic reduction stage spent approximately 10% of the total execution time, the DNS detection phase spent approximately 37% of the total execution time, and the TCP detection phase spent approximately 53% of the total execution time. Therefore, it suggests that dispatching the DNS detection phase and the TCP detection phase to multiple servers is able to effectively improve the detection efficiency.

Table II. Detection accuracy of the proposed solution.

	Data set M	Data set B1	Data set B2
Type	Malicious	Benign	Benign
Number of traces	240	695	5
Captured size	3.4 GB	32.4 GB	910 MB
Average reduction rate (%)	75.4	77.3	73.1
Correctly classified	230	671	5
Incorrectly classified	10	24	0
True positive rate (%)	95.83	N/A	N/A
True negative rate (%)	N/A	96.55	100
False negative rate (%)	4.17	N/A	N/A
False positive rate (%)	N/A	3.45	0

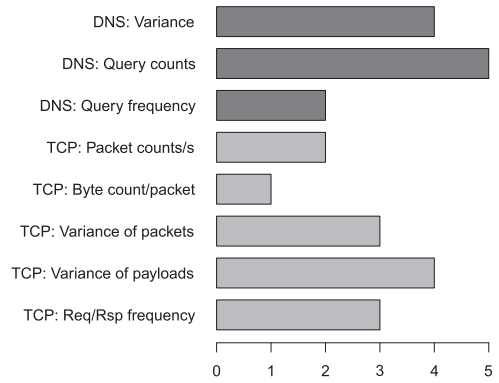


Figure 9. The statistics on the origin of false positives.

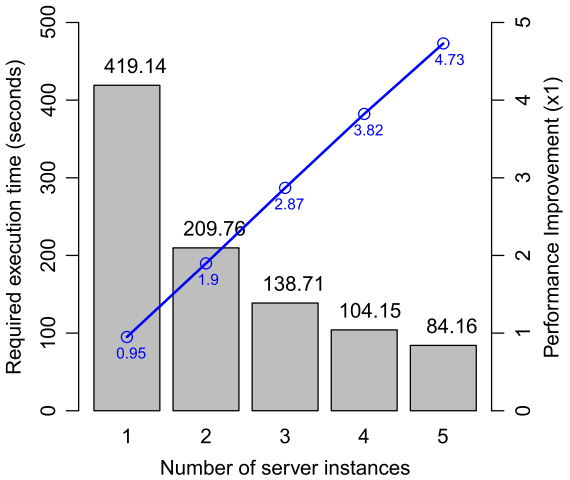


Figure 10. The total execution time and the performance improvements in a cloud-based configuration. We used the Windows Azure cloud service to implement the detection servers.

Figure 10 shows total execution time with respect to number of server instances of the proposed botnet detection system in a cloud-based configuration. We conducted the experiments in both a single-host configuration and a cloud-based configuration. In the single-host configuration, we used a commodity personal computer equipped with an AMD Athlon X2 4000+ (2.1 GHz) dual-core CPU and 2 GB of RAM. In the cloud-based configuration, we varied the number of allocated servers from one to five to evaluate how the performance of the detection load distributed to a various number of servers. Note that we adopted the Windows Azure cloud service to host the detection servers. Each machine is equipped with two 1.6 GHz CPUs and 3.5 GB of RAM. However, it is able to be deployed in any infrastructure as a service architecture. The experimental results show that the total execution time

Table III. Comparison of the proposed solution against other previous works.

	The proposed solution (BBDP)	Park *et al.* [9]	Yu *et al.* [10]	Wang *et al.* [8]
Basic idea	Fuzzy pattern recognition with data partitioning	Static analysis and data-mining approaches	Feature stream	Fuzzy pattern recognition
Evaluated bots	250 IRC+HTTP (real bots)	110 IRC+HTTP (real bots)	4 IRC (self-made bots)	250 IRC+HTTP (real bots)
True positive rate (%)	95.83	94.35	100	90.41
False positive rate (%)	3.45	4.39	14.70	9.59
False negative rate (%)	4.17	5.65	0	5.41
Required execution time (s)	398.1	N/A	N/A	N/A
(with five servers) (s)	(84.2)	N/A	N/A	N/A

BBDP, behavior-based botnet detection in parallel; IRC, Internet Relay Chat; HTTP, hypertext transfer protocol.

can be reduced almost linearly to the number of allocated servers. The single-host configuration requires 398.1 s to finish processing all the 2-h data sets. However, due to extra overheads in the cloud-based configuration, running a single botnet detector under the cloud-based configuration requires 419.14 s. Nevertheless, when there are five servers allocated, the cloud-based configuration requires only 84.16 s, which is 4.73 times faster than the single-host configuration.

4.6. Summary

We finally compared the proposed BBDP against several alternative solutions to detect botnets activities. The comparison is shown in Table III. Although most existing researches evaluated their bot detection solution using self-made or limited number of bot samples, we used 250 real bot samples to evaluate BBDP. The proposed solution performs better than compared solutions except Yu's *et al.* work [10]. However, their work was only evaluated by four self-made bots. Their performance is not known when working with real bots. One special note for Wang's *et al.* work [8] is that, we used exact the same 250 bots to reproduce their experiments. Therefore, the numbers shown in the table is different from that in their paper, which conducted experiments with only 44 live bots. For the detection efficiency in terms of the required detection time, we only show our numbers because we did not have numbers for the other solutions. However, on the basis of the design of the previous algorithms, we believe that it is difficult for those detection algorithms to scale up by using multiple servers.

5. CONCLUSION

In this paper, we presented BBDP. BBDP extends our previous work in terms of detection accuracy and efficiency. The detection accuracy is improved by tuning the membership functions used by the fuzzy pattern recognition approach. In contrast, the detection efficiency is improved by dispatching workloads to multiple servers concurrently. We implemented BBDP on the Windows Azure cloud service and evaluated it using a large number of benign traces (generated from more than 670 hosts) and malicious traces (generated from 240 live bots). Experiments show that BBDP is able to detect more than 95% of bots and only has a false positive rate lower than 3.5%. In addition to good detection accuracy, the implementation shows that the proposed parallel process architecture improves the detection efficiency linearly to the number allocated detection servers. We believe that the demand on scaling out detection servers would be necessary when monitored networks get larger and more complex.

ACKNOWLEDGEMENTS

This research was supported in part by National Science Council under the grants NSC 99-2221-E-009-081-MY3, NSC 102-2219-E-009-012, and NSC 102-2219-E-019-001. We would also like to thank the anonymous reviewers for their valuable and helpful comments.

REFERENCES

1. Roesch M. Snort - Lightweight intrusion detection for networks, *Proceedings of the 13th USENIX Conference on System Administration (LISA'99)*, Seattle, Washington, USA, 1999; 229–238.
2. Paxson V. Bro: a system for detecting network intruders in real-time. *Computer Networks* 1999; **31**(23–24): 2435–2463.
3. Lu W, Tavallaee M, Rammidi G, Ghorbani AA. Botcop: an online botnet traffic classifier, *Proceedings of the 7th IEEE Annual Communications Networks*

and Services Research Conference, Moncton, New Brunswick, Canada, 2009; 70–77.
4. Alserhani F, Akhlaq M, Awan IU, Cullen AJ. Detection of coordinated attacks using alert correlation model, *Proceedings of IEEE International Conference on Progress in Informatics and Computing (PIC)*, Shanghai, China, 2010; 542–546.
5. Szymczyk M. Detecting botnets in computer networks using multi-agent technology, *Proceedings of the 4th International Conference on Dependability of Computer Systems*, Brunow, Poland, 2009; 192–201.
6. Braun L, Munz G, Carle G. Packet sampling for worm and botnet detection in TCP connections, *Proceedings of IEEE Network Operations and Management Symposium (NOMS)*, Osaka, Japan, 2010; 264–271.
7. Sinha P, Boukhtouta A, Belarde VH, Debbabi M. Insights from the analysis of the mariposa botnet, *Proceedings of the 5th IEEE International Conference on Risks and Security of Internet and Systems (CRiSIS)*, Montréal, Québec, Canada, 2010; 1–9.
8. Wang K, Huang CY, Lin SJ, Lin YD. A fuzzy pattern-based filtering algorithm for botnet detection. *Computer Networks* 2011; **55**(15): 3275–3286.
9. Park Y, Zhang Q, Reeves D, Mulukutla V. Antibot: clustering common semantic patterns for bot detection, *Proceedings of the 34th IEEE Annual Computer Software and Applications Conference*, Seoul, Korea, 2010; 262–272.
10. Yu X, Dong X, Yu G, Qin Y, Yue D, Zhao Y. Online botnet detection based on incremental discrete fourier transform. *Journal of Networks* 2010; **5**(5): 568–576.
11. Perdisci R, Ariu D, Giacinto G. Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks* 2013; **57**(2): 487–500.
12. Hsu CH, Huang CY, Chen KT. Fast-flux bot detection in real time, *The 13th International Symposium on Recent Advances in Intrusion Detection (RAID-2010)*, Ottawa, Ontario, Canada, 2010; 464–483.
13. Lin HT, Lin YY, Chiang JW. Genetic-based real-time fast-flux service networks detection. *Computer Networks* 2013; **57**(2): 501–513.
14. Shirley B, Babu L, Mano C. Bot detection evasion: a case study on local-host alert correlation bot detection methods. *Security and Communication Networks* 2012; **5**(12): 1277–1295.
15. Huang CY. Effective bot host detection based on network failure models. *Computer Networks* 2013; **57**(2): 514–525.
16. Lin YD, Chen IW, Lin PC, Chen CS, Hsu CH. On campus beta site: architecture designs, operational experience, and top product defects. *IEEE Communications Magazine* 2010; **48**(12): 83–91.