BotCloud: Detecting Botnets Using MapReduce

Jérôme François, Shaonan Wang, Walter Bronzi, Radu State, Thomas Engel

University of Luxembourg – Interdisciplinary Center for Security, Reliability and Trust
6 rue R. Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg – firstname.lastname@uni.lu

***Abstract*—Botnets are a major threat of the current Internet. Understanding the novel generation of botnets relying on peer-to-peer networks is crucial for mitigating this threat. Nowadays, botnet traffic is mixed with a huge volume of benign traffic due to almost ubiquitous high speed networks. Such networks can be monitored using IP flow records but their forensic analysis form the major computational bottleneck. We propose in this paper a distributed computing framework that leverages a host dependency model and an adapted PageRank [1] algorithm. We report experimental results from an open-source based Hadoop cluster [2] and highlight the performance benefits when using real network traces from an Internet operator.**

I. Introduction

The attacks in Internet and their variety have greatly increased leading to the emergence of defense techniques including firewalls, IDS (Intrusion Detection System), antivirus software... However, recent studies have shown that new attacks are hard to detect [3], [4]. Thus, forensics is required for understanding these attacks and measuring their impacts. This is helpful to recover the system back to a safe state and counter them in the future. From a network point of view, attacks are more distributed. Botnets are one of the most major threat [5] and have evolved from a centralized model towards a decentralized, highly scalable architecture [6] based on peer-to-peer (P2P) networks [7].

Thus, detection and forensics analysis have to be shifted from edges to the core of the network, *i.e.*, the operators (Internet Service Providers - ISP). However, the ISPs have to deal with a huge volume of traffic although fast and efficient analysis is required. Many forensic tools still rely on manual analysis [8]. Hence, new assisted approaches have appeared relying on host dependencies [3], profiling host behaviors [9] or using deep packet inspection [10]. Due to scalability issues in high speed networks, common solutions exclude that and focus exclusively on Netflow [11] data. This is an aggregated view of the network traffic excluding content and, thus, avoid many privacy issues which have to be considered in forensic analysis [9].

Therefore, we propose to detect new generation of botnets from large dataset of Netflow data, such as those gathered by each individual operator. Our previous approach [12] is extended by leveraging cloud computing paradigms especially MapReduce [13] for detecting densely interconnected hosts which are potential botnet members.

A botnet description is given in section II. Our approach is described in III. Section IV explains the host dependency model and the algorithms. Section V is the experimental evaluation. Section VI summarizes related work. Conclusion and future work are included in section VII.

II. Botnets

A botnet is a network of compromised hosts (bots) which are controlled by an attacker also called the botmaster. The botmaster sends commands via a C&C (Command and Control) channel. Although first botnets relied on a central architecture with a core network of few interconnected IRC [14] (Internet Relay Chat) servers, current botnets are based on P2P technologies [15], [7]. In a P2P architecture, each bot acts as a client and a server. Hence, for sending a command to certain bots, other bots are involved. Therefore, P2P bots are well interconnected and that is why we argue in this paper that analyzing interactions between hosts is valuable for detecting botnets.

III. System Overview

A. Cloud computing

Today, storing large volumes of data is possible but analyzing them is still a problem. In our case, 720 millions netflow records (77GB) covering only 23 hours were collected from a major Internet operator in Luxembourg. Hence, distributed computing might be the only viable solution. The main idea of cloud computing is to provide a simple interface to clients who do not want to manage hardware related details such as the resource allocations. The cloud computing service aims to be very scalable and on demand without long delays: computing power should be available instantaneously. In brief, it can be seen as an abstraction layer taking benefit of recent virtualization outcomes in order to provide a simple way for final users to run tasks requiring intensive computing and storage.

The popularity of such services is highlighted for instance by Amazon EC2 [16] which also introduces a new economic model where a cluster of machines can be easily rented. In this way, individuals or companies can take advantage of distributed computing without a huge financial investment.

B. MapReduce

MapReduce [13] is a high-level abstraction of parallel computing introduced by Google. Although traditional approaches need to define exactly the way to carry out the data to process, MapReduce programming model focuses on the processing code. The key idea of the method is to shift the network transfer from the data to the code. In brief, the data is distributed a priori using a distributed file system. For

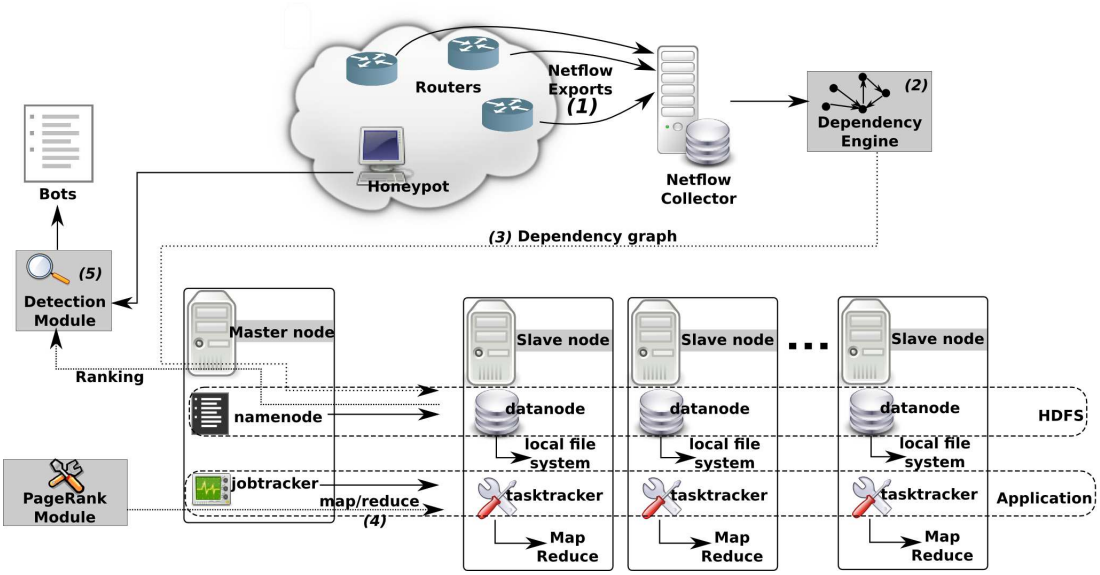


Fig. 1: BotCloud framework

achieving a task, the code is then distributed where the needed data is. Therefore, the cloud computing paradigm is well suited for problem dealing with huge volumes of data.

MapReduce comes from functional programming concepts with functions (*map* and *reduce*) that take other functions as inputs. The *map* aims to divide input data into multiple inputs for applying a function on each of them (mapper). The *reduce* function applied by reducers aggregates the individual results from the mappers. These tasks are attributed by a master machine to slave ones. The master is also responsible to detect node or network failures by a ping mechanism in order to reassign tasks to others nodes.

From a theoretical point of view, the input data is composed of a set of key-value pairs such as (k_i, v_i) and the map function is applied to each of them to produce a list of intermediate key-value pairs:

$$map : (k_i, v_i) \rightarrow list(k'_j, v'_j)$$

There is no relationship between the number of initial and intermediary keys. This intermediate list of key-value pairs represents the intermediary outputs produced by mappers which have to be merged by the reducers:

$$reduce : (k2, list(v_2)) \rightarrow list(k_3, v_3)$$

The input of the reduce function is an intermediate key with a list of all intermediate values generated for this key by all mappers. Therefore, the reducer can generate the aggregated result for each key passed as argument of the reduce function. More details about MapReduce are given in [17].

C. Hadoop

Our botnet detection method is based on Hadoop [2], an open source implementation of MapReduce. A common deployment of an Hadoop cluster is represented in the lower part of figure 1 with a master and slave nodes. The first key component is the Hadoop Distributed File System (HDFS) for storing data. The namenode daemon maintains the file namespace (directory structure, the location of file blocks). However, the blocks are directly stored on the slaves (datanodes) and guarantee a redundancy. Although the master node is the entry point to locate data, the scalability is enforced thanks to direct data transfer between entities (slave machines, user) without being forwarded by the master.

Considering the application side, the jobtracker takes as input a MapReduce job and is responsible to coordinate (task assignment) and monitor the map and reduce tasks. For improving the robustness, the jobtracker and namenode daemons may be executed on different master machines. Unlike the number of reduce tasks, the number of map tasks is automatically determined (the user can only give a hint) regarding the data distribution on HDFS. This is the application of the main MapReduce paradigm which aims to upload the code where the data is. When a node is overloaded, it will transfer data blocks to another which will then run the code. Obviously, the user has not to deal with these aspects which is a strength of the Hadoop.

D. Detection

Figure 1 shows the main steps of our approach using Hadoop. The first step is to gather Netflow records (1) via exports from routers to a collector. Then, the interactions between hosts are analyzed in order to produce a dependency graph (2). This graph is the input to PageRank [18] to figure out hosts which are well interconnected themselves such as within a P2P network. Our first experiments showed that the bottleneck of our approach is the PageRank algorithm which has to iterate many times before exhibiting stable results. Therefore PageRank is executed on Hadoop by distributing the adjacency matrix of the dependency graph among all datanodes (3) before executing map and reduce tasks (4).

The reducers write the scores of the different nodes of the dependency graph into the datanodes and that is why the detection module has to access to them for retrieving all the scores. Based on them, hosts are ranked and bots should be highly ranked. However, normal P2P nodes can also have high rankings and, to distinguish them from bots, we consider that some of them are prior known using a honeypot [19]. In this case, the PageRank algorithm can be tuned (node weight in section IV-B) to increase the ranking of well interconnected hosts (graph partition) containing such hosts. Thus, legitimate P2P traffic can be discarded.

IV. Algorithms

A. Dependency Graph

Netflow [11] is a standard tool for today's large-scale network monitoring. A record represents a series of IP packets sharing the same source, destination address, associated ports, and protocols. A flow collecting architecture consists of probes and collectors. Probes or sensors are devices deployed in different network locations, and are responsible for capturing flow data and forwarding it to the collector. Almost all modern commercial-grade routers support Netflow or an equivalent format export.

Because of its wide availability and intrinsic host communication information, we utilize flow data and link analysis to detect structured botnets. The rationale behind our approach is that P2P structured botnets exhibit a distinguishable communication pattern among bots. Each node in a dependency graph represents a host, an edge pointing from node A to node B indicates that there exists at least one flow which originates from host A and destinates to host B. Hence, bots belong to the same P2P network are linked with either directly or via other bots.

Regarding privacy issues, Netflow records do not include content and our method relies only on host IP addresses which have been rendered anonymous using a reversible one-to-one function only known by the data owner. Thus, an ISP may use an outsourced Hadoop cluster without a high financial investment.

B. PageRank algorithm

The PageRank algorithm [1] is a link analysis algorithm used by the Google web search engine to weight the relative importance of web pages on the Internet. It ranks each web page according to the hyperlink structure among web pages. The importance of a web page is determined by two factors:

1) how many pages contain a hyperlink pointing to it
2) the importance of the pointing pages

Intuitively, pages receiving many links are important, and also pages receiving links from important pages are important.

Considering a dependency graph, an arbitrary initial score is attributed to each node which only impacts the convergence time. At each iteration, the current score of each node is distributed through its outgoing links, and the new ranking score for each node is the sum of distributed ranking scores that all the incoming links bring. The iterations continue until the score changes are sufficiently small. In case of nodes without outgoing links (dangling nodes) we assume the scores are distributed evenly to all nodes. To ensure convergence at primary eigenvalue, extra links are added between each pair of nodes. A certain portion of ranking scores is distributed through original links, and the rest is distributed through the added links. The portion of scores, distributed through the original links, is call damping factor d, and is fixed at 0.5 as an indication of how link structure influence the final rankings. We let the $1-d$ node scores distributed to all the graph nodes according to our subjective preference of each node (a higher node weight is assigned to nodes representing infected honeypots). Assuming n nodes/hosts, $\mathbf{P_t}$ and $\mathbf{W}$ are n-dimensional vectors representing the ranking scores and weights of the nodes, each sums to one, and let $\mathbf{A}$ be the adjacency matrix modified to link dangling pages to all the graph nodes, then the PageRank computation can be formalized as:

$$\mathbf{P_t} = (1-d)\mathbf{W} + d\mathbf{A}^T\mathbf{P_{t-1}} \tag{1}$$

The direction of edges indicates where the ranking scores are distributed. For flow monitoring, when edges point from the source to the destination host, the ranking scores are distributed towards the final destination of traffic, and is named hub rank (hr) in this paper; on the other hand, when pointing from the destination to the source host, the ranking scores are distributed towards the origins and it is named authority rank (ar). Equations (2) and (3) formulate this idea. Note that the adjacency matrix for hub rank and for authority rank are the transpose of each other.

$$hr_t = d\mathbf{A}hr_{(t-1)} + (1-d)\mathbf{W} \tag{2}$$

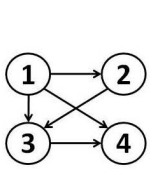
$$ar_t = d\mathbf{A}^T ar_{(t-1)} + (1-d)\mathbf{W} \tag{3}$$

Regarding our context, both scores are relevant because P2P bots act as client and server meanwhile.

C. MapReduce Implementation

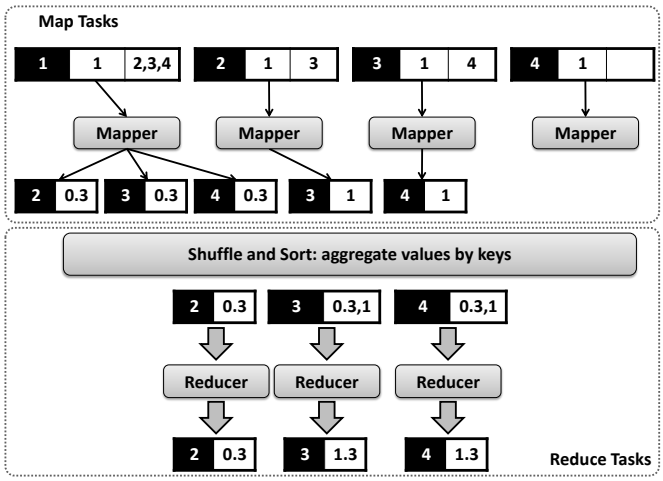
PageRank is well fitted for MapReduce and this section reviews the basics of executing PageRank in the MapReduce context without taking into account the dangling nodes or the damping factor for clarity sake (details can be found in [17]). As highlighted in section III-B, the first step is to divide data into multiple key-value pairs. Like other graph algorithms, a key is a node ID and the corresponding value is the list of adjacent nodes derived from the adjacency matrix $\mathbf{A}$. In our case, each key (node ID) is also associated to the current PageRank score which is initialized at 1 for the toy example in figure 2(a). For each iteration of PageRank, a mapper distributes the current score of a node to its adjacent ones. A reducer is dedicated to sum all scores which were computed for a specific node ID.

This process is illustrated in figure 2(b) for the first iteration. For instance, the first mapper distributes the score of node 1 to nodes 2, 3 and 4. Hence, they get $\frac{1}{3}$ each. Before the



Key	Value	
	Current Score	Adjacency List
1	1	2 3 4
2	1	3
3	1	4
4	1	

(a) Topology representation



(b) MapReduce

Fig. 2: PageRank toy example

reduce step, the results are aggregated and sorted by keys by the *shuffle and sort* phase. Thus, the node 3 has two scores which are processed by a single reducer to obtain a global PageRank score of 1.3. This score is then reused for computing the next iteration including a set of map and reduce tasks again. Figure 2(b) also highlights precisely the border between map and reduce.

From a practical point of view, Hadoop master node assigns the different tasks to slave nodes. Besides, intermediate key-value pairs are just stored locally on mapper nodes which transmit them directly to reducer nodes without being forwarded by the master. Each Hadoop input data split corresponds to several key-value pairs for avoiding wasting too much time to divide data. Furthermore, there are several optimizations such as the local aggregation of intermediate results. Therefore, if a single node computes several scores for the same intermediate key, it only returns the sum of values (as for instance if a slave node deals with keys 1 and 2 in figure 2(b), it has to sum the two values produced for node ID 3). We used the different optimizations provided by the Cloud9 library [20].

V. Evaluation

A. Datasets and methodology

Because no public dataset with labeled botnet traffic at an equivalent level than an ISP is available, our evaluation is based on NetFlow data provided by a major Luxembourg Internet Operator considered as free of botnet as it was checked by a traffic screening solution dedicated to ISP [21].

This dataset involves around 16 million hosts within 720 million of netflow records. The corresponding dependency graph contains 57 millions links. Then, synthetic additional records reflecting the topology of three P2P protocols are added in a similar manner than in [12]: chord [22], kademlia [23] and koorde [24]. A P2P network maps each peer to an ID defined in a huge ID space, as for example with a maximal node ID equal 2^{160}. This paper focuses on structured P2P protocols since they guarantee high performances. Hence, Chord is a pioneering work where the routing has a complexity equals to $log(N)$. Koorde is an extension of Chord with a lower complexity: $O(log(N)/log(log(N))$. Finally, Kademlia was chosen because it is well used in real world for file sharing but also for botnet communications [7].

Although the efficiency evaluation was done using the entire dataset (section V-C), generating synthetic botnet traces needs a lot of computation. Hence, the first experiment (section V-B) about the detection accuracy is based on a subpart of the dataset: 2,133k records and 323k hosts. To strengthen the evaluation, only 1% of IP addresses are used to generate bot traces (stealthy botnet).

Our Hadoop configuration is composed of 11 slave nodes: 5 Intel Core 2 Duo 2.13Gz with 4 GB of memory and 6 Intel Pentium 4 3GHz with 2GB of memory. This small cluster is ideal for testing Hadoop capabilities to cleverly assign the tasks even if the machines are heterogeneous.

B. Botnet detection

Figure 3 shows the ROC (Receiver Operating Characteristic) curves without strengthening initial weight of a prior known bot nodes for computing PageRank. Therefore, all hosts presenting a hub or authority value higher than a threshold are considered as bots. In this figure, this threshold varies for calculating the TPR (True Positive Rate) which is the proportion of bots correctly detected and the FPR (False Positive Rate) which is the proportion of benign hosts detected as bots. Both of these metrics are calculated in terms of number of counted IP addresses. As highlighted, the hub values are better to figure out bots. It means that the client role of P2P bot is more discriminative than the server role. There are also differences between topologies. More precisely, Kademlia botnet detection is very effective because this protocol is fault tolerant by guaranteeing multiple paths for a single destination which is similar to increase the linkage level of nodes measured by PageRank. Chord topology is more compact (without redundant paths) and that is why the detection accuracy is a bit lower. Bots based on these topologies are well detected (close to 100%) with few false positives (3%). Finally, a Koorde-based botnet is the worst to detect since the linkage level of node is very low due to its underlying topology (each node has few connections and shares them with its direct neighbors). However, botnets usually relies on robust overlay like Kademlia since Koorde can be easily disrupted by disconnecting few nodes.

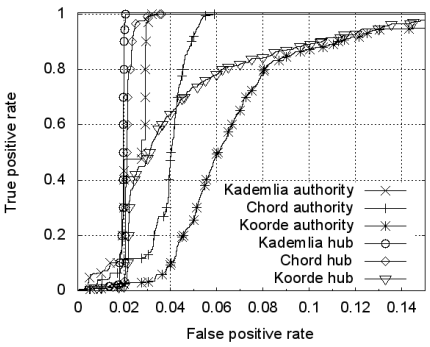
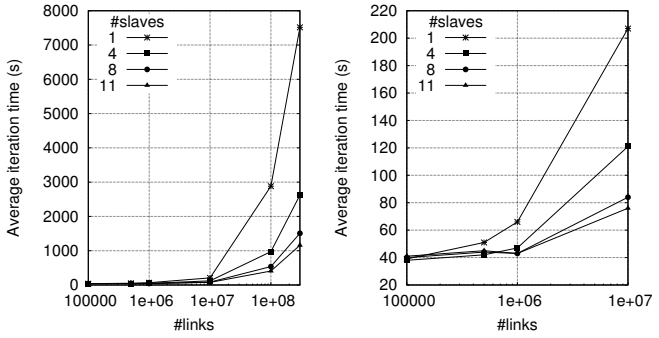


Fig. 3: Botnet detection



(a) All results (b) Zoom in range from 1e+5 to 1e+7

Fig. 4: Average execution for a single PageRank iteration

Knowing initially few bots using honeypot (around 20), the accuracy is improved (TPR = 99% with FPR $< 0.1\%$ knowing initially). However, this paper is not dedicated to accuracy results which are fully covered in [12].

C. Efficiency

We generated different datasets from the original one by extracting a subpart or by adding links in order to test configurations having between 100k and 300 millions of links. Preliminary experiments have shown that the number of nodes has no impact on the execution time. Because scores are propagated through them, the number of intermediate MapReduce key-value pairs is dependent on the number of links. Furthermore, the execution time of a single iteration is approximatively the same for a given configuration because the algorithm do exactly the same process at each iteration.

In figure 4, the configuration of the cluster varies with different numbers of slave nodes. As previously argued, the number of links highly impacts on the execution time. Because the x-axis has a logarithmic scale, the curves seem exponential but they are almost linear with different slopes (except for configurations having few links). For example, the execution time is multiplied by about 6 when the number of links is multiplied by 10 with 8 slaves. Obviously, when more machines are involved for running tasks, the execution time decreases. With few links, the differences are small as highlighted in figure 4(b). In fact, distributing the work using MapReduce entails a lot of additional tasks (data splitting, reduce phase) and also more delays due to network communications. Therefore, for problems with few links and so few computations, MapReduce is useless. Our experiments show that using Hadoop ($\#slaves > 1$) is useful with at least 1 million links (figure 4(b)) and having more than 4 slave machines is efficient with at least 10 millions links which is the case with the original dataset (57 millions). Finally, using all our machines reduces the execution time by 7 comparing to a single machine with a huge dataset since this ratio seems stable for 100 millions links or more.

VI. Related Work

Using NetFlow records were also employed in the past from detecting various attacks including botnets [25]. As well as the whole paper, this section focuses on traffic analysis for botnet detection, which is one of the major threat in Internet.

Since tools and/or datasets of other approaches are not publicly available, only qualitative aspects are considered. By tracking hosts looking for specific DNS names, infected hosts might be detected [26]. For P2P botnets, a honeypot must be active to retrieve infected hosts by crawling the P2P network [27]. Another way to detect bots is to look for malicious activities such as spamming or scanning [28]. Correlating such activities with potential C&C communication patterns may be helpful [29], [30].

Considering our approach, there are also other works considering graphs especially host interactions [31] and complex analytic methods may be employed to detect P2P networks [32], [33]. Such traffic classification is also performed by BLINC [34]. Discovering service or host dependencies may be helpful for network management tasks [35]. BotGrep [36] uses on a random walk technique to discover botnet cluster within an interaction graph. Link analysis is leveraged in our previous work based on flow dependency graphs [37] and host dependency graphs [38] for finding the root cause of attack traffic. This technique is refined in this paper to detect botnets. Although in [12] we focused our evaluation regarding detection performance in different scenarios, this paper is dedicated to show the viability of the approach in a cloud-computing environment by highlighting computational performance benefits and specific use cases where it is more relevant (depending on the underlying data to deal with). Other works, like [39], aim at dividing the traffic to analyze among different intrusion detection systems before correlating the generated alerts. Finally, [40] is the closest relative work since they propose also a cloud-computing based botnet detection. The main difference with our approach is that it is focused on detecting malicious activities, especially spam, to create a correlation graph. Our model is dedicated to botnet C&C channel always used even no malicious activity is observable.

VII. Conclusion

This paper describes a scalable method for detecting P2P botnets regarding the relationships between hosts. Our evalu-

ation shows a good detection accuracy and a good efficiency based on a Hadoop cluster. Our approach is easily usable, at low costs, thanks to existing cloud computing services such that Amazon EC2 [16]. We plan to increase our cluster capacity and to execute all steps including the dependency graph generation on Hadoop. In the future, cooperation between ISP will be also studied and involves many issues, in particular related to exchange anonymous information.

ACKNOWLEDGMENT

This work is partly funded by OUTSMART, a European FP7 project under the Future Internet PPP programme, and by MOVE, a CORE project funded by FNR in Luxembourg.

Note: figures include wikimedia contents (visit http://commons.wikimedia.org for license information).

REFERENCES

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," 1998.

[2] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, June 2009.

[3] W. Wang and T. E. Daniels, "A graph based approach toward network forensics analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 1, pp. 1–33, 2008.

[4] N. Liao, S. Tian, and T. Wang, "Network forensics based on fuzzy logic and expert system," *Computer Communications*, vol. 32, no. 17, pp. 1881–1892, 2009.

[5] A. networks, "Worldwide infrastructure security report (2009 report)," Tech. Rep., 2010.

[6] G. Masters, "Mariposa Botnet Mastermind Nabbed," July 2010. [Online]. Available: http://www.scmagazineus.com/mariposa-botnet-mastermind-nabbed/article/175721/

[7] P. Porras, H. Sadi, and V. Yegneswaran, "A Multi-perspective Analysis of the Storm (Peacomm) Worm." [Online]. Available: http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf

[8] "Safeback," http://www.forensics-intl.com/safeback.html.

[9] J. McHugh, R. McLeod, and V. Nagaonkar, "Passive network forensics: behavioural classification of network hosts based on connection patterns," *SIGOPS*, vol. 42, no. 3, pp. 99–111, 2008.

[10] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg, and J. V. Bokkelen, "Network forensics analysis," *IEEE Internet Computing*, vol. 6, pp. 60–66, 2002.

[11] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954 (Informational), Oct. 2004.

[12] J. François, S. Wang, R. State, and T. Engel, "Bottrack: Tracking botnets using netflow and pagerank," in *Proceedings of IFIP/TC6 Networking*, 2011.

[13] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Symposium on Opearting Systems Design & Implementation (OSDI)*. USENIX Association, 2004.

[14] J. Oikarinen and D. Reed, "rfc 1459: Internet relay chat protocol," United States, 1993.

[15] I. Arce and E. Levy, "An analysis of the slapper worm," *IEEE Security and Privacy*, vol. 1, no. 1, pp. 82–87, 2003.

[16] A. Inc, *Amazon Elastic Compute Cloud (Amazon EC2)*, 2008. [Online]. Available: http://aws.amazon.com/ec2/

[17] J. Lin and C. Dyer, *Data-Intensive Text Processing with MapReduce (Synthesis Lectures on Human Language Technologies)*. Morgan and Claypool Publishers, 2010.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," 1998.

[19] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[20] J. Lin and M. Schatz, "Design patterns for efficient graph algorithms in mapreduce," in *MLG '10: Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. New York, NY, USA: ACM, 2010.

[21] Arbor Networks, "Peakflow SP: Traffic anomaly detection," accessed on 29/03/11. [Online]. Available: http://www.arbornetworks.com/en/peakflow-sp-traffic-anomaly-detection.html

[22] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[23] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS '01: International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 53–65.

[24] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *International workshop on Peer-To-Peer Systems (IPTPS)*, 2003.

[25] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[26] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *ACM SIGCOMM conference on Internet measurement (IMC)*, 2006, pp. 41–52.

[27] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm," in *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–9.

[28] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*. USENIX, 2007.

[29] J. François, G. C. M. Moura, and A. Pras, "Cleaning your house first: Shifting the paradigm on how to secure networks," in *International Conference on Autonomous Infrastructure, Management, and Security (AIMS)*, Springer, Ed., Nancy, France, June 2011.

[30] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *USENIX Security Symposium (SS)*, San Jose, CA, July 2008, pp. 139–154.

[31] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in *Recent Advances in Intrusion Detection (RAID)*, 2007.

[32] M. Iliofotou, M. Faloutsos, and M. Mitzenmacher, "Exploiting dynamicity in graph-based traffic analysis: techniques and applications," in *ACM International conference on Emerging networking experiments and technologies (CoNEXT)*, 2009, pp. 241–252.

[33] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *ACM CoNEXT Conference*, 2008.

[34] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: multilevel traffic classification in the dark," in *ACM Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2005.

[35] S. Kandula, R. Chandra, and D. Katabi, "What's going on?: learning communication rules in edge networks," *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pp. 87–98, 2008.

[36] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding p2p bots with structured graph analysis," *Proceedings of the 4th International Conference on Autonomous Infrastructure, Management and Security (USENIX Security Symposium)*, 2010.

[37] S. Wang, R. State, M. Ourdane, and T. Engel, "Mining netflow records for critical network activities," *Proceedings of the 4th International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, 2010.

[38] ——, "Riskrank: Security risk ranking for ip flow records," *Proceedings of the 6th International Conference on Network and Services Management (CNSM 2010)*, 2010.

[39] Z. Li, Y. Chen, and A. Beach, "Towards scalable and robust distributed intrusion alert fusion with good load balancing," in *ACM SIGCOMM workshop on Large-scale attack defense*.

[40] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum, "Botgraph: large scale spamming botnet detection," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. USENIX Association, 2009.