J Comput Virol Hack Tech (2015) 11:247–261
DOI 10.1007/s11416-015-0250-2

ORIGINAL PAPER

BotSpot: fast graph based identification of structured P2P bots

Bharath Venkatesh[1] · **Sudip Hazra Choudhury**[1] · **Shishir Nagaraja**[2] · **N. Balakrishnan**[1]

Received: 8 March 2015 / Accepted: 10 August 2015 / Published online: 2 September 2015

Abstract An essential component of a botnet is the Command and Control (C2) channel (a network). The mechanics of C2 establishment often involve the use of structured overlay techniques which create a scaffolding for sophisticated coordinated activities. However, it can also be used as a point of detection because of their distinct communication patterns. Achieving this is a needle-in-a-haystack search problem across distributed vantage points. The search technique must be *efficient* given the high traffic throughput of modern core routers. In this paper, we focus on efficient algorithms for C2 channel detection. Experimental results on real Internet traffic traces from an ISP's backbone network indicate that our techniques, (i) have time complexity linear in the volume of traffic, (ii) have high F-measure, and (iii) are *robust* to the partial visibility arising from partial deployment of monitoring systems, and measurement inaccuracies arising from partial visibility and dynamics of background traffic.

✉ N. Balakrishnan
balki@serc.iisc.in

Bharath Venkatesh
bharath@ssl.serc.iisc.in

Sudip Hazra Choudhury
sudip@ssl.serc.iisc.in

Shishir Nagaraja
s.nagaraja@lancaster.ac.uk

[1] Supercomputer Education Research Centre, Indian Institute of Science, Bangalore 560012, Karnataka, India

[2] School of Computing and Communications, Lancaster University, Balrigg, Lancaster LA1 4WA, UK

1 Introduction

Botnets are networks of compromised computers (called 'zombies' or 'bots'). They are distributed attack platforms, and have been used to send SPAM, launch Distributed Denial of Service (DDoS) attack and perform click fraud [40]. An essential component of the botnet lifecycle is the establishment and use of a Command and Control Channel (C2) between the botmaster and the zombies. Our goal is to propose *efficient* techniques to detect C2 channels based on *structured Peer-to-Peer* (*P2P*) *architectures*. Most early botnet C2 designers adopted a centralized topology where control was provided by one or more C2 servers. These servers were easy target points for dismantling the channel. Recent C2 designs mostly adopt P2P topologies [36] and thus have become more difficult to dismantle. P2P topologies can be structured or unstructured. Unstructured designs use flooding, random walks or gossip protocols for routing. Structured P2P designs rely on Distributed Hash Tables (DHT) for routing, and have been shown to be a robust and efficient way for the attacker to coordinate attack activity [14]. They enable the attacker to control millions of malware instances across different locations, and are able to handle the typical churn associated with hosts being powered on and off. The Storm, Conficker and the current TDL-4 botnet are examples that use structured P2P designs. The TDL-4 botnet had infected about 4.5 million hosts worldwide [20].

Botnet Detection has been an active area of research (Sect. 5). Most existing methods of botnet C2 Detection rely on computing statistical features of flow/packet traffic or in some cases even deep packet inspection. Such techniques require the comparison of each traffic flow to all the others in order to isolate C2 traffic, thus *limiting their efficiency*. Such techniques are also deficient in that attackers can evade detection by making minor changes in C2 flow statistics to

present a moving target for botnet defenses. For instance, by the use of variable length encryption or changes in packet structure leads to new behavioural characteristics.

An alternate approach is to use graph analysis. We demonstrate that this is fundamentally more efficient than flow clustering approaches since it avoids the need to cross compare flows across the dataset. Graph analysis offers *three key benefits*. First, it permits *collection efficiency* – the communication graph $G(V, E)$ with the vertices as hosts and edges as flows, can be constructed in linear time by parsing only the source and destination addresses at the Internet Protocol (IP) header. In the context of backbone networks, such a compressed representation of traffic data consumes much lesser computational and storage resources relative to the existing methods. Second, we devise fast ($O(|E|)$) detection algorithms that consume this relatively compressed representation. Third, and most important, graph analysis leverages the *fundamental* properties of structured P2P botnets. C2 mechanisms must be robust to node take-downs, to remain functional. This demands relatively high internal connectivity among the peer nodes. Another important feature of structured P2P subgraphs is their assortative nature, where nodes of similar degree (number of connections) connect to each other. Assortativity makes botnets robust and resistant to takedown [47]. This is in sharp contrast to benign traffic, which does not show both assortativity in degree and higher internal connectivity.

P2P C2 leads to distinguishable topological features in the communication graph. BotGrep [32] works on such a graph constructed from network traffic, and uses the topological properties of botnet command and control (C2) communication graphs to separate them from benign traffic. BotGrep exploits spatial relationships in traffic graphs to a greater extent than other graph based botnet detection methods [9,17,24,25,29] (Sect. 5). BotGrep relied on the fast-mixing nature of structured P2P subgraphs relative to benign traffic in order to detect them. BotGrep is tested on synthetic botnet topologies superimposed on a graph constructed from real world backbone traces, and is found to give high accuracy on the datasets tested.

Main contribution We exploit the relative difference in assortativity and density properties to detect P2P C2. Compared with prior art, we achieve an $O(log|V|)$ improvement in theoretical time complexity (and smaller constants) over BotGrep while maintaining its accuracy. Our method (BotSpot) relies on an efficient local search optimization method to identify dense subgraphs, followed by a filtering phase to detect assortative graphs using a novel measure. The proposed method is evaluated on synthetic structured P2P topologies embedded in background graphs generated from real world network traces and found to be comparable to prior art in terms of accuracy, while being much faster in execution.

The rest of the paper is organized as follows. Section 2 describes the system architecture of BotSpot and the approach it follows. Section 3 describes the implementation details of the algorithm. Section 4 contains a comprehensive evaluation of the algorithm, its efficiency, accuracy and discusses its resilience to evasion. Section 5 contains a brief survey of the related work.

2 System architecture and approach

The aim of the work is to detect hosts that are a part of structured P2P C2 by analysing an IP-IP graph. The architecture of such a detector is illustrated in Fig. 1.

The IP-IP graph $G(V, E)$ considered in this work is constructed by flows or packets, the vertex set V is the set of IP addresses, and the edge set E contains pairs of vertices (V_1, V_2) if there has been any communication (packet sent/received) of any kind between V_1 or V_2. The directionality is also ignored, and the presence of an edge implies that either V_1 or V_2 could be the source. The protocol and the size or number of the packets/flows are ignored to obtain a unweighted graph, making any method that relies on this graph robust against evasion by randomizing protocols, ports and statistical flow features. The IP-IP graph will contain a set of hosts S that are a part of a structured P2P C2 subgraph. Each structured P2P Botnet will be a subgraph $G_i^S(S_i, E_i^S)$ of $G(V, E)$, such that $S_i \subseteq V$; $S = \bigcup_i S_i$ and $E_i^S \subseteq E$. The remaining vertices (hosts that do not participate in structured P2P C2) of the graph are referred to as N. There can be many connections among the set of hosts who participate in structured P2P traffic, S to the set N. This is because each host in S also participates in web and mail traffic.

BotSpot takes a IP-IP graph as input outputs a set of hosts $C \subseteq V$ that are a part of a structured P2P subgraph. It does this by filtering out hosts that are a part of set N. The output set C closely matches the set S. It has a low rate of false positives, i.e. hosts from the set N that are wrongly classified as Structured P2P traffic. BotSpot is complimentary to traffic classification approaches such as anomaly or misuse detection techniques [46] that differentiate between structured P2P botnets and legitimate applications that use structured P2P topologies. BotSpot scales the application of traffic classification techniques by significantly reducing the input data volume. It generates efficiently and with high Recall, a set of hosts that participate in Structured P2P C2.

2.1 Approach

BotSpot exploits the differences in the degree assortativity and density between S and N traffic, especially the behaviour

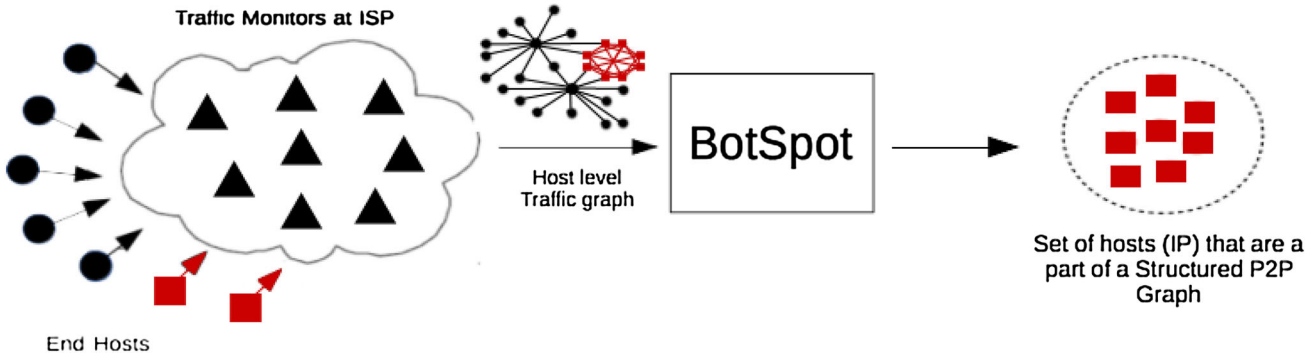


Fig. 1 System architecture

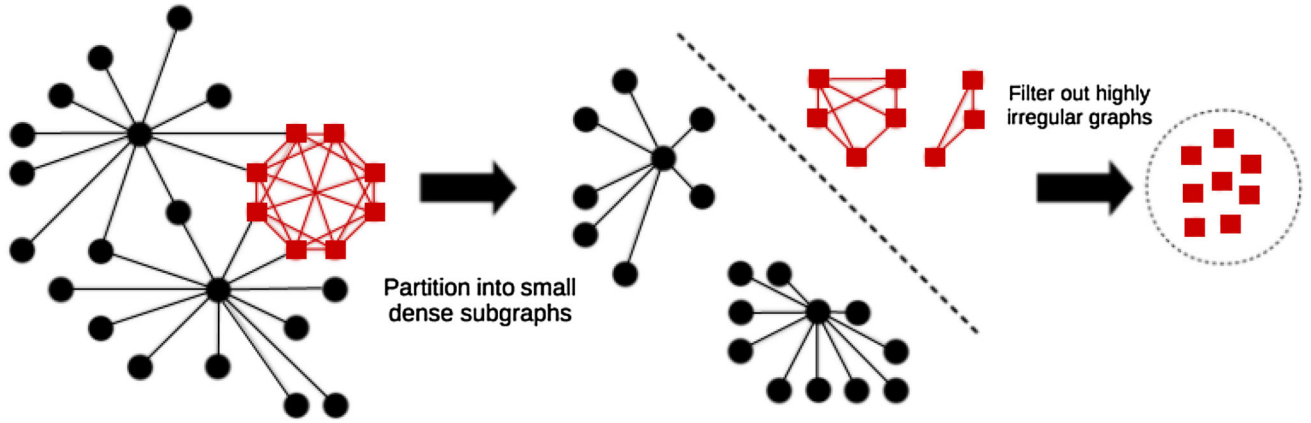


Fig. 2 BotSpot Algorithm overview: the hosts represented by the squares participate in a structured P2P C2

exhibited by each subgraph when the IP-IP graph is partitioned into dense subgraphs. A novel measure Mean Regular Degree that captures both properties assortativity and density is defined and used to effectively filter out N hosts. This is illustrated in Fig. 2.

2.1.1 Degree assortativity

The degree assortativity of the network measures the propensity of a vertex to connect to vertices which have degree (number of connections) similar to its own. Hosts that are a part of S have a higher assortativity than hosts in set N.

Structured P2P subgraphs are regular graphs i.e. all vertices have the same degree. Hosts that are a part of a structured P2P network typically maintain a fixed number of hosts in their routing tables. Thus a structured P2P graph exhibits a high degree of assortativity as all nodes connect to other nodes with the same degree. This is illustrated by the red coloured nodes in the first graph in Fig. 2.

On the other hand, hosts in the set N rarely connect to other hosts in the set N with similar degree and exhibit disassortative mixing. IP-IP graphs exhibit a skewed or a power-law degree distribution. Such a distribution is characterised by a set of hubs (popular web, email and content delivery servers etc.) which account for the most number of connections. The hubs rarely connect to each other, and low degree end hosts or clients connect preferentially to these hubs resembling a connected set of "star" graphs similar to black coloured nodes in the first graph in Fig. 2.

2.1.2 Edge density

The edge density for any graph is defined as the ratio of the number of edges of the graph to the number of edges in a complete graph. Structured P2P Graphs typically have a higher number of edges when compared to a client server IP-IP graph of the same number of nodes. These differences can be clearly observed by comparing the subgraph of red nodes (Structured P2P Traffic) and the subgraph of black nodes in Fig. 2.

The edge density is an important property of a structured P2P subgraph as it controls the robustness of the P2P network. Structured P2P graphs will have a high value of density as they have to be robust against node failures by adding redundant paths between hosts.

For IP-IP subgraphs induced by the set N, majorly consisting of client-server traffic, each client connects to a set of servers. As these are not explicitly designed to be robust against hosts going down, such a graph will have a fewer

number of edges relative to a structured P2P graph of the same size.

2.1.3 Partitioning into small and dense subgraphs

As bot-infected hosts exhibit both C2 traffic as well as benign traffic, IP-IP graphs have structured P2P subgraphs deeply embedded among hosts in N. One approach would be to employ Community Detection Algorithms (CDA) to partition the graph to isolate structured P2P subgraphs as communities.

CDA aim to partition a graph into densely connected subgraphs or communities [16]. CDA have been used successfully in both social and biological networks.

While density is one feature of structured P2P subgraphs, a direct application of CDA may not always yield structured P2P C2 subgraphs as communities. This is mainly because they are not separated by a small cut (edges) from the rest of the graph, a requirement for effective Community Detection. However they can be used to generate homogeneous subgraphs (S hosts only or N hosts only). Such subgraphs can then be filtered out and combined to output the set C.

A large number of CDA exist, several of them have been surveyed in [11,16,37]. Most methods aim at maximising partition quality functions which are defined in several ways. A popular family of partition quality functions $Q(P)$ [44] are defined based on internal weight w and the total volume v of the nodes in the community and are given in Eq. 1.

$$Q(P) = \sum_{C \in P} f(w(C), v(C)) \tag{1}$$

$$w(C) = \sum_{i,j \in C} \frac{A_{ij}}{2|E|}$$

$$v(C) = \sum_{i \in C} \frac{d_i}{2|E|}$$

where A_{ij} is the boolean adjacency matrix of the graph, the degree of each node $d_i = \sum_{j \in V} A_{ij}$ and $P = \{C_i \subset V\}_{i=1}^{k}$ is a partition of the vertex set into k subsets V, such that $V = \bigcup C_i$ and $C_i \cap C_j = \phi \; \forall i, j$. f is a function of $w(C)$ and $v(C)$, and assigns a quality score to a single cluster.

A good community is characterised by the majority of the volume v being accounted for by the internal weight w.

Modularity [34] is a widely used partition quality function in literature. It compares the observed number of internal edges of a community to a rewired random graph, that preserves the degree sequence of the original graph. It is defined as

$$Q_{Modularity}(P) = \sum_{C \in P} \frac{1}{2|E|} \sum_{i,j \in E} \left(A_{ij} - \left(\frac{d_i d_j}{2|E|} \right) \right)$$
$$= \sum_{C \in P} w(C) - (v(C))^2 \tag{2}$$

For a weighted graph, the same formulae apply, with A_{ij} replaced by a real valued matrix W_{ij} and the degree of a node is given by $d_i = \sum_{j \in V} W_{ij}$.

Algorithm 1: *LocalSearchOptimization*

```
Input : The graph G, Objective Function Q
Output: The partition P of G into communities
begin
    initialize P such that each node is in its own community;
    Q_prev ⟵ Q(P);
    Q_current ⟵ ∞;
    while Q_current > Q_prev do
        for node i ∈ V do
            MaxGain ⟵ 0;
            for node j adjacent to i do
                ΔQ(i, j) ⟵= gain in Q(P) when node i is
                moved to node j's community;
                MaxGain ⟵ max(MaxGain, ΔQ(i, j));
            end
            move the node i to that community that results in max
            gain;
            Q_prev = Q_current;
        end
    end
end
```

In this work, we focus on a local search method (Algorithm 1) that forms the basis of the Louvain method [6], which has been used to partition graphs of millions of vertices and hundreds and millions of edges. Local search algorithms aim to do a greedy maximization of partition quality functions. The algorithm initially assigns each node to its own community. Then iteratively every node is moved to the community that results in the largest gain in the objective function. The time complexity of the algorithm is decided by the computation of the gain in objective function. Local Search Optimization can optimize the family of objective functions in Eq. 1 including Modularity.

In a recent paper, VanLaarhoven and Marchiori [44] introduce a new objective function $Q_{w-log-v}$. This objective function is defined as

$$Q_{w-log-v}(P) = -\sum_{C \in P} w(C) log(v(c)) \tag{3}$$

The objective function in Eq. 3 prefers smaller and denser communities relative to $Q_{Modularity}$, when optimized by Local Search Optimization. This is because of the higher penalty on the volume of the community owing to the use of the log function in $Q_{w-log-v}$.

When this objective function is optimized in order to detect communities, each structured P2P C2 will fragment into smaller connected subgraphs. The high degree nodes (hubs) tend to form small communities dragging in their immediate neighbourhood consisting of the adjacent nodes.

The vertices adjacent to hubs may be part of S or N. However, the vertices which are a part of S will behave differently from nodes in N. The N nodes will attach to the hubs forming star-like communities. The subgraph corresponding to such communities will be disassortative as well as relatively sparse.

The S hosts will be moved by the algorithm to a community with other similar nodes owing to the high internal connectivity and will resist being moved into the community of the hub. At the same time, the hub will resist joining the community of the S nodes as it will tend to increase the total degree of the community owing to its high degree. Therefore it will be placed by the algorithm in a different community.

Due to the symmetry associated with the near regular nature of the whole structured P2P graph, the structured P2P C2 will tend to break into subgraphs that are also near regular. This makes the subgraphs of the structured P2P C2 assortative and relatively denser than star-like communities.

2.1.4 Mean regular degree

In order to differentiate between the small but homogeneous communities of S hosts and communities of hosts in N that result after the partitioning, we define a measure, *mean regular degree* or m_{reg} in this paper. It captures the mean number of connections of the node to other nodes with similar degrees.

$$m_{reg}(G) = \frac{1}{|V|} \sum_{i,j \in E} \frac{1}{1 + |d_i - d_j|} \tag{4}$$

The above measure m_{reg} accounts for the assortativity of a graph with the denominator term that depends on the difference of the degrees. This measure is also dependent to the total degree of the community, accounting for the density. The time taken to evaluate this metric is $O(|E|)$ as each edge is considered only once during the computation, allowing it to be computed efficiently.

2.1.5 Properties of mean regular degree

For a k-regular graph (degree of each node is k), the value of

$$m_{reg}^{k-regular} = \frac{1}{|V|} \sum_{i,j \in E} 1 = \frac{2|E|}{|V|} = \frac{k|V|}{|V|} = k$$

Thus a ring graph ($k = 2$) of any size will have $m_{reg} = 2$ and a clique or a complete graph of size $|V|$ will have $m_{reg} = |V|$. Thus the value of m_{reg} increases as the degree of each node increases.

A star graph of size $|V|$ has one vertex of degree $|V| - 1$ and $|V| - 1$ vertices of degree 1. For each vertex, the absolute difference in degrees is $|V| - 1 - 1 = |V| - 2$. Thus for a star graph with $|V| > 1$, the value of

$$m_{reg}^{star} = \frac{1}{|V|} \sum_{i,j \in E} \frac{1}{1 + |V| - 2} = \frac{2(|V| - 1)}{|V|(|V| - 1)} = \frac{2}{|V|}$$

Thus for a star graph, the value of m_{reg} decreases as the size of the graph increases.

A dyad (two nodes with a single edge) which is both a 1-regular graph as well as a star graph of 2 nodes will have $m_{reg} = 1$. A graph that is larger than 2 nodes and is star-like will have small values of $m_{reg} < 1$.

The next section details our proposed BotSpot algorithm which uses these ideas.

3 BotSpot

BotSpot consists of two stages. In the first stage, communities are obtained by greedy optimization of $Q_{w-log-v}$ followed by the discarding of N communities using m_{reg}. In the second stage, the communities retained from the first stage are collapsed to form a weighted graph, and greedy modularity optimization is carried out followed by another filtering of communities in order to obtain a final set of nodes C that are a part of a structured P2P C2 (Fig. 3).

3.1 Stage 1

The stage 1 of the proposed algorithm runs the greedy optimization of $Q_{w-log-v}$ (Eq. 3). This will result in homogeneous communities. The m_{reg} for each community is obtained by extracting its corresponding subgraph and computing it according to Eq. 4. A filtering of the communities with $m_{reg} > 1$ at this point would result in poor Recall. This is because some communities of S hosts may be too small. These communities would thus be indistinguishable from other communities of N hosts owing to values of $m_{reg} < 1$. Therefore another chance has to be given for these communities to merge with other bot communities to increase the m_{reg} value. Communities with m_{reg} less than the median value of m_{reg} are selected for the next stage. This median value will be less than 1 as the number of structured P2P hosts are less than 50 % of the total nodes of the graph. The median value of m_{reg} is preferred over the mean value over all community subgraphs as the sizes of the communities are skewed, thus the mean m_{reg} will be affected strongly by the larger and more regular communities. Stage 1 is outlined in Algorithm 2.

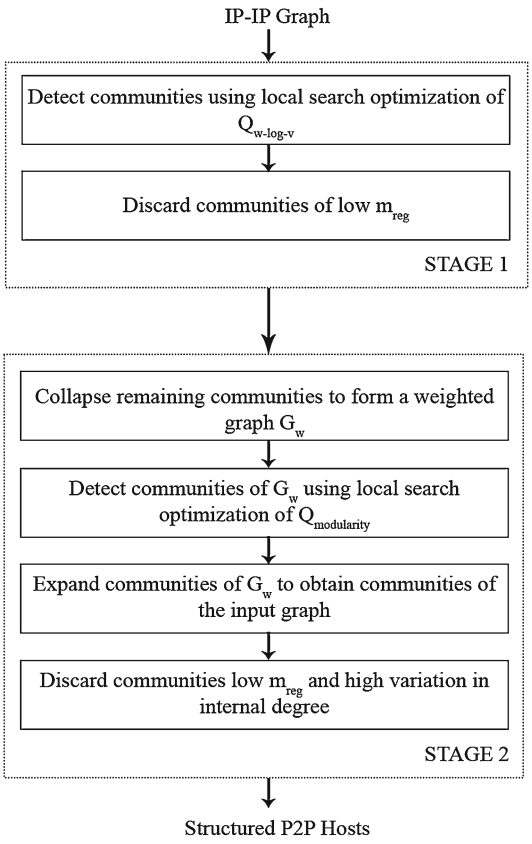


Fig. 3 The BotSpot Algorithm

Algorithm 2: BotSpot - Stage 1

Input : The graph G
Output: A set of selected candidate bot communities $P_{selected}$
begin
  $P \leftarrow LocalSearchOptimization(G, Q_{w-log-v})$;
  for *community* $C_i \in P$ **do**
    $G_{C_i} \longleftarrow SubGraph(G, C_i)$;
    $m_{C_i} \longleftarrow m_{reg}(G_{C_i})$;
  end
  $m_{med} \longleftarrow Median(m_{C_1}, m_{C_2} ... m_{C_k})$;
  $P_{selected} \longleftarrow \{C_i : m_{C_i}) > m_{med}\}$
end

3.2 Stage 2

In order to allow some small S communities to merge, the local search optimization of $Q_{Modularity}$ is run on a weighted graph.

The weighted graph $G_w(V^{P_{selected}}, W)$ consists of the node set $V^{P_{selected}} = \{C_i : C_i \in P_{selected}\}$. The set W consists of weighted edges with $W_{C_iC_j} = \sum_{i \in C_i, j \in C_j} A_{ij}$.

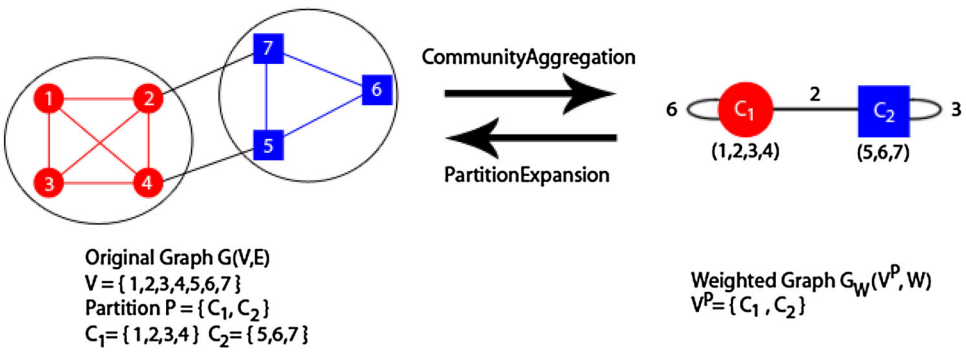
Thus the weighted graph is obtained by collapsing the set of communities $P_{selected}$ obtained from Stage 1, similar to the process in [6].

The edge weights between any two communities C_i and C_j are the number of edges between a node in community C_i and a node in community C_j. Each node also has a self loop, with edge weight equal to the number of edges among nodes in the same community. This process is illustrated in a toy example in Fig. 4.

Here $Q_{Modularity}$ is used as the objective function as it allows for larger communities than $Q_{w-log-v}$ (Eq. 3). The aggressive merging due to optimization of modularity is reduced by the removal of many hubs in Stage 1, which connect together several communities and play a role in the merging of communities. The communities thus obtained are for the weighted graph, and are converted back to communities of the original graph by expanding each node of the weighted graph.

Finally the filtering step needs to be applied to separate out communities that are not very regular. Communities of bots, being assortative and denser will have values of $m_{reg} > 1$ (Eq. 4), whereas the star like communities of N nodes will have values of $m_{reg} < 1$ making it a good rule for filtering out these communities. The communities with $m_{reg} \leq 1$ can be discarded at this stage. There can be cases of certain non-regular communities which may have a value of $m_{reg} \geq 1$, this is because m_{reg} is proportional to the internal degree of the community as well. Structured P2P C2 are near regular communities. Thus they have $\sigma_{deg}(G_C) > \mu_{deg}(G_C)$ and are discarded by the algorithm. Stage 2 is outlined in Algorithm 3.

Fig. 4 Creation of weighted graph



Algorithm 3: BotSpot - Stage 2

Input : The graph G,the set of selected communities from Stage1 $P_{selected}$
Output: A set C_{final} consisting of structured P2P botnet nodes
begin
  $G_w \longleftarrow CommunityAggregation(G, P_{selected})$;
  $P_w \leftarrow LocalSearchOptimization(G_w, Q_{Modularity})$;
  $P \longleftarrow PartitionExpansion(G, G_w, P_w)$;
  for *community* $C_i \in P$ **do**
    $G_{C_i} \longleftarrow SubGraph(G, C_i)$;
    if $m_{reg}(G_{C_i}) > 1$ *and* $\sigma_{deg}(C_i) < \mu_{deg}(C_i)$ **then**
      Add all nodes in C_i to C_{final};
    end
  end
end

The overall time complexity of the algorithm remains linear in the number of edges, and thus is efficient and applicable to detect structured P2P botnets. This results in an improvement of $O(log|V|)$ over the time complexity of BotGrep which scales as $O(|E|log|V|)$ for sparse graphs.

4 Evaluation

4.1 Dataset

Due to a lack of labelled botnet network trace data, especially at the infrastructure level, BotSpot is tested on graphs constructed from real world traffic traces of different durations with synthetic botnet topologies embedded in them. The background, botnet graph construction and embedding are done according to the procedure followed by [32].

The description of this dataset generation process is provided in this section.

4.1.1 Background traffic traces

The first set of background traffic data - the Abilene dataset consists of NetFlow traces captured at three core routers of the Internet2 ISP located at Washington D.C (WA) and Chicago IL (CH). The data was captured over 1 day (1D) on the 1st of December 2008 and has no payload information. We also extract a 1 hour (1H) subtrace of both of these datasets to evaluate the impact of the density of the background graph on the accuracy of the algorithm.

The second set is a larger set of packet traces captured at a OC192 (10 Gbps) Internet Point of Presence (PoP). The data was captured for 1 h between 13:00 and 14:00 on the 17th of February 2011 on a backbone link between Chicago (CA-CH) and San Jose City (CA-SJ) as a part of the 2011 Anonymized Internet Traces Dataset offered by CAIDA [45] and contains no payload information.

4.1.2 Background graphs

The IP-IP graph is constructed from only the information extracted from the Internet Protocol (IP) layer of the network traces. Only the source address and destination address fields of each packet of the trace are parsed. The nodes are IP addresses, and there is an edge added between nodes A and B if there is a packet sent from an IP address A to IP Address B or vice-versa, i.e. the directionality and the number of packets, number of bytes, the port numbers etc. are ignored to create an undirected and unweighted graph with no self loops and no multiple edges. The following Table 1 contains the details of the graphs extracted from the network traces.

4.1.3 Botnet graphs and embedding

Since the focus is on modelling and exploiting the connectivity features of only the Command and Control (C2) communication of a structured P2P botnets we generate graphs of popular Distributed Hash Tables (DHT) - KADEMLIA (KAD) [31], CHORD (CHO) [42] and KOORDE (KOO) [28].

A single botnet graph $G^S(S, E^S)$ is embedded in one of the background graphs to create a single dataset. The botnet graphs are embedded in the background graph by mapping each node to a node in the background uniformly at random. Then the edges in the botnet graph are added to the edges of the background graph for all the nodes in order to create the final superimposed graph. Thus each bot node will also have edges to other benign (N) nodes in addition to other bots. We embed two different sizes (1000, 10000) nodes for each DHT in order to examine the effect of botnet size on the accuracy of the algorithm.

4.2 Metrics

Consider C, the set of detected hosts output from BotSpot and S, the set of bot nodes from ground truth. BotSpot is validated on the datasets and standard metrics, Precision, Recall and FScore are computed to compare the sets C and S.

Table 1 Properties of graphs extracted from the network traces

Background graph	$\|V\|$	$\|E\|$
WA-1H	119203	967002
CH-1H	206390	1639800
WA-1D	217504	5188343
CH-1D	297732	12156830
CA-CH-1H	2716915	8867080
CA-SJ-1H	8427335	28109401

$$P = \frac{|S \cap C|}{|C|} \tag{5}$$

Precision (P) represents the purity of each community by considering the fraction of botnet nodes to the total nodes in the community.

$$R = \frac{|S \cap C|}{|S|} \tag{6}$$

Recall (R) quantifies the detection rate, counting the total fraction of bots identified.

$$F = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{7}$$

FScore (F) is the harmonic mean of Precision and Recall and summarises both values. As a benchmark, BotGrep [32] is implemented and tested on the above datasets, the parameter values are as per those specified in the original reference. All experiments have been performed on a machine with 32-core AMD Opteron 2.4 GHz based processor with 128 GB of RAM, only a single core is used.

4.3 Accuracy

In this section the accuracy of the proposed method is compared to BotGrep in the case of the Abilene Datasets and are provided in Table 2a, b. Each dataset is labeled as $< DHT > - < Size >$, i.e. CHO-1000 represents a 1000 node CHORD DHT. The implementation of BotGrep was not able to provide results for the CAIDA datasets in reasonable time (24 h) in our execution environment. The results obtained using the proposed method are provided in Table 3a, b. In most cases, the proposed method results in FScores of over 0.85 and outperforms BotGrep. The Recall is always above 0.9, indicating a good detection rate. For very small botnets embedded in larger graphs, such as CHO-10000 and KOO-10000, the Precision brings down the FScore (0.7, 0.58 respectively). This is due to the fact that BotSpot relies on the relative density of the structured P2P subgraphs. If the structured P2P subgraph is too small relative to the whole graph, then number of false positives tend to increase. However the Precision is still around 0.5, allowing efficient refinement. This is possible as the false positives are of the order of the size of the structured P2P subgraphs and not the size of the entire IP-IP graph.

BotGrep is a two stage method consisting of the prefiltering and refinement stages [32]. The Recall is dependent on the first stage, and the main aim of the much slower second stage is to remove false positives. In our experiments, we found that the first stage sometimes results in very poor Recall, high Precision and low FScore. In some other cases the Recall is very high but at the cost of Precision. The second stage predominantly and consistently improves the Precision and hence is more effective in the later cases. This has resulted in FScore assigning the cup sometimes to one and sometimes to the other. Though the Precision of BotSpot is lower than BotGrep, as can seen from Table 2 that the two are close to each other. Further, the second stage of filtering techniques of BotGrep can also be used to improve the results further. However, this will add to the computation time.

Table 2 Performance comparison of BotSpot and BotGrep on Abilene Trace Graphs: Precision, Recall and FScore

Botnet	%Bots	BotGrep			BotSpot		
		P	R	F	P	R	F
(a) WA-1D							
CHO-1000	0.46	1.00	0.77	0.87	0.87	0.94	0.9
CHO-10000	4.60	0.98	0.79	0.79	0.97	0.92	0.94
KOO-1000	0.46	1.00	0.61	0.76	0.96	0.94	0.95
KOO-10000	4.60	1.00	0.76	0.86	0.98	0.90	0.94
KAD-1000	0.46	1.00	0.77	0.87	0.96	0.94	0.95
KAD-10000	4.60	0.98	0.80	0.88	0.97	0.93	0.95
(b) CH-1D							
CHO-1000	0.34	1.00	0.96	0.98	0.70	0.86	0.77
CHO-10000	3.36	1.00	0.35	0.52	0.97	0.87	0.92
KOO-1000	0.34	1.00	0.97	0.98	0.85	0.87	0.86
KOO-10000	3.36	1.00	0.40	0.57	0.95	0.80	0.87
KAD-1000	0.34	1.00	0.54	0.7	0.74	0.88	0.87
KAD-10000	3.36	1.00	0.43	0.6	0.97	0.89	0.93

Table 3 Performance of BotSpot on CAIDA Trace Graphs: Precision, Recall and FScore

Botnet	%Bots	BotSpot		
		P	R	F
(a) CA-CH-1H				
CHO-10000	0.37	0.94	0.93	0.93
CHO-100000	3.7	1	0.92	0.96
KOO-10000	0.37	0.9	0.95	0.92
KOO-100000	3.7	0.97	0.94	0.96
KAD-10000	0.37	0.62	0.94	0.75
KAD-100000	3.7	0.95	0.93	0.94
(b) CA-SJ-1H				
CHO-10000	0.12	0.56	0.95	0.7
CHO-100000	1.2	0.97	0.95	0.96
KOO-10000	0.12	0.42	0.94	0.58
KOO-100000	1.2	0.83	0.93	0.88
KAD-10000	0.12	0.83	0.93	0.88
KAD-100000	1.2	1	0.92	0.96

Table 4 Performance of BotSpot under conditions of partial visibility (40 % botnet edges removed) on Abilene Trace Graphs

Botnet	%Bots	BotSpot		
		P	R	F
(a) WA-1D				
CHO-1000	0.46	0.96	0.80	0.87
CHO-10000	4.60	0.99	0.77	0.87
KOO-1000	0.46	0.97	0.85	0.91
KOO-10000	4.60	0.99	0.83	0.90
KAD-1000	0.46	0.85	0.81	0.83
KAD-10000	4.60	0.98	0.88	0.93
(b) CH-1D				
CHO-1000	0.34	0.76	0.62	0.68
CHO-10000	3.36	0.98	0.78	0.87
KOO-1000	0.34	0.70	0.74	0.72
KOO-10000	3.36	0.94	0.64	0.76
KAD-1000	0.34	0.62	0.51	0.56
KAD-10000	3.36	0.94	0.64	0.76

Table 5 Performance of BotSpot under conditions of partial visibility (40 % botnet edges removed) on CAIDA Trace Graphs

Botnet	%Bots	BotSpot		
		P	R	F
(a) CA-CH-1H				
CHO-10000	0.37	0.77	0.88	0.82
CHO-100000	3.7	0.85	0.93	0.89
KOO-10000	0.37	0.74	0.78	0.76
KOO-100000	3.7	0.93	0.81	0.87
KAD-10000	0.37	0.78	0.92	0.84
KAD-100000	3.7	0.86	0.95	0.9
(b) CA-SJ-1H				
CHO-10000	0.12	0.17	0.94	0.28
CHO-100000	1.12	0.61	0.94	0.74
KOO-10000	0.12	0.16	0.91	0.28
KOO-100000	1.2	0.61	0.9	0.72
KAD-10000	0.12	0.17	0.95	0.28
KAD-100000	1.2	0.61	0.95	0.74

4.3.1 Partial visibility

This section evaluates the robustness of BotSpot under conditions of partial visibility arising from partial deployment of monitoring systems.

According to [32], it was found that 60 % of inter-bot communication could be observed at the level of Tier-1 ISPs. This was concluded by using a list of IP Addresses of Storm botnet affected hosts obtained by the authors. With this information, in this section, we test the performance of BotSpot if only Tier-1 ISP's cooperate.

The dataset we use is the unaltered background graphs of the Abilene and CAIDA datasets described in Sect. 4.1, with the difference that we remove 40 % of the botnet edges for each botnet. It must be noted that community detection is the basis for BotSpot, and the lack of visibility of 40 % of the botnet edges will vastly reduce the density of the botnet, making detection harder.

Tables 4 and 5 show the Precision, Recall and FScore for the Abilene and CAIDA datasets. It can be observed that BotSpot is still able to achieve an FScore above 0.7 in most cases.

4.3.2 Multiple botnets

In the earlier experiments, BotSpot was tested with a single instance of a botnet. In a real setting multiple botnets exist, thus in this part, we demonstrate the performance of BotSpot on graphs where three different botnets have been embedded. The background datasets used are the Abilene WASH-1D, CHIC-1D traces and the CAIDA CH-1H, SJ-1H traces. In each of the four background graphs, three botnets - KAD, KOO and CHORD of same size are embedded, retaining only 60 % of the botnet edges so as to make them harder to detect.

Table 6 shows the results obtained from this experiment. It can be seen that BotSpot can detect multiple botnets with high Recall (>0.7). The Precision can be adjusted by increasing the minimum requirement of *mean regular degree* to be >2 in the final filtering step. This is left for future work.

4.4 Scalability and efficiency

The runtime for the proposed method is dominated by the first call to $GreedyOptimizationQ_{w-log-v}$ (Algorithm 3), which takes $O(|E|)$ time [6]. In order to test the running time of the proposed method and compare it with BotGrep, we consider the Abilene background datasets. We embed a CHORD graph of size 1000 in all cases to profile the running time of the method. BotGrep consists of two stages - prefiltering and refinement, in some cases the first stage of $O(E)$ complexity, results in a valid partition causing the algorithm to terminate before the more expensive $O(ElogV)$ refinement stage, resulting in non-monotonic scaling behavior. This can be observed in the time vs graph size curve for BotGrep, where there is an outlier at $E = 1639800$ for the CH-1H dataset

In Fig. 5, the runtime is plotted against the size of the graph. It can be clearly observed from the figure that the proposed method scales significantly better that BotGrep, being about 350 times faster in graphs with more than 10 million edges. The proposed method was able to handle the largest

Table 6 Performance of BotSpot under conditions of partial visibility (40 % botnet edges removed) with 3 botnets of total 30000 nodes

Dataset	%Bots	BotSpot		
		P	R	F
WA-1D-CHO-KOO-KAD-3000	1.61	0.98	0.75	0.85
CH-1D-CHO-KOO-KAD-3000	1.18	0.92	0.67	0.77
CAIDA-CH-CHO-KOO-KAD-30000	1.29	0.88	0.75	0.81
CAIDA-SJ-CHO-KOO-KAD-30000	0.42	0.37	0.8	0.5

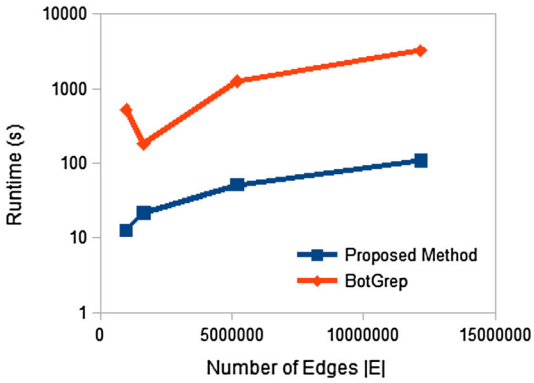


Fig. 5 Runtime comparison of proposed method and BotGrep: Abilene 1 Day Traces and 1000 Node CHORD Botnet

graph (8 million nodes, and 27 million undirected edges) in under 20 min on a single core, which was shorter than the duration of the trace.

4.5 Evasion

Any intrusion detection system/anomaly detection system is subject to several evasion strategies. Stinson and Mitchell [41] characterise several such evasion strategies that can be adopted by botnets. Zhang et al. [49] discuss some evasion strategies specific to P2P botnets. We discuss the adoption of such strategies in order to evade BotSpot.

4.5.1 Encryption

Many signature based botnet or intrusion detection systems are easily defeated by encrypting the communication channel. BotSpot, however does not use payload based features, and hence cannot be defeated by encryption.

4.5.2 Limiting botnet activity

This strategy aims at spacing out attacks or other *malicious* activity carried out by the bots, which can be done so as to beat detector thresholds, BotSpot relies only the Command and Control (C2) traffic and not on attack traffic. Even sleeper botnets with no exhibited attacks can be isolated.

BotSpot does not depend on the number of packets exchanged between hosts, and thus is unaffected by reducing the number/size of packets exchanged between the bots.

Any modifications to the C2 by introducing churn for example can be detrimental to the botnet as there would be routing failures, data inconsistencies and complete disruption of the overlay [49].

4.5.3 Flow perturbation and randomizing communication behaviour

Most detection techniques rely on statistical features obtained from the flow. Payload based features include character distributions. Non payload based features include bytes per flow (bpf), packets per flow (ppf) and other similar features. These can be randomized by the botnet to fool detection systems trained on these features. BotSpot only relies on the presence of communication, not on the volume or rate of communication, and thus is immune to flow perturbation attacks.

4.5.4 Addition of spurious connections

The main feature exploited by BotSpot is the *regular structure* of the botnet C2 topology and attempts can be made to evade detection. As discussed earlier, sparsifying the topology by limiting botnet activity can be detrimental to the botnet, but the botmaster may try and evade detection by having bots make spurious connections among themselves or benign nodes. Further the connections can be made such they are purely random (beacon traffic) or made in such a way as to resemble a benign host (strategic cover traffic). A benign host will normally connect with higher probability to other popular benign hosts (such as Google), which have a high degree or a large amount of incoming connections. This results in four possible scenarios:

- spurious connections to other bots to resemble benign traffic.
- spurious connections to other bots randomly.
- spurious connections to benign hosts to resemble benign traffic.
- spurious connections to benign hosts randomly.

In order to test the method for evasion, we simulate the four possible scenarios of spurious addition of edges according to the above models. We use the WASH-1D dataset as the background, and a 10000 node CHORD botnet. In each case, the edges added per each bot is increased, and the effect of it on Precision and Recall is recorded and plotted in Figs. 6 and 7. In each figure both the addition of random beacon traffic and strategic cover traffic are analysed. The probability of adding an edge as random beacon traffic to a node i in the target set S is according to the Erdos–Renyi (ER) model [15] and is given by

$$p_i = \frac{1}{|S|}$$

The probability of adding an edge as cover traffic to a node i in the target set S is according to the Albert-Barabasi preferential attachment (AB) model [3] and is given by

$$p_i = \frac{d_i}{\sum_{i \in S} d_i}$$

The addition of edges directly proportional to the degree of the target host models Internet traffic, where a few hosts will get the most connections.

Scenario 1 and 2 - Spurious connections within the botnet In these scenarios, the botmaster can attempt to introduce spurious connections so as to get the assortativity of botnet to match that of benign traffic. This can be done by changing the number of connections each bot makes to other bots by preferential addition or random addition. In this case a random addition of edges does not reduce the assortativity of the botnet as the degree of each bot in expectation will remain the same. This can be noticed in the Fig. 6a, b which show that the edge addition by the ER model does not bring about any change in the performance of BotSpot. However in the case of preferentially connecting bots so as to bring about a skewed degree distribution does change the assortativity of the ring. This is confirmed in Fig. 6a, b, where the Recall of BotSpot is affected on an addition of a large amount of spurious edges that are connected according to the AB model. However BotSpot is able to tolerate the addition of $4N$ connections, where N is the size of the Botnet. Addition of many connections within the botnet will make it dense allowing community detection algorithms to isolate them. Furthermore it amounts to an increase in communication volume and may make detection easier for other traffic statistic based methods.

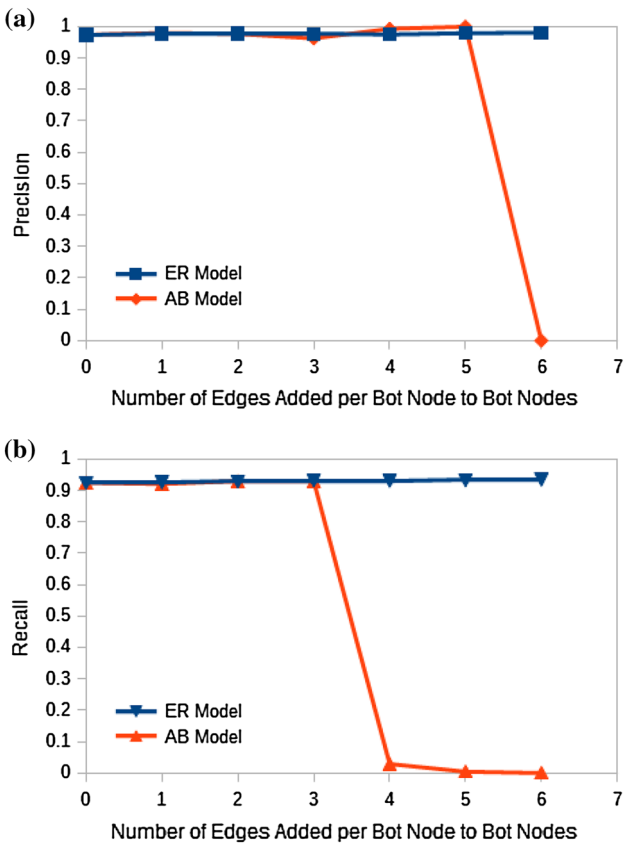


Fig. 6 Evasion resilience of BotSpot to addition of spurious within Botnet edges

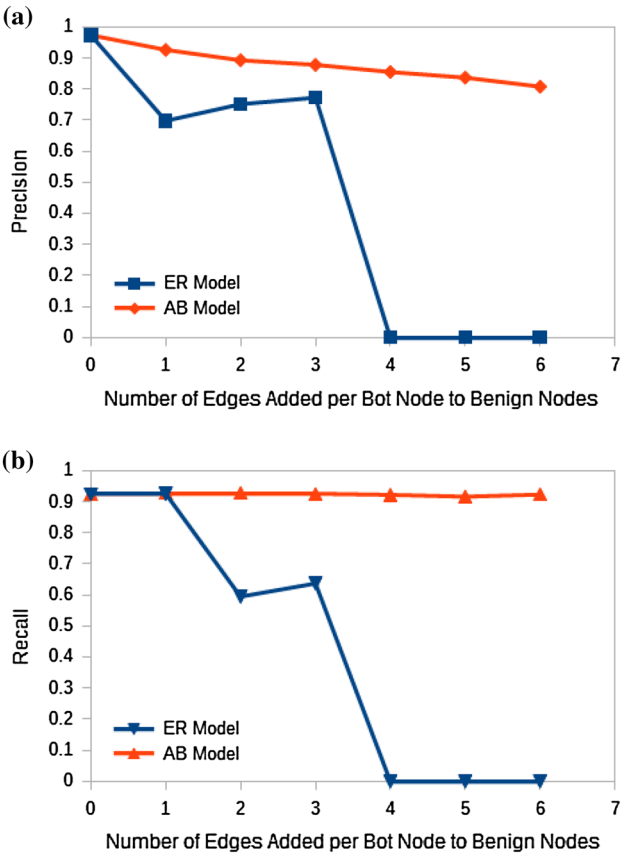


Fig. 7 Evasion resilience of BotSpot to addition of spurious edges to Benign Nodes from each Bot

Scenario 3 and 4 - Spurious connections to benign hosts In these scenarios the botmaster can attempt to increase the number of connections from bots to benign nodes to blend in with benign traffic. Doing this will reduce the relative density of connections among the botnet with respect to the observed connection density among normal traffic. The bots can again make connections preferentially towards high degree hosts (AB Model) or at random to a benign host (ER Model).

In the case of preferential connections to high-degree benign hosts (AB Model), it can be observed in Fig. 7a, b that BotSpot is unaffected by such an attempt as there is no change in the Recall of the algorithm and it takes about $6N$ edges to reduce the Precision to 0.8.

In the case where connections are made at random to a benign host (ER Model), it can be observed in Fig. 7a, b, that the performance of the algorithm - both Precision and Recall is affected severely. This is because the assortativity of the bots is reduced. However in a realistic setting, it will be difficult for a botnet to truly connect to a benign node at random without generating a lot of scan traffic. This scan traffic can be picked up by other detection systems.

4.5.5 Blending with existing DHT based P2P traffic

The botnet may function as a part of a larger DHT-based P2P network as in the case of the Storm botnet. BotSpot will detect the entire P2P network and it has to be clubbed with existing detection systems such as [46] to further isolate the botnet. As discussed in [32], our method can be combined with misuse detection methods at the edge level to build a distributed detection system. The large reduction of traffic offered by BotSpot can make these more accurate but slower methods more feasible to use.

4.5.6 Anonymisation using Tor

Tor hidden services can be used to build a structured DHT. In such a case, a direct application of BotSpot is not possible, however the complexity of the system increases for the botmaster. However, strategies like [5] allow locating such hidden service requests by setting up detector Tor nodes. Once these nodes are set up, BotSpot could be run with suitable modifications on a similar graph extracted from data collected by the detectors.

5 Related work

Host based methods operate similar to anti-virus systems and detect activities of the bot in the host system. Network based methods rely on features obtained by passive monitoring of network traffic. Network based approaches are the most popular owing to the relative ease of deployment. A variety of techniques from different areas have been applied for network based detection of botnets. Traffic mining, clustering, correlation, entropy analysis, stochastic modeling, time series analysis and other machine learning based techniques have been proposed in literature and surveyed in [19,40]. A classification and a brief description of the methods follow.

Honeynet based Detection

Early methods of botnet detection and analysis [1,10,18] relied on the use of honeypots and honeynets. A honeypot is a set of hosts that are made intentionally vulnerable, so as to attract attackers. Honeypots get infected with various malware, and since they are under the control of the security community, they can be used to infiltrate a botnet, and study it. Honeynets are networks of honeypots, to observe the network behaviour of malware, and can be used to detect botnets in a small scale.

DNS based detection

Many centralized botnets need to rely on the Domain Name Service (DNS) in order to obtain the IP address of the C2 servers. Some botnet detection approaches aim to detect temporally correlated and periodic DNS queries [8,13]. Other approaches focus on detecting hosts that query domain names which receive Non-Existent Domain (NXDOMAIN) responses [39]. Recent approaches deal with Domain Generating Algorithms(DGA) used by Botnets, and aim at identifying domains generated by these DGA's [2,33,38]. P2P botnets may not rely on the use of the DNS, and hence cannot be detected by these approaches.

Correlation of various lifecycle behaviours

A bot exhibits several stages in it's lifecycle - infection, egg download, bootstrapping/communicating with the botmaster, scanning and propagation, and attack. All of these stages can be detected separately from network flow data and the correlation of the detector outputs can be correlated in order to detect botnet flows. This approach is followed by [4,21,22,26]. These methods typically detect anomalies using payload level features and are thus ill suited for large scale detection.

Group activity and similarity

Bots that are a part of the same botnet will similar observable characteristics in terms of packet content, velocity and size, similar response time. Packet content similarity approaches include work by [23,30]. Gu et al. have proposed BotSniffer [23] use payload n-gram features for content similarity, and also perform a spatio-temporal correlation of flows in order to detect groups of similarly behaving hosts. Lu and Ghorbani [30] propose a method of clustering similar hosts based on flows that have similar payload character frequencies. These methods rely on deep packet inspection, and cannot scale to high traffic volume. Strayer et al. [43] train machine learning classifiers based on packet summary features and packet velocity(such as packets per second, bits

per second, bytes per packet) which are relatively faster to collect, but prone to randomization.

P2P Botnet Detection

Detection of P2P traffic poses several challenges, P2P networks normally employ strong encryption, rendering use of payload based methods difficult, and use random port numbers, including ports reserved for well known services and try to blend in with normal traffic in order to avoid detection [40]. The methods described in this section detect C2 flows.

Jiang et al. [27] have proposed a method to detect P2P botnets by discovering flow dependencies in C2 traffic. Dependencies of flows are extracted by identifying pairs of flows that occur together many times in a given observation time, and the extracted two-level dependencies are used to obtain higher level dependencies by combining flows. Flows are then clustered based on the Jaccard similarity of the extracted flow dependencies. This approach scales quadratically with the number of flows, and thus is not a scalable technique. Zhang et al. [48] have described a method to detect stealthy P2P botnets. Their approach is a multi-stage approach which rely on reduction of flows by retaining flows of nodes which exhibit many failed outgoing connections, followed by clustering using flow based features, followed by filtering on the basis of temporal persistence, finally relying on the overlap of peers among the nodes in the clusters and traffic similarities to identify P2P bots. A parallelized more scalable version is built up in [49]. BotSpot is faster than this approach in detecting structured P2P traffic as it relies only on the IP-IP graph, and does not have to look at DNS traffic or maintain state to detect failed connections, making collection of traffic easier.

Rahbarinia et al. propose PeerRush [35], a technique that aims to not only detect but also characterize P2P traffic with very high accuracies in a two step process. The first step detects P2P traffic, and the second step uses machine learning classifiers to characterize the traffic. Though the technique does not rely on deep packet inspection, the P2P detection component requires detection, computation and storage of state of failed connections and correlation of connections with DNS queries similar to [49]. BotSpot can be used as the P2P traffic detector in PeerRush.

Graph based P2P Botnet Detection

There have been some graph-based P2P Botnet Detection methods proposed in literature. Collins and Reiter [9] propose a method to detect worms by noting that scanning behaviour initiated by worm infected hosts would tend to connect different disconnected components of protocol specific traffic graphs. Graption [25] is a graph-based method that identifies P2P flows by calculating the in-degree to out-degree ratio of hosts in protocol traffic graphs. This method can be defeated by protocol randomization.

Bot-Track, proposed by Francois et al. [17] is a method that works on a directed traffic graph constructed from NetFlow traces, and computes the hub and authority centrality of each host, and clusters hosts based on these values using the DBSCAN algorithm. However, similarly to previous works, BotTrack rely mostly on the degree of nodes and does not exploit group connectivity patterns. Coskun et al. [12] have described a graph-based method to identify other members of an unstructured P2P botnet within a network, when given a known bot. A mutual contacts graph is constructed, and a dye diffusion from the source node is simulated. Other members of the same botnet are identified by thresholding on the final dye concentrations on the nodes. Hang et al. [24] use community detection based clustering to identify long-lived low intensity flows. Li et al. [29] have proposed a method of detecting P2P botnet communities using Latent Dirichlet Allocation (LDA). Their approach relies only on the who talks to who graph, and can handle overlapping botnets. However their method requires a prior filtering approach to remove non P2P communities, and cross validation to estimate the number of groups and thus difficult to employ in the backbone. However both [24] and [29] approaches rely on only edge density and does not utilize the topological properties associated with structured P2P botnets. Our approach is more scalable than the current work in field owing to the reduced input as well as the inference algorithm.

Our work needs only the source IP address and destination IP address from a given packet. It does not even need the port and protocol information, thus very efficient hardware collectors can be built to collect this at wire speed without sampling. Further our work uses a very efficient graph clustering strategy based on the recent advances of community detection algorithms. The method is fast, scaling linearly in the order of the number of edges of the graph owing to the use of local search optimization.

6 Conclusion and future work

In this paper, BotSpot, a graph based method is developed to detect nodes that are a part of a structured P2P botnet given a traffic graph. BotSpot relies on an efficient local search optimization method to identify dense subgraphs, followed by a filtering phase to detect assortative graphs using a novel measure which can also be computed efficiently. The proposed method is validated, and found to be comparable in performance with BotGrep. The runtime of the algorithm is found to be 300 times lower than BotGrep on graphs of tens of millions of edges. and thus a suitable technique for dealing with backbone scale traffic. This work also shows how primitives from the state of the art in community detection literature can be adopted and modified to suit different requirements, such as discovering assortative subgraphs effectively and efficiently. Future extensions of the work include:

Parallelization This work can be easily optimized for multi-core execution with a strategy described in [7], that proposes a synchronous version of local search optimization.

Local and Distributed Computation There exist several local community detection algorithms, which aim to detect the community by expansion, given a node. Thus with a seed list of suspected bots, the local community detection algorithm can be modified to expand the seed set according to m_{reg}. This can be easily distributed among graph collector nodes, and can even be used to detect overlapping structured P2P subgraphs.

Unstructured P2P subgraphs This work demonstrates the generic nature of community detection algorithms, thus appropriate topological discriminative features of unstructured P2P botnets can replace the features of structured P2P botnets.

Augmenting the Traffic Graph The unweighted traffic graphs used for analysis can be augmented with other robust features derived from traffic which will serve as edge weights for density based partitioning.

References

1. Abu Rajab, M., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: MC '06 Proceedings of the 6th ACM SIGCOMM on Internet Measurement, pp. 41–52. New York, New York, USA, Oct. 2006. ACM Press. ISBN 1595935614. doi:10.1145/1177080.1177086. http://dl.acm.org/citation.cfm?id=1177080.1177086. http://portal.acm.org/citation.cfm?doid=1177080.1177086
2. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou II N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: detecting the rise of dga-based malware. In: USENIX Security Symposium, pp. 491–506. (2012)
3. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
4. Binkley, J.R., Singh, S.: An algorithm for anomaly-based botnet detection. In: Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI), pp. 43–48 (2006)
5. Biryukov, A., Pustogarov, I., Weinmann, R.: Trawling for tor hidden services: Detection, measurement, deanonymization. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 80–94. IEEE (2013)
6. Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10):6 (2008). ISSN 1742-5468. doi:10.1088/1742-5468/2008/10/P10008. http://stacks.iop.org/1742-5468/2008/i=10/a=P10008?key=crossref.46968f6ec61eb8f907a760be1c5ace52. arXiv:0803.0476
7. Browet, A., Absil, P.-A., Van Dooren, P.: Fast community detection using local neighbourhood search (2013). arXiv preprint. arXiv:1308.6276
8. Choi, H., Lee, H., Kim, H.: Botgad: detecting botnets by capturing group activities in network traffic. In: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, pp. 2–10. ACM (2009)
9. Collins, M.P., Reiter, M.K.: Hit-list worm detection and bot identification in large networks using protocol graphs. In: RAID'07 Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection, pp. 276–295. (2007). ISBN 3-540-74319-7, 978-3-540-74319-4. http://dl.acm.org/citation.cfm?id=1776434.1776456
10. Cooke, E., Jahanian, F., McPherson, D.: The Zombie roundup: understanding, detecting, and disrupting botnets. In: SRUTI'05 Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop, pp. 6–12 (2005). http://dl.acm.org/citation.cfm?id=1251282.1251288
11. Coscia, M., Giannotti, F., Pedreschi, D.: A classification for community discovery methods in complex networks. Stat. Anal. Data Min. ASA Data Sci. J. **4**(5), 512–546 (2011)
12. Coskun, B., Dietrich, S., Memon, N.: Friends of an enemy. In: ACSAC '10 Proceedings of the 26th Annual Computer Security Applications Annual Conference, pp. 131–140, New York, New York, USA, Dec. 2010. ACM Press. ISBN 9781450301336. doi:10.1145/1920261.1920283. http://dl.acm.org/citation.cfm?id=1920261.1920283
13. Dagon, D., Gu, G., Lee, C.P., Lee, W.: A taxonomy of botnet structures. In: ACSAC '07 Proceedings of the 23rd Annual Computer Security Applications Annual Conference, pp. 325–339. IEEE, Dec. 2007. ISBN 0-7695-3060-5. doi:10.1109/ACSAC.2007.44. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4413000. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4413000
14. Davis, C.R., Neville, S., Fernandez, J.M., Robert, J.-M., Mchugh, J.: Structured peer-to-peer overlay networks: ideal botnets command and control infrastructures? In: Jajodia, S., Lopez, J. (eds.) ESORICS '08 Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security. Lecture Notes in Computer Science, vol. 5283, pp. 461–480, Berlin, Heidelberg, Oct. 2008. Springer, Berlin. ISBN 978-3-540-88312-8. doi:10.1007/978-3-540-88313-5. http://dl.acm.org/citation.cfm?id=1462455.1462495. http://www.springerlink.com/index/10.1007/978-3-540-88313-5
15. Erdos, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci. **5**, 17–61 (1960)
16. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174, (2010). ISSN 03701573. doi:10.1016/j.physrep.2009.11.002. http://linkinghub.elsevier.com/retrieve/pii/S0370157309002841
17. François, J., Wang, S., State, R., Engel, T.: BotTrack: tracking botnets using NetFlow and PageRank. In: NETWORKING '11 Proceedings of the 10th International IFIP TC 6 Conference on Networking, pp. 1–14, May 2011. ISBN 978-3-642-20756-3. doi:10.1007/978-3-642-20757-0_1. http://dl.acm.org/citation.cfm?id=2008780.2008782. http://hal.inria.fr/docs/00/61/35/97/PDF/networking11CR.pdf
18. Freiling, F.C., Holz, T., Wicherski, G.: Botnet tracking: exploring a root-cause methodology to prevent distributed denial-of-service attacks. In: Vimercati, S.D.C., Syverson, P., Gollmann, D. (eds.) ESORICS'05 Proceedings of the 10th European Conference on Research in Computer Security. Lecture Notes in Computer Science, vol. 3679, pp. 319–335. Berlin, Heidelberg, Sept. 2005. Springer, Berlin. ISBN 978-3-540-28963-0. doi:10.1007/11555827. http://dl.acm.org/citation.cfm?id=2156732.2156751
19. Gardiner, J., Cova, M., Nagaraja, S.: Command & control: understanding, denying and detecting (2014). arXiv preprint. arXiv:1408.1136
20. Golovanov, S., Soumenkov, I.: TDL4-Top Bot (2011). http://securelist.com/analysis/36152/tdl4-top-bot/
21. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: detecting malware infection through IDS-driven dialog correlation. In: USENIX Security '07 Proceedings of the 16th USENIX Secu-

rity Symposium, pp. 1–16. Aug. 2007. ISBN 111-333-5555-77-9. http://dl.acm.org/citation.cfm?id=1362903.1362915
22. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: USENIX Security '08 Proceedings of the 17th USENIX Security Symposium, pp. 139–154, July 2008. http://dl.acm.org/citation.cfm?id=1496711.1496721
23. Gu, G., Zhang, J., Lee, W.: BotSniffer: Detecting botnet command and control channels in network traffic. In: NDSS '08 Proceedings of the 15th Annual Network and Distributed System Security Symposium, pp. 1–18. Citeseer (2008). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.8092&rep=rep1&type=pdf. http://users.csc.tntech.edu/~weberle/Fall2008/CSC6910/Papers/17_botsniffer_detecting_botnet.pdf
24. Hang, H., Wei, X., Faloutsos, M., Eliassi-Rad, T.: Entelecheia: detecting p2p botnets in their waiting stage. In: IFIP Networking Conference, 2013, pp. 1–9. IEEE (2013)
25. Iliofotou, M., Kim, H.-C., Faloutsos, M., Mitzenmacher, M., Pappu, P., Varghese, G.: Graption: a graph-based P2P traffic classification framework for the internet backbone. Comput. Netw. **55**(8):1909–1920 (2011). ISSN 13891286. doi:10.1016/j.comnet.2011.01.020. http://dl.acm.org/citation.cfm?id=1982705.1983058
26. Jaikumar, P., Kak, A.C.: A graph-theoretic framework for isolating botnets in a network. Secur. Commun. Netw. (2012). ISSN 19390114. doi:10.1002/sec.500. http://onlinelibrary.wiley.com/doi/10.1002/sec.500/full. http://doi.wiley.com/10.1002/sec.500
27. Jiang, H., Shao, X.: Detecting P2P botnets by discovering flow dependency in C&C traffic. Peer-to-Peer Netw. Appl. (2012). ISSN 1936-6442. doi:10.1007/s12083-012-0150-x. http://www.springerlink.com/index/10.1007/s12083-012-0150-x
28. Kaashoek, M.F., Karger, D.R.: Koorde: a simple degree-optimal distributed hash table. In: Peer-to-Peer Systems II, pp. 98–107. Springer, Berlin (2003)
29. Li, L., Mathur, S., Coskun, B.: Gangs of the internet: towards automatic discovery of peer-to-peer communities. In: 2013 IEEE Conference on Communications and Network Security (CNS), pp. 64–72. IEEE (2013)
30. Lu, W., Tavallaee, M., Ghorbani, A.A.: Automatic discovery of botnet communities on large-scale communication networks. In: ASIACCS '09 Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, pp. 1–10, New York, New York, USA, Mar. 2009. ACM Press. ISBN 9781605583945. doi:10.1145/1533057.1533062. http://dl.acm.org/citation.cfm?id=1533057.1533062
31. Maymounkov, P., Mazieres, D.: Kademlia: a peer-to-peer information system based on the xor metric. In: Peer-to-Peer Systems, pp. 53–65. Springer, Berlin (2002)
32. Nagaraja, S., Mittal, P., Hong, C.-Y., Caesar, M., Borisov, N.: BotGrep: finding P2P bots with structured graph analysis. In: USENIX Security'10 Proceedings of the 19th USENIX Security Symposium, p. 7, Aug. 2010. ISBN 888-7-6666-5555-4. doi:10.1.1.172.8756. http://static.usenix.org/event/sec10/tech/full_papers/Nagaraja.pdf
33. Nelms, T., Perdisci, R., Ahamad, M.: Execscent: mining for new c&c domains in live networks with adaptive control protocol templates. In: USENIX Security, pp. 589–604 (2013)
34. Newman, M.E.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
35. Rahbarinia, B., Perdisci, R., Lanzi, A., Li, K.: Peerrush: mining for unwanted p2p traffic. J. Inf. Secur. Appl. **19**(3), 194–208 (2014)
36. Rossow, C., Andriesse, D., Werner, T., Stone-Gross, B., Plohmann, D., Dietrich, C.J., Bos, H.: Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 97–111. IEEE (2013)
37. Schaeffer, S.E.: Graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007). ISSN 15740137. doi:10.1016/j.cosrev.2007.05.001. http://linkinghub.elsevier.com/retrieve/pii/S1574013707000020
38. Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S.: Phoenix: Dga-based botnet tracking and intelligence. In: Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 192–211. Springer, Berlin (2014)
39. Schonewille, A., van Helmond, D.-J.: The domain name service as an IDS. Research Project for the Master System-and Network Engineering at the University of Amsterdam (2006)
40. Silva, S.S., Silva, R.M., Pinto, R.C., Salles, R.M.: Botnets: a survey. Comput. Netw. (2012). ISSN 13891286. doi:10.1016/j.comnet.2012.07.021. http://linkinghub.elsevier.com/retrieve/pii/S1389128612003568
41. Stinson, E., Mitchell, J.C.: Towards systematic evaluation of the evadability of bot/botnet detection methods. In: WOOT'08 Proceedings of the 2nd Conference on USENIX Workshop on Offensive Technologies, pp. 1–9, July 2008. http://dl.acm.org/citation.cfm?id=1496702.1496707
42. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. In: ACM SIGCOMM Computer Communication Review, vol. 31, pp. 149–160. ACM (2001)
43. Strayer, W., Walsh, R., Livadas, C., Lapsley, D.: Detecting Botnets with Tight Command and Control. In: Proceedings. 2006 31st IEEE Conference on Local Computer Networks, pp. 195–202. IEEE, Nov. 2006. ISBN 1-4244-0418-5. doi:10.1109/LCN.2006.322100. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4116547
44. van Laarhoven, T., Marchiori, E.: Graph clustering with local search optimization: the resolution bias of the objective function matters most. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **87**(1), 012812 (2013). ISSN 1550-2376. http://www.ncbi.nlm.nih.gov/pubmed/23410393
45. Walsworth, C., Aben, E., Claffy, K., Andersen, D.: The CAIDA UCSD anonymized internet traces (2011). http://www.caida.org/data/passive/passive_2011_dataset.xml. Accessed 8 Mar 2015
46. Yen, T.-F., Reiter, M.K.: Are your hosts trading or plotting? Telling P2P file-sharing and bots apart. In: ICDCS '10 IEEE 30th International Conference on Distributed Computing Systems, pp. 241–252. IEEE, June 2010. ISBN 978-1-4244-7261-1. doi:10.1109/ICDCS.2010.76. http://dl.acm.org/citation.cfm?id=1845878.1846291. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5541681
47. Yen, T.-F., Reiter, M.K.: Revisiting botnet models and their implications for takedown strategies. In: Degano, P., Guttman, J.D. (eds.) POST'12 Proceedings of the First International Conference on Principles of Security and Trust. Lecture Notes in Computer Science, vol. 7215, pp. 249–268. Berlin, Heidelberg, Mar. 2012. Springer, Berlin. ISBN 978-3-642-28640-7. doi:10.1007/978-3-642-28641-4. http://dl.acm.org/citation.cfm?id=2260577.2260591. http://www.springerlink.com/index/10.1007/978-3-642-28641-4
48. Zhang, J., Perdisci, R., Lee, W., Sarfraz, U., Luo, X.: Detecting stealthy P2P botnets using statistical traffic fingerprints. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN), pp. 121–132. IEEE, June 2011. ISBN 978-1-4244-9232-9. doi:10.1109/DSN.2011.5958212. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958212. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5958212
49. Zhang, J., Perdisci, R., Lee, W., Luo, X., Sarfraz, U.: Building a scalable system for stealthy p2p-botnet detection. IEEE Trans. Inf. Forensics Secur. **9**(1), 27–38 (2014)