Community detection in dynamic networks with spark

Priyangika R. Piyasinghe*

Department of Computer Science,
Iowa State University,
Ames, IA, 50011, USA
Email: rumesh@iastate.edu
*Corresponding author

J. Morris Chang

Department of Electrical Engineering,
University of South Florida,
Tampa, FL, 33620, USA
Email: chang5@usf.edu

Abstract: Detecting the evolution of communities within dynamically changing networks is important to understand the latent structure of complex large graphs. In this paper, we present an algorithm to detect real-time communities in dynamically changing networks. We demonstrate the proposed methodology through a case study in peer-to-peer (P2P) botnet detection which is one of the major threats to network security for serving as the infrastructure that is responsible for various cyber crimes. Our method considers online community structure from time to time and improves efficiency by maintaining the same level of accuracy of community detection over time. Experimental evaluation on Apache Spark implementation of the method showed that the execution time improves over dynamic version of Girvan-Newman community detection algorithm while having a higher accuracy level.

Keywords: dynamic networks; community detection; Girvan-Newman algorithm; large graphs; spark.

Reference to this paper should be made as follows: Piyasinghe, P.R. and Chang, J.M. (2018) 'Community detection in dynamic networks with spark', *Int. J. Data Science*, Vol. 3, No. 3, pp.236–254.

Biographical notes: Priyangika R. Piyasinghe received his BS in Computer Science from the University of Peradeniya Sri Lanka in 2009. He joined the Department of Computer Science at Iowa State University in 2012 and received his MS in Computer Science in 2014 before working toward the PhD. His research interests include cyber security and big data analytics.

J. Morris Chang is a Professor at the University of South Florida. He received his PhD from the North Carolina State University. His past industrial experiences include positions at Texas Instruments, Microelectronic Center of North Carolina and AT&T Bell Labs. He received the University Excellence in Teaching Award

at Illinois Institute of Technology in 1999. His research interests include cyber security, wireless networks, and energy efficient computer systems. Since 2012, his research projects on cyber security have been funded by DARPA. His current DARPA project focuses on privacy-preserving computation over the internet. Currently, he is an Editor of *Journal of Microprocessors and Microsystems* and the Associate Editor-in-Chief of IEEE IT Professional. He is a Senior Member of IEEE.

1 Introduction

A network is said to have a community structure if it divides naturally into groups of nodes with dense connections within groups and sparser connections between groups. Community detection is an important tool to reveal the latent structure of the graphs. Many concrete applications are there for communities such as Web clients who have similar interests and are geographically near to each other served by dedicated mirror servers to improve performance of services provided on the World Wide Web (Krishnamurthy and Wang, 2000), group detection on social and collaboration networks (Girvan and Newman, 2002), protein community discovery in protein-protein interaction networks (Zhang, 2009) and efficient recommendation systems set up by identifying clusters of customers with similar interests in the network of online purchase relationships between customers and products (Reddy et al., 2002). A variety of approaches for community detection have been proposed due to the wide range of these applications. For example, modularity based (Blondel et al., 2008; Clauset et al., 2004), weighted community clustering (WCC) based (Prat-Perez et al., 2012), graph partition approach (Fortunato and Castellano, 2012) and statistical inference approach (Holland, 1973).

Mainly network structures with respect to time can be identified as two types: static and dynamic. Static networks are the graphs with permanently fixed nodes and edges over time considered. Much of the current work in community detection in complex networks are based on static networks that either is derived from the aggregation of data over all time or a single snapshot at a particular time. However, some networks (e.g., social networks) evolving over time making topological changes such as adding new connections, removing the existing connections, merging the properties, etc. which can have major impacts on the dynamics over the network. A dynamic representation of complex networks where nodes and edges shift according to changes in the system reflects this reality more closely. When faced with dynamic networks, traditional community mining methods may lead to unrealistic divisions (Kumar et al., 2010).

A number of studies on analysing communities and their evolution in dynamic networks have been proposed (Toyoda and Kitsuregawa, 2003; Kumar et al., 2003). However, there are some limitations in them. For example, the communities and their changes are studied separately or the results rely on extensive human interpretation (Lin et al., 2008). Most community detection methods treat each configuration over time as a separate network even though the changes are not that significant from the last time stamp. For example, networks may vary by only one node or one edge. Redundant computation is required When the information regarding communities from the previous configuration is not used and the communities have to be recomputed as a whole. The efficiency of these algorithms can be

greatly improved if the recomputation is limited only to the portions of the network that are affected by the modifications.

In this paper, we propose a new algorithm for community identification to overcome the limitations in existing methods. Our work is based on the fact that most communities tend to evolve gradually over time without dramatic changes from one time step to another.

We used peer-to-peer (P2P) botnet detection as case study for community detection in dynamic networks. A botnet is a collection of compromised machines that are remotely controlled by one or more botmasters through a command and control (C&C) channel. Botnet infrastructure is responsible for a variety of cyber crimes. Government departments, commercial institutions and common users suffering a lot from net thefts, distributed denial of service (DDoS) attacks and a mass of spams caused by botnet.

Botnets can be divided into two sets based on the architecture:

- the centralised architecture, which uses C&C channels such as internet relay chat (IRC) to receive instructions from a remote controller (botmaster)
- the decentralised architecture that utilises a P2P protocol to coordinate its operation.

When considering these two sets, the centralised botnets contain a single point of failure which is a disadvantage. Therefore, most of the recently developed botnets attempt to build on the P2P architecture (Buford et al., 2009) to take advantage of the resilience offered by the architecture (Rossow et al., 2013). In a P2P botnet, even if a certain number of bots are identified and taken down (Vacca, 2012), the other peers can still work and carry out attacks. Therefore, it is important to identify all the bots and take them down. In this work, we detect such P2P botnets as a community detection problem in dynamic networks. We present an efficient and fully distributed method to detect dynamic communities on Spark (Apache Spark) to identify P2P botnets. The proposed detection method works in real-time. To process graphs efficiently, we use GraphX component on Spark which exposes fundamental operators of Pregal API.

Our botnet detection starts from building a mutual contact graph (MCG) for all of the targeted hosts coming from the monitored network trace. Then, it applies a community detection method on Spark to the whole graph into several different communities (subgraphs) and clustering each bot that comes from the same botnet into the same community. Detection system clusters each bot into its own botnet communities and distinguishes the legitimate hosts and bots into different communities. Detection system then applies an outlier selection strategy on the list of communities utilising the community behaviour level features to detect potential botnet communities and further identifies potential bots from each botnet community by considering community behaviour level features. We have designed our experiments with real network trace from a public traffic archive and 2 botnet datasets (13 Storm and 3 Waledac). Test performed on 40 different randomly generated datasets that each of which contains 5000 targeted hosts and achieve both higher detection rate with very low false positive rate (FPR) and improved execution time. The main contributions of this paper are as follows:

- we have greatly improved the efficiency of the community detection algorithm by limiting the recalculations at each step to the portions of the network that are affected by the modifications
- we have implemented a scalable dynamic community model in distributed environment based on Apache Spark using Pregel computational model

The rest of this paper is organised as follows: Section 2 provides the motivation and the summary of the main features specified in the proposed approach. Section 3 presents the proposed detection system design and implementation. Section 4 describes the list of extensive experiments, results and corresponding analysis of the proposed detection system. Section 5 reviews the previous studies related to this research area. Section 6 includes a discussion. Finally, Section 7 concludes the paper with some future work suggestions.

2 Dynamic community to detect P2P botnets

In this section, we present the basic idea and the details of the proposed algorithm. We first begin with an intuitive explanation in the next subsection and then provide a more detailed and formal explanation in subsequent subsections.

2.1 Basic idea

This section presents the basic ideas applied in our approach. We start by giving an explanation of the foremost feature that has been applied in our approach, the mutual contacts. These give us the basic idea to formulate a P2P botnets clustering problem into a graph community detection problem. Then, we provide a more detailed discussion about the community behaviour features, which contain numerical community features and structural community features that have been applied to both botnet community detection and final bot candidates selection.

In order to illustrate the community behaviour features and statistical traffic features, we conduct a list of preliminary experiments on a dataset obtained from (Rahbarinia et al., 2013) which we selected a sub-dataset that contains 24 h traffic trace of four popular legitimate P2P applications: eMule, FrostWire, uTorrent and Vuze and two P2P botnets: Storm and Waledac. Table 1 shows the summary of our preliminary experiment dataset.

Table 1 Preliminary experiment dataset summary

Application	*Number of hosts*	*Hours*	*Average number of flows*	*Protocol*
eMule	2	24	175,151	TCP/UDP
FrostWire	2	24	217,330	TCP/UDP
uTorrent	2	24	965,893	TCP/UDP
Vuze	2	24	652,029	TCP/UDP
Storm	13	24	666,238	UDP
Waledac	3	24	439,455	TCP

2.2 Mutual contact graph

Mutual contact graphs are used in different perspectives (e.g., Coskun et al., 2010). A mutual contact between a pair of hosts is defined as a set of shared contacts or connections between the corresponding pair of hosts.

Consider the network illustrated in Figure 1(a) which contains an internal hosts set (HostA, HostB, HostC, HostD and HostE) and an external hosts set (Host1, Host2, Host3

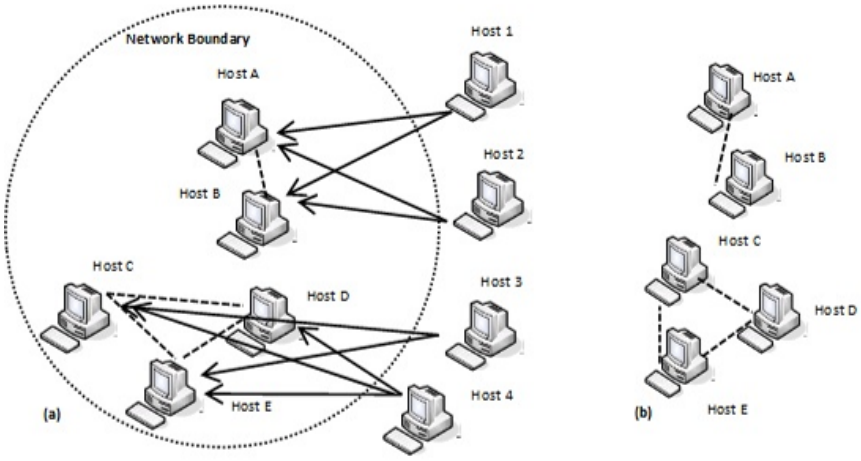
and Host4). A link between a pair of hosts means they have at least one connection between them. In Figure 1(a), Host1 and Host2 are the mutual contacts shared by HostA and HostB.

If we simply divide all of the hosts into legitimate hosts and different types of botnets, we can divide host pairs into four categories:

- legitimate-legitimate, a pair of legitimate hosts
- legitimate-bot, a pair of hosts between legitimate host and bot
- bot1-bot1, a pair of bots within the same botnet
- bot1-bot2, a pair of bots from different botnets.

To illustrate the mutual contacts patterns among different types of host pairs, we have conducted a preliminary experiment using the datasets in Table 1 that contains 8 legitimate hosts (running 4 P2P applications: eMule, FrostWire, uTorrent and Vuze), 13 Storm hosts and 3 Waledac hosts. As shown in Figure 2, different types of host pairs have different mutual contacts patterns. For instance, compared with legitimate-bot and bot1-bot2, host pairs within bot1-bot1 and legitimate-legitimate have a larger number of mutual contacts. Moreover, the host pairs within bot1-bot1 have the largest number of mutual contacts.

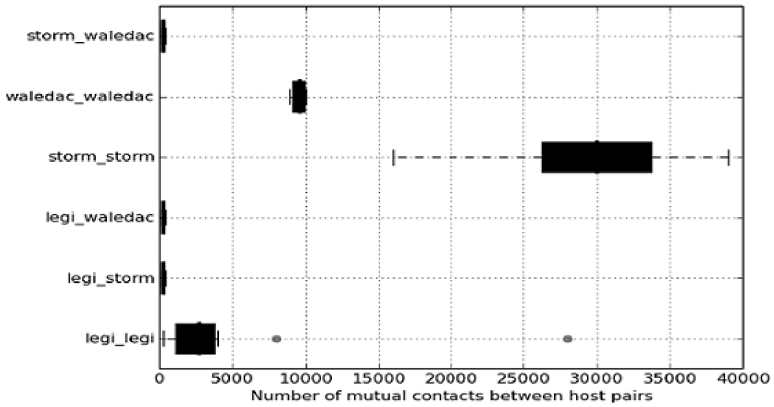
Figure 1 (a) Mutual contact communities for a network and (b) mutual contact graph, in which there exist two communities (C1: Host A, B, and C2: Host C, D, E)



Compared with legitimate hosts, there is a much more significant probability that a pair of bots within the same botnet have a mutual contact (Coskun et al., 2010) since bots within the same P2P botnet tend to receive or search for the same in the set of its IPs. First, P2P applications usually have a large number of distinct degrees because the peer IPs are usually spreading across a large number of networks. Moreover, in order to prevent peers from churning in a P2P botnet, botmaster has to check each bot periodically, which translates into a convergence of contacts among peers within the same botnet. However, since bots from different botnet are controlled by different botmasters, they will not have a large number of mutual contacts. Legitimate host pairs may also have a small set of mutual contacts since nearly all hosts would communicate with a list of very popular servers such as google.com,

facebook.com etc. Furthermore, the host pairs running the same P2P applications may also result in a decent number of mutual contacts, if they are accessing the same resource from the same set of peers by coincidence. However, in reality, different legitimate P2P hosts usually will not search for the same set of peers intentionally. Therefore, we can utilise these different mutual contacts patterns among different types of host pairs to cluster bots within the same botnet into the same communities.

Figure 2 Number of mutual contacts between different types of host pairs contain in Table 1 dataset



To utilise mutual contacts feature, we begin with constructing an MCG. The basic idea is illustrated in Figure 1, in which HostA and HostB are linked together in Figure 1(b) since they have mutual contacts Host1 and Host2 in Figure 1(a). Similarly, HostC, HostD and HostE are linked to each other (in Figure 1(b)) since every pair of them is sharing at least one mutual contacts (in Figure 1(a)). The detail of implementation is given in Section 3.2.

2.3 Dynamic community

In this section, we present a list of community behaviour based features that can be utilised to identify potential botnet communities. We assume that the botnet communities must have community-level behaviour features that are distinguishable from legitimate hosts or legitimate communities since botnet always works as a group/community. However, since the community behaviours of single bots are usually changing dynamically, it would be very difficult to identify a single bot from a number of legitimate hosts, only based on the single host's behaviours. Community behaviour features can be roughly divided into two categories:

- numerical community features
- structural community features.

2.3.1 Numerical community features

Numerical community features contain a list of community-level numerical statistics of each community such as the average number of mutual contacts between each pair of hosts within the same community, the average number of degrees or the number of contacts

among hosts within the same community and the average traffic statistics or connection characteristics.

In this work, we do not utilise any traffic statistics related features to detect botnets since they can be randomised or changed dynamically without much influence on the primary functions of a botnet community. Our work mainly focuses on two numerical community features: the average degree and mutual contacts ratio.

Average degree: P2P bots within the same botnet tend to have a similar number of degree. These bots are directly controlled by the same machine without human involvement. As a result, they receive the same C&C messages and conduct similar malicious activities. The average degree of a P2P botnet community is much higher than a legitimate community. Although an individual legitimate host (e.g., P2P hosts) may have a large degree, the other legitimate hosts within the same community may not have similar large degree. For example, even if a legitimate community contains several high degree hosts, the average degree of that community might not be that significant. Furthermore, legitimate P2P hosts usually have a larger degree in the same time period compared to P2P bots since botnets usually act stealthily that generate a low volume of traffic. Therefore, we consider the average degree among all hosts within the same community as the first community behaviour features.

Average mutual contact ratio: Defined as the number of mutual contacts between a pair of hosts divided by the number of all contacts of those pair of hosts. This feature is based on three assumptions:

- P2P botnet community contains at least two bots since one member communities cannot have this feature
- the mutual contact ratio between a pair of bots is much higher than that between a pair of legitimate hosts
- each pair of bots within the same botnet has similar mutual contact ratio.

Therefore, we consider the average mutual contacts ratio among all pairs of hosts within the same community as the other community behaviour features.

2.3.2 Structural community feature

This feature captures the structural characteristics of a botnet community. As discussed in previous subsections, every pair of bots within the same botnet tends to have a considerable number of mutual contacts. Therefore, if we consider each host as a node and draw an edge between two nodes if the pair of hosts represented by those nodes has a certain number of mutual contacts, then the bots within the same botnet should form a clique (a complete graph). In contrast, the contacts sets among legitimate P2P host usually tend to diverge into different networks which result in a relatively low probability of forming a clique. Hence, we can translate P2P botnets detection problem into a complete graph detection problem that can detect complete graphs with certain requirements based on the node and edge weights. However, since clique detection problem is NP-complete, it is not algorithmically feasible to apply a clique detection to detect botnets. Therefore, we need to combine both numerical and structural features to identify P2P botnets.

Detecting communities dynamically is an iterative computational process. When the graph size gets larger over the time, it requires efficient graph processing platform to find the communities within. Spark has received a lot of attention in recent past as a platform to

process large datasets in relatively quick time. GraphX component under the Spark specially built to analyse huge graphs using fundamental operators like subgraph, joinVertices, aggregateMessages, etc. which have benefited from resilient distributed datasets (RDDs) – highly optimised data abstractions perfectly tailored for iterative algorithms (Zaharia et al., 2012).

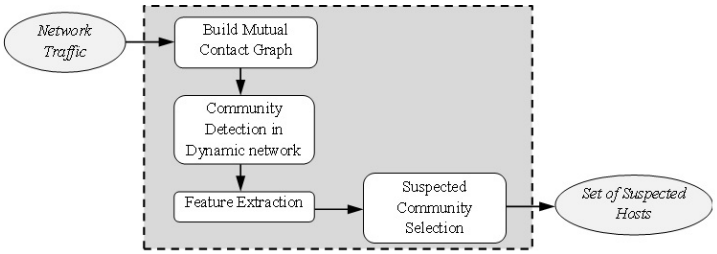
3 System design

In this section, we present the detailed description of system design to detect P2P botnets. The proposed system consists of three main components, that work synergistically to

- construct MCGs from network trace
- cluster bots into its botnet communities during dynamic community detection process
- identify potential botnets communities and further identify potential bots from each botnets community.

Figure 3 illustrates the system framework and the description of each component are given below.

Figure 3 An overview of proposed community detection in dynamic network



3.1 Mutual contact graph component

Mutual contact graph is a weighted undirected graph where each node represents a host. An edge in MCG implies that the pair of hosts corresponding to the two nodes of the edge is sharing at least one mutual contact. To utilise the mutual contacts feature mentioned in Section 2.2, we start from building an MCG from the network trace. This MCG is updated during the online detection process as shown in Figure 4.

Consider applying our botnet detection procedure on the network boundary (e.g., firewall, backbone link) illustrated in Figure 1(a). The input for this component is a list of internal network hosts V_{in}, such as HostA, HostB, HostC, HostD and HostE in Figure 1(a), and a set of netflow trace $F = (S, D)$, where $S = \{s_1, s_2, \ldots, s_n\}$ is a set of hosts appeared in the netflow trace, including both internal network hosts V_{in} and external network hosts, and $F = \{f_1, f_2, \ldots, f_{|F|}\}$ is a set of traditional 5-tuple flows. The output is the MCG $MCG(V, E)$, where each node $s_i \in V$ is an external contact corresponding to internal host and each edge $e_{u_i v_j} \in E$ contains a weight attribute that shows the ratio of mutual contacts

between nodes u_i and v_j. Weight attribute of an edge is same as the Jaccard Similarity between the two nodes. Algorithm for this component is given in Algorithm 1. Below is the detailed description of the main steps in this component.

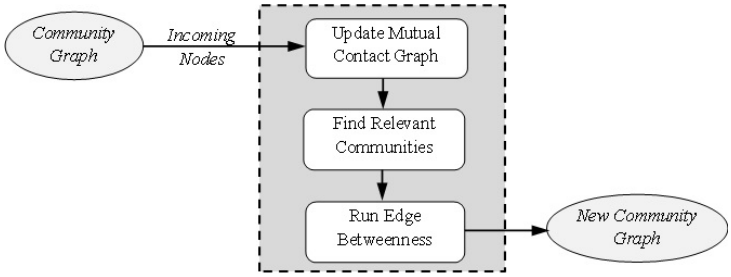
Algorithm 1 Construct mutual contact graph

Data: $F(S, D)$: Flow such that $F(S, D) = \bigcup f(s_i, d_i)$ where $f(s_i, d_i)$ is the flow from source node s_i to destination node d_i and $S = \bigcup s_i$ and $D = \bigcup d_i$, V_{init} : Set of existing nodes in initial graph

Result: $MCG(V, E)$: Mutual contact graph where V = Set of nodes and E = Set of edges

V = $\emptyset$
E = $\emptyset$
$Cont_S$ = Contact set of S = $\emptyset$
$Cont_D$ = Contact set of D = $\emptyset$
for *each flow* $f(s_i, d_i)$ *in* $F(S, D)$ **do**
  if $s_i \in V_{in}$ **then**
    $Cont_S = Cont_S \cup s_i$
  end
  else
    $Cont_D = Cont_D \cup d_i$
  end
end
for *each node* u_i *in* V_{in} **do**
  $Cont_{u_i}$ = Contact set of u_i
  Set the degree of node u_i to $|Cont_{u_i}|$
  $V = V \cup u_i$
end
for *each pair of nodes* (u_i, v_j) *in* V_{in} **do**
  $e_{u_i v_j}$ = Edge between nodes (u_i, v_j)
  $Cont_{u_i}$ = Contact set of u_i
  $Cont_{v_j}$ = Contact set of v_j
  Set the mutual contact ratio of edge $e_{u_i v_j}$ to $\frac{|Cont_{u_i} \cap Cont_{v_j}|}{|Cont_{u_i} \cup Cont_{v_j}|}$
  $E = E \cup e_{u_i v_j}$
end

Figure 4 Detailed view of mutual graph component given in Figure 3



3.1.1 Generating contact sets

Assume we only monitor the network at the network boundary (e.g., firewall, backbone link). Then, detection system only considers the bot/non-bot membership of the hosts belonging to the internal network. If an external host has communication with an internal host, we call the external host as a contact of the corresponding internal host. In this step, for each internal host $h_i \in V_{in}$, it will generate a contact set S_{h_i} for each internal hosts. This process contains:

- initialising an empty contact set S_{h_i} for each internal host $h_i \in H_{in}$
- add new contacts into each contact set based on the source/destination hosts (IP) information of each flow.

3.1.2 Computing edge and vertex attributes

As mentioned earlier, both nodes and edges in MCG have weight attributes. Vertex weight attribute d_i is the cardinality of that node's contact set, and edge weight attribute w_{ij} is the ratio of cardinalities of the common contact set and the total set that they own.

3.2 Dynamic community detection component

Dynamic community detection consists of two parts:

- update MCG
- community detection.

3.2.1 Update mutual contact graph

When a new node is connecting to an existing graph, the MCG $MCG(V, E)$ is also need to be updated. To update MCG, the adjacency list of incoming node v_{in} has to be considered with the adjacency list of existing nodes in the graph. To keep this much data available, we used Redis (Redis) in-memory data structure along with RDD in Spark. The corresponding algorithm is given in Algorithm 2. Here, if v_{in} is connected to v_j, then contact set of v_j will be updated. Therefore, the weights of edges already connected to v_j need to be updated using ADJ_{v_j}. This step takes linear time.

3.2.2 Community detection

Many community detection methods have already been discussed in Holz et al. (2008). We use Girvan-Newman community detection algorithm (Girvan and Newman, 2002) considering its' simplicity in adaptation to run parallelly on Spark environment, which will discuss later in this section. Girvan-Newman algorithm runs in four steps:

a find shortest path between pairs

b find the edge betweenness

c remove the edge with the highest betweenness

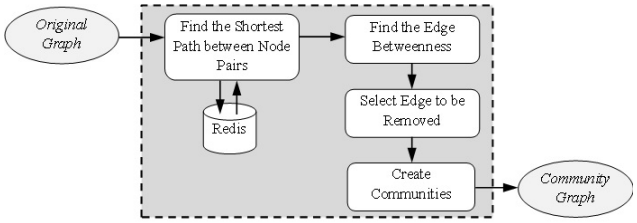
d repeat step (b) and (c) until no remaining edges.

Algorithm 2 Update mutual contact graph

Data: $MCG(V, E)$: Mutual contact graph where V = Set of nodes and E = Set of edges, v_{in} : Incoming node, $V_{connecting}$: Subset of nodes in MCG in which v_{in} is connecting to
Result: Updated MCG, V_{update} : Set of nodes changed in MCG
$ADJ_{v_{in}}$ = Adjacency list of v_{in}
$V_{update} = \emptyset$
for *each node* v_j *in* $V_{connecting}$ **do**
  ADJ_{v_j} = Adjacency list of v_j
  Update MCG by inserting edge (v_{in}, v_j) and calculate mutual contact ratio of (v_{in}, v_j) using $ADJ_{v_{in}}$ and ADJ_{v_j}
  $V_{update} = V_{update} \cup v_j$
end

The performance bottleneck of Girvan-Newman algorithm is a step (a) which always of complexity $O(n^2)$. By storing the shortest path in Redis memory structure we make dynamic community detection efficient at the entrance of new node into existing community structure. Figure 5 illustrates these steps.

Figure 5 Implementation of Girvan-Newman algorithm on Spark using Redis memory structure to store shortest paths between node pairs



At the end of each MCG update, the new community structure CMM_G is derived by considering the shortest paths $SPATH_{CMM\_G}$ of the previous community. This process generates, considerable amount of metadata. Again, to hold the data efficiently in memory, we use Redis data structures. The corresponding algorithm is given in Algorithm 3.

When the node joins existing graph, it can either

a joins to the existing community or

b divide existing community to smaller communities.

For case (a), updating shortest paths $SPATH_{CMM\_G}$ with new node takes linear time since the new node always becomes a leaf node (an end node) in the resulting community. For case (b), shortest paths between every pair of nodes need to be recalculated in the resulting community. Here the runtime depends on how well the graph is connected. However, by definition a community is a tightly connected knot in Graph. Hence, the number of iterations in message passing considerably reduced. For example, if we need to find minimum node in a clique, it will only require a single round of message passing in Spark GraphX.

3.3 Suspicious botnet detection

To detect suspicious community in CMM_G, both average degree AVG_Deg and average mutual contact ratio $AVG\_Mut\_Cont\_Rat$ are considered. Let AVG_Deg_{ci} and $AVG\_Mut\_Cont\_Rat_{ci}$ be the average degree and average mutual contact ratio of community c_i respectively. If the product of AVG_Deg_{ci} and $AVG\_Mut\_Cont\_Rat_{ci}$ is greater than the predefined threshold T, then that community c_i declared as a suspicious community. The value of T is decided at the training phase of the experiment. Initially, all the nodes in the community declared as suspicious, but this leads to the false positives. Later in the experiment, a host is declared as suspicious based on the flow classification of hosts in a suspicious community. Algorithm 4 shows the algorithm of suspicious botnet detection.

Algorithm 3 Update community graph

Data: CMM_G : Community Graph such that $CMM\_G = \cup c_i$ where c_i is i^{th} community, $SPATH_{CMM\_G} = \bigcup spath_{c_i}$ where $spath_{c_i}$ is the list of shortest paths between each pair of nodes in community c_i , V_{update} : Set of nodes that changed in mutual contact graph
Result: Updated CMM_G, Updated $SPATH_{CMM\_G}$

```
for each node c_i in CMM_G do
    for each node v_j in V_update do
        if v_j belongs to c_i then
            spath_{c_i} = List of shortest paths between each pair of nodes in c_i
            Run Girvan-Newman on c_i community based on v_j and spath_{c_i} and get
            the updated CMM_G
            Update spath_{c_i}
        end
    end
end
```

Algorithm 4 Suspicious botnet detection

Data: CMM_G : Community graph such that $CMM\_G = \bigcup c_i$ where c_i is the i^{th} community, T : Threshold
Result: SSP_CMM :Suspicious community

```
SSP_CMM = ∅
for each community c_i in CMM_G do
    Node_{c_i} = Set of nodes in c_i
    Edge_{c_i} = Set of edges in c_i
    AVG_Deg_{c_i} : Average degree of c_i = (Σ_{p∈Node_{c_i}} d_p) / |Node_{c_i}| where d_p is the degree of p^{th}
    node of c_i
    AVG_Mut_Cont_Rat_{c_i} :
    Average mutual contact ratio of c_i = (Σ_{(p,q)∈Edge_{c_i}} w_pq) / |Edge_{c_i}| where w_pq is the mutual
    contact ratio of (p, q) edge of c_i
    if AVG_Deg_{c_i} × AVG_Mut_Cont_Rat_{c_i} > T then
        SSP_CMM = SSP_CMM ∪ c_i
    end
end
```

4 Experimental evaluation

4.1 Experiment setup

4.1.1 Experimental environment

The experiments are conducted on a PC with a 12 core Intel i7-5820 Processor, 32GB RAM, 470GB SSD and 4TB HHD, and on 64-bit Ubuntu14.04 LTS operating system. Spark-1.4.1 is used with GraphX running in local mode.

4.1.2 Datasets and analysis tool

To evaluate our system, we utilise botnet dataset DB and legitimate dataset DL. DB is obtained from the University of Georgia (Rahbarinia et al., 2013), which contains 24 h real network trace from 13 hosts infected with Storm and 3 hosts infected with Waledac. The Storm dataset contains 666,238 flows and 145,972 unique IPs other than 13 Storm hosts. The Waledac dataset contains 439,455 flows and 29,973 unique IPs other than 3 Waledac hosts. No malicious activities can be observed in this botnet dataset.

DL is downloaded from the MAWI Working Group Traffic Archive (MAWI Working Group Traffic Archive) which contains a 24 h anonymised network trace at the transit link of WIDE (150Mbps) to the upstream ISP on 2014/12/10 (sample point F). DL contains approximate 407,523,221 flows and 48,607,304 unique IPs. 79.3% flows are TCP flows and the rest are UDP flows.

We utilise ARGUS (ARGUS) to process and cluster network trace into 5-tuple TCP/UDP flows.

4.2 Experimental dataset generation

In order to evaluate our system, we generate three main datasets by mixing the network trace from DB and the DL. Table 2 illustrates the summary of three main datasets. Each DS and DW contains 20 sub-datasets and each sub-dataset includes the network trace from 5000 internal hosts sampled from DL. Among each sub-dataset's 5000 internal hosts in DS, 13 hosts are mixed with Storm network trace. Similarly, among each sub-dataset's 5000 internal hosts in DW, 3 hosts are mixed with Waledac network trace. DS+W contains 40 sub-datasets and each sub-dataset includes the network trace from 5000 internal hosts sampled from DL where 3 hosts are mixed with Waledac network trace and the other 13 hosts are mixed with Storm network trace. A sub-dataset is the basic unit for one test. In the rest of this section, the term 'graph' represents the graph generated using a sub-dataset.

Table 2 Experiment dataset summary

Datasets	*Number of graphs*	*Bots/Internal hosts*	*Average number of totals hosts*
DS	20	13/5000	786,494
DW	20	3/5000	543,913
DS+W	40	16/5000	1,209,112

The first step of generating experimental sub-datasets is to sample a list of background hosts from DL. As discussed in Section 2, the design of our system is to deploy at a network

boundary (e.g., firewall, gateway, etc.) where the network forms a bipartite structure so that we can only capture the connection between internal hosts and external hosts. Therefore, we need to sample a list of internal hosts such that any pair of them should not have any connections to each other.

To maintain a bipartite network structure of botnets network trace, we eliminate all of the flow between every two bots in DB. Further to make sure after mixing both botnet and legitimate network trace, each graph still maintains a bipartite structure. To select 5000 nodes, we used two-colouring of the graph.

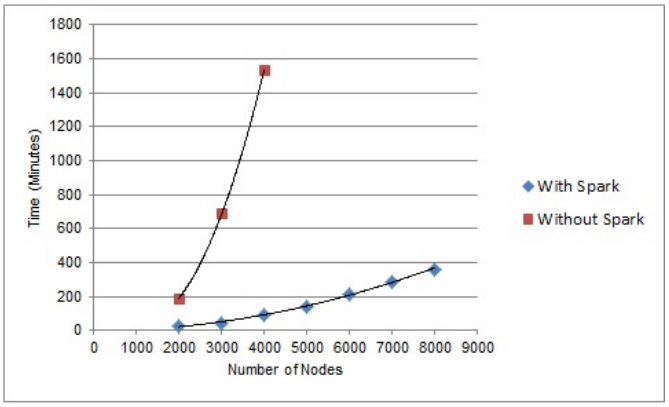
To mix the botnet trace with sampled legitimate trace, for DS, each time we randomly select 13 legitimate hosts out of the 5000 hosts, map 13 Storm hosts IPs to the 13 legitimate hosts IPs, and merge the corresponding network trace. DS and DS+W are generated using the same procedure.

This experimental dataset generation process is repeated for a total of 80 times (each time we start from a different random host; 40 times for DS+W, 20 times for DS and 20 times for DW). Finally, we obtained DS, DW and DS+W to evaluate our system respectively. From the generated graphs, 40 graphs are used to find the threshold and rest used for testing the online detection of communities.

4.3 Results

Figure 6 illustrates the comparison of time taken to detect communities in the dynamic version of original Girvan-Newman that runs natively and in Spark. According to the figure, there is a significant improvement of execution time in Spark by restricting recalculations at each step only to the portion of the network that affected by modifications and keeping most of the metadata that required for recalculations in memory.

Figure 6 Comparison of time taken to detect dynamic communities natively and in Spark (see online version for colours)



By using Redis in-memory data components we can further reduce writing to disk by Spark in between the jobs. Hence performance is increased as shown in Figure 7.

We utilise precision and recall as our evaluation criterion:

$$\text{Precision} = TP/(TP + FP)$$
$$\text{Recall} = TP/(TP + FN),$$

where TP stands for true positive, FP stands for false positive, FN stands for false negative.

Figures 8 and 9 show the precision and recall on 40 graphs tested using our model respectively. The graphs having bot count of 16 with both Waldec and Storm have the highest average precision of 0.923. Graphs with only 3 Waldec bots have an average precision of 0.63. From this result, we can see graphs with higher bot count gives less false positive. We will discuss about possible reasons in Section 6. Average recall for all graphs is 1 that implies higher detection from our model.

Figure 7 Comparison of time taken to detect dynamic communities in Spark with and without using Redis (see online version for colours)

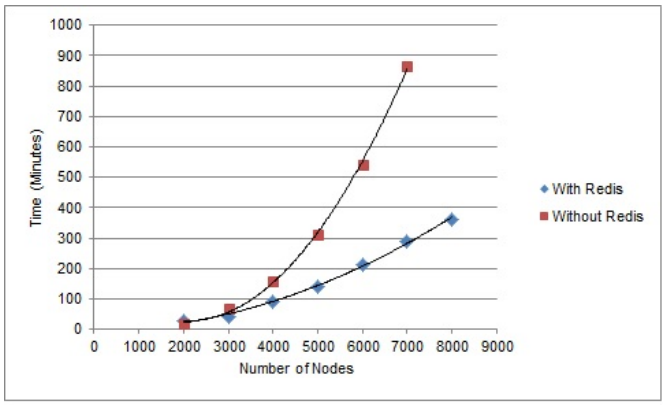
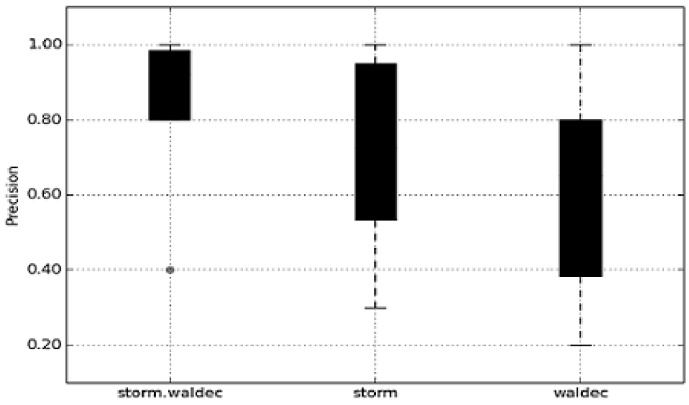


Figure 8 Precision on 40 graphs tested using proposed model

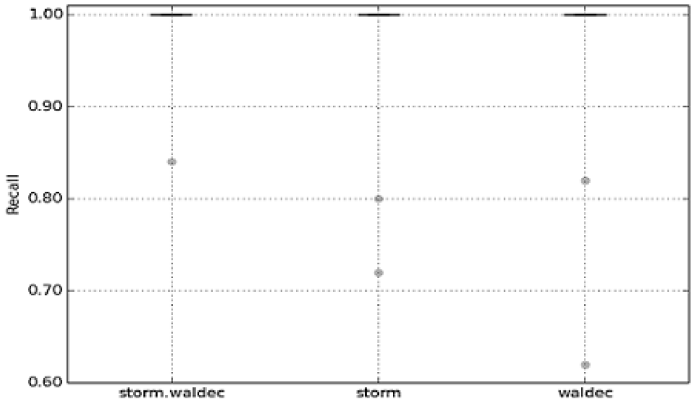


5 Related work

Community identification for dynamic networks has received less attention in the field compared to identifications of communities in static networks. The community detection

methods in dynamic networks can be categorised into two classes: incremental or online community detection where data is evolving in real-time; and offline community detection where all the changes of the network evolution data are known at the beginning.

Figure 9 Recall on 40 graphs tested using proposed model



In offline community detection Tantipathananandh et al. (2007) propose clustering framework based on finding optimal graph colourings which are proved to be an NP-hard problem. They present heuristic algorithms which find near optimal solutions and are demonstrated on small networks with little evolution. However, when scalability is considered, their algorithm in the current form is not ideal for large networks.

In online community detection, evolution clustering proposed by (Chakrabarti et al., 2006) simultaneously optimises two potentially conflicting criteria. First, the clustering at any point should remain fixed to the current data as much as possible. Then, the clustering should not dramatically move from one time step to the next time step. One can obtain a balanced community structure between the quality of present clustering result and the previous result. However, the network structure at each time step is usually required to be clustered separately, which results in a higher complexity. Therefore, it is difficult to apply to identify and analyse the large-scale dynamic networks.

Ning et al. (2007) propose an incremental algorithm which is initialised by a standard spectral clustering algorithm, followed by updates of the spectra as the dataset evolves. Compared with recomputation by standard spectral clustering for web blog data, their algorithm achieves similar accuracy but smaller computational costs. Leung et al., 2009 discuss theoretical label propagation algorithm for dynamic network data. The label propagation algorithm initialises each node with a unique label, and proceeds by allowing each node to adopt the label most popular among its neighbours. This iterative process produces densely connected groups of nodes from the current state of the network. The static version of the label propagation algorithm is efficient. However, they did not discuss how efficient algorithm can be in dynamic networks.

As a solution to large data growth over past few years, researchers have proposed scalable data analytic methods along with storage and processing models (Kumar and Kumar 2015, Sharma, et al., 2014). Known implementations to detect communities using Hadoop MapReduce big data framework include: SLPA using MPI (Kuzmin et al., 2013), the Louvian method using Apache Graph (Distributed Louvain Modularity), the propinquity

dynamics method using Hadoop MapReduce (Zhang et al., 2009), Scalable Community Detection (Perez et al., 2014) and many others. However these methods address scalability of the community detection, but in our work, we address more of efficiency in order to detect communities online in real-time.

Apache Spark implementation of a community detection method has been proposed in Buzun et al. (2014). They used label propagation method with the help of friendship groups of individual users in the social network to identify the communities. In their work, the detection process is viewed as static rather than dynamic. To the best of our knowledge, this work is the first to report on an Apache Spark implementation of community detection in dynamic networks. Firstly, we make use of GrpahX ecosystem that comes with Spark to do the operations like message passing, merging and aggregating. Secondly, we benefit from RDDs – highly optimised data abstractions perfectly tailored for iterative algorithms (Zaharia et al., 2012).

6 Discussion

As shown in the experimental results, the proposed system achieves higher average precision and higher average recall. Moreover, the comparison with a dynamic version of Girvan-Newman algorithm shows relatively efficient runtime. Despite these significant improvements, the proposed system is not without limitations.

Although the proposed system is capable of detecting all the bots in the network, the existence of false positives is not fully eliminated. By analysing the each trace, we have identified following reasons may lead to an existence of false positives.

- If one-degree legitimate hosts only connection is a bot, then it will be clustered into the same community as the corresponding bot. When selecting botnet communities that one-degree host will also be considered as a bot.
- The best community selected in the current time stamp to join the incoming node might not be the ideal community based on the connections that node will have in the future time stamps.

The first reason can be overcome by eliminating very low degree nodes while selecting bot candidates from suspicious communities. For the second one, we can keep all the connection properties of the incoming node with the help of the distributed storage and consider all these information when a connection appears with existing node.

7 Conclusion

In this paper, we propose a novel way of detecting communities in dynamic networks on Spark. Our major contributions in this paper include

- the improvement of the community detection algorithm efficiency
- the implementation of scalable dynamic community model in distributed environment.

The proposed system is capable of detecting P2P botnets through an MCG-based botnet community detection approach. We introduce an algorithm for dynamic community detection by extending the original Girvan-Newman algorithm and benchmarking our results with the results of the original algorithm. It is important to note that we are not aiming to improve the original algorithm in term of accuracy.

In the future work, we will focus on developing this dynamic community method with more robust fast community detection algorithms. Moreover, we will go on optimising our method and test it on more datasets.

References

Blondel, V.D., Guillaume, J.L., Lambiotte, R. and Lefebvre, E. (2008) 'Fast unfolding of communities in large networks', *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 10, pp.P10008.

Buford, J., Yu, H. and Lua, E. (2009) *P2P Networking and Applications*, Morgan Kaufmann.

Buzun, N., Korshunov, A., Avanesov, V., Filonenko, I., Kozlov, I., Turdakov, D. and Hangkyu, K. (2014) 'EgoLP: fast and distributed community detection in billion-node social networks', *2014 IEEE International Conference on Data Mining Workshop(ICDMW)*, IEEE, pp.533–540.

Chakrabarti, D., Kumar, R. and Tomkins, A. (2006) 'Evolutionary clustering', *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp.554–560.

Clauset, A., Newman, M.E. and Moore, C. (2004) 'Finding community structure in very large networks', *Physical Review E 70*, Vol. 6, pp.066111.

Coskun, B., Dietrich, S. and Memon, N. (2010) 'Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts', *Proceedings of the 26th Annual Computer Security Applications Conference*, ACM, pp.131–140.

Fortunato, S. and Castellano, C. (2012) 'Community structure in graphs', *Computational Complexity*, Springer, New York, pp.490–512.

Girvan and Newman J. (2002) 'Community structure in social and biological networks', *Proceedings National Academy of Sciences of the United State of America*, Vol. 99, No. 12, pp.7821–7826.

Holland, P.W. (1973) 'Introduction to Bayesian inference and decision', *Technometrics Journal*, Vol. 15, pp.938–939.

Holz, T., Steiner, M., Dahl, F., Biersack, E. and Freiling, F. (2008) 'Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm', *Proceedings First USENIX Workshop on Large Scale Exploits and Emergent Threats, LEET'08*, Vol. 8, No. 1, pp.1–9.

Krishnamurthy, B. and Wang, J. (2000) 'On network-aware clustering of web clients', *SIGCOMM Comput.Commun. Rev.*, Vol. 30, No. 4, p.97.

Kumar, A. and Kumar, T.V. (2015) 'Big data and analytics: issues, challenges, and opportunities', *International Journal of Data Science*, Vol. 1, No. 2, pp.118–138.

Kumar, R., Novak, J., Raghacan, P. and Tomekins, A. (2003) 'On the busty evolution of blogspace', *Proceedings of the 12th International Conference on World Wide Web*, ACM, pp.568–576.

Kumar, R., Novak, J. and Tomkins, A. (2010) 'Structure and evolution of online social networks', *Link Mining: Models, Algorithms, and Applications*, Springer, New York, pp.337–357.

Kuzmin, K., Shah, S.Y. and Szymanski, B.K. (2013) 'Parallel overlapping community detection with SLPA', *International Conference in Social Computing (SocialCom)*, IEEE, pp.204–212.

Leung, I.X.Y., Hui, P., Lio, P. and Crowcroft, J. (2009) 'Towards real-time community detection in large networks', *Physical Review E*, Vol. 79, pp.066107.

Lin, Y., Chi, Y. and Zhu, S. (2008) 'FacetNet: a framework for analyzing communities and their evolutions in dynamic social networks', *Proceedings of the 17th International Conference on World Wide Web*, ACM, pp.685–694.

Ning, H., Xu, W., Chi, Y., Gong, Y. and Huang, T. (2007) 'Incremental spectral clustering with application to monitoring of evolving blog communities', *Proceedings of the 2007 SIAM International Conference on Data Mining*, Society for Industrial and Applied Mathematics, pp.261–272.

Prat-Perez A., Dominguez-Sal, D. and Larriba-Pey, J-L. (2014) 'High quality, scalable and parallel community detection for large real graphs', *Proceedings of the 23rd International Conference on World Wide Web*, ACM, pp.225–236.

Prat-Perez, A., Dominguez-Sal, D., Brunat, J.M. and Larriba-Pey, L. (2012) 'Shaping communities out of triangles', *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ACM, pp.1677–1681.

Rahbarinia, B., Perdisci, R., Lanzi, A. and Li, K. (2013) 'PeerRush: Mining for unwanted p2p traffic', *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer Berlin Heidelberg, pp.62–82.

Reddy, K.P., Kitsuregawa, M., Sreekanth, P. and Rao, S. (2002) 'A graph based approach to extract a neighborhood customer community for collaborative filtering', *International Workshop on Databases in Networked Information Systems*, Springer Berlin Heidelberg, pp.188–200.

Rossow, C., Andriesse, D., Werner, T., Stone-Gross, B., Plohmann, D., Dietrich, C. and Bos, H. (2013) 'P2PWNED: modeling and evaluating the resilience of peer-to-peer botnets', *IEEE Symposium on Security and Privacy (SP)*, IEEE, pp.97–111.

Sharma, S., Tim, U.S., Wong, J., Gadia, S. and Sharma S. (2014) 'A brief review on leading big data models', *Data Science Journal*, Vol. 13, pp.138–157.

Tantipathananandh, C., Berger-Wolf, T. and Kempe, D. (2007) 'A framework for community identification in dynamic social networks', *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp.717–726.

Toyoda, M. and Kitsuregawa, W. (2003) 'Extracting evolution of web communities from a series of web archives', *Proceedings of the Fourteenth ACM conference on Hypertext and Hypermedia*, ACM, pp.28–37.

Vacca, J. (2009) *Computer and Information Security Handbook*, Newnes.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S. and Stoica, I. (2012) 'Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing', *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, USENIX Association, pp.2–2.

Zhang, A. (2009) *Protein Interaction Networks*, Cambridge University Press, Cambridge, UK.

Zhang, Y., Wang, J., Wang, Y. and Zhou, L. (2009) 'Parallel community detection on large networks with propinquity dynamics', *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp.997–1006.

Websites

Apache Spark, http://spark.apache.org/

ARGUS, http://qosient.com/argus/

MAWI Working Group Traffic Archive, http://mawi.wide.ad.jp/mawi/

Distributed Louvain Modularity, http://sotera.github.io/distributed-louvain-modularity/

Redis, http://redis.io