Czech Technical University in Prague
Faculty of Electrical Engineering

Doctoral Thesis

August 2017 *Jan Jusko*

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Graph-based Detection of Malicious Network Communities

Doctoral Thesis

Jan Jusko

Prague, August 2017

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Supervisor: Martin Rehák, Ph.D., Ingénieur ECP
Supervisor-Specialist: Ing. Tomáš Pevný, Ph.D.

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisors, Martin Rehák and Tomáš Pevný. Martin introduced me to the field of network security and his enthusiasm persuaded me to eventually join his research team. Ever since, Martin and Tomáš have been constantly helping me with all aspects of my research. I especially appreciate and enjoy our discussions – we might often disagree, but their ideas always push my research forward. My sincere thanks also goes to Michal Pěchouček who provided me with an opportunity to join the Agent Technology Center group at the Czech Technical University.

I would like to thank my fellow graduate students and co-workers for providing a stimulating and fun environment in which to learn and grow. I am especially grateful to Karel Bartoš, Martin Grill and Jan Stiborek with whom I have worked closely over the last seven years. I am also very grateful to all collaborators who contributed to this research. Namely I would like to thank Verónica Valeros, with whom I worked closely on transforming my research project into production systems with global impact. I would like to thank Petr Somol and Lukáš Machlica for valuable comments on this thesis.

I also would like to thank my family. I am thankful to my parents for their support, encouragement and guidance that helped me immensely to get where I am today. They were always a source of inspiration for me. I also want to say a big thank you to Jazmína and Leonka for their love, patience and understanding. They were always happy to put a smile on my face when I needed it the most. Thank you.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by the ITC-A of the U.S Army (contracts N62558-07-C-001, W911NF-08-1-0250, W911NF-10- 1-0070 and W911NF-12-1-0028); ONR Global under the Department of the Navy (contract N62909-11-1-7036); Air Force Office of Scientific Research, Air Force Material Command, USAF (grant FA8655-10-1-3016); Czech Ministry of Education (grants ME10051 and MEB111008) and the Czech Ministry of Interior (grants VG2VS/189 and VG20122014079).

Abstract

In this thesis, we use graph based methods in conjunction with behavioral modeling to uncover hidden malicious communities and peer-to-peer traffic.

The nature of malicious traffic, and its tendency to rally in order to communicate with its owner opens a possibility to detect malicious traffic by revealing hidden sub-structures of network traffic. In fact, besides discovering the presence of an infection, analyzing network traffic also enables inference of valuable context information about the malicious campaign as a whole, often leading to a more precise attribution than is possible using only a host-based solution. In this work, we focus on the detection approaches that observe the hidden structures and exploit them to uncover malicious command & control (C&C) servers.

Peer-to-peer (P2P) protocol is a popular choice with malware authors to be used as a C&C channel. Therefore, we propose a unified solution to identify P2P communities operating in a monitored network. We propose an algorithm that is able to 1) progressively discover hosts in the monitored network that cooperate in a P2P network and to 2) identify that P2P network. Starting from a single known host, other hosts participating in the P2P network are identified through the analysis of widely available and standardized IPFIX (NetFlow) data. It is able to identify a large range of both legitimate and malicious P2P networks, is highly scalable and the use of standard meta-data without access to traffic content makes it easy to deploy and justify from privacy protection perspective.

Even malware families that do not rely on a P2P-based C&C channels resort to highly dynamic C&C structures to counter security industry approaches based on blacklisting known malicious domains. It is therefore important to automatically follow the migration of C&C servers. We propose to use a well-known Probability Threat Propagation (PTP) with a novel graph representation capturing connections from clients to servers. The proposed graph representation is highly condensed, preserves privacy, allows us to find malicious domains that cannot be found using existing graph representations and is harder to evade by malware authors.

We propose two behavioral models for HTTP traffic together with kernel-based similarity and distance functions that can be conveniently used to extend the findings of PTP. For any domain marked as malicious by PTP we can find other domains with identical or similar behavior, which are likely also malicious. This significantly increases the number of discovered malicious domains.

All proposed algorithms and representations are verified using extensive data sets spanning hundreds of independent networks. The validity of proposed approaches was further verified in a large-scale deployment within the Cisco Cognitive Threat Analytics.

Abstrakt

V tejto práci využívame grafové metódy v spojení s behaviorálnym modelovaním na odhalenie skrytých spoločenstiev škodlivých serverov a škodlivých peer-to-peer (P2P) sietí.

Povaha sieťovej komunikácie malwaru a jeho tendencia koordinovane sa napájať na kontrolné servery s cieľom komunikovať s jeho vlastníkom otvára možnosť odhaliť sieťovú komunikácia malwaru odhalením skrytých subštruktúr sieťovej prevádzky. Analýza sieťovej prevádzky okrem zistenia prítomnosti infekcie malwarom umožňuje odvodenie hodnotných kontextových informácií o kampaniach malwaru ako celku, čo často vedie k presnejšiemu určeniu pôvodu malwaru, než je možné len na základe antivírového riešenia. V tejto práci sa zameriavame na prístupy detekcie, ktoré pozorujú tieto skryté štruktúry a využívajú ich na detekciu serverov slúžiacich na koordináciu malwaru.

Autori malwaru často používajú P2P siete ako prostriedok komunikácie s malwarom. Preto navrhujeme jednotné riešenie na identifikáciu P2P komunít pôsobiacich v monitorovanej sieti. Navrhujeme algoritmus, ktorý je schopný 1) postupne objavovať zariadenia v sledovanej sieti, ktoré spolupracujú v P2P sieti a 2) identifikovať danú P2P sieť. Toto riešenie je schopné na základe znalosti jedného zariadenia s využitím analýzy sieťovej komunikácie v štandardnom formáte IPFIX (NetFlow) dohľadať ďalšie zariadenia, ktoré s ním spolupracujú v rovnakej P2P sieti. Navrhnutý algoritmus je schopný identifikovať veľký rozsah legitímnych aj škodlivých P2P sietí a je vysoko škálovateľný. Používanie štandardných formátov bez prístupu k vlastnému obsahu sieťovej komunikácie umožňuje jednoduché nasadenie a ochranu súkromia užívateľov.

Skupiny malwaru/škodlivého softvéru, ktoré sa nepoužívajú P2P siete na komunikáciu so svojím autorom, využívajú často sa meniace sady serverov aby sa vyhli detekcii pomocou blacklistov – zoznamov známych škodlivých serverov. Je preto dôležité sledovať automaticky migráciu serverov škodlivých serverov. Navrhujeme použiť známy algoritmus Probabilistic Threat Propagation (PTP) v spojitosti s novým grafom, ktorý popisuje sieťové spojenia medzi klientami a servermi. Navrhovaný graf zachytáva relevantné informácie o sieťovej komunikácii v komprimovanej forme, ktorá zachováva súkromie užívateľov zariadení, umožňuje nájsť škodlivé servre, ktoré nie je možno nájsť pomocou existujúcich grafových reprezentácií a obmedzuje možnosti autorov malwaru vyhnúť sa detekcii.

Navrhujeme dve techniky modelovania HTTP sieťovej komunikácie, spolu s funkciami podobnosti a vzdialenosti založenými na kernel funkcii. Tie sa dajú použiť na nájdenie dodatočných škodlivých serverov na základe serverov nájdených použitím algoritmu PTP. Použitím modelovania sieťovej komunikácie, nájdeme pre každý server označený ako škodlivý algoritmom PTP ďalšie servery s rovnakým alebo podobným správaním, ktoré sú pravdepodobne taktiež škodlivé. To výrazne zvyšuje počet nájdených škodlivých serverov.

Všetky navrhované algoritmy boli overené na rozsiahlych množinách dát, ktoré pokrývajú stovky nezávislých sietí. Úspešnosť navrhovaných prístupov bola ďalej overená vo veľkom meradle v rámci produktu Cisco Cognitive Threat Analytics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Malware is a shortcut for *malicious software*. It is designed to infect a device and either provide profit to its author or cause harm to the owner of the device. In the past, malware was created mostly as a prank or to demonstrate skill. Nowadays, it is mostly used to create profit for its authors, steal information, or attack specific organizations or nation states with the goal of causing harm.

Malware thus became an umbrella term for a whole family of software with malicious intents. In fact, whether a software is considered to be a malware is based on the intent of its author, not on the actual harm/profit made by the software. Malware can be categorized into several groups, either according to its purpose or means of spread. Distinguishing by purpose, there are four common groups of malware:

- *adware* which makes profit to its owner by click fraud,
- *spyware* whose goal is to steal personal information such as credit card numbers, or industrial and/or state secrets
- *ransomware* which encrypts the files on infected devices and demands ransom to decrypt them, and
- *spamware* that makes profit by sending spam emails from the infected devices.

When distinguishing malware by spreading mechanism, there are basically two types of malware – one can spread without user interaction, for example by exploiting known or not yet disclosed device or network vulnerabilities; the other spreads by tricking a human user into thinking that the given malware actually has a legitimate purpose.

Two main strategies [122] exist to protect devices from infection by malware. The classical, host-based approach is an anti-virus software installed on a device, that scans in real time all executed files. If a file is found to be malicious, its execution is blocked and it is removed from the system. A more recent approach is to protect devices on the network level. Private companies and governmental organizations employ firewalls that check all traffic coming to a network from the Internet. Such solutions block downloads of malicious binaries or connections to/from sites that serve exploit kits. However, malware with infection vectors other than the network is not covered by this solution.

Host-based and network-based strategies of infection prevention are based on fundamentally different observations. First, during and after infection, any malware needs to embed

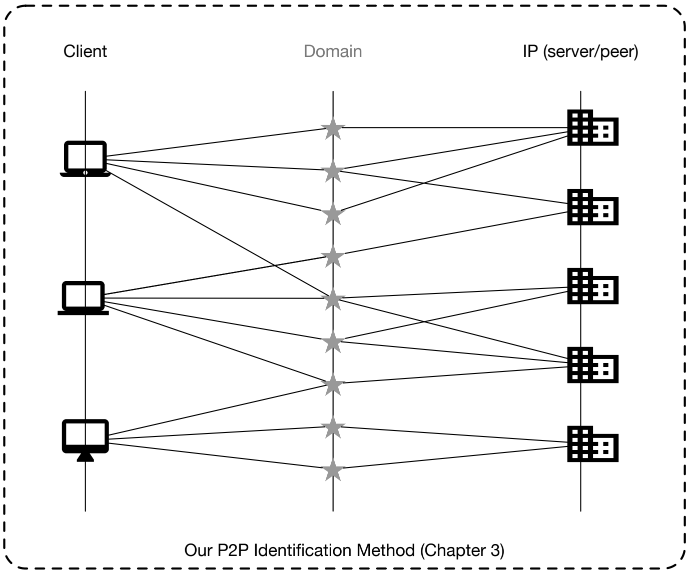


Figure 1.1: Scope of the peer-to-peer discovery algorithm.

itself into the device's system, e.g. by installing code and/or making changes in registries so that the instance of malware is started every time the device is rebooted. Second, malware needs to communicate over Internet to bring economic benefit to its owner. The communication may be either to receive new commands or monetize the infected machine in any way, e.g. ex-filtrate data.

Host-based and network-based detection approaches are thus complementary. Malware authors can avoid detection by either, but evading both approaches working in unison poses new challenges to malware authors.

Another important distinction is whether the protection system is set-up for *prevention* or *detection*. Intrusion prevention systems (IPS) attempt to prevent the infection itself – firewall blocks download of a malicious file and anti-virus blocks its execution. But prevention is not always effective. First, IPS is often biased to prefer precision over recall. This prevents false alarms, but weakens the coverage of rapidly developing new threats. Second, malware can exploit an infection vector that is not covered by the installed IPSs. Malware thus can infect a system despite IPS being in place. On the other hand, Intrusion detection systems (IDS) passively monitor what is happening either in the network or on the infected hosts in order to uncover successful infections.

Many detection approaches focus on statistical properties of malicious network communication. The nature of malicious traffic and its tendency to rally in order to communicate with its owner opens a possibility to detect malware activity by revealing hidden sub-structures of network traffic. In fact, besides merely discovering the presence of an infection, analyzing network traffic on global scale also enables inference of valuable context information about the malicious campaign as a whole. This often leads to a more precise attribution than is possible using only a host-based solution. Our work specifically focuses on the detection of malicious communication structures in network traffic and uses them to uncover malicious command & control (C&C) servers that manage the botnets. Once discovered, the C&C servers are

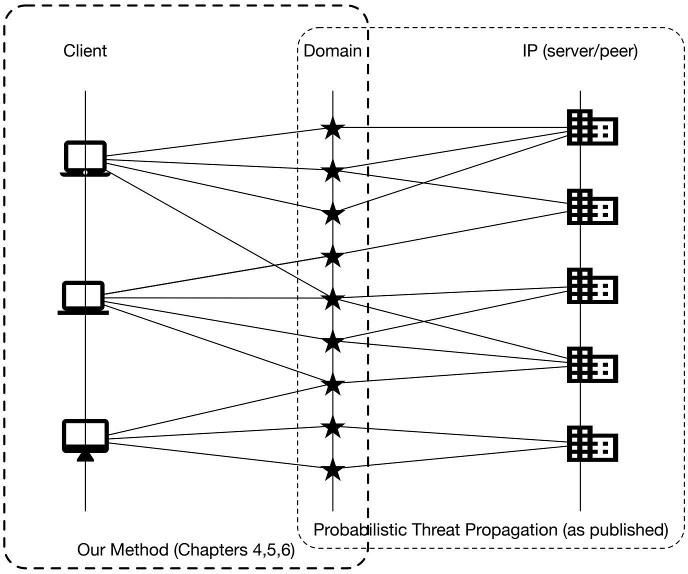

Client
Domain
IP (server/peer)
Probabilistic Threat Propagation (as published)
Our Method (Chapters 4,5,6)

Figure 1.2: Scope of our method when compared to Probabilistic Threat Propagation.

used by other methods (not discussed in this thesis) to associate threat intelligence and other analytical results to a specific botnet.

Malicious C&C channels can have many shapes and forms. In recent years, C&C channels of several major botnets, i.e. virtual networks of cooperating infected hosts, were based on peer-to-peer (P2P) networks. P2P networks are an ideal choice for botnet owners – P2P networks do not have a central point of failure, which was typical for botnets with centralized C&C architecture. At the same time, any computer infected by a P2P-based malware can disseminate new commands to the rest of the botnet, protecting the identity of the botnet master. Accordingly, we will also address the problem of malicious P2P discovery. The scope of the proposed P2P discovery algorithm is depicted in Figure 1.1.

Our approach to the problem differs from the prevailing approach (Probabilistic Threat Propagation) in the domain. Instead of using the information contained in the relationship between a domain name and the associated IP address(es), we rely on the analysis of network metadata associated with the host. A comparison of the scopes of the two approaches is depiceted in Figure 1.2. The distinction is important mainly from security perspective, as it allows us to find a different set of malware when compared to the traditional approach. Traditional methods excel in identification of malware that heavily relies on changing set of DNS records on a limited set of server hosts. This case is typical, as it avoids domain-based blocking on host and network prevention devices and does not force the attackers to laboriously migrate the infrastructure. However, a growing proportion of diverse malware deviates from this scheme and avoids the detection by domain-based analytics completely. Our approach has been designed to be robust w.r.t. this evasion technique and therefore detects a different, novel set of malware.

The input metadata is typically available in standardized formats, such as Netflow/IPFIX records and/or HTTP access logs. When collected on a perimeter device, such as a router, a firewall or a proxy, it contains full information about network connections between the monitored host and Internet. Following main elements can be extracted from each flow or request record: client host (internal IP, username and device ID), server domain and server IP address. Additional information can be extracted from some records (such as HTTP(S) header elements).

Our work builds a graph (or, rather, several graphs) that represents a set of connections between one monitored network and Internet. Then we use the analysis techniques discussed below to identify peer-to-peer activity and infected hosts in the network. Compared to traditional IDS systems or anomaly detectors, this approach is harder due to the huge data volumes (our system processes more than 10 billion flows every day) and associated computational cost. Therefore, our techniques need to be meticulously optimized for efficiency and scalability and many of the design choices have been driven by efficiency constraints.

1.1 Research problems

The main topic of this dissertation is **identification of collaboration structures from client-server connection information**, with particular emphasis on discovery of malicious communities. This can be decomposed to two specific research problems:

- *RP1: How to detect all P2P networks in a monitored physical network?*

  P2P protocol is a popular choice with malware authors to be used as a C&C channel. Several of the biggest botnets observed recently used P2P as their C&C channel [117]. Malicious P2P networks aside, even legitimate P2P networks may have a detrimental effect on the existing anomaly detection approaches [53]. Therefore, traffic related to P2P traffic should be identified and processed separately. Categorization of detected P2P traffic is then needed to enable a proper response of network administrators.

- *RP2: How can we use connection information to reveal C&C structures of malicious botnets?*

  Malware C&C structures are becoming increasingly dynamic, as the malware operators counter security industry approaches based on blacklisting known malicious domains. It is therefore important to automatically follow the migration of C&C servers. Connection information between infected hosts and external domains can be mined to discover the identity of migrated malicious servers and should be robust w.r.t. adversary manipulation. Intuitively, when a thousand infected hosts visit a single domain, this domain should be considered suspicious - unless it is called google.com. The real research problem is to identify the malicious domains between the millions of legitimate servers, given the immense variety of network communication patterns.

1.2 Key contributions

In the following we propose solutions to the outlined research problems.

1. **Detection of all P2P networks active in a monitored network and their classification** [65] (Chapter 3) We present a unified solution to identify Peer-to-Peer (P2P) communities operating in the monitored network. We propose an algorithm that is able

to 1) progressively discover hosts in the monitored network that cooperate in a P2P network and to 2) identify that P2P network. Starting from a single known host, other hosts participating in the P2P network are identified through the analysis of widely available and standardized IPFIX (NetFlow) data. Instead of relying on the analysis of content characteristics or packet properties, we monitor connections of hosts that are known to participate in a P2P network and then progressively discover other hosts through the analysis of their shared contacts outside of the monitored network. The algorithm presented in Chapter 3 is able to identify a large range of both legitimate and malicious P2P networks. It is highly scalable and the use of standard meta-data without access to traffic content makes it easy to deploy and justify from privacy protection perspective.

2. **Identification of additional malicious domains based on the information about known malicious domains** [66] (Chapter 5) Probabilistic Threat propagation is one of the leading algorithms to discover malicious communities in positive-unlabeled data. So far, it has not been used on the data capturing connections between clients and servers. Our contribution is the application of Probabilistic Threat Propagation to a unipartite graph that we have designed to capture the connection data in a highly condensed and privacy-sensitive form. The use of connection data instead of ip-domain hosting information used in prior art [19] allows us to find different malicious domains using the same algorithm. At the same time, our focus on the connection data makes the detector harder to evade.

3. **Definition of similarity for HTTP servers** [66] (Chapter 6) We propose two distinct behavioral models to represent domains based on the HTTP request towards them. We also propose corresponding kernel functions that can be used to calculate either similarity, distance or can be used directly in the kernelized versions of classifiers or clustering algorithms. Behavior similarity can be used to extend the information about the maliciousness of a domain inferred using the algorithm from Chapter 5. Behavior similarity can therefore find other domains used for C&C of a specific botnet even without a single mutual connection on the set of monitored hosts. This significantly increases the number of malicious domains discovered by our algorithm from Chapter 5.

Chapter 2

Related work

In this chapter, we provide overview of

- peer-to-peer (P2P) networks,
- P2P-base command & control channels (C&C),
- methods to detect P2P networks,
- guilt by association approaches and
- behavioral modeling of HTTP traffic.

Each section should provide a comprehensive overview of prior art.

2.1 Basic graph notation

In this section, we introduce basic graph notation used throughout the thesis. It can be read prior the following chapters, as well as used as a reference if necessary.

Weighted undirected graph is defined as

$$G = (V, E, w) \tag{2.1}$$

where V is a set of nodes, $w : E \rightarrow R$ is an edge weight function and $E \subseteq \{\{u, v\} | u, v \in V\}$ is a set of edges.

A graph with a specific layout, a bipartite graph, can be defined as

$$B = (U \cup V, E, w) \tag{2.2}$$

where U and V are set of nodes called partitions, $U \cap V = \emptyset$, $E \subseteq \{\{u, v\} | u \in U, v \in V\}$ is a set of edges and $w : E \rightarrow R$ is an edge weight function. Bipartite graph can be further generalized in n-partite graph:

$$N = (\bigcup_i V_i, E, w) \tag{2.3}$$

where U_i are partitions, $U_i \cap U_j = \emptyset, \forall i, j$, $E \subseteq \{\{u, v\} | u \in U_i, v \in U_j, i \neq j\}$ is a set of edges and $w : E \rightarrow R$ is an edge weight function.

For any graph G we can define a *node neighborhood function*

$$\mathcal{N}(u) = \{v|\{u,v\} \in E\}, \forall u \in V \tag{2.4}$$

that returns neighbors, i.e. adjacent vertices, of u in the graph G.

Weight function w can be also specified using an adjacency matrix $W \in R^{|V|\times|V|}$ where

$$W_{ij} = w(v_i, v_j). \tag{2.5}$$

The adjacency matrix is always symmetric for undirected graphs. The same does not hold for directed graphs.

Closely related to the adjacency is the degree matrix $D \in R^{|V|\times|V|}$. Its elements are defined as

$$\begin{aligned} D_{ii} &= \sum_j W_{ij} \\ D_{ij} &= 0, \forall i,j; i \neq j. \end{aligned} \tag{2.6}$$

D is thus a diagonal matrix.

2.2 P2P networks

In this section we explore various peer-to-peer network architectures that serve as a basis for all P2P applications, including botnets with P2P-based C&C. In this work, we distinguish three types of peer-to-peer architectures [81]:

- structured,
- unstructured,
- hybrid.

2.2.1 Structured Architectures

Structured P2P networks have a known structure and it is undeniably easier to quantify their network performance. Among the most noted structured architectures are the classic DHT (*distributed hash tables*) designs: *Chord*, *CAN*, *Pastry* and *Tapestry.*

Chord Chord, introduced in [130], is among the first attempts to design a robust P2P architecture for data storage and lookup. Chord uses a consistent hash function (SHA-1) to describe data that is shared within the overlay. Every piece of data (e.g. file) shared is assigned an m-bit identifier. The hash function of choice should distribute hashes uniformly since it is critical for chord performance. Every node in the Chord network is also assigned an m-bit identifier, and the node is responsible for providing files whose identifiers are closest to his own. The identifier space is organized in a ring — a circular list of numbers from 0 to $2^m - 1$. It is only very rare that the Chord network contains as many nodes as there are available identifiers, therefore we define successor(i) to be the node with closest higher identifier than i. In this ring, every node maintains a small routing table, called *finger table*, which contains $O(\log N)$ items. Specifically, a node with identifier n has nodes $s_i = \text{successor}((n + 2^{i-1}) \bmod 2^m)$,

$1 \leq i \leq m$ in its finger table. The i-th entry contains interval $[s_i, s_{i+1})$ thus it is easy for node n to determine which peer is responsible for a given range of identifiers. With this setting, the Chord requires each node to keep $O(\log n)$ other nodes in the finger table; the lookup complexity is also $O(\log N)$ and node joining or leaving a network entails only $O(\log^2 N)$ messages.

CAN CAN, which stands for *Content Addressable Network* was first presented in [116]. In CAN, nodes are spread into a d-dimensional Cartesian space and each node is responsible for a part of the space. For each data with key K, the key is hashed into d values and is then represented by a point in the coordinate space. The data is stored by the node that is responsible for the portion of the coordinate space in which the point resides. In the CAN architecture, each node needs to keep record of addresses of its neighboring nodes. In this d-dimensional space, two nodes are neighbors if their regions of responsibility overlap in *exactly* $d - 1$ dimensions. It follows, that each node keeps record of $2d$ neighbors. Routing in this architecture is straightforward — node just forwards the request for a specific resource to the neighbor that is closer (in terms of Cartesian distance) to the requested resources coordinates. If all the regions belonging to the N nodes in the network are same, the mean routing path length is $(d/4)(n^{1/d})$. This architecture is also resilient to random node failures, since there is more than one pathway between any two points in the coordinate space.

Pastry & Tapestry The remaining two prominent DHT designs are Pastry [118] and Tapestry [150]. In Pastry, each node identifier is a 128-bit number from a circular space. Every node keeps a routing table with $\log_b N$ rows, with b being a system specific integer value. In each row there are addresses of other nodes matching prefix one bit longer than those in the previous row. Routing is done by searching for nodes in the routing table that have the largest common prefix with the requested resource. Message routing in Pastry is $O(log_b N)$. The same routing complexity holds for Tapestry.

Interconnection of Structured Architecture There have also been attempts to interconnect these networks. For example, one may use nodes that are capable of communicating using two or more protocols and use those as a proxy between two networks [44]. This requires introduction of *supernodes* that have enhanced capabilities compared to other nodes in the network. Another option is to use so-called *truncated pyramid* where different P2P overlays are interconnected by trees [102].

2.2.2 Unstructured Architectures

In unstructured P2P networking architecture, nodes simply choose their peers randomly. The prominent examples of unstructured P2P networks are Gnutella or BitTorrent. These two are also examples of two different approaches to the unstructured overlay network construction — Gnutella is purely random without any central authorities and BitTorrent utilizes a centralized tracker (which is not necessary in the latest protocol versions). The two protocols are further described in Section 2.2.3. Overall, the unstructured P2P networks offer several good characteristics, like low diameter and network resilience to node churn and random failures.

Hybrid Architectures

Hybrid architectures try to bring the best from both worlds. We do not deal with them later in the text, but just to give one example we may refer reader to [101]. In it, a DHT ring of smaller unstructured networks called load balancing clusters is proposed. It is a two-layer network, with supernodes in the upper layer. Each supernode participates in the DHT network and manages a smaller unstructured network. This load balancing cluster shares the workload put on the managing supernode.

2.2.3 Popular peer-to-peer networks

In this section, we shortly describe the following P2P protocols:

- Skype,
- BitTorrent,
- Gnutella,
- Kademlia.

One can find studies of BitTorrent in [98], BitTorrent's DHT [37], KAD (which is based on Kademlia) in [127, 86] and Gnutella in [84, 3, 92]. In the following we provide their short description.

Skype Skype protocol is proprietary and has not been publicly described yet. Moreover, Skype encrypts all its communication, it is thus difficult or even impossible to reverse engineer it. It is believed that Skype P2P protocol is based on the FastTrack P2P protocol and closely related to KaZaa [49]. While this has not been proven directly, there are stark similarities between them, e.g. both FastTrack and Skype use two layer overlay architecture with supernodes in one layer and ordinary nodes in the other.

A basic description of Skype architecture and an analysis of the message workflow can be found in [34]. Authors have also created several signatures to detect Skype traffic based on their findings. Detection using signatures requires packet payload inspection, which is processor intensive and introduces significant latency. Therefore, it is not suitable for high-bandwidth networks. Another analysis of Skype can be found in [49]. In this work, the authors also provide an analysis of user-behavior and estimate network workload generated by Skype.

Another approach to detect Skype was proposed in [15]. Two detection methods are offered – one based on Pearson's Chi Square test and the other using Naive Bayes Classifiers. Both methods require packet payload inspection and therefore their deployment on backbone networks can be problematic. These two methods were further modified and their performance analyzed in [121]. Packet inspection and flow properties are also used for detection in [148]. Another approach, introduced in [139], is based solely on the flow data from the network.

An interesting Skype detection method designed specifically for 3G networks can be found in [134]. The authors exploit a specific property of 3G networks whereby every packet is associated with a specific mobile device (and thus user). However, it is still necessary to inspect packet payloads in order to detect active Skype node in the network.

Skype is analyzed from a rather different perspective in [138] where two Skype outages are analyzed and compared. One of the interesting observations of this work is that Skype temporarily centralized its topology by the introduction of so-called *mega-supernodes*, consequently speeding up the network recovery.

BitTorrent Bittorrent is a popular file sharing solution nowadays. It differs from another P2P network in that it joins a particular overlay only when it wants to download a file. This overlay is called a *swarm* and has form of a mesh. Before a node joins a swarm, it must retrieve set of already participating nodes from a tracker to bootsrap. This might be considered a single point of failure; as a countermeasure distributer tracker functionality over DHT (actually Kademlia-based) was added to the later versions of Bittorrent protocol. To enable multi-source downloads, the shared file is split into segments and chunks. Each participating node can download different chunks from different users. They key mechanisms of Bittorrent are *choke algorithm* and *chunk selection algorithm.* Choke algorithm is an incentive algorithm to promote fair usage of the network. In basics, every 10 seconds the peer decides to which of its neighbors it will send data. These neighbors are selected based on they willingness to share and speed of download from them. Chunk selection algorithm controls in what order are the chunks downloaded. Since the nodes exchange information about already downloaded parts, each node has an information about rarity of particular chunks. Then it chooses to download the rarest chunks first to ensure it will be able download the whole file. This was of course only a very short introduction, but since it is not the main aim of our work, we will not go any further. An interested reader may find more detailed information in [83].

Bittorrent from the point of view of an ISP was described in [124]. According to their analyses, flashcrowds (which are often the reason for the scalability studies in Bittorrent) appear always around midnight. Another interesting observation is that Bittorrent DHT dominates the BitTorrent traffic, even compared to data transfers. This suggests that BitTorrent has a high overhead when using distributed trackers.

The distributed tracker functionality over DHT is studied in [63]. They investigate why the real deployment of distributed trackers is not as successful and optimal as simulations suggest. According to their results, it seems that majority of Bittorrent nodes using DHT are behind NAT, therefore are unreachable by external hosts which severly disrupts the DHT overlay.

Last but not least, a detailed analysis of Bittorrent traffic on the both application- and flow- level in a broadband network can be found in [98]

Gnutella Gnutella is an unstructured P2P network used for file sharing. In its first version, it used purely random peer selection. Upon start of the node, it flooded the network sending *ping* messages. If it contacted another Gnutella host, it received a *pong* response. Searching in Gnutella network is done by flooding the overlay network with TTL-aware (time to live) query messages, where each query is forwarded to all known peers unless the TTL is 0 in which case it is discarded. A node usually had 4 to 5 peers. With the introduction of a new version of the protocol, so-called *supernodes* were introduced. Supernodes maintain many more connections to other supernodes. Gnutella overlay these days has a tree-like structures.

Many studies of Gnutella have been published to this day. A study concerned with structural properties of Gnutella overlay can be found in [132]. This study contradicts many other studies of Gnutella by claiming that its overlay in fact does not follow a power-law degree distribution. It also states that Gnutella forms a small-world network.

Another work focused on Gnutella focuses on its messages inter-arrival times [97]. It argues that although many previous works assume that Gnutella has inter-arrival times following the Poisson distribution, it is in fact not true. They show that inter-arrival times are much better approximated by bi-modal Poisson distribution.

Observations from a 15-months long study of the two-tier Gnutella overlay network are described in [115]. They made an interesting observation, that while locality is not enforced in Gnutella, most of the connections exhibit a strong bias towards the intra-continent connectivity. Besides that, they observed that the architecture was starting to lose balance during its evolution and rapid increase in the number of nodes. However, changes in the client application managed to reverse the effect and the network regained its stability.

Kademlia Kademlia is a DHT P2P network algorithm, first introduced in [93]. It communicates over UDP. We will not describe internals of the Kademlia implementation, however we point out to several interesting works dealing with Kademlia.

An implementation of Kademlia called KAD was monitored for over a year in [128]. One of the key observations is that user session times in the network are Weibull distributed, exploiting which may improve the publishing mechanism in KAD. Also, the measurements imply that KAD ID, assigned to each user is not static for all users (as was originally assumed). Some users change their KAD ID as often as once a day.

Kademlia is also known to be implemented as an overlay for several P2P botnets. Two articles dealing with Kademlia and its advantages and/or disadvantages for botmasters are considered in Section 2.3.2.

2.2.4 Peer Selection in P2P Networks

Peer selection in P2P networks is a broad and complex topic and we are interested only on a small practical implication of the chosen peer selection strategy. Therefore, we provide only a brief overview of the topic. Peer selection strategy can be either uniform random or biased (random). In case of a uniform random peer selection, node selects several peers randomly using uniform sampling. Such peer selection is used for example in BitTorrent to obtain the first set of peers from a tracker when joining a swarm. The goal of biased peer selection strategies is either to improve data locality [137, 123, 145], minimize delay in multimedia streaming [143], adapt to heterogeneous bandwitdhs within the population [143, 11] or solve scalability issues of existing networks [23].

2.3 P2P as a botnet C&C

2.3.1 Viability of P2P networks for C&C

In this section we review related work that analyze the ability of various P2P architectures to be used as a C&C channel. The focus lies mainly in the ability of the P2P protocols to disseminate new commands, update malicious binaries, and to withstand targeted attacks and enumeration attempts.

Researchers do not have direct global access to the P2P overlay networks used by botnets, therefore, when studying properties, mitigation strategies, and potential improvements of P2P-based botnets, they turn to theoretical models that approximate real P2P overlay networks. Structured P2P networks are commonly represented by the *Erdös-Rényi* random graph model and unstructured P2P networks are represented as *Barabási-Albert* scale free network model. This choice is based on the observed degree distributions of nodes in graph representations of P2P networks. Structured P2P networks tend to keep uniform degree distribution of its nodes, which is typical the case of the Erdös-Rényi model. On the other hand, unstructured networks typically have scale-free degree distribution, which is typical for Barabási-Albert scale free model. Published works sometimes also analyze *Watts-Strogatz* small world model, despite the fact that overlay networks with the similar structure are extremely rare; one example of such network is Zindos [27].

Research community is understandably interested in strategies to disrupt botnets. Across the literature, three disinfection strategies are considered when evaluating resilience of a P2P overlay:

- random disinfection,
- tree-like disinfection,
- globally-optimal disinfection.

Random disinfection is equivalent to a random node failure in the study of resiliency of general peer-to-peer networks. In such a case, an infected computer is cleaned once a malware is detected without using any knowledge that can be gained from the malware instance. In the tree-like disinfection, all the peers known to the already cleaned instance are also cleaned and so on. Globally optimal disinfection assumes knowledge of the whole overlay and picks nodes to be disinfected in the order of their importance to the network, e.g. highly connected nodes first.

To what extent a disinfection strategy disrupts a peer-to-peer overlay is determined by various metrics. For example, effectiveness, efficiency and resilience of P2P overlay network were evaluated in [27]. By effectiveness of a botnet, the authors understand an estimate of overall utility to accomplish a given purpose. By efficiency the authors understand the communication efficiency within the overlay network. Finally, robustness represents to ability of the overlay network to withstand the removal of peers. Effectiveness is measured by two metrics – size of the largest connected component and average bandwidth of the infected clients; efficiency is measured by the inverse geodesic length [55], and robustness is measured by the local clustering coefficient. The authors evaluate these properties of three theoretical models:

- *Erdös-Rényi* random graph model,
- *Barabási-Albert* free scale network model and
- *Watts-Strogatz* small world model.

The conclusion is that the Erdös-Rényi and Watts-Strogatz behave similarly if the targeted disinfection is being performed, outperforming the Barabási-Albert model. On the other hand, the Barabási-Albert model is more resilient to the random disinfection. As a conclusion, Erdös-Rényi is considered as a good choice for overlay network.

In [28] a similar study is performed on a mix of theoretical models and existing peer-to-peer overlays:

- Overnet (Kademlia/DHT),

- Gnutella,
- Erdös-Rényi random graph model,
- Barabási-Albert scale free network model

using different performance measures – *reachability from a given node*, *shortest path length sets* and *diameter of the network graph* and the three disinfection strategies. Experiments in the paper suggest that Gnutella, as an unstructured peer-to-peer network can be can be very well approximated by the Barabási-Albert model. On the other hand, Overnet, as a structured peer-to-peer network is not approximated correctly by the Erdös-Rényi model, mainly due to the degree restriction in the Overnet, which is not present in the theoretical model. The conclusions of the paper seem to be consistent with [27]: Gnutella & Barabási-Albert model are more resilient to the random disinfection, while Overnet and Erdös-Rényi model are more resilient to targeted and tree-like disinfection. However, Overnet is even more resilient than the Erdös-Rényi model. Similar observations were made in [6].

Another types of attacks on P2P botnets do not rely on node removal, but rather try to shut down the overlay itself. They were analyzed on an example of Kademlia in [50]. In it, several measures to identify important nodes in the overlay are proposed, among them

- degree centrality,
- eigenvector centrality [17],
- betweenness centrality,
- closeness centrality,
- routing–based centrality.

According to the experiments, none of the proposed measures fared particularly well in identifying nodes in the overlay that see most of the P2P network's traffic. Even for the betweenness centrality, which appears to be the best metric for choosing the nodes that see the most traffic, this metric achieved only around 13% overlap of top 1000 nodes with respect to the portion of observed traffic with top 1000 nodes chosen by the measure. When comparing top 2000 nodes, this number jumps to around 22%.

Further on, betweenness centrality was used as a measure to identify the most important nodes in the overlay for the three mitigation techniques:

- poisoning attack [56],
- Sybil attack [32],
- eclipse attack [20].

In the poisoning attack, several nodes controlled by an attacker join the P2P network and publish keys belonging to the content being attack. Nodes controlled by the attacker provide fake content under those keys. In the case of Peacomm botnet, the attacker would publish keys corresponding to the URL for download of a new malware binary with an empty string. Other nodes in the network that do not belong to the attacker would slowly rewrite the original keys with the fake content. In the experiment, several poisoning attacks were conducted for various numbers of poisoned nodes. It is shown that while betweenness centrality is not a good criterion for traffic observation, it does fairly well in choosing nodes for the poisoning attack. Even with only 10 initial bots, the authors were able to poison 25% of the population (of 2000); with 100 initial bots, they were able to poison half of the population. However, with 400 of the initially infected nodes, the poisoning hits the plateau and adding more poisoned nodes makes almost no difference.

In the Sybil attack, several fake nodes with various IDs are added to the node population. These nodes behave differently than the legitimate nodes. In the Sybil attack, the Sybil nodes are passive, thus do not forward any messages routed to them. The results of their experiment suggest that only a small fraction of traffic is forwarded to the Sybil nodes, thus these are not particularly effective in disturbing the operation. On the other hand, when intercepting traffic, the Sybil nodes learn the identity of other bots which suggests that this approach of enumerating botnet can be quite successful. It has also been shown, that it is more efficient to insert addresses of a few Sybil nodes into routing tables of many legitimate nodes, rather than inject addresses of many Sybil nodes into routing tables of only a few legitimate nodes.

In the eclipse attack, the attacker tries to use his attack nodes to isolate particular nodes from the rest of the node population; this attack was not successful at all in [50].

Formal models of P2P overlays and attacks on them were described in [117]. Two main types of attacks on P2P-based botnets were considered:

- intelligence gathering,
- disruption and destruction.

Intelligence gathering represents the enumeration attacks, goal of which is to reconstruct the graph representation of the overlay network (*crawling*) or at least identify IP addresses of machines participating in the P2P network (*injecting sensor nodes*). Attacks classified as disruption and destruction are *partitioning*, *sinkholing* and *poisoning*. Goal of the partitioning attack is to split the overlay into several disconnected parts, which is similar to the eclipse attack. The goal of sinkholing is to replace peer list of all participating peers in the P2P overlay with sinkholed hosts, and thus effectively destroying the overlay. Poisoning attack is basically the Sybil attack described above. In the article, enumeration attacks are attempted on **all active** P2P-based botnets. Disruption attacks are modeled theoretically using the formal models proposed in the paper. Authors observed that in the enumeration attack, using sensor nodes in addition to crawling considerably increased number of observed peers. It was also observed that while Zeus and Sality are highly resilient botnets, Kelihos and ZeroAccess contain weaknesses that can be used to disrupt them.

2.3.2 Peer-to-peer as a botnet C&C channel

Botnets have been utilizing P2P overlay as their C&C for almost two decades now. A list of early botnets accompanied by the history of peer-to-peer and their introduction as the C&C channel can be found in [48]. In short, one of the first P2P botnets was *Sinit* that appeared in 2003 and used random scanning to find peers. In 2004 another botnet, *Phatbot*, emerged whose C&C was based on WASTE [2]. Two P2P botnets appeared in 2006, *SpamThru* and *Nugache*. SpamThru used a custom P2P protocol; and we mention Nugache in a bit more detail later in this section. More recent botnets that use peer-to-peer networks are described in [117]. Figure 2.1 provides the information about the life span of the modern P2P botnets.

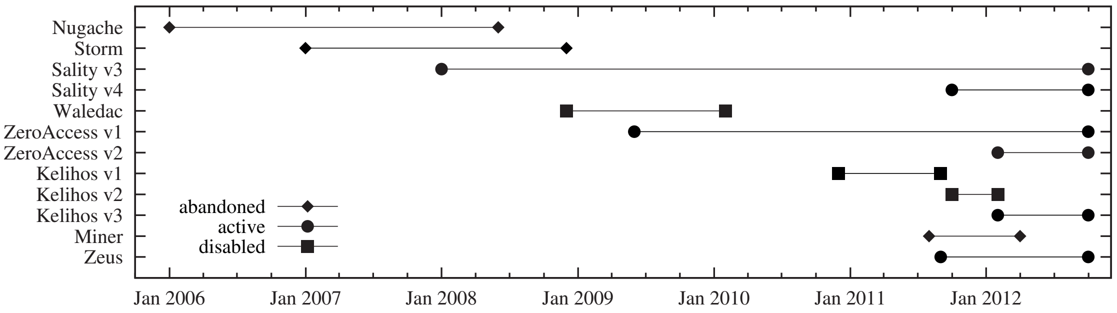


Figure 2.1: Visualization of life span of known P2P-based botnets. Adapted from [117].

All identified peer-to-peer based botnets were studied by the research community; for example, one can find studies of Nugache [31], Peacomm/Storm [68], ZeroAccess [144], Sality [38], Waledac [129], Kelihos [71] and Miner [107]. In the following, we provide a short overview of several peer-to-peer based botnets. Their life span is visualized in Figure 2.1.

Nugache

Nugache first appeared in 2006 and was easily identified by AV software because of several known signatures it contained and use of port 8/tcp to listen for incoming connections [31]. However, Nugache became much more elusive after it was updated to a newer version which ceased to use port 8 and moved to a randomly chosen high port. Nugache first used a combination of P2P and centralized C&C architecture and all connection within the P2P overlay were encrypted. With time, the malware shifted completely to P2P, abandoning its IRC channels. At the time, the two C&C architectures were used together, the P2P overlay was used only for rallying bots to visit the botmaster's IRC server when he needed them [31].

Another interesting thing is that when taking snapshots of peers of the one particular bot in different points in time, the sets overlap only marginally. This might be due to the churn in the network (new bots appearing, old bots being eliminated) or due to the use of dynamic IP addresses. Also, there is only a very small overlap between set of bots communicating with the particular observed bot over P2P and those encountered on the IRC channel. It is suspected, that the P2P network was designed in such a way, that upon observing a bot and monitoring its connection, one would not see more than a 10% of all bots in the botnet [31]. Also, connections within the P2P network are lightweight to avoid detection.

While monitoring a Nugache bot, the authors also witnessed several DDoS attacks and commands to infect other machines. The bot also received several commands to scan for vulnerabilities on hosts in the reserved ranges (reserved IP ranges containing non-routable IP addresses). It is interesting though, that scanning and propagation commands were received through the IRC channel, and command to perform DDoS was received over the P2P overlay.

Storm (Peacomm)

Peacomm, more commonly known as *Storm Worm*, uses two-step injection. In the primary injection, the base binary with the capability to join a P2P network is installed after the user runs an infected application. The bot then connects to the P2P network and downloads a secondary injection, that gives him the actual malicious capabilities (SPAM campaigns, DDoS campaigns, ...). Peacomm employed two distinct P2P overlays. First from January to October 2007 — this network co-opted Overnet (based on Kademlia), shared the same ID space, message types and semantics as Overnet but employed a different Routing algorithm. This network was followed by new, encrypted network which run separate from Overnet [68]. Interesting thing about Peacomm is that it uses P2P network only to search for the URL from which it can download a new version. In comparison to Nugache, where bots kept low profile and did not heavily communicate within the P2P network, Peacomm bot contacted or was contacted by enormous number of peers as can be seen in Figure 2.2. At some point, Peacomm created a layered P2P overlay, with bots in possession of public IP addresses being considered more valuable and placed in the upper layer. The lower layer contained bots behind NAT that were used for example for spamming campaigns [68].

Sality

Sality is a file infector that appeared in several versions. Since 2008 it uses P2P to distribute URL of malware payloads to be downloaded and executed by infected hosts. Upon infection, infected binaries join Sality P2P overlay using the 1000 bootstrap peers embedded in the binary. There are two parallel versions of Sality that use P2P overlay for their C&C – v3 & v4; v4 fixes a critical vulnerability of v3 where files downloaded by the malware are not digitally signed and thus can be tampered with. P2P overlays of these two versions are not interconnected [117].

Waledac

Waledac appeared in 2008 and used a combination of centralized and P2P architecture. Waledac C&C architecture was split into three layers. The lowest layer contained spammers, that were infected machines with no public IP address and which, as the name suggest, sent out spam. The fact that these were hidden behind NAT complicated their identification. In the next layer were *repeaters*, which were the entry point for newly infected machines as well as a place where *spammers* went to receive instructions. Repeaters also hosted fast-fluxing domains used by the botnet. While *spammers* and *repeaters* formed an actual P2P network, the highest layer containing backend servers was centralized and only repeaters had access to it [129].

Kelihos (Hlux)

Kelihos existed in three variants. The first variant was used for spamming and ID theft. It was blackholed in September 2011. The second version of Kelihos appeared shortly after, with mostly the same structure but new capabilities, such as performing DDoS attacks, stealing bitcoin wallets, intercepting passwords. This version of Kelihos was blackholed in March 2012. The third version was sinkholed in 2013. Kelihos has the same structure of P2P overlay as Waledac, therefore it is sometimes considered to be its successor. It was spread by social networks, especially by Facebook worm which lured victims into downloading a photo album [109, 71].

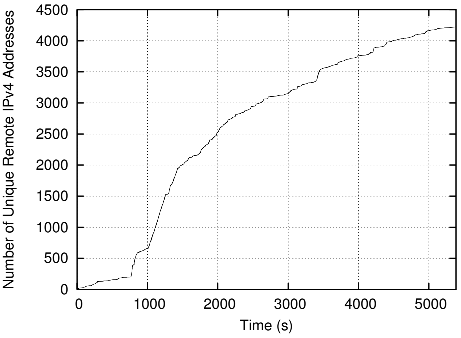


Figure 2.2: Number of contacted peers along with time for a Peacomm (Storm Worm) malware instance. We can see that Peacomm is very active and within 100 minutes it contacted or was contacted by more than 4000 peers. [68]

ZeroAccess

Existence of ZeroAccess was first reported in 2011. Its size was estimated to around 1 million of infected machines [144]. It is a malware dropper and uses P2P network for its C&C channel. Thanks to its P2P-based C&C, it survived a takedown attempt by Microsoft in 2011 [111]. It exists in 2 variants and has 7 separate P2P overlays for its various minor versions – i.e. 32 bit and 64 bit versions use different overlay networks [117]. It has been estimated that ZeroAccess can bring up to $\$100,000$ of profit to its owners daily [144].

Miner

Miner was a botnet using unstructured P2P overlay for its C&C and operated between August 2011 and March 2012 [117]. It was split into two disjoint networks consisting around $38,000$ nodes [107]. One of its purposes was to mine bitcoins [142].

Zeus

Zeus has existed in several variants. Its first variants used centralized architecture and were target of many studies and take-down attempts. One of the variants switched to an unstructured P2P overlay in September 2011. Zeus uses its overlay to relay commands, stolen data, configuration and binary updates. One of the monetization techniques of Zeus is theft of online banking credentials [117].

2.4 Botnet detection

In this section, we provide an overview of related work dealing with botnet detection. Two specific approaches that are the closest to our contribution are discussed in Sections 2.4.1 and 2.4.2. We then follow with the overview with other related work in the field. Categorization of related work can be found in Table 2.1.

2.4.1 Monitoring mutual contacts

A method how to identify local members of a botnet that uses *unstructured random P2P network* for its overlay if one bot from the botnet is already known was proposed in [26].

Table 2.1: *Overview of methods and goals of the related work.*

		goals			
		P2P properties	P2P-based C&C properties	P2P detection	botnet detection
used methods	empirical studies	[3, 37, 78, 84, 86, 92, 98, 108, 127]	[48, 117]	-	-
	theoretical analysis	-	[27, 28, 50, 117]	-	-
	thresholding	-	-	[10]	-
	graph methods	[58, 59]	-	[26, 42, 57, 58, 59, 61, 73, 99, 42, 149]	-
	statistical approaches	-	-	[100, 151, 10]	[42]
	persistence	-	-	-	[46]

Identification is done by post-mortem analysis of the network traffic. Once one bot is discovered in the network, the traffic preceding its discovery is collected (say 24 hours) and analyzed. The detection of other hosts is centered around the term *mutual contact*. We say two hosts in the network have a mutual contact if they both exchange traffic with the same host from outside of that network.

Before we move on, let us define a *mutual contacts graph*. It is a graph $G = (V, E, w)$ where vertices V in the graph represent hosts in the protected network (for example identified by an IP address) and E is a set of weighted undirected edges; w is a weighting function which assigns each edge weight equal to the number of mutual contacts of the two incident nodes. When creating the graph, only mutual contacts are that were contacted by at most k local hosts are considered. This removes nodes representing popular servers that create a lot of edges in the graph and do not bring any valuable information.

Given the identity of the first infected host, referred to as *seed node* further on, mutual contacts graph is created and on it so called *dye-pumping algorithm* is applied. Dye-pumping algorithm calculates the likelihood of belonging to the same P2P network as the seed node, for all nodes in the graph. It is an iterative algorithm that can be illustrated as pumping dye into the graph through the seed node. From the seed node, the dye distributes to the other nodes in the graph proportionally to *dye-attraction coefficient* γ_{ji}. γ_{ji} specifies what portion of the dye arriving to node j will be distributed to node i in the next iteration. Its value is determined as

$$\gamma_{ji} = \frac{E_{ji}}{(D_i)^{\beta}}$$

where E_{ji} is weight of the edge connecting nodes j and i, D_i is degree of node i (number of adjacent vertices) and β is *node degree sensitivity coefficient.* This coefficient limits the amount of dye arriving to highly connected nodes, since it is unlikely that these nodes are parts of the botnet (which authors assume has only few bots in the protected network).

The algorithm has three inputs:

- matrix $E = (E_{ji})$ representing all edge weights,
- index s of the seed node N_s,
- i — the maximum number of iterations of the algorithm.

With these inputs at hand, the algorithm first computes the dye-attraction coefficients and forms the transition matrix $\mathbf{T}$ such that:

$$\mathbf{T}(i,j) = \gamma_{j,i} = \frac{E_{ji}}{(D_i)^\beta}$$

where $i, j \in 1, ..., v$ and $i \neq j$. We set $T(i,i) = 0$ and finally v is number of nodes in the graph. This matrix is further normalized so that all columns sum to 1. The algorithm further needs *dye-level vector* L, where $L(i)$ specifies how much dye has accumulated in the i-th node. Before the first iteration, all the dye is in the seed node and other nodes have no dye in them, therefore:

$$L(i) = \begin{cases} 1, & \text{when } s = i \\ 0, & \text{otherwise} \end{cases}$$

At each iteration, dye from nodes that have it is diffused to their neighborhoods. Amount of dye at each node can be calculated as

$$L(i) = \sum_{j=1}^{v} \mathbf{T}(j,i)L(j)$$

which is equivalent to $L = \mathbf{T}L$. After each iteration, L is updated, such that $L(s) = L(s) + 1$ and then normalized. After i iterations, L indicates the confidence for each node that it belongs to the same P2P network as the seed. We can define some threshold *thr* and nodes with confidence exceeding this threshold are returned by the algorithm as the nodes in the same P2P botnet as the seed node. The algorithm summary can be found in Algorithm 1. While this algorithm has cubic complexity, matrix $\mathbf{T}$ and vector L are usually sparse and one can take advantage of this.

The paper also discusses theoretical grounds on which the detection method is based. It is shown that for P2P networks where each peer selects its neighbors using uniform random peer selection, the probability that two hosts in the network share a mutual contact is surprisingly high. At the same time, any biased peer selection strategy increases this probability. It is also shown that assuming uniform random peer selection, edge probability p_e between any two nodes can be computed using hypergeometric distribution, so that

$$p_e = 1 - \frac{\binom{C}{0}\binom{B-1-C}{C}}{\binom{B-1}{C}}$$

This probability rises quickly as the number of contacted nodes increases. Besides the probability of edge, its weight is also important for the algorithm to work correctly. In the random

Algorithm 1 Dye-pumping algorithm.

```
function DYE_PUMPING($\mathbf{E}$, $s$, $maxIter$)
    $\mathbf{T} \leftarrow computeTransitionMatrix(\mathbf{E})$
    $tex\bar{t}bfT \leftarrow normalize(\mathbf{T})$
    $L \leftarrow [0, 0, ..., 0]^{tr}$ initialize L as a zero vector
    for $iter = 1 : maxIter$ do
        $L(s) \leftarrow L(s) + 1$ Pump dye from the seed node
        $L \leftarrow \frac{L}{\sum L(i)}$ Normalize dye level vector
        $L \leftarrow tex\bar{t}bfTL$ Distribute dye in network for one iteration
    end for
    output $L$
end function
```

peer selection model, the probability of a peer to be contacted by two other peers is

$$(\frac{C}{B})^2$$

and since there are B peers at total, we can write the expected capacity of edges $E[Cp]$ as

$$E[Cp] = (\frac{C}{B})^2 B = \frac{C^2}{B}.$$

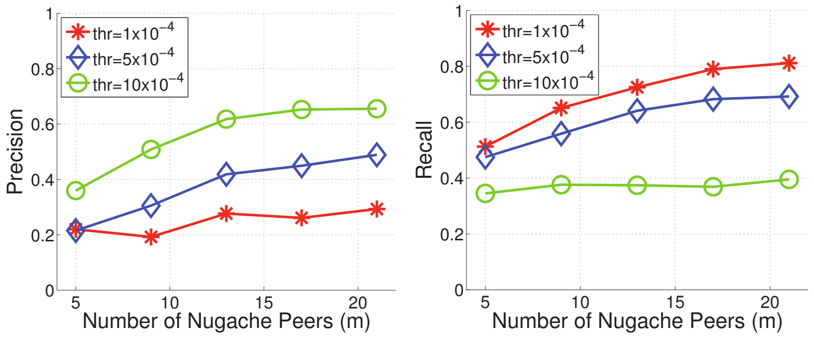
The expected edge weight increases as the number of contacted peers increases regardless of the botnet size. Overall, these two observations suggest that bot will be closely connected by high weight edges. Since we assume random graph model with edge probability being same for each pair of nodes, we are in fact considering Erdös-Rényi model [35, 36]. It has been shown that there is a sharp threshold of p_e exceeding which results in the disappearance of isolated vertices in the graph [36]. In particular this threshold has value of

$$\frac{\ln v}{v}$$

where v is number of nodes in the graph. This implies, that increase in the number of infected hosts in the network eases the detection of those hosts by this algorithm.

We would like to note, that while in [26] it is suggested that Erdös-Rényi model is used to model random unstructured P2P network, in other related work [27, 28], Erdös-Rényi model is used to model structured P2P networks. The reasoning in [26] is that peers choose their neighbors uniformly at random, therefore there is no “intentional” structure and such P2P network can be approximated by random graph model. In contrast to that, reasoning of [27, 28] is based on the similarity of degree distributions seen in the random graphs and real-world structured P2P networks. This discrepancy however does not compromise the analysis of the probability of having a mutual contact provided in [26].

Authors evaluated the detection performance using the Nugache botnet. Nugache can be crawled since its protocol supports querying for the known and recently contacted peers. For the experiment, authors performed several crawls over the Nugache network and based on the data about recently contacted peers they created mutual contacts graph. This graph was overlaid with mutual contacts graph created from background traffic collected for 9 days at the border of NYU network. They overlay was created in such a way that hosts from the



(a) Precision (b) Recall

Figure 2.3: Precision and recall as function of various number of Nugache peers in the network and thresholds to select the suspicious peers. [26]

NYU network were randomly mapped to the Nugache peers. In result, those hosts appeared as being infected by Nugache. For the completeness, we note that before the mutual contacts graph for the background traffic was created, all external IP addresses that did not exchange at least 256 bytes in both directions with at least one internal IP were removed. This way they were able to get rid of scans of the network. Also, all DNS servers were removed from the background mutual contacts graph. It was observed, that the final mutual contact graph is more clustered than comparable random graph (in number of nodes end edges).

For the evaluation, the number of iterations was set to 5, the privacy threshold was set to 2 and $\beta = 2$. The measured values of precision and recall can be found in Figure 2.3. Precision is set to be the ratio of the number of returned Nugache peers to the overall list length and recall is ratio of the returned Nugache peers to all present Nugache peers. In accordance with the random graph theory, both precision and recall rise as the number of Nugache peers in the network increases.

Overall, monitoring mutual contacts is an interesting idea, which we use ourselves in our work. However, the method as presented cannot be used as an automated P2P botnet classification system due to low values of precision and recall. We believe this is caused by the fact that the algorithm works with IP addresses. We present an improvement in this matter in Section 3. Moreover, the graph as presented can be only created offline, therefore it is useful only for post-mortem analysis. If the graph were to be constructed on-line it would have to keep track of contacted nodes, in which case a more convenient way of representing mutual contacts is in question. We present our improvements in this matter in Section 3 as well.

2.4.2 Persistence analysis

In this section we describe a method that was designed to reveal infected but yet dormant hosts in the network. The method measures *persistence* of connections, which will be properly defined later, but intuitively we can consider it a measure of longevity or periodicity of the given connection. The method was proposed with centralized botnets in mind, however, as will be shown in the review of the paper, it managed to detect several instances of a P2P based botnet. The method was originally proposed in [46].

Detection algorithm is designed to be deployed on a host machine. It records all outgoing connections from the host. The endpoint of an outgoing connection is called a *Destination Atom* (referred to as DA further on) and is defined as a vector (service, port, protocol) where port and protocol are properties of the outgoing connection and service is determined as follows:

- if source IP and destination IP belong to different domains, `service` is defined as a second-level domain,
- if source IP and destination IP belong to the same domain, `service` is defined as a third-level domain,
- for higher layer protocols that use ephemeral ports, such as FTP, only one DA is created. This DA covers all ephemeral ports,
- if IP cannot be resolved to a host name, it is used as a `service` name.

The regularity of the connection to a given DA is observed by sliding window W, which is split into n bins. This window is called *observation window* and bins are called *measurement windows*. We can write $W = [b_1, b_2, b_3, ..., b_n]$ where b_i are bins of size $\frac{W}{n}$. To quantify the "regularity" of the DAs, a metric called `persistence` is defined as follows:

$$\mathrm{p}(d, W) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{d,b_i}$$

where d is the DA, W is the observation window and function $\mathbf{1}_{d,b_i}$ is equal to 1 if at least one connection from the host to the DA d occurred during the measurement window b_i, otherwise the function is equal to 0. If we take a look at the definition of the persistence, we see it is just the portion of bins in the *observation window* that had any connection to the DA. This measure gives the same weight to the lightweight connections as it gives to the heavy traffic connections. Thanks to this one can discover "regular" but at the same time lightweight communication. A DA is considered to be persistent, if its persistence exceeds the *persistence threshold*, which is set by means of a prior knowledge or experiments. In [46] authors show on statistics from their experimental data, that less than 20% of all DAs has persistence higher than 0.2. This is in accordance with the assumption that most of the endpoints are connected to very sporadically or only once. The data also show, that persistence threshold should be chosen somewhere between 0.5 and 0.8. Authors choose 0.6 as their threshold value.

There are many different types of botnets from different authors with different C&C setup. As a result, one can't expect all of them to follow the same pattern of connections to its master. Choosing only a single measurement window size might lead to good results detecting one type of botnets, but other botnet families might evade detection. For example, measurement window size of 1 hour cannot detect a bot that contacts its master only once in a day, this C&C connection never becomes persistent and raises any suspicion. For this reason, several *observation window* and *measurement window* sizes are usually used. These ensure that the detection method can detect more types of botnets. When utilizing different *observation window* and *measurement window* sizes, vector (W, b) where W is the *observation window* and b is the size of *measurement window* is called a *time scale*. Persistence is calculated for each *time scale* and the total persistence is the maximum of all persistence values for individual *time scales*:

$$p(d, W) = \max_j p^{(j)}(d, W).$$

Table 2.2: List of sampled botnet binaries with clear identifiable C&C traffic

AV Signature	C&C Type	# of C&C Atoms
Trojan.Aimbot-25	port 22	1
Trojan.Wootbot-247	IRC port 12347	4
Trojan.Godbot.T	IRC port 66659	1
Trojan.Godbot-14	IRC port 6667	2
Trojan.Aimbot-5	IRC via HTTP proxy	3
Trojan.IRCBot-776	HTTP	16
Trojan.VB-666	IRC port 6667	1
Trojan.IRC-Script-50	IRC ports 6662 — 6669, 9999, 7000	8
Trojan.Spybot-248	port 9305	4
Trojan.MyBot-8926	IRC port 7007	1
Trojan.IRC.Zapchast-11	IRC ports 6666, 6667	9
Trojan.Peed-69 [Storm]	P2P/Overnet	19672

The detection algorithm uses whitelist and requires a learning phase. Before the detection phase we need to run the algorithm in a learning phase. In the learning phase, the detection algorithm processes clean traffic and every DA that exceeds the persistence threshold (therefore is persistent) is added to the whitelist. DAs detected in the learning phase are assumed to represent legitimate destinations regularly visited by the user. After the learning phase and creation of the whitelist, the algorithm can start the detection phase. In this phase, every DA that is persistent and is not included in the whitelist rises alarm to the network administrator.

To evaluate the detection performance, authors collected traffic for four weeks from 350 users. Only 157 were active during the whole period and thus the algorithm was evaluated on their respective network traces. To gather malicious traffic from bots the C&C server they manually executed 55 botnet samples in a virtual machine. Out of them, only 12 generated enough traffic to be included in the experiment. Overview of the used malware can be found in Table 2.2. Since the method requires a learning period, the 4 weeks were divided into halves, first half serving for the learning phase and second half (intermixed with the traffic traces of malware instances) to evaluate the performance.

The algorithm was run with measurement window sizes $s \in \{1, 4, 8, 16, 20, 24\}$ and having 10 bins in the observation window. Persistence threshold was chosen to be 0.6. With these setting, all bot instances in the experiment were detected. Specific time windows that raised an alarm for particular bot samples can be found in Table 2.3. Based on the ROC curve in Figure 2.4 we can say that 0.6 is indeed the optimal choice for the persistence threshold. The authors also advertise low number of false alarms — on average, any user receives less than one false alarm every two days.

Surprisingly, despite the fact that persistence monitoring was designed to work with centralized botnets, it was also able to detect STORM bots in the network, which is a P2P botnet. We believe it is due to the fact that STORM might keep its peer list stable thus resulting into persistent connections.

We believe that monitoring temporal persistence as it is proposed in [46] can be considered an anomaly detection — all whitelisted destinations are "normal" persistent destinations and

Table 2.3: C&C detection performance

Botnet	**Persistence**	**Timescale**	**# of DA**
Trojan.IRCBot-776	1.0	(10,1)	1
Trojan.IRCBot-776	0.8	(200,20)	2
Trojan.Aimbot-5	1.0	(10,1)	1
Trojan.Aimbot-5	1.0	(40,4)	1
Trojan.Aimbot-5	1.0	(160,16)	1
Trojan.MyBot-8926	0.6	(160,16)	1
Trojan.IRC.Zapchast-11	1.0	(40,4)	3
Trojan.Spybot-248	1.0	(10,1)	2
Trojan.IRC-Script-50	1.0	(10.1)	7
Trojan.VB-666	0.7	(10,1)	1
Trojan.Godbot-14	1.0	(10,1)	1
Trojan.Godbot.T	1.0	(10,1)	1
Trojan.Wootbot-247	1.0	(10,1)	3
Trojan.IRC.Zapchast-11	1.0	(10,1)	6
Trojan.Aimbot-25	1.0	(10,1)	1
Trojan.Peed-69 [Storm]	1.0	(10,1)	$\gg 1$

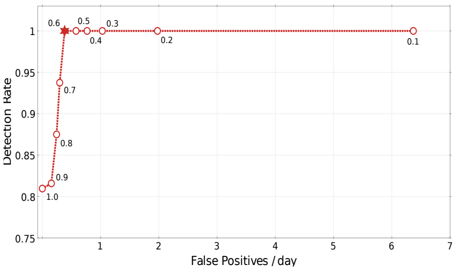


Figure 2.4: ROC curve of the detection performance of persistence measurement method. [46]

any other is an anomaly that rises an alarm. Definition of Destination Atom also brings several challenges. P2P applications, for example, contact their peers by IP address and not by a host name. Of course, Destination Atom can work with IP addresses, but only as a fallback mechanism. Also, a P2P application may connect to plethora of peers, some of them persistently. And at each start of such application, the peers tend to change. This might result in dramatic increase in false alarm rate since the method was only evaluated only on a corporate network, where users are generally well-behaved.

2.4.3 Other methods

A method using a graph of flows was proposed in [73]. It is a two step detection mechanism where in first step they identify several P2P flows and in the second step, using this several P2P flows identify the remaining flows. While the method is presented using a graph formalism, it is in fact a collection of simple rules that are called recursively over the set of flows. In the first step several P2P flows are identified based on their statistical properties. For example, flows originating from a host that has high in- and out-degree (that is, it connect to many peers and many peers connect to it). The other possibility is to identify these flows by monitoring ports that are known to be used by P2P applications. Once an initial set of P2P flows F_i from the observed flows F is found, it is extended by moving flows from $F \setminus F_i$:

- any flow in $F \setminus F_i$, that has same source and destination hosts as some flow already in F_i, it is added to F_i (it has been shown that any any traffic exchanged between two hosts belong to the same application with 99.5% probability [64]),
- any flow in $F \setminus F_i$ that target the same destination IP and port as the any of the flows in the F_i is added to F_i,
- flows from $F \setminus F_i$ that originated from the same IP as any flow in F_i within one second form it are added to F_i.

These rules are applied iteratively until the set converges and F_i is also the output of the algorithm. These rules can be also formalized as a graph algorithm, that creates edges between flows based on the same rules. According to the experiments, 90% of all detected P2P flows were within one giant connected component. Evaluation was done as a comparison to the signature-based detection. Since they had signatures only for several P2P networks, they were only able to determine false negatives. Altogether, they were able to identify 99% of all flows labeled by the packet-based classifier. This approach is an alternative to established methods that use hosts as vertices in the graphs, and puts flows in the center point of interest. According to the presented results, this algorithm is unable to distinguish between various P2P networks, since several hosts are "super-clients" that support more than one network and inter-connect various P2P networks. We also must object to the third rule used in the iterative extension of the flow set. We believe that false-positive rate might be unnecessarily increased this way, especially if the are several streams active on the host (for example, while downloading files, one might be also listening to online radio stream, etc.).

An unified framework that combines structural and statistical based detection of P2P networks was proposed in [99]. It first creates graph G based on the observed communication in the network. Nodes of such graphs are IP addresses that are endpoints of communications, and edges represent network flows. Each edge is assigned a feature vector that is based on the properties of network communication that the edge represents. Graph G is then converted to a dual graph, where edges from G are represented as nodes, and two nodes (formerly edges) are connected by an edge if they

- share a common node in G,
- have similar feature vectors; similarity of feature vectors is determined using euclidean distance between them.

Such dual graph is then clustered and clusters containing malicious flows are evaluated for precision and recall. Intriguing property of this approach is the conversion of the original graph to the dual graph. This way, the actual clustering is done to partition connection according to their behavior, as well as their structural properties. On the other hand, the size of the dual graph may be considerable, which can be countered by thresholding edge weights in the dual graph. In comparison to our algorithm, there are two conceptual differences:

- graphs represent different entities,

- our proposed algorithm is on-line.

While the graphs we propose to be used for P2P detection in Chapter 3 and graph proposed in [99] are conceptually different, they both address and unspoken limitation of the graphs with IP addresses as nodes, commonly used in the field – one client or device can participate in multiple P2P overlays at the same time and these graphs cannot distinguish between them.

P2P detection method using directed communication graphs and PageRank was proposed in [42]. In the communication graph, node represent IP addresses and a directed edge between the nodes signifies the existence and the specific direction of communication between them. PageRank, which identifies "important" nodes is run on the communication graph twice; in the second run the direction of the edges is reversed. It was shown that if IP addresses are represented in a two-dimensional space with coordinates specified by the two PageRank runs, IP addresses participating in a P2P are in dense, but disjoint clusters. The precision and recall of the method can be influenced by the quality of information the PageRank is initialized with. Its personalized version can be used to detect a specific P2P network. We are curious whether the fact that PageRank score is higly correlated with in-degrees of nodes [8, 41] can be used to simplify this method – instead of running the PageRank twice, only in/out-degrees of nodes would be used as two dimensions to which the IP addresses are projected. This would align with the intuition that IP addresses participating in P2P network have a high number of communication partners with both incoming and outgoing connections.

Another approach to classify communication between IP addresses, instead of addresses themselves was proposed in [100]. Specifically, in this approach *conversations* are classified. *Conversation* is a sequence of packets between two hosts, where conversation ends after the communication stopped for at least some time period. Four features are calculated for each conversation - duration, number of exchanged packets, volume of data and median of inter-arrival times of packets. A Bayesian network, decision trees and boosted REP tress are trained. These classifiers are then able to classify conversation as belonging to any of the P2P networks used in training. In contrast to our work, it classifies conversations instead of endpoints (i.e. ip addresses). It is a supervised approach, therefore for each new P2P network the classifier needs to be retrained to detect it. Surprisingly, no background traffic was used in evaluation of the classifier. Therefore, even though the capability of the trained classifier to distinguish the P2P networks in the training set was demonstrated, it is unclear the true precision in the presence of background (non-P2P) traffic.

A multi-layer system for detection of P2P networks was proposed in [149]. The system is based on several observations about P2P traffic:

- P2P applications mostly connect to IP addresses directly, without resolving them using DNS request first,
- P2P applications create unusually high number of failed connections,
- P2P-based malware is active as soon as the infected device starts, whereas legitimate applications need to be started by a user who also shuts them down after he/she is done using them,
- flows are similar for nodes in the same P2P network, and vary depending on the P2P protocol and network in use
- nodes that are part of a malware P2P overlay are more likely to share remote peers than nodes participating in a legitimate P2P network.

Description of the whole system is out of scope of this overview; therefore we refer the interested reader to [149]. In principle, the authors solve the same problem as do we in Chapter 3, however focus mostly on different inherent properties of P2P overlay networks and peers participating in them. Also, in our approach we avoid clustering or any other graph-based algorithms, in favor of carefully designing a graph construction and manipulation algorithm that directly describes the overlay of a P2P network.

A detector of P2P flows based on two heuristics about properties of P2P traffic was proposed in [69]. The heuristics used were:

- many P2P networks tend to use both TCP and UDP connections between two same endpoints specified by IP address and port,

- due to port randomization in the latest P2P networks, when an endpoint specified by IP address and port is contacted by other peers in the network, each unique IP address also uses a unique port.

Using these two heuristics in a rule-based system, authors were able to identify 99% of P2P network flows, while limiting false positives to less than 10%.

Graph based detection of P2P networks, that is agnostic of any specific peer-to-peer protocol features is proposed in [25]. It creates a connection graph of communication seen in a network for each specific port and protocol. Based on the graph diameter and number of hosts that function as both client and server, the method determines whether they constitute a peer-to-peer network. Assumption of this method is that all (or at least many) peers in P2P networks listen to incoming communication on the same port which is in stark contrast to [69].

A supervised approach to detect flows originating from P2P-based botnets was proposed in [151]. Flow(s) between two IP addresses captured within a 5-minute interval are represented by 12 statistical features. Two classifiers are trained to classify conversations as either malicious or legitimate. This is an approach based purely on supervised learning, therefore is not closely related to our approach.

A flow-based method to detect peers using inherent properties of peer-to-peer networks is introduced in [10]. The method itself does not use any protocol-specific features and thus, in theory, might be used for any peer-to-peer network. The authors validate the method on BitTorrent and Gnutella networks.

In the detection of P2P networks, and thus P2P botnets, a lot of attention is given to *Traffic Dispersion Graphs* (TDG). A TDG is a graphical representation of various interactions of a group of nodes [59]. In IP networks, nodes in TDG represent hosts in the network identified by their IP address. However, definition of an edge in such a graph is more complicated — to give variability to TDG to describe various forms of interactions, the edges can be defined arbitrarily. They can be either directed or undirected. Edge between two nodes can represent various form of behaviors, eg. two nodes can have an edge between them i.f.f. when they exchange at least 100 kB of data; or if there was an ICMP communication between the two hosts. Clearly, we can not provide an exact definition for TDGs in the general discussion, but every time we present specific experimental results, we explicitly state the particular TDG definition.

Static properties of various TDG were first analyzed in [59]. A network classification system Graption is based on these properties [57]. In the further work, Ilifotou *et al.* investigate also dynamic properties of traffic dispersion graphs [58]. TDGs were also used to evaluate the limits of local approaches in the P2P botnet detection [61].

Static metrics on TDGs (which were used in Graption) are good to distinguish between *collaborative* (includes protocols where hosts are required to communicate with large number of other hosts and have interchangeable roles of client and server) and *non-collaborative* (client-server) applications, but it is harder to distinguish P2P applications from the rest of collaborative applications (SMTP, DNS, NTP). Graption performance for such distinction was not evaluated. Improvement in the distinction between P2P and non-P2P collaborative applications is provided by dynamic metrics. These describe how TDGs evolve with time.

2.5 Guilt by association methods

Guilt by association approach can be loosely defined as: *Given the graph with several nodes labeled as either positive or negative and assuming that neighbors influence each other, find class memberships for all remaining nodes in the graph* [77]. The strength of the influence is specified by the edge weights. Two types of influence are distinguished in the literature – homophily and heterophily. Homophily implies that nearby nodes connected with edges with high weights should have the same label. Heterophily, on the other hand, implies that the neighboring nodes connected by an edge should have different labels.

A number of algorithms belong to the family of guilt by association methods, among them belief propagation (BP), semi-supervised learning (SSL), random walks with restart (RWR) and probabilistic threat propagation. With the exception of belief propagation, which addresses both homophily and heterophily, all mentioned algorithms can be used only with homophilic relationships. RWR is used as a basis of Google's famous PageRank algorithm [17].

BP [104] is an inference algorithm for graphical models such as Random Markov Fields and Bayesian networks. Given the observations of some nodes, it calculates the marginal distributions for unobserved nodes. BP or its generalization - Generalized Belief Propagation [146] were used in a number of fields, such as error-correcting codes [79], stereo imaging in computer vision [39], fraud detection [94, 103] and malware detection [22].

SSL has grown into a large topic and SSL methods now can be divided into three groups, depending on assumption about the data they make [21]. These assumptions are necessary for the semi-supervised learning to make sense and provide more accurate predictions compared to the supervised setting. Of the three groups - generative models, low-density separation, and graph methods, the most relevant for this work are the graph-based approaches. Their interesting property is that they are transductive in nature, meaning that labels can be only predicted for examples present in the time of training.

It was shown in [77] that SSL is equivalent to Gaussian BP; and it can even produce identical results to RWR given a specific choice of parameters. This correspondence illustrates that while the three methods are not identical, their rationales are very similar.

BP was used to improve detection rate of malicious binaries [22]. The method was built on an intuition that:

- malicious binaries can be found on infected computers,
- malicious binaries are not present on computers that are not infected,
- legitimate binaries are more common across the user base than malicious binaries.

Using a bipartite graph and belief propagation, the authors were able to identify additional malicious binaries given the knowledge of some malicious and legitimate binaries with fairly high precision. In this case, the domain knowledge was encoded in both the graph and the method – a specific potential functions were chosen to reflect the domain knowledge. Bipartite graph used in that work has a strong resemblance to the bipartite graph from which we derive our proposed graphs. The links between executables (nodes of interest) are created through machines where they reside, whereas in the graph used in Chapter 5 the domains (nodes of interest) are made through clients (possibly machines) that connect to them.

Same task is solved in [136]. The authors propose to use an unipartite graph where nodes are files/binaries for which they want to infer whether they're legitimate or malicious. They propose an edge weight function based on Jaccard index, which is identical to one of the edge weight functions we propose to use. However, the intention to create an unipartite graph fails due to tremendous memory requirements for its storage. This is a topic that we slightly touch in Section 5.6. Therefore, the authors settle to using a bipartite graph that is identical to the one used in [77]. In spite of giving up on using an unipartite graph, the authors propose an efficient approximation to the edge weight function based on Jaccard index using MinHashing. In our approach, we do not resort to approximate calculation of the edge weights, since we are able to calculate the exact version of the edge weight thanks to the specialized algorithms and graph pruning.

A bipartite graphs with "custom" algorithm is used in [140] to find file hosting servers spreading malicious binaries. In the bipartite graph, one partition contains file hosting servers and the other partition contains websites that point to these file hosting servers. There is an edge between a website and a file hosting server iff the given website points to the given file hoster. These edges are asymmetrically weighted, i.e. weight of an edge depends on the direction it is traversed and the specific weights are determined by the actual number of downloaded files. The iterative algorithm in each iteration sets the *malwareness* of a node as an weighted average of *malwareness* of its neighbors. While the algorithm is described without any reference to prior art, it is basically an iterative approximation of the harmonic minimal function of the Gaussian field for a specific choice of energy function [153]. This harmonic function also has an analytical solution and is a basis of semi-supervised learning in graphs [153].

Problem of identification additional malicious domains given the knowledge of some malicious domains is addressed in [85]. Authors propose to use an inference in a bipartite graph. In the bipartite graph, one partition contains domains and the other partition contains clients that connect to these domains. Weighted edges are used to connect the clients with domains. Anomaly value of a network connection from client to domain, determined by an arbitrary anomaly detection engine, is used as an edge weight between the given client and domain. The inference algorithm used on top of this graph was adopted from [152] which was inspired by [153]. The used edge weight is intriguing in that it should restrict spread of the probability of being malicious along the legitimate edges (as deemed by the anomaly detection engine). One would assume that threat should stay localized in a small part of the graph. On the other hand, consider a situation in which a few infected clients created an anomalous connection towards a well-known and legitimate site. If other users connect to it via normal (i.e. not anomalous) connections the most of the weight mass of incident edges of the well-known legitimate site would connect it to the infected users and consequently malicious domains. Then the probability of being malicious would be rather high. Usage of such edge weight function can thus have both positive and negative effects. Conceptually, by using such edge weight function authors lean away from the idea that the edge weight should be proportional to the similarity of the two nodes or to some other relationship between them. As a final remark, we note that not all sources of data may contain information about anomaly of connections. Another issue may arise when the data is aggregated from various systems. Even if all connections had an anomaly value, there would be a question of correspondence between anomaly values produced by various systems; which is something that was not addressed in the paper.

Identical problem was also addressed in [89]. Just like in the previous case, a bipartite graph was used, this time unweighted. On top of the bipartite graph, a BP was used to infer the probability of being malicious for unlabeled nodes. The difference to our work is in the ground truth – examples from both malicious and legitimate classes are available. Additionally, we consider pruning of the graph in order to increase precision, which is not discussed in [89].

Unweighted bipartite graph mapping clients to domains was used to find additional malicious domains also in [62]. In this work, it was constructed based on the failed DNS requests. According to observations made in [62], failed DNS requests are usually caused by malware trying to reach old, now defunct, domains that were used as C&C servers. Three types of dense subgraph were observed in the graph – near-star structure with a few clients in the center (client-star), near-star structure with a few domains in the center (domain-star) or bi-mesh, a heavily connected subgraph with many users and domains. It was theorized that client-star subgraph may contain clients infected by a malware using domain-fluxing C&C together with malicious domains; domain-star contains a spread-out malware with several C&C domains some of which might be already blocked. Instead of graph inference, graph partitioning is used to identify dense subgraphs that likely represent malware infrastructure.

In [112], DNS data is used to construct a bipartite graph with clients and domains. However, no graph-based inference algorithm is used. Instead, graph is used merely to calculate features of domains in the graph. Features are then used to train a classifier. Advantage of such approach is that the trained classifier can be used for new, unseen data as well. This goes in contrast with the transductive nature of the graph-based inference algorithms. This is the only related work that considers removing low-prevalence domains, albeit without any detailed analysis of the impacts of such pruning.

Guilt by association was used in other fields as well. As an example we can mention identification of gold miners in online games [5]. Another example may be an identification of top k items an user is most likely buy next, based on his reviews of previous items he bought [51]. In the following section we deal extensively with Probability Threat Propagation, which we use in Chapter 5.

2.5.1 Probabilistic threat propagation

Probability Threat Propagation (PTP) is a graph-based community discovery algorithm used for tasks such as blacklist extension and discovery of peers in P2P networks [19]. The algorithm assumes existence of two communities of interest – *malicious* and *legitimate*. It uses the knowledge of some members of these communities to calculate the probability of other nodes belonging to either them. It can be thus also considered to be a guilt by association approach. In further text, we refer to the known members of the communities as *seeds*. The probability of being a member of the malicious community is denoted $P(x)$ and is also referred to as *threat*. Accordingly, the probability of belonging to the legitimate community is $1 - P(x)$.

All instances considered in the inference are nodes in a weighted graph

$$G = (V, E, w). \tag{2.7}$$

The weight of an edge is expected to be proportional to the strength of the relationship. The weight function does not necessarily need to be symmetric, that is $w(x_i, x_j) \neq w(x_j, x_i)$.

Threat of the seeds is fixed; and is specified by the user. Let $S \subset V$ be set of seeds, then $P(x_i) = \alpha_i, \forall x_i \in S$. Threat of all other nodes in the graph is calculated recursively as

$$P(x_i; G) = \sum_{x_j \in N(x_i)} \frac{w_{ij}}{d_i} P(x_j | x_i = 0; G), \tag{2.8}$$

which describes the rationale of the method, that the threat of a node in the graph is equal to the weighted average of threats of its neighbors. Note the conditional probability, which means that the threat of the neighbors was calculated in the absence of x_i. Point of this is to prohibit x_i from increasing its own threat.

There is a closed-form exact solution for trees with only a single seed which is equivalent to the well-known *Belief Propagation*. Tree can be always rearranged in such a way that the seed poses as the root. Then, for each node x_i in the tree, there is a path

$$P_i = \{x(1), x(2), ..., x(m)\} \tag{2.9}$$

to the root, where $x(1) = x_i$ and $x(m)$ denotes the root (seed). The exact formula for threat of x_i is:

$$P(x_i) = P(x(m)) \prod_{j=2}^{m} \frac{w_{(j-1)(j)}}{d_{j-i}}. \tag{2.10}$$

On the other hand, exact inference is infeasible on large graphs, since one would need to calculate $P(x_j | x_i = 0)$ for all pairs of nodes, which is a $O(n^2)$ operation. Hence an approximate solution to Equation 2.8 is proposed in [19]. The approximate threat is calculated iteratively. Instead of conditional probabilities it uses marginal probabilities calculated at previous iteration and subtracts the threat transferred in the opposite direction in the previous iteration. Let $P^k(x_i)$ the threat of a node x_i at iteration k and $C^k(x_i, x_j)$ be a threat transferred from x_i to x_j in k-th iteration, then P^k can be calculated as

$$P^k(x_i) = \sum_{x_j \in N(x_i)} w_{ij}(P^{k-1}(x_j) - C^{k-1}(x_i, x_j)), \tag{2.11}$$

$$C^k(x_i, x_j) = w_{ji}(P^{k-1}(x_i) - C^{k-1}(x_j, x_i)), \tag{2.12}$$

$$\tag{2.13}$$

with P and C initialized as

$$P^0(x_i) = \alpha_i; \forall x_i \in S, \tag{2.14}$$

$$P^0(x_i) = 0; \forall x_i \in V \setminus S, \tag{2.15}$$

$$C^0(x_i, x_j) = 0; \forall x_i, x_j \in V. \tag{2.16}$$

$C^k(x_i, x_j)$ tracks the threat passed between two nodes. Subtracting it in the threat calculation guarantees that threat of x_i is not increased by the threat that x_i itself transferred to its neighbors in the previous iteration. In other words, x_i does not increase its own threat. This is an approximation to the calculation of $P(x_j|x_i = 0)$.

It can be also expressed via matrix operation as

$$C = W.D^{-1}.diag(P) - (W.D^{-1}) \circ C^T, \tag{2.17}$$

$$P = C.1. \tag{2.18}$$

where $\circ$ denotes a Hadamard product of two matrices and D is the degree matrix defined in Equation 2.6.

Graph used for blacklist extension

When designing graph for the task of blacklist extension, three observations were considered:

- it is common for a malicious domain to be hosted on several IP addresses [19],
- it is common for several malicious domains to be hosted at the same IP address [19],
- IP addresses hosting malicious domains tend to concentrate in the IP ranges belonging to hosting providers that are oblivious to the malicious activities in their networks [24, 147]

These three observations lead to a proposal of a bipartite graph, with one partition containing domains and the other partition containing IP addresses on which these domains are hosted. There is an edge between a domain and an IP iff the given domain was hosted on the given IP address. Formally, a bipartite graph is defined as

$$D = (V = C \cup S, E)$$

where set S contains domains, and C contains IP addresses on which the domains are hosted. We refer to this graph as *ip-domain* graph.

2.6 Behavioral modeling of HTTP traffic

First notable work using behavioral modeling based on HTTP traffic is [106]. It aims to use modeling of HTTP traffic to cluster malware samples from the same malware family and generate reliable network signatures for that malware family. Each malicious sample is represented by a set of HTTP requests it issued while executed in a sandboxed environment. Malware samples are grouped using two-stage clustering. In the first stage, malware samples are clustered according several simple statistical properties, such as number of HTTP requests, average number of query parameters, etc. HTTP behavioral similarity was used in the second stage of the clustering. For that, authors propose a distance function between two HTTP requests and define distance between malware samples as the average minimum distance between sequences of HTTP requests from the two samples. The distance between two HTTP requests is calculated as a weighted average of distances based on HTTP method used, path, parameter names and parameter values. Clustering of malware samples is proposed as a two-stage clustering because calculating distance between two samples is very expensive. Performance optimization to the proposed method were later introduced in [105].

HTTP traffic modeling and clustering was used for classification of malware samples also in [72]. Input data used for this approach are again HTTP request made by malware samples while executed in a sandboxed environment. The method even uses a two-stage clustering just like the previous one. The differences between the two methods are two-fold:

- in the first stage of the clustering, three concatenated character distributions, one for paths, one for query parameter names and one for their values, are used instead of a short vector containing simple statistics like length of the path,

- clustering is done on HTTP requests, instead of malware samples.

While no exact definition of the similarity function between two HTTP requests is provided, it is clear that the algorithm tries to infer the type of values being sent as a query parameter. This is approach very similar to one we propose in 6.1.1. In addition to the types of parameters used in this work, the query similarity function we propose also tries to distinguish basic types of query parameter values, such as *string* or *integer*. In our experience, several flows might be needed to infer the basic type of a parameter correctly, thus we find better to cluster the entities that are represented by HTTP requests, rather than requests themselves.

A more general approach towards generating network signatures to detect malware families was proposed in [110]. Besides HTTP, they also consider network communication over other protocols. In this case, the clustering is not multi-stage, even though specific traffic types, among them HTTP, are clustered separately from other network protocols. In clustering of HTTP traffic, the requests are clustered based on path, query parameter names and selected HTTP header values. Instead of clustering, connected components of the HTTP-requests graph pose as clusters. In such graph, two HTTP requests are connected by an edge if

- they target the same path,

- Jaccard index of the query parameters name is > 0.4,

- headers of specific types are identical in both requests (e.g. User-Agent).

Found clusters are again used to generate signatures to detect malicious traffic.

Signatures for pay-per-install malware samples are created based on the traffic they generate when run in a sandboxed environment in [18]. Similarly to [110], all traffic regardless of the protocol is used for clustering and consequent signature generation. Instead of a dedicated clustering for selected protocols, all communication is grouped in a single run. In the grouping, each malware sample is represented by a feature vector that has features such as list protocols used, list of query parameters seen in HTTP requests, and so on. Path part of the URL is ignored, due to the authors' observation that it changes frequently even for a single malware family. There is no clustering algorithm used per-se. First, samples with identical feature vectors are grouped into initial clusters. Further merging of clusters is done manually. Using the proposed method, authors were able to map market of pay-per-install providers and were able to detect 12 out of 20 known pay-per-install families.

Viability of the behavioral clustering of malware samples, regardless of the used behavioral representation is investigated in [12]. They consider a case where attacker attempts to bias the results of the used clustering algorithm by inserting samples with poisoning behavior. They demonstrate a poisoning attack on a single linkage hierarchical clustering (SLINK) [126] used for example in [106]. They show that only a few poisoning samples need to be inserted to destroy the recovery of malware families.

Classifiers for malicious HTTP requests using behavioral features are proposed in [88]. Behavioral features are based on the path and query paths of the URL. In general, the URL is tokenized using special symbols, such as `.` and `/` and bag of words model is used to create a feature vector. Besides behavioral-based features, the authors use other features based on information from DNS & WHOIS, geographic properties and information from blacklists. Constructed feature vectors are tested with SVM, logistic regression and naive Bayes. According to their experiments, using host-based information from WHOIS, DNS and various blacklists reduces false positives.

Chapter 3

P2P networks detection

3.1 Introduction

Although Peer-to-Peer networks are mostly known for file sharing applications, they are also used for VoIP applications (Skype), malware's Command & Control (C&C) channel and media streaming (Spotify). Interestingly, there is also an online currency that relies on P2P architecture for its ecosystem — BitCoin [1]. Last but not least, P2P networks might be used for military purposes [141]. Since VoIP applications, media streaming and especially file sharing applications generate large amount of network traffic, the P2P networks generate a significant amount of traffic in today's Internet.

For the sake of both network management and network security, it is important to be able to identify the network traffic generated by P2P networks. With the knowledge what network traffic is generated by specific P2P application, one can more effectively manage networks and provide a better quality of service. From the security standpoint, blocking P2P based C&C is very effective for disrupting botnet activity. In addition to these two points, it was also shown that peer-to-peer traffic can degrade the performance of anomaly detection techniques. The detection rate can decrease by up to 30% and false positive rate can increase by up to 45% [53].

This chapter deals with the issue of finding P2P peers in the network for all P2P protocols in general, without focus on any particular protocol. That is thanks to exploiting the intrinsic properties that are common for all P2P protocols and which cannot be effectively avoided by any P2P network. Specifically, we are looking for answers to the following questions:

1. Having one peer in an unknown P2P network, are we able to find other peers in the respective network?

2. Can we determine what particular P2P network it is?

3. Can we enumerate all P2P networks and their peers that are active in the monitored network?

Answers to these questions are presented in Sections 3.2, 3.3 and 3.4. As a result, we get a detector that provides information about all active P2P networks in the network. The detector utilizes NetFlow data and uses only information about communication endpoints without using any flow-based statistics. This shows to be an advantage since such statistics (e.g. flow duration) could be distorted for example in the case of a DDoS attack on the monitored network [119].

In our evaluation, we show that the proposed detector is able to link hosts cooperating in the usual peer-to-peer networks such as KAD, Gnutella, BitTorrent and Skype P2P network as well as hosts infected by the same malware using peer-to-peer as its C&C channel. Besides knowledge about the cooperating hosts, the detector is capable of identifying detected P2P networks if they are of a known type.

3.2 Finding cooperating hosts

Even if an Intrusion prevention system (IPS) is deployed in the network and host machines are protected by an anti-virus, hosts can get infected — be it because of the zero-day attack or because of the infection vector that is not covered by the security solution. When such compromised host is found, it is a good practice to look for other hosts in the network that exhibit similar behavior to find other potential victims of the infection.

One way to do this is to look at what hosts the infected machine communicates with, because these would include the perpetrator. And if any other machine shares a similar set of peers[1], it might be infected as well. If two machines share a remote peer, we say they are *cooperating* in at least one P2P network. In some cases, the shared P2P network is benign, like Skype, whereas in other cases, the shared P2P network will be malicious.

Comparing the hosts' peers can be viewed as looking for sets intersection — for each host we keep a set of IP addresses of communication peers and look for pairwise intersections. The shortcoming of this method is that it does not acknowledge that one host might participate in several P2P networks at the same time. For a host that is being infected by a P2P-based malware and running Skype at the same time, looking just at the intersections of sets containing IP addresses we end up marking all other hosts in the network using Skype as suspicious, and we cannot distinguish between the two. Another issue with this approach is that this analysis is done offline, after the incident occurs. If we chose to do it in the real time, the definition of sets would have to be extended, e.g. to allow forgetting of peers. It also brings the requirement of some algorithm extension, such as definition of a time window in which the two hosts must have an intersection of the peer sets.

Both issues can be addressed by graph-based models. Graphs are commonly used to represent a P2P network and graph formalism has been used before in the task of P2P network detection; please refer to Section 2 for more details. However, the proposed graph formalism brings two novelties — different node representation and graph dynamicity.

Most graph methods in the state of the art used nodes to represent hosts or IP addresses. We move from this definition towards using nodes to represent tuples `(IP, port)` which we call *endpoints*. This mitigates the first shortcoming of the "intersection" method. If a host is participating in several P2P networks, its IP is the same, but the ports used for communication in different networks are different. Therefore, a single host can be represented by several endpoints, each associated with a specific network. One host can certainly use several ports for communication within one P2P network, but there always needs to be one port listening for incoming connections that does not change (often) so that other peers in the P2P network can contact it. Therefore, one endpoint should be dominant among all endpoints associated with a single host and P2P network.

[1] by peers we refer to other hosts on the Internet communicating with the host in question

Additionally, to enable real-time detection we employ a graph algorithm that not only constructs the graph, but also modifies it with time, thus assuring that the graph describes the current state of the part of the P2P network that we can observe. The notion of time which is necessary for such graph dynamicity is captured by edge weights.

To detect endpoints participating in a P2P network within our network we use a 3-partite weighted graph

$$G = (V, E, w)$$

where

$$V = V_c \cup V_s \cup V_r.$$

V_c is a set of endpoints from our network we *believe* are participating in a P2P network, V_s is a set of endpoints from our network that we *suspect* are participating in a P2P network and V_r is a set of endpoints from outside of our network communicating with nodes from $V_c \cup V_s$. E is a set of edges, where edge connects two endpoints iff we observed a communication between them. Function w assigns each edge a weight — a value equal to time when the edge was added to the graph.

We ignore all intra-network communication[2] and cannot see communication between the endpoints that are outside of our network. Therefore, the defined graph is indeed a 3-partite weighted graph. This also implies that $G = ((V_c \cup V_s) \cup V_r, E)$ can be viewed as a bipartite graph.

3.2.1 The graph algorithm

Since P2P networks are dynamically changing, so should the graph that represents a P2P network. The detection algorithm monitors network traffic and constructs (modifies) the graph based on the observed network activity in the following way:

1. the graph starts with only the seed node $n \in V_c$,
2. when a network connection occurs between any node $n \in V_c$ and some node m outside of our network then there are two options:
   (a) $m \in V_r$; in this case we just update $w(\{m, n\}) = current\_time()$,
   (b) $m \notin V_r$; in this case we add m to V_r and $\{m, n\}$ to E and set $w(\{m, n\}) = current\_time()$,
3. when a network connection occurs between any local node not yet in the graph and some node $m \in V_r$, we add n to V_s, add $\{m, n\}$ to E and set $w(\{m, n\}) = current\_time()$,
4. any edge $e \in E$ for which $t_{now} - w(e) > m$ is removed from the graph,
5. any node $n \in V$ is removed from the graph when it does not have any incident edge (it has a zero degree),
6. for $l \in \{m \in V_s | \exists n \in V_c :| \mathcal{N}(m) \cap \mathcal{N}(n) |> o\}$ we move l from V_s to V_c; where $\mathcal{N}(n)$ is a neighborhood function defined in Equation 2.4.

[2] Based on the deployment location of the NetFlow probes, the intra-network communication may or may not be available. We chose not to use this information to avoid susceptibility of the algorithm to the probe deployment location.

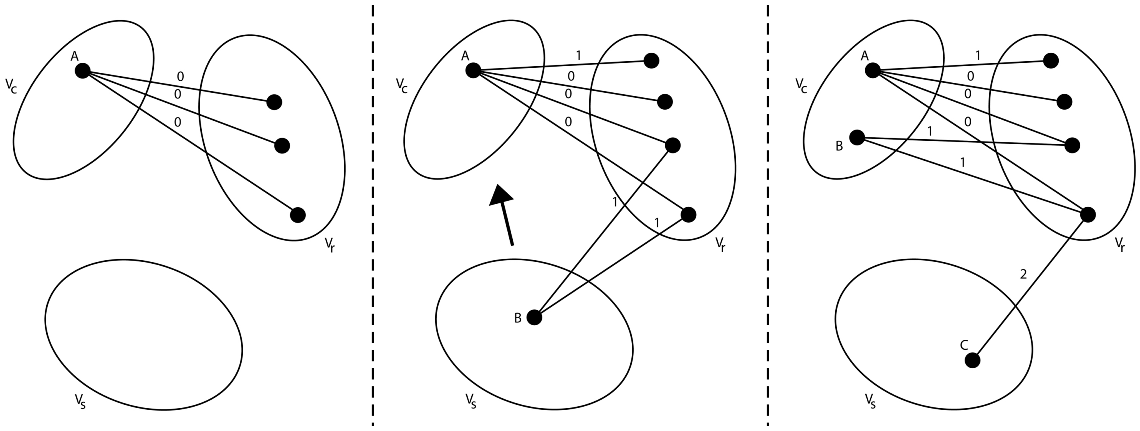


Figure 3.1: Algorithm illustration. We start with a seed node A with 3 recorded contacts. In the second time interval, another node, B, is observed, sharing two mutual contacts with A. If we consider $K = 2$, then in the third step, node B is already moved to the V_c. Moreover, the algorithm detected yet another node, which has only one mutual contact with a node from V_c. Note that the weights of the edges in the graph are determined by the time step in which they occurred most recently.

There are two parameters used in this algorithm:

- a *memory limit*, m, which specifies how long a recorded connection (an edge in the graph) is kept in memory,
- a *mutual contacts overlap threshold*, o, which specifies how many mutual adjacent vertices a node from V_s needs to have with any node from V_c to be moved to V_c.

The first three steps of the algorithm with a simple example graph are depicted in Figure 3.1. The set V_c, at any given moment, contains a list of active P2P nodes in the local network participating in the same P2P network.

There is, of course, a possibility that the graph algorithm will not be able to find any cooperating hosts for certain seed. This might happen when the seed is the only peer of the respective P2P network in the network, or when the seed node around which we tried to construct a graph was not participating in any P2P network.

A discussion about the practical implications of the parameter choice for the graph model can be found in Section 3.7.1. In the following we analyze the optimal choice of o.

On the theoretical bounds of o and m

Choice of o should reflect properties of the P2P networks that are to be detected by the proposed algorithm. Specifically, the choice of o affects the probability with which suspicious nodes are promoted in the graph module – which is in fact equal to the probability of two hosts participating in the same P2P network having o mutual contacts.

This probability depends on three properties of a P2P network:

- size of the P2P network,
- number of neighbors (peers to select) and

- peer selection strategy.

Clearly, these properties are P2P network specific and can even change in time for any given P2P network. Size of the network is determined by the popularity of a P2P application (or infection success rate of P2P-based botnet). Number of peers kept in the neighborhood list is a parameter specified by the P2P network's authors/owners and can be identified by for example reverse engineering. Peer selection strategy specifies how peers participating in the P2P network select their neighbors, i.e. other peers they communicate with. It can be either uniform random or biased (random). We reference several contributions discussing peer selection in Section 2.2.4.

Uniform random peer selection can be modeled as a sampling without replacement from an uniform distribution. It was shown [67] that intersection sizes of two sets created by sampling without replacement from uniform distribution can be described as a hypergeometric distribution. Same observation was used in [26] to analyze the probability of two nodes in the peer-to-peer network to have a mutual contact. It was shown that this probability is surprisingly high.

On the other hand, biased peer selection is basically sampling without replacement from non-uniform distribution which can be modeled by multivariate Wallenius' noncentral hypergeometric distribution [40]. To our knowledge, no probability distribution has been shown to model intersection sizes of two draws from identical non-uniform distributions. Therefore, we resort to Monte Carlo methods to estimate the intersection probabilities under the biased peer selection.

In the following we perform a simple experiment to get insight into possible choices and optimal ranges of o. We compare three selection strategies:

- uniform random,
- biased random from identical geometric distribution,
- biased random from identical mixture of uniform distributions.

Sampling from geometric distribution approximates P2P networks that have a few nodes whose degree is considerably higher than those in the rest of the network. This is the case for ZeroAccess botnet [52]. On the other hand, in the Sality's P2P network, there exist two groups of (super) nodes that can be selected as peers. The two groups have different popularity, one being considerably more popular than the other [52]. This setup is approximated by mixture of uniform distributions; an example of mixture of uniform distributions can be found in Figure 3.2.

Figure 3.3 and Figure 3.4 plot probabilities of having 1, 3, 5 or 7 mutual peers using the three selections strategies with varying P2P network sizes and numbers of selected peers. The results are in line with [26], which states that the probability of having mutual contacts is the lowest when using the uniform random peer selection scheme. In general, the probability of having mutual contacts (any amount) is positively correlated with number of neighbors and negatively correlated with the size of the P2P network. In other words, fixing all other parameters, the probability of having a mutual contact increases with increasing the number of neighbors; and decreases with increasing the P2P network size.

The estimated probabilities of having mutual contacts and their impact on the proposed algorithm assume that a node communicates with all its selected peers at least once within the time frame specified by m. The period within which a node contacts all its neighbors

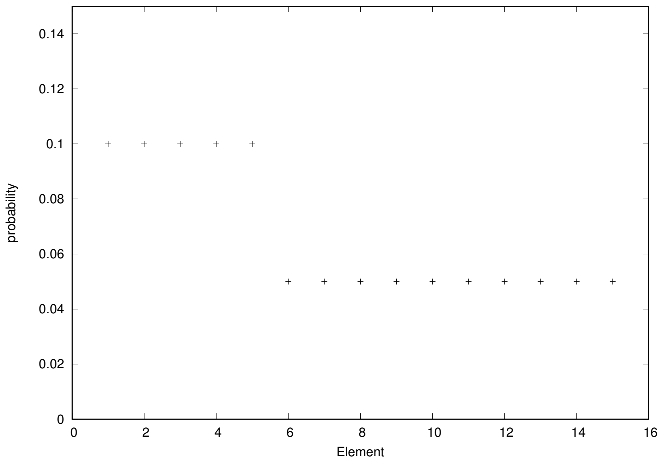


Figure 3.2: Example of a mixture of uniform distributions.

varies depending on the specific P2P network. An overview of communication periods can be found in [117] or in a Table 3.1 adapted from the same paper. This communication period can be also taken as a guideline for the choice of m.

It would be incorrect to assume that a node in a P2P network communicates only with peers in its neighborhood list. Any node can also appear in the neighborhood list of other peers, with which it also communicates. Also, any node can also replace peers in its neighborhood list with other peers. If it does so frequently, it leads to an effective increase of number of neighbors in the communication graph. That increases the probability of having mutual neighbors with other participating peers. Therefore, the estimated probabilities found in this section can be taken as a lower bound, provided that m was chosen such that it respects the communication period of the P2P network(s) it is intended to detect.

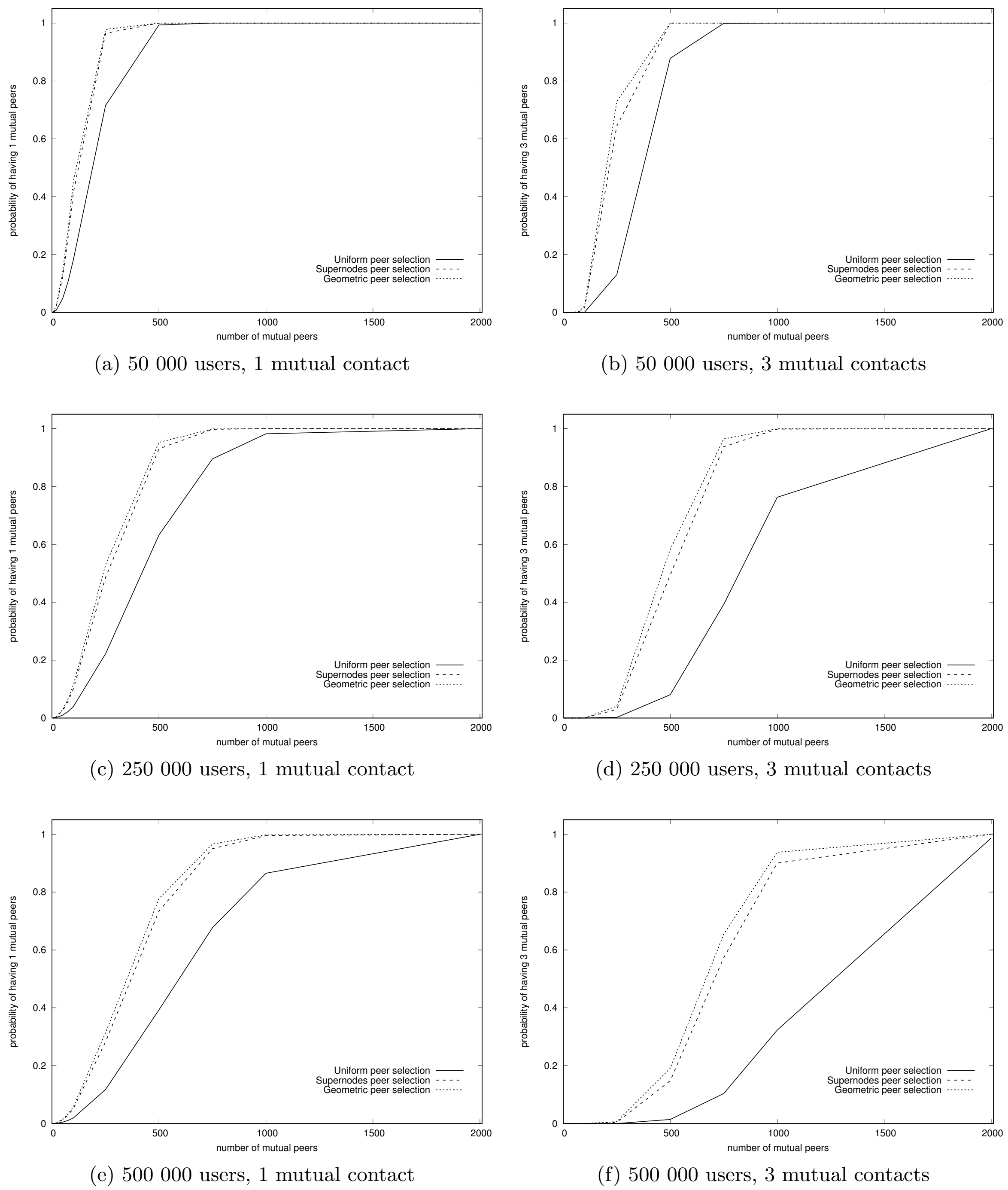
Since the probability of having mutual contacts depends on P2P-network specific factors, we cannot determine the globally optimal choice of the o based on these theoretical estimates. Instead, we choose the parameter based on the empirical results, which we provide in Section 3.6.

3.3 Identifying the revealed P2P network

Assuming we revealed a P2P network, the next step is to determine what particular P2P network it is. Most of the P2P network identification methods rely on deep packet inspection or on flow-based statistics as we point out in Section 2. The proposed method is able to identify the P2P protocol solely based on the information available in the created graphs.

We propose to identify P2P network based on port distribution of peers participating in the P2P network. The port distribution is defined as an empirical probability distribution of ports used by peers that can be represented by a normalized vector with 65535 elements. While it is often argued that port-based P2P protocol identification is useless due to the port randomization [69, 100], the port distribution for the whole P2P network is surprisingly stable.

The proposed detector does not have the full knowledge of the whole P2P network, but we argue that even the partial knowledge is sufficient to identify the P2P network using the **port**



(a) 50 000 users, 1 mutual contact

(b) 50 000 users, 3 mutual contacts

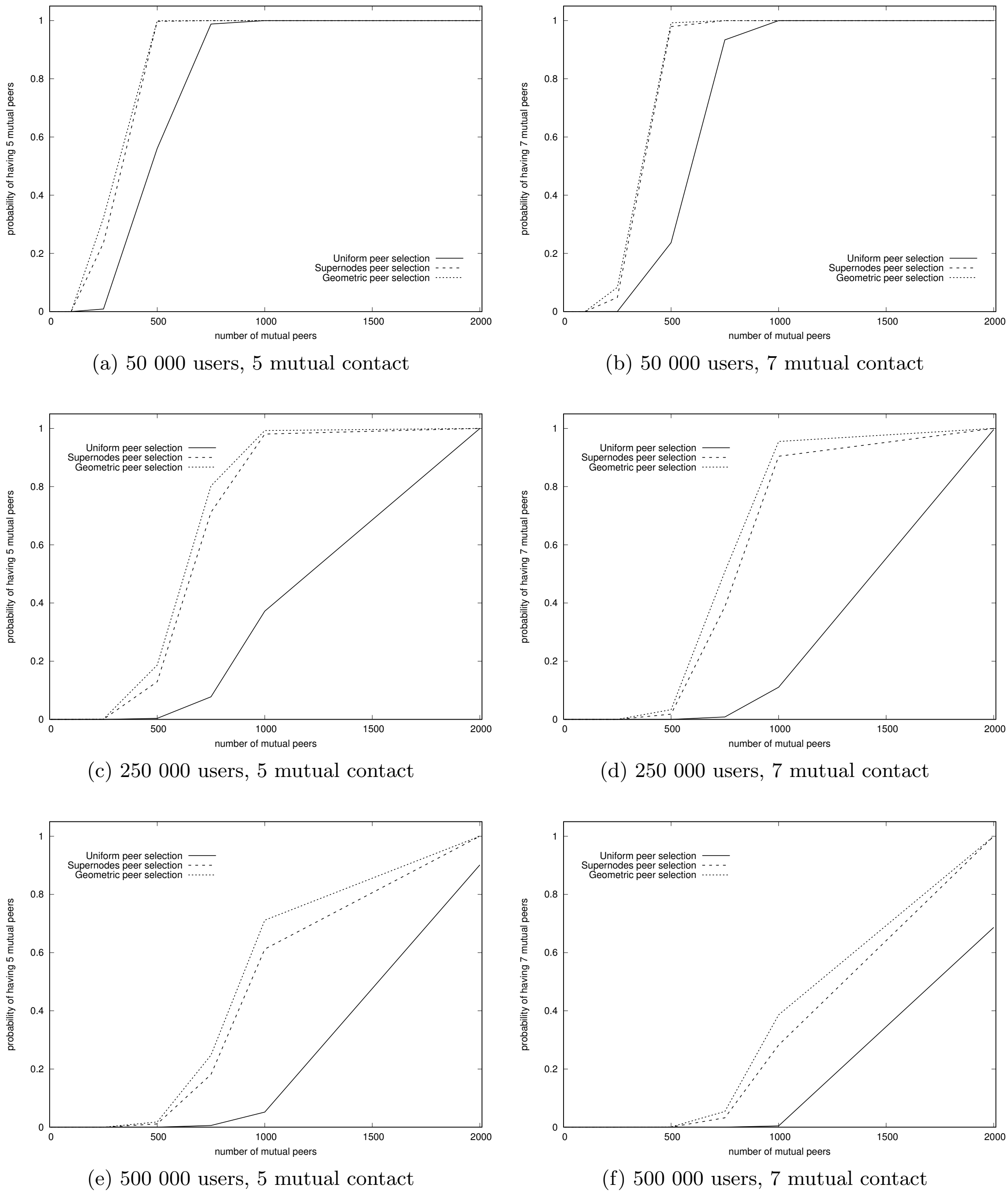
(c) 250 000 users, 1 mutual contact

(d) 250 000 users, 3 mutual contacts

(e) 500 000 users, 1 mutual contact

(f) 500 000 users, 3 mutual contacts

Figure 3.3: Probability of having 1 and 3 mutual contacts for various sizes of peer-to-peer networks and various numbers of selected peers.



(a) 50 000 users, 5 mutual contact

(b) 50 000 users, 7 mutual contact

(c) 250 000 users, 5 mutual contact

(d) 250 000 users, 7 mutual contact

(e) 500 000 users, 5 mutual contact

(f) 500 000 users, 7 mutual contact

Figure 3.4: Probability of having 5 and 7 mutual contacts for various sizes of peer-to-peer networks and various numbers of selected peers.

Table 3.1: Period within which a node exchanges communication with all its peers for selected peer-to-peer botnets. Note that Nugache's period is random, and can be up to 4.5 hours, which is considerably longer than any other botnet in the list. Adapted from [117].

botnet	period
Kelihos v1	10m
Kelihos v2	10m
Kelihos v3	10m
Miner	30m
Nugache	random
Sality v3	40m
Sality v4	40m
Storm	10m
Waledac	30s
ZeroAccess v1	15m
ZeroAccess v2	256s
Zeus	30m

distribution of known remote peers communicating with confirmed nodes. Once the cooperating peers in the network are found, a port distribution over their remote peers is captured and matched against the port distributions of known P2P networks. The graph is considered to represent the P2P network that has the most similar port distribution to the port distribution created over a subset of V_r. As of now, we have defined port distributions for a few major P2P networks — Skype, BitTorrent, KAD (Kademlia) and Gnutella. Their respective distributions can be found in Figure 3.5.

The algorithms can be formalized as a multinomial classification with rejection option [33] — a classifier decides whether the graph represents one of the several known P2P networks and if none of the known P2P networks is similar enough it does not make a decision. The classifier is using one-vs-all strategy, where for each class there is a binary classifier $f_i(\cdot)$ that classifies elements in the respective class. The overall classifier can be defined as

$$f(x) = \arg\max_i f_i(x).$$

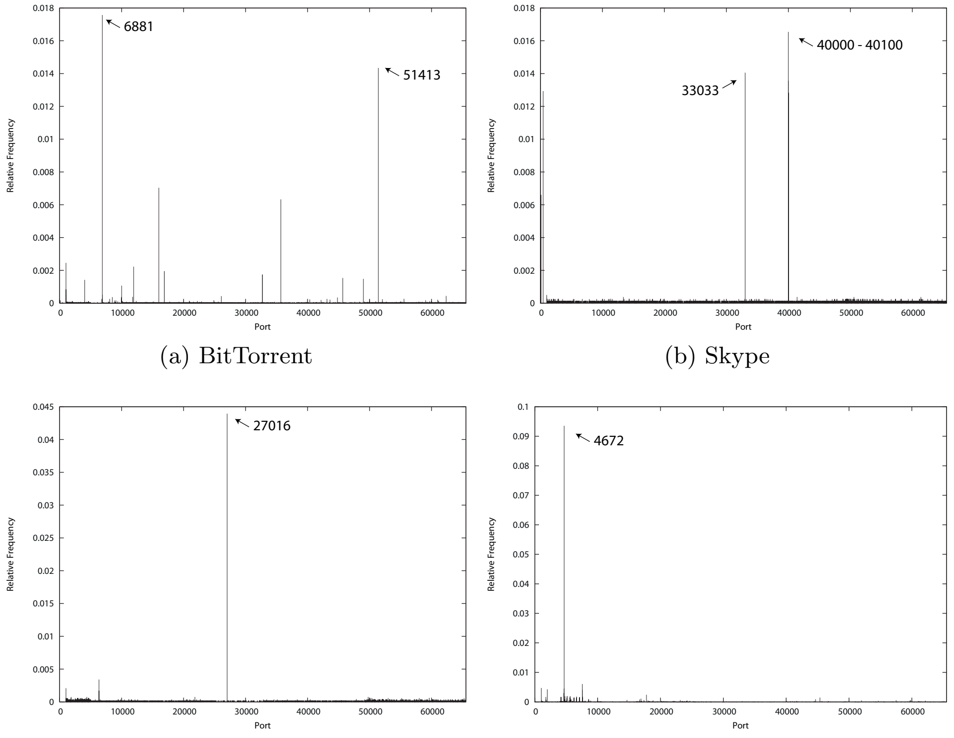
In our case, each binary classifier is represented by a class prototype – a port distribution of peers participating in a known network. Classifier score of each binary classifier is calculated as a dot product of the captured port distribution and the class prototype. One can easily see that the classifier score attains value between $[0, 1]$ with the value increasing with the higher similarity of two port distributions. Since we do not have prototype vectors for all existing P2P networks and we never will, it is crucial that the classifier has the rejection option.

The classification of graph $G = (V_c \cup V_s \cup V_r, E, w)$ proceeds as follows:

1. we create port distribution vector d:

   (a) we instantiate an vector d where

$$d_i =\mid \{n | n \in V_r \wedge port(n) = i \wedge \exists m \in V_c, n \in \mathcal{N}(m)\} \mid, i \in [1, 65535] \tag{3.1}$$



(a) BitTorrent

(b) Skype

(c) Gnutella

(d) KAD

Figure 3.5: Port distributions of some of the major P2P networks. These distributions are distinctively different. For example, port distribution for Skype has strong peaks on port 33033 which is the legacy control port. Ports around 40000 are the new control ports that have been introduced after the Microsoft's acquisition of Skype. On the other hand, the BitTorrent port distribution has peaks at 6881 and 51413 which are default ports for several BitTorrent client applications.

i.e. each element of the vector contains the value equal to the number of remote peers of confirmed nodes whose port is equal to the element index,

(b) we normalize d,

2. for each known class C we define $f_i = \langle d, e_C \rangle$ where e_C is the class prototype; $\langle \cdot, \cdot \rangle$ denotes a dot product of the two vectors; then the overall classifier is defined as $\arg\max_C \langle d, e_C \rangle$
3. we select class C for which $\langle d, e_c \rangle$ is maximized as a possible match,
4. the maximal score $f_{max} = \max_i f_i$ is compared with the *rejection threshold* T, if $f_{max} > T$ the classifiers identifies the graph as representing class C, otherwise no decision is made.

The rejection option from step 4 is crucial. One of the dot products will always have the maximal value. This value might be very low which signals that the classifier is not very sure what P2P network is represented by the graph. In such case it is undesirable to make a decision. In our implementation, we set rejection threshold to 0.3. According to our observation, this value is high enough to avoid "accidental" classification because vectors in high dimensions are typically orthogonal to each other. On the other hand, it is low enough to allow variances of captured port distributions caused by a sampling of peers in the P2P network. Graph has only a partial visibility of the P2P network, therefore the captured port distribution may vary from the true distribution to some extent.

3.3.1 P2P prototype stability

The classification approach requires that the port distributions are stable in time. To show that, we reconstructed P2P networks of BitTorrent and Skype using the graph algorithm described in Section 3.2.1 on two different networks at different points in time and compared their port distributions. The capture times were approximately 9 months apart. For BitTorrent the dot product of the vectors consisting of the port relative frequencies of the two distributions yielded a value of 0.93. This indicates that BitTorrent peers' port distribution is stable both in time and across various network environments.

For Skype the dot product of the vectors created the same way yielded only a value of 0.4 indicating that Skype peers' port distribution is less stable. This can be explained by a major Skype network architecture change that took place between the two points at which we reconstructed the Skype network. A large number of Skype supernodes were moved to the Microsoft's own data centers, relying less on nodes running on users' computers. New supernodes usually listen on ports between 40000 and 40100. Also, the port 33033 which was associated with supernodes owned by the Skype itself, is less prominent after the architecture change. Even after such major architectural change, the proposed peer-to-peer identification method would produce correct classification with the specified choice of the rejection threshold.

The remaining two P2P networks shown in Figure 3.5 were not detected in one of the networks, therefore the comparison cannot be made.

3.4 Enumerating all active peer-to-peer networks

The previous two sections enable us to find a P2P networks given the knowledge of one peer in the P2P network and possibly identify it, if it is of a known type. To find and enumerate all active P2P networks in the monitored network using the same technique, one needs a seed node for each of the active P2P networks. If we were to find one seed node for each active P2P network in the network, we would need to know

- what P2P networks are active in the network,
- how to pinpoint a node participating in a given P2P network.

This is a rather recursive problem. To find the seed nodes we need the knowledge that is sufficient to solve the original problem at hand — to find an enumerate all active P2P networks. To circumvent this issue, we can select all endpoints likely to be peers in some P2P network and grow graphs around them. If any of the chosen endpoints is an active peer, the graph would represent the P2P network it belongs to. The guidelines for selecting seeds as likely peers are based on two intrinsic properties of all P2P networks that always hold:

1. all peers need to listen for incoming connections on an arbitrary but stable port,
2. every peer communicates with at least two other peers.

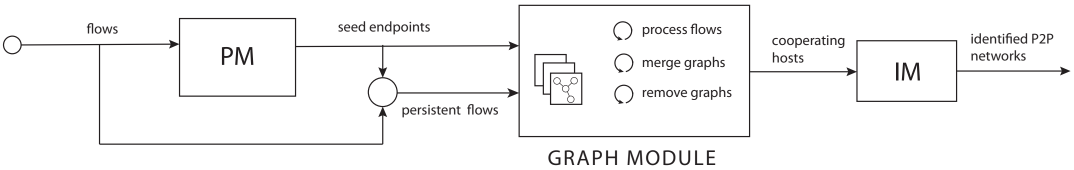


Figure 3.6: Schema of the detector. As an input it takes flows from the network which are processed by the Persistence Module (denoted PM) that provides seeds. The set of seed endpoints is then transferred to the Graph module which processes the flows induced by persistent endpoints and merges and deletes graphs as needed. Sets of cooperating peers are sent to the Identification Module (denoted IM). The output of the detector are sets of endpoints that appear to be cooperating in peer-to-peer networks with identification of the peer-to-peer network if available.

The first property emerges from the observation that each peer both receives and initiates connections to other peers, otherwise it would defer to the client/server paradigm. In other words, each peer behaves like both the client and the server, thus it has to keep a port open for incoming connections. Also, this port cannot be changed often because it produces overhead in the exchanged messages and decreases the effectiveness of the P2P network. In structured P2P networks, each change of listening port is equivalent to leaving and rejoining network with different address, which for example in Chord overlay [130] requires $O(\log^2 n)$ messages [81]. In an unstructured network, changing the listening port is also functionally equivalent to leaving and rejoining network. This does not require any update to the routing table, but former peers of the rejoining peer will not be able to contact it any more. A new search in the network has to take place to reestablish the connection. Therefore, the peers try to avoid changing the listening port while being part of the P2P network.

Moreover, it is not sufficient and/or desirable for peers in a P2P network to be in touch with only one other peer. In the worst-case scenario, where each peer knows address of only one other peer, we effectively end up with centralized or poorly connected network, such as star or round robin. Star is effectively a centralized structure and we can no longer talk about peer-to-peer network. Round robin is inferior in terms of search within the P2P network since the data transfers would often have to pass many unrelated nodes that are not interested in the transferred data and/or information. Thus, we expect nodes participating in a P2P network to have at least two other peers.

3.5 Joint detector design

The two properties discussed in Section 3.4, joined together with information from Section 3.2 and Section 3.3 are used to construct the proposed detector with three modules. Architecture of the detector is depicted in Figure 3.6. The Graph Module revolves around the graph algorithm described in Section 3.2.1, maintaining several graphs simultaneously. The Identification module represents the classifier described in Section 3.3. Finally, the Persistence Module brings the ability to select seeds around which the graphs are constructed based on the intrinsic properties of P2P networks.

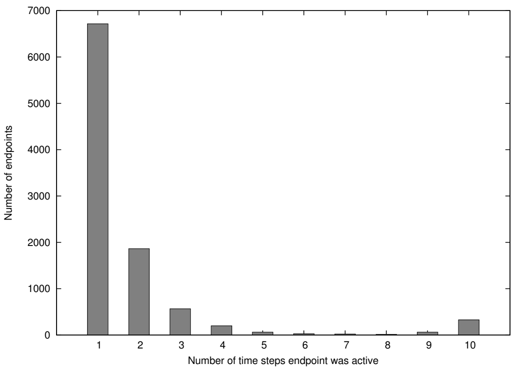


Figure 3.7: The histogram shows that majority of endpoints are active only one or two time intervals. Then we can see only a marginal number of endpoints being active between three and nine time steps. All services that run steadily and are regularly used are active all 10 time windows.

3.5.1 Persistence module

The graph algorithm used in the graph module to be discussed in Section 3.5.2 needs a *seed* node around which it constructs the connection graph. The sole purpose of persistence module is to find such nodes. We utilize two criteria described recently that are based on basic P2P networks' properties — the *persistence criterion* and the *peers count* criterion to select such seeds.

Persistence criterion Peers need to communicate continually with each other to keep the P2P network functional. They exchange messages for the purpose of routing table updates, searches, data exchanges etc. The criterion states that we choose endpoints that are *persistent*, i.e. are sending or receiving data for a longer period of time. During normal network activity, a single host uses many ports to communicate. Most of these ports are used only for a short period of time, these are called *ephemeral* ports. However, there are some ports that are kept open — these are usually used for listening for incoming connections.

To illustrate this, we performed a small experiment on a CTU University network, in which we were monitoring network traffic in ten 5-minute intervals. In the first time interval, we recorded all active endpoints in our network. In the following nine time intervals, we recorded whether the given endpoints were reused. This way, we were able to create a histogram showing the number of endpoints used in either one, two or up to ten time intervals. The histogram can be found in Figure 3.7. We can see that most endpoints were used only in one time interval during the experiment. Then the trend is declining with exception of endpoints that were used during all time intervals. These are, among others, the endpoints that represent services (such as web servers or IMAP servers) or active peers of P2P networks.

To define persistence of endpoints formally, we use simplified method of measuring persistence introduced in [46] and described in detail in Section 2.4.2. The original method was focused on revealing hidden C&C channels. We are interested only in persistence of endpoints, no matter where they connect to. Also, we are not trying to detect exact periodicity of connections, but an ongoing character of a connection. The regularity of endpoint's activity is observed by a sliding window W, which is split into n bins $b_1, ..., b_n$. This window is called *observation window* and bins are called *measurement windows.* We can write $W = [b_1, b_2, b_3, ..., b_n]$. We then define persistence of an endpoint as:

$$p(e, W) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{e,b_i}$$

where e is the endpoint for which the persistence is calculated, W is the observation window and function $\mathbf{1}_{e,b_i}$ is equal to 1 if at least one connection to or from the endpoint e occurred during the measurement window b_i, otherwise it is 0.

The persistence calculation itself depends on three parameters — *measurement window size*, which states how long the connections are recorded into one bin before proceeding to another, *observation window size* n, which determines how many bins there are in the observation window and the *threshold persistence* p, which determines what persistence must an endpoint reach to be considered for seed selection.

Peer count criterion To keep the P2P network running, peers need to communicate with at least two other peers. Therefore, all endpoints that had at least two distinct peers in the current observation window pass this criterion. This removes long lasting connections between two peers on static ports such as clients downloading large files from the Internet, users connecting to other computers via Remote Desktop or SSH or any application keeping open and active connection to a dedicated server.

Each time the module is queried for seeds, it calculates persistence for all recorded endpoints and selects those with persistence exceeding the persistence threshold p. Those selected are then checked against the second criterion, which is the number of contacted peers during the last observation window.

Of course, not all selected seeds represent active peers in some P2P network. However, we argue that *all active peers should be selected as seeds.*

3.5.2 Graph module

The graph module

- constructs graphs around the seed endpoints received from the persistence module,
- merges similar graphs,
- removes graphs that failed to find any cooperating host for the given seed endpoint.

Graph module uses the graph algorithm presented in Section 3.2.1. Knowing that all active peers should be represented by an endpoint that is persistent, we can also reduce number of flows we process in each graph. This reduction is attained by removing all flows that do not originate from or are not directed towards a persistent endpoint since this traffic should not be part of the P2P network. Clearly, by increasing p we reduce the number of flows that are processed. As can be seen in Figure 3.7, even using p as low as 0.3 leads to a significant reduction of number of processed flows. In fact, persistence requirement on the local endpoints of flows processed in the graph module can be independent of p. We can require seeds to have persistence of 0.8 to be used as a seed and at the same time in the graph module process all flows whose local endpoint's persistence is higher than 0.3. We denote persistence threshold enforced in the graph module as p_g. Impact of splitting the two is demonstrated in the evaluation.

However, before the module can construct any graph, it first needs to receive seed endpoints from the persistence module. The persistence module feeds seed endpoints to the graph module periodically. When the module receives the first set of seed endpoints it creates a graph for each of them. For every subsequent set of received seed endpoints it checks whether given seed endpoints are already recorded in any of the graphs. For those that are not, it creates new graphs. This way we prevent the creation of unnecessary duplicate graphs.

We expect that graph algorithm finds cooperating endpoints of the provided seeds, and that the persistence module selects all seeds participating in a P2P network. As a consequence, the graph module should after some time contain several graph models that are very similar and describe the same P2P network, despite starting from different seed endpoint. There is no point in keeping such graphs separate, therefore the module joins them together. It raises a question how to define "similarity" of two graphs.

Two graphs that represent the same P2P network should have similar sets V_c, but since both graphs were iteratively constructed from different seed nodes, they do not necessarily contain similar sets of edges or set V_r. Therefore, we define *similarity* of two graphs G_1 and G_2 as

$$s(G_1, G_2) = \frac{\mid V_c^{G_1} \cap V_c^{G_2} \mid}{min(\mid V_c^{G_1} \mid, \mid V_c^{G_2} \mid)}$$

where $V_c^{G_1}$ resp. $V_c^{G_2}$ represents V_c of graph G_1 resp. G_2. This is also known as meet/min similarity measure [47]. Meet/min ensures that similarity of two graphs G_1, G_2 is high (in fact equal to 1) if $V_c^{G_1} \subset V_c^{G_2}$ and $\mid V_c^{G_1} \mid \ll \mid V_c^{G_2} \mid$. This is a case of two graphs that represent the same P2P network but one of them is much smaller (either because it was created later or because the seed was not as "active" as the seed of the bigger graph). We merge two graphs if their similarity is greater than the *merge overlap threshold*, r, which is another algorithm parameter. Note that meet/min is similar to the *Jaccard index*. The principal difference of the two similarity measures is that for two sets with highly imbalanced sizes, meet/min typically assigns larger values than Jaccard Index.

There is a possibility that the graph algorithm will not be able to find any cooperating hosts for certain seed. This might happen when the seed is the only peer of the respective P2P network in the network, or when the seed node around which we tried to construct a graph was a service, e.g. an email server. If any graph fails to find at least one cooperating endpoint in the network after certain period of time called the *tryout period* it is removed from the module. Even though we remove the graph, it might be recreated next time the seed nodes are received from the persistence module, because the endpoint might be active despite the fact it has no cooperating nodes. Therefore we define another time parameter, the *ignore period*, that determines how long after removing a graph with a specific seed, this seed may not be used to construct another graph. We do not want to ignore the given seed endpoint forever, because service using the port may change or cooperating peers might appear later.

3.5.3 Identification module

Identification Module simply accepts graphs from Graph Module, extracts port distribution for the remote peers and performs protocol identification as described in Section 3.3.

Table 3.2: List of peer-to-peer networks with their respective clients installed on the client hosts in the control set. Last column specifies how many hosts are running given client application.

network	client application	hosts
Skype	official client	18
BitTorrent	μTorrent	26
KAD	eMule	15
Gnutella	Phex	18

3.6 Evaluation

We evaluate both detection and computational performance of the proposed detector. For each evaluation part, we use a different data set. On both of them the traffic was collected in form of NetFlow data by a network probe. Flows were always collected for five minutes and then sent in a batch to the detector. Favoring flow batch processing over stream processing and batch size are settings of the anomaly detection engine in which the detector was deployed. However, the detector can process flows in a streaming fashion or work with batches of a different size.

The computational performance is evaluated on a relatively large telco provider network to test the detector under a heavy load. Since we could not tamper with the network in any way, the detection performance evaluation was done on a much smaller university network. In this network, we could deploy our own P2P nodes and thus establish the ground truth.

Several parameters can be set for the detector. In our experiments, we fixed the value of tryout period to 1 hour. Ignore period was set to 1 hour as well, and it increases by 1 hour for the given seed every consecutive time the graph around that seed is removed because it failed to find any cooperating peers. Measurement window size is set to 5 minutes. Each observation window is composed of 10 bins, which we believe offers a good granularity of information. Persistence requirement on the local endpoints of flows processed in the graph module, p_g, is set to 0.3. Settings of other parameters that we experimented with in evaluation to find the best combination can be found in Table 3.3.

3.6.1 Data sets description

The University data set was collected in the University network consisting of approximately 1000 hosts. The network traffic was collected for 20 hours during a working day. Since we did not have access to all the computers and could not establish the ground truth concerning the overall network activity, i.e. what service did every endpoint in the local network belong to, we chose 155 hosts from two subnets as a small control set.

The first subnet contains 36 hosts of which 18 are running either Windows or Linux. We refer to these hosts as *client hosts*. The client hosts were engaged in casual Internet activity, such as browsing the web, working with email, listening to music, watching videos, etc. Client hosts can have one or more P2P client applications installed on them, and thus participate in several peer-to-peer networks. The list of installed client applications can be found in Table 3.2.

To examine whether the algorithm is capable of linking hosts participating in a botnet, we infected three computers with *ZeroAccess* botnet, which uses a P2P network for its C&C [95,

Table 3.3: Parameters and their values used in the experiment. Parameter values used in the final evaluation are printed in bold face.

symbol	parameter	values
p	persistence threshold	$0.5, \mathbf{0.8}$
o	mutual contacts overlap threshold	$3, 4, \mathbf{5}, 6$
m	memory limit	$60, \mathbf{90}, 120$ minutes
r	merge overlap threshold	$\mathbf{0.3}, 0.5, 0.7$

Table 3.4: Properties of P2P networks, adapted from [52]. Please note that Sality snapshot contains only supernodes, while ZeroAccess snapshot contains both supernodes and ordinary nodes.

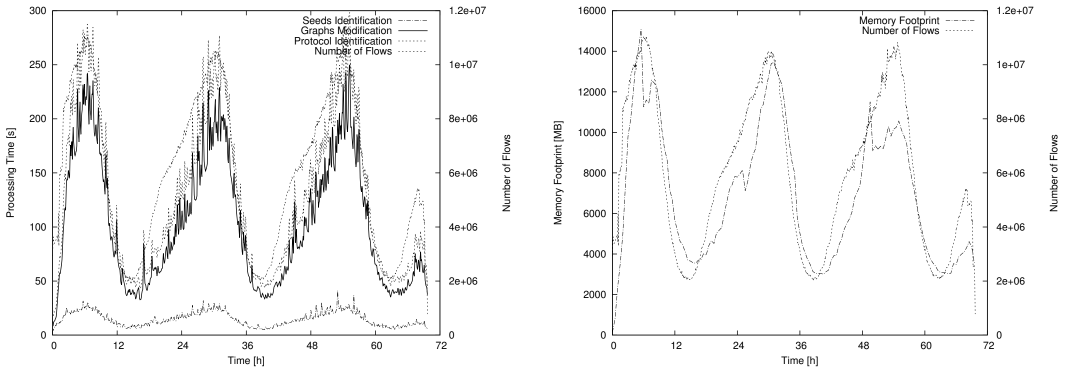
	Sality (supernodes)	ZeroAccess (all nodes)
date created	24.02.16 03:34	24.02.16 13:52
nodes	1 422	4 805
edges	592 646	900 527
average node degree	834	375
in-degree	0/ 44 / 1 244	0/ 55 /4 562
in-degree	0/ 457/ 537	0/ 212/ 233
diameter	3	4

117, 144]. We set all client applications belonging to the same peer-to-peer network to use the same port to ease up evaluation of the results. This has no effect on detection capabilities of our algorithm.

The second subnet contains servers – we refer to this hosts as *server hosts*. None of the them is running any of the aforementioned applications. They run many services, such as web servers, IMAP/POP services and other.

The *TelCo data set* contains traces mainly of homes with DSL uplink. The network encompasses tens of thousands of users and has throughput of 40 Gbps with number of flows per 5 minutes ranging from 2 million to 11 million. The data set spans 3 days in November 2011. For the computational performance evaluation, we are only interested in size of the network, therefore we do not provide any additional information about the network.

The *Overlay data set* contains P2P network snapshots of two recent peer-to-peer botnets - Sality [38] and ZeroAccess [117, 144]. The P2P network snapshots were produced by an enumeration attack in [52]. The properties of the two P2P network snapshots can be found in Table 3.4. Both P2P networks networks are represented as a directed graph, where edge originating from node A targeting node B signifies that that node B is in the neighborhood set of node A. Let us note that in this data set, the nodes are linked to the peers according to their neighborhood lists, in contrast to the linking according to the observed network connection. The distinction is further described in Section 3.2.1.



(a) Processing Times

(b) Memory Footprint

Figure 3.8: Stacked plots of processing time and memory footprint as function of time (and thus number of flows). The graph algorithm takes most of the processing time required by the detector. Strong trends are visible in the data that are caused by the users usually using computers in the evening / at night.

3.6.2 Computation performance evaluation

For the sake of performance evaluation, the detector was deployed on a 24-core Intel Xeon computer (24 virtual on 12 physical cores) and tested on the TelCo data set. We were monitoring processing times of the three modules and retained memory as a function of time (and thus number of flows in the network).

In Figure 3.8a, we can see a stacked plot of the processing times of the three modules. It is obvious that the graph algorithm takes most of the time, whereas seed selection and P2P identification take only a fraction of time. There is also a clear relationship between number of flows and the processing time. One can see trends in the traffic as the users use their computers most in the evening / at night. The computation time peaks at around 300 seconds, which is actually the time span of the processed batch of flows. The detector is reaching its limits when dealing with 10M+ flows (approximately 34k+ flows per second), at this throughput it is still capable of “real-time” processing of data. While is is partially parallelized, it could certainly be optimized to allow for bigger throughput.

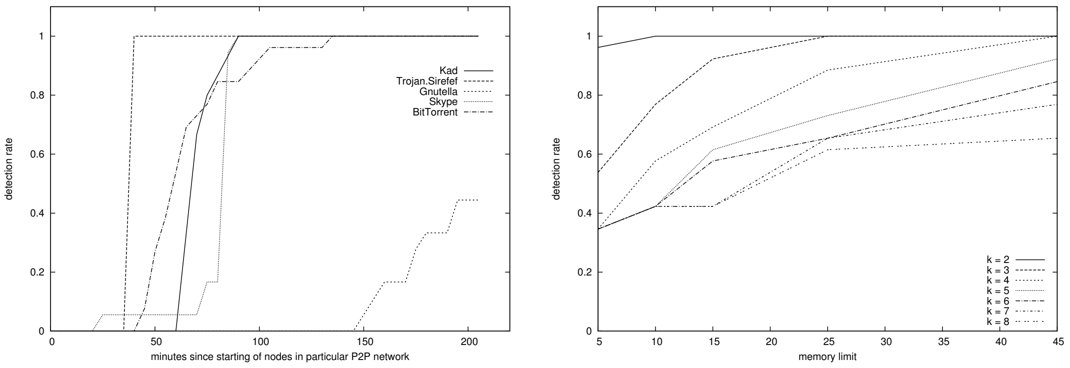
As for memory consumption that is shown in Figure 3.8b, it exhibits the same dependence on number of flows. One more thing can be noted from the plot — the memory footprint is slightly lower in the third peak, which is caused by blacklisting of seeds. The memory footprint ranges from 3 GB to 15 GB during the peak hours.

3.6.3 Detection performance evaluation

Detection rate:

Since the algorithm runs continually and modifies the graphs according to the changes in the network we measure detection rate in time. This way we can see how much time the detector needs to detect a P2P network since start of the client application.

After every batch of flows (every 5 minutes) we query the Identification Module for the list of recognized P2P networks and nodes that participate in them. As can be seen in Figure 3.9a, the algorithm was able to find all hosts participating in Skype, BitTorrent, Kademlia and



(a) Detection rate as a function of time.

(b) Detection rate as a function of m and o.

Figure 3.9: Detection rate of the proposed detector.

ZeroAccess peer-to-peer networks. Detection rate for Gnutella was considerably lower — 44%.

Figure 3.9a also shows that detection of P2P nodes is not immediate and the algorithm needs some time before it detects them. All P2P networks except Gnutella were at least partially detected within an hour since the client applications were started. One can also notice that some Skype nodes were identified even earlier than endpoints representing them reached the required p. This happens thanks to p_g being set only to 0.3 and the other Skype nodes commonly running in the University network. The graph for Skype was already present when we started the Skype clients in the control set and endpoints belonging to these clients were simply added to the graph without the need of becoming a graph seed. This illustrates an important property of the detector — peers that join the P2P network for which a graph already exists are detected much faster than the first peer(s) of a P2P network that does not have corresponding graph yet.

False positive rate:

For various P2P networks we use different methodologies for evaluation of false positives. For KAD, Gnutella, BitTorrent and ZeroAccess, we consider every detected endpoint not associated with the host from the control set and the respective listening port of the client application to be a false positive. Since these peer-to-peer networks are used only rarely at the University, such approach is viable. Using this approach, we determine the upper bound of the false positives detected by our algorithm. We cannot do the same with Skype since it is very popular at the University. Instead, we evaluate false and true positives only on the control set.

There were **no false positives** for four of the P2P networks — namely Skype, KAD, Gnutella and ZeroAccess. Only one false positive was found when linking cooperating hosts in the BitTorrent network. We refrain from calculating the false positive rate, since it would only have a negligible value due to the low number of false positives.

3.6.4 Recall on the P2P network snapshots

The *Overlay data set* provides us with the opportunity to estimate the recall achieved by the proposed method for the Sality and ZeroAccess P2P botnets. We cannot estimate precision of the method since the data set does not contain any negative examples. This data is fundamentally different to the data used by the proposed algorithm, thus we need to make several assumptions and consequently convert the *P2P overlay network* graph to *network communication* graph.

Presence of a node in the neighborhood list of another node does not say anything about the intensity of their communications. Therefore, in the evaluation we assume that all peers participating in the P2P network fulfill the selection criteria of the persistence module (i.e. long-term activity and at least two peers within 5 minutes) and are therefore added to the graph module. There are two parameters used by the graph module – m and o. Since the P2P network snapshot was created within a short period of time, we assume that communication between all peers linked by an edge occurs within the memory span of the graph module; according to discussion in Section 3.2.1 and data in Table 3.1 this assumption is likely to be met. We evaluate recall for various values of the mutual contacts threshold, to see what impact its choice has for the two botnets.

Data containing the P2P network snapshot, including the participating peers, is anonymized to the extent that we are not able to group peers by the network they belong to. However, the proposed algorithm assumes that we are able to identify the network boundary and thus split observed communication endpoints into local and remote endpoints. Instead, we choose a random number of randomly selected peers to represent the local peers, and consider all remaining peers to be remote. For each of the local peers we create a graph model and observe how many of the local peers are added to the model and promoted to *confirmed* nodes. We use the most successful model to determine recall. The attained recall depends on the connectivity of the chosen peers, and consequently the choice of the "local" peers influences the attained recall. Therefore, we perform several random local peers selections and present the reader with the average attained recall.

The results, which can be found in Table 3.5, suggest that the proposed method should be able to successfully reconstruct the P2P network. For Sality this estimate may be actually far from real performance, since the snapshot contains only supernodes that are very well connected. Ordinary nodes that are behind NATs may exhibit completely different connectivity. Unfortunately, Sality's ordinary nodes behind NATs cannot be found an enumeration attack on its P2P network and therefore are not available in the data set.

3.7 Discussion

To explain the inferior detection performance for Gnutella, we need to investigate how nodes in various P2P networks communicate. Some P2P networks use UDP-based communication while other use TCP-based communication or combination of the two. If the P2P network is UDP-based, both incoming and outgoing connections use the main port on which our detector focuses. On the other hand, if the P2P network is TCP-based, the main port is used only by incoming connections. Outgoing connections use ephemeral ports assigned by the operating system which change frequently[3]. Therefore, for TCP-based P2P networks, our detector can

[3] Endpoints representing ephemeral ports might occasionally appear in the graph of the P2P network. In a rigorous understanding, these endpoints are *true positives* because they are used for the communication in the

Table 3.5: Estimate of the attained recall of the proposed algorithm on the two P2P networks of Sality and ZeroAccess.

o	Sality	ZeroAccess
2	0.999	0.997
3	0.998	0.995
4	0.997	0.994
5	0.996	0.994
6	0.996	0.993
7	0.995	0.993
8	0.995	0.993
9	0.995	0.993
10	0.995	0.991

only take advantage of node's incoming connections (because those target the main port). Of the P2P networks we do our experiments with, Kademlia and ZeroAccess both use UDP for their P2P network, Gnutella uses TCP for its P2P network, and Skype and BitTorrent use combination of the two. Therefore, in order to detect Gnutella there needs to be a reasonable number of incoming connections from remote peers to increase the chance of having mutual contacts with other local nodes in the Gnutella network. Gnutella has two types of peers, *leaf nodes* and *ultrapeers*. Leaf nodes only connect to ultrapeers and ultrapeers connect to both ultrapeers and leaf nodes. Also, ultrapeers have higher frequency of connections with other peers. Therefore it is much more probable to detect and link together Gnutella ultrapeers than ordinary peers. And indeed, most of the cooperating hosts found for the Gnutella network were in fact ultrapeers. The longer ramp-up period for Gnutella is due to the fact that it took time until the detected nodes became ultrapeers.

We mentioned that Skype and BitTorrent use both TCP and UDP. Skype uses both protocols in a single P2P network and exchanges them as necessary. On the other hand, BitTorrent keeps separate P2P network on TCP and UDP. TCP is used in the original BitTorrent protocol for downloading files in the swarm where the first set of peers is received from a tracker. UDP is used in the newest implementations for distributed tracker functionality to avoid using centralized trackers when it is not necessary. This P2P network uses BitTorrent's own *Distributed Hash Tables* (DHT) implementation. Therefore, there are two possibilities how to detect BitTorrent clients — via DHT-based P2P network or via the original BitTorrent P2P network. Our experiments show that the detector is capable of linking BitTorrent clients using either protocol.

Here we need to realize the difference between the original BitTorrent protocol and other P2P protocols in this evaluation. While other P2P applications are participating in the P2P network at all times, the BitTorrent is intermittent. The client joins the P2P network only when it wants to download a file and joins a swarm (unless it is using DHT). Therefore, when we talk about detecting cooperating hosts for BitTorrent using only the BitTorrent protocol, we mean hosts that are members of the same swarm not all BitTorrent clients in the network.

We mentioned that our detector is able to identify both P2P networks run by the BitTor-

P2P network. On the other hand, they are present in the same graph as endpoint representing main listening port on the same host. We have therefore ignored these ephemeral endpoints in the evaluation.

rent client. Of the two, detection of the UDP-based DHT implementation is faster because communication in this P2P network starts as soon as the client application is launched without any user activity. In fact, all P2P networks in our experiment with the exception of BitTorrent's original protocol were detected without any user activity. The original BitTorrent P2P network can be observed only after user initiates a file download.

3.7.1 Parameters and their impact

Persistence module and Graph module can be tuned using several parameters that were introduced throughout the text; see Table 3.3.

Parameters of the Persistence module impact mainly computational performance, but they can also effect the detection performance if chosen improperly. *Tryout period* and *ignore period* affect only computation performance and have no impact on the detection or false positive rate. *Measurement window size* and *observation window size* determine which endpoints are selected as seeds. As the measurement window size increases, endpoints need to be active for a longer period of time to be selected. Observation window size impacts the granularity of persistence values. For example, using only two bins, an endpoint can have only one of the three values of persistence — 0, 0.5 or 1. Making the measurement window too small, even ephemeral ports can have high persistence which would increase the number of graphs to be processed by the Graph module. The last parameter used by the Persistence module is p. Choice of its value impacts the computational performance — the higher the value, the less models are created in the graph module and thus less resources are needed to process the graphs. Another important impact of this parameter is on the duration of the ramp-up period in detection. The higher the value, the longer it takes for the detector to find nodes participating in new P2P networks since the client application is started. Each endpoint needs to be active for a specified time before a graph is created for the given endpoint. As p increases, the time required gets longer.

Graph module uses three parameters — r, o and m. Graph module is responsible for merging graphs that represent the same P2P network and r is the parameter that states how strict the module is when deciding whether two graphs represent the same P2P network. In our experiments, we used three values of this parameter without any impact on the detection results. However, the value of this parameter cannot be arbitrary as choosing it very low could cause even unrelated graphs to be merged. On the other hand, choosing its value too high could have performance penalty because the Graph module would keep working with several very similar graphs. Both o and m have significant impact on the detection rate and some impact on false positive rate. Having m fixed, increasing the o decreases the detection rate and false positive rate. Similarly, having o fixed, increasing m increases the detection rate and false positive rate. Impact on the false positive rate is usually only marginal, but setting o to a very low value can rapidly increase the false positive rate. For example, if we set o to 1 and the network is target of a scan then all scanned endpoints are added to the graph, increasing the false positive rate considerably. As can be seen in Figure 3.9b, these two parameters can compensate for each other. Increase in o decreases the detection rate unless m is increased appropriately as well.

3.8 Conclusion

In this chapter, we presented a detector that is able to link hosts cooperating in the same P2P network and identify this P2P network if it is of a known type. The detector uses only inherent properties of P2P networks. It reconstructs the P2P network based on the observed connection in the network. Since the method uses neither packet payloads nor flow statistics, it is a viable option for deployment on the backbone network where computationally expensive models are not an option. Identification of the P2P network is based on port distributions which we show are stable in time.

In the process of designing the detector we address the following questions:

1. Having one peer in an unknown P2P network, are we able to find other peers in the respective network?

2. Can we determine what particular P2P network it is?

3. Can we enumerate all P2P networks together with peers participating in them in the monitored network?

In Section 3.2 we show how the proposed graph algorithm can find other hosts participating in the same P2P network as the first peer. The algorithm is based on monitoring mutual peers of hosts. The next logical step is to identify the observed P2P network; and we present a simple classification method based on remote peers port distribution in Section 3.3. Finally, in Section 3.4 we show how to select peers that are likely participating in a P2P network and which are used as an input to the graph algorithm.

The method managed to detect all cooperating peers in most of the P2P networks and attained almost zero false positive rate in the controlled experiment.

We believe that this method presents a viable approach to detecting peers in P2P networks, both well-known file sharing networks and specialized peer-to-peer networks used by botnets as a C&C channel. It has been used as a part of anomaly detection engine for several years now.

Chapter 4

Probabilistic threat propagation analysis

In the following we analyze PTP presented in Section 2.5.1 in order to gain a better understanding of requirements on graphs to be used with PTP. As we demonstrate in the evaluation, the analysis enables us to define a graph that improves the performance of PTP, compared to the originally proposed graph.

The approximate solution to PTP presented in Equation 2.11 gives little insight to what values the threat of nodes actually converge. To better illustrate the process, in the following we derive exact formulas for calculating threat of a node in the first three iterations. We assume that PTP is run on a general graph G, with nodes of the graph denoted as x_i and edge weight of an edge between two nodes x_i and x_j denoted w_{ij}.

The value of threat of seeds is fixed to some arbitrary values in $[0,1]$ and remains the same in all iterations. Threat of the remaining nodes is recalculated in every iteration. Before the beginning, the 0-th iteration, threat of all non-seed nodes is 0:

$$P^0(x_i) = \begin{cases} \alpha_i, & \forall x_i \in S \\ 0, & \forall x_i \in V \setminus S. \end{cases} \tag{4.1}$$

In the first iteration, threat of the non-seed nodes is

$$P^1(x_i) = \sum_{x_j \in \mathcal{N}(x_i) \cap S} \frac{w_{ij}}{d_i} P^0(x_j) \tag{4.2}$$

which is an average threat of seeds neighboring with node x_i. Nevertheless, it can be also interpreted as a sum of probabilities of all paths of length 1 that end in a seed multiplied by threat of that seed. Similarly, P^2 can be expressed as

$$P^2(x_i) = P^1(x_i) + \sum_{x_j \in \mathcal{N}(x_i) \setminus S} \frac{w_{ij}}{d_i} (\sum_{x_k \in \mathcal{N}(x_j) \cap S} \frac{w_{jk}}{d_j} P^0(x_k)) \tag{4.3}$$

which is equal to sum of path probabilities of all paths with length at most 2 that end in a seed multiplied by threat of that seed. Finally, P^3 is calculated as

$$P^3(x_i) = P^2(x_i) + \sum_{x_j \in \mathcal{N}(x_i) \setminus S} \frac{w_{ij}}{d_i} (\sum_{x_k \in \mathcal{N}(x_j) \setminus (S \cup x_i)} \frac{w_{jk}}{d_j} (\sum_{x_l \in \mathcal{N}(x_k) \cap S} \frac{w_{kl}}{d_k} P^0(x_l))) \tag{4.4}$$

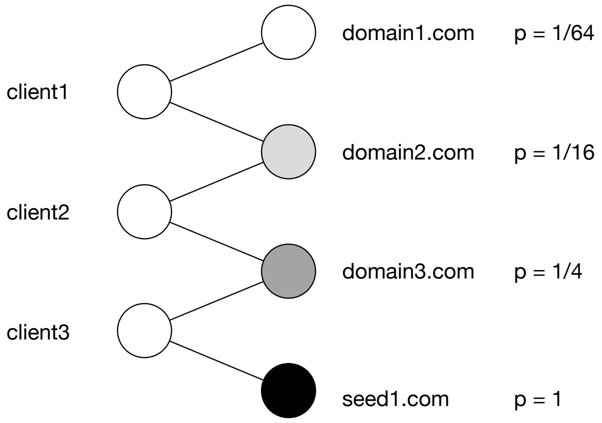


Figure 4.1: A simple bipartite graph that shows how quickly the threat of a node decreases with increasing distance from the seed.

which is basically a sum of path probabilities of all paths with length at most 3 that end in a seed multiplied by threat of that seed, with the exception of paths with cycles of length 2. Ignoring such paths is equivalent to subtracting $C^k(x_i, x_j)$ in Equation 2.11. More detailed derivation of $P^2(\cdot)$ and $P^2(\cdot)$ can be found in Section 4.1.

The limitation to paths that do not have cycles of length 2 is an artifact of the approximate solution. Exact solution considers only simple paths - those without any cycles. The exact solution for a threat of a node can be then expressed as:

$$P(x) = \sum_{s \in S} P(s) \sum_{R \in \mathcal{R}(s)} p(R) \tag{4.5}$$

where $\mathcal{R}(s)$ is a set of simple paths from x to s. For a cycle-free graph with only a single seed, we get back to the exact solution from Equation 2.8.

Interestingly,

$$\sum_{R \in \mathcal{R}} p(R) \tag{4.6}$$

is also known as *cycle-free escape probability* used for calculation of *cycle-free effective conductance* proposed as a measure of proximity in networks [75].

It was pointed out in [75] that the path probability decreases exponentially with the length of the path. A simple example can be found in Figure 4.1. This observation has implications on the choice of the graph representation used with Probabilistic Threat Propagation, which is discussed in Chapter 5.

4.1 Full equations

$$P^0(x_i) = \begin{cases} \alpha_i, & \forall x_i \in S \\ 0, & \forall x_i \in V \setminus S. \end{cases} \tag{4.7}$$

$$\tag{4.8}$$

Threat of seed nodes does not change

$$P^k(x_i) = P^0(x_i), \forall k \in \mathbb{N}, \forall x_i \in S. \tag{4.9}$$

The following holds for $\forall x_i \in V \setminus S$:

$$P^1(x_i) = \sum_{x_j \in \mathcal{N}(x_i) \cap S} \frac{w_{ij}}{d_i} P^0(x_j), \tag{4.10}$$

$$P^2(x_i) = \sum_{x_j \in \mathcal{N}(x_i)} \frac{w_{ij}}{d_i}(P^1(x_j) - \frac{w_{ji}}{d_j} \underbrace{P^0(x_i)}_{=\,0}), \tag{4.11}$$

$$= \sum_{x_j \in \mathcal{N}(x_i) \cap S} \frac{w_{ij}}{d_i} \underbrace{P^1(x_j)}_{=P^0(x_j)} + \sum_{x_j \in \mathcal{N}(x_i) \setminus S} \frac{w_{ij}}{d_i}(P^1(x_j)) \tag{4.12}$$

$$= \sum_{x_j \in \mathcal{N}(x_i) \cap S} \frac{w_{ij}}{d_i} P^0(x_j) + \sum_{x_j \in \mathcal{N}(x_i) \setminus S} \frac{w_{ij}}{d_i}(\sum_{x_k \in \mathcal{N}(x_j) \cap S} \frac{w_{jk}}{d_j} P^0(x_k)) \tag{4.13}$$

$$P^3(x_i) = \sum_{x_j \in \mathcal{N}(x_i) \cap S} \frac{w_{ij}}{d_i} P^0(x_j) + \sum_{x_j \in \mathcal{N}(x_i) \setminus S} \frac{w_{ij}}{d_i}(\sum_{x_k \in \mathcal{N}(x_j) \cap S} \frac{w_{jk}}{d_j} P^0(x_k)) \tag{4.14}$$

$$+ \sum_{x_j \in \mathcal{N}(x_i) \setminus S} \frac{w_{ij}}{d_i}(\sum_{x_k \in \mathcal{N}(x_j) \setminus (S \cup x_i)} \frac{w_{jk}}{d_j}(\sum_{x_l \in \mathcal{N}(x_k) \cap S} \frac{w_{kl}}{d_k} P^0(x_l))). \tag{4.15}$$

Chapter 5

Identification of malicious domains using graph inference

Blocking communication with known malicious domains is a common approach to block malicious traffic adopted by many Network Intrusion Prevention Systems (NIPSs). Malicious domains are collected into blacklists that are then shared among providers of NIPS solutions. Blacklists are however known to suffer from very low recall [80]. There is a considerable delay between the time a malicious domain becomes active and the time it is added to a blacklist. Many malicious domains are never added to blacklists, simply due to their large amount. Some malware families are even known to actively avoid having their active domains blacklisted by frequent transitions to new domains [9].

Malicious domains are traditionally identified by methods that can be split into two families – classification based and anomaly detection based. Both groups of techniques have different roles. Classification identifies malicious behavior which we have already seen and trained a classifier for; anomaly detection points to traffic that somewhat differs from "normal" in the given environment. Recently, a third family of techniques – *guilt by association* – has seen an adoption in the field of network security.

Classifiers and anomaly detectors differ conceptually. While classifiers excel at identifying behaviors similar to those they were trained for, they (by design) fail to identify new classes of malicious behaviors that are fundamentally different from those in the training set. On the other hand, anomaly detectors are able to detect anomalous behavior, however one cannot guarantee any correspondence between anomalous and malicious behaviors [45]. *Guilt by association* methods can be used with advantage to bridge the difference between anomaly detection and classification techniques, in that they use prior knowledge, unlike anomaly detection and at the same time can find principally new threats, unlike classifiers.

Guilt by association approach can be loosely defined as: *Given the graph with several nodes labeled as either positive or negative and assuming that neighbors influence each other, find class memberships for all remaining nodes in the graph* [77]. The strength of the influence is specified by the edge weights. Two types of influence are distinguished in the literature – *homophily* and *heterophily*. Homophily implies that nearby nodes connected with edges with high weights should have the same label. Heterophily, on the other hand, implies that the nearby nodes connected by edges should have different labels.

In the field of network security, guilt by association approach was used with graph capturing homophilic relationships to improve recall of blacklists [19]. However, the graph proposed in [19] and described in Section 2.5.1 is susceptible to relatively easy evasion by malware authors. Therefore, we propose an alternative view of the network communication that can be used for the same task and makes it **harder to evade detection**.

In the field of network security, it is common to have only positive-unlabeled data available. This forces the choice of the guilt-by-association algorithm to those that can be used in such setting. We use Probabilistic Threat Propagation which has been used for the same task before [19].

In this Chapter we solve the following problem: *given the knowledge of some malicious domain, find additional malicious domains* using the guilt-by-association approach. We propose new graphs that can be used with any existing guilt by association algorithm. These graphs are based on intrinsic properties of malware communication making it harder to avoid than the existing graphs used in the field.

The main contributions of this chapter are

- analysis of the existing graphs used with guilt by association methods in the field of network security,
- proposal of novel graphs to be used with guilt by association approaches that makes evasion of detection much harder.

Finally, we demonstrate that the proposed graphs can offer roughly comparable detection performance to the existing graphs used in the field, while making it considerably harder to evade detection by malware authors.

5.1 The *ip-domain* graph

Guilt by association approaches have already been used to detect malicious domains. In [19] authors proposed to use a graph that maps domains to the IP addresses they are hosted on. Therefore we refer to it as an *ip-domain* graph. It is formally defined as

$$D_{ip} = (C, S, E), \tag{5.1}$$

where C is the set of IP addresses, S is the set of domains and E is the set of edges. An edge links an IP address to a domain if the domain was hosted on that IP address. Application of the *ip-domain* graph is based on the observation that it is common for a malicious domain to be hosted on several IP addresses and several malicious domains to be hosted on the same IP address. At the same time, it was observed that IP addresses hosting malicious domains tend to concentrate in the IP ranges belonging to hosting providers that are oblivious to the malicious activities in their networks [24, 147].

However, our observations suggest, that the approach to hosting malicious domains is shifting. Hosting providers catering to malicious actors are withdrawing due to the easy blackholing of their ranges and reluctance of others to peer with them [7]. Nowadays, we observe a number of malicious domains hosted in public clouds where they often share IP addresses with legitimate sites. Frequent sharing of infrastructure by malicious and legitimate domains leads to *false* edges in the graph and possibly false alarms. We are aware of a number of malicious campaigns that do not host more than one malicious domain per server IP address.

Furthermore, analyses of black markets [43, 154] suggest that infected clients are often resold to new perpetrators that can utilize them for additional monetization activities. Other reports [16] suggest that modern malware is modular and can be equipped with various monetization capabilities, possibly owned by different actors. Any new monetization activity from a different actor yields communication to previously unseen malicious domains. Contrary to the assumption of the *ip-domain* graph, these likely do not share the same infrastructure.

5.2 The *client-domain* graph

To avoid issues described above, we propose to monitor connections between clients and domains. Such connections are independent of specific hosting infrastructure used by malware authors. The proposal is based on the properties typical for many malicious domains and infected clients:

- malicious domains are visited predominantly by infected clients,
- infected client usually visits multiple malicious domains,
- typical malicious domain is visited by several infected clients.

Analogous observations were also used in [89, 112, 85]. Admittedly, not all malicious domains have these properties, but based on the results reported in [89, 112], many of them have.

In [112, 89, 85] connections between clients and domains are represented by a bipartite graph

$$D_{client} = (C, S, E) \tag{5.2}$$

where C is the set of clients connecting to domains in set S. Set E contains edges that link clients with domains they connect to. We refer to this graph as *client-domain* graph. We have identified several issues with using a bipartite representation of these connections, especially when used with Probabilistic Threat Propagation (PTP). In the following text we describe three types of issues we have identified.

5.2.1 Interpretation issues

Definition of the guilt by association problem states that the graph contains nodes for which we want to infer the class membership and edges encode the strength of influence between those nodes. Therefore, edge weights encode pairwise similarities of nodes or some other form of their direct relationship. Weights are assumed to be proportional to the similarity or strength of the relationship.

Specifically, in the solved problem, nodes of the graph should be domains and edge weights should encode their similarity or some other form of relationship. That is not true for the *client-domain* graph, since one of the partitions contains clients. PTP does not discriminate between node types and calculates probabilities of belonging to the malicious community even for clients, even though they are not subject of the community discovery task. Clearly, neither [89] considers clients to be part of the inference task, since the calculated probability of being malicious for clients is ignored in the evaluation.

5.2.2 Practical issues

There is a plethora of work covering the task of inference and/or community detection in bipartite graphs, e.g. [87, 133]. In these, there is a specific purpose to use a bipartite graph. PTP can be naturally applied to a bipartite graph, under an assumption that nodes from both partitions belong to one of the classes. *However, in the case of the bipartite graphs in question, the partition of clients is used merely to encode similarities of domains in the other partition of the graph.* Partition of IP addresses in the *ip-domains* has the same purpose.

There are also practical implications of using a bipartite graph to encode similarities of nodes from one of the partitions. For this comparison we assume there is a unipartite graph $G = (S, E, w)$ containing *only* domains with edge weight between any two domains proportional to their similarity and/or relation. Exact edge weight function is not important, the only restriction on the edge weight function is that two domains have an edge between them iff they have a mutual neighbor in D_{client}. Clearly, any path between two domains is twice as long in D_{client} as it is in G. In Section 4 we show that the calculated threat of a domain (its probability of being malicious) is proportional to the probabilities of a random walks taking paths starting in the domain in question and reaching a known malicious node. Knowing that probability of taking a specific path between any two nodes in the *ip-domain* graph decreases exponentially with path's length (number of hops) [75], it is clear that the probabilities calculated by PTP using bipartite graph are lower than those calculated using an unipartite graph.

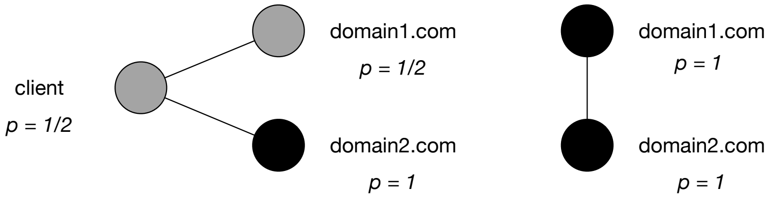
This skew of probabilities towards zero undermines one of the claimed advantages of PTP – resulting threats are probabilities of being malicious, as opposed to other methods that produce a score without a probabilistic interpretation [26]. The calculated probability score can be then used as a prior to the consequent classification, or decision making in general, including automated alarms and incidents [19]. While this is also true for a bipartite graph, the skew of calculated probabilities towards zero renders any automated post-processing based on the calculated probability impractical. Note that the scale of the skew towards 0 depends also on the degree of nodes in the partition that encodes similarities. Therefore, the skew towards 0 is more pronounced in the *client-domain* graph than in *ip-domain* graph, which is caused by overall higher degree of clients.

Using PTP with a bipartite graph can also lead to counter-intuitive results. As an example, consider two domains hosted on the same ip address or visited by the same user, with no other domains in the graph as depicted in Figure 5.1. *Domain2.com* is a known malicious domain, therefore its threat is equal to 1. If we run PTP on the two graphs, calculated threat of *domain1.com* in G is 1. However, calculated threat of the same domain in D_{client} is always less than 1. If both edges in D would have the same weight, the threat score of *domain1.com* would be $\frac{1}{2}$. In this case, we find the threat score of 1 be a more intuitive result.

5.2.3 Leaking personally identifiable information

Computer network owners have various policies for providing network traffic logs, which are needed to construct graphs, to third parties. They may not allow the data to leave their premises, and some countries impose strict geographic restrictions on data location, retention and processing.

The *client-domain* graph potentially contains user names or some other identification of client devices. If the guilt by association inference is to be done globally to improve the



(a) bipartite graph (b) general graph

Figure 5.1: Comparison of calculated values of threat for domain1.com *using PTP seeded with* domain2.com*. In the bipartite graph, threat is calculated for the client as well. In the end,* domain1.com *attains threat of* 0.5*. On the other hand, in the general graph, threat of* domain2.com *is* 1*, which in our opinion better reflects the situation.*

coverage of the inference algorithm, the participating network owners need to share their data.

Of course, the *client-domain* can be anonymized before leaving the network boundary. The anonymization at the network level is certainly possible. Global graph is then constructed from partial anonymized graphs. If we assume that no client can be present in two networks, lossless reconstruction of the global graph is possible. However, the communication patterns of specific clients are still visible, and an attacker might attempt to de-anonymize the graph [76, 82].

5.3 From bipartite to unipartite graphs

In the previous section, we show that bipartite graphs are not a natural graph format to be used with PTP. This results into uncharacteristically skewed distribution of calculated probabilities towards 0. At the same time, explicit capture of the communication in the network poses a threat of de-anonymization attack. None of these issues are relevant for the unipartite graph.

Clearly, there is a merit to converting graph $D = (C \cup S, E_D)$ to graph $G = (S, E_G, w)$ with properly chosen edge weight function w. In a way, conversion from D to G is a compression of information, and some of it is inherently lost. Therefore, the edge weight function needs to be carefully chosen to preserve as much information as possible.

The systematic study of similarity functions used in the graph construction, and their impact on the graph-based algorithms is lacking, especially in the field of network security. To our knowledge, specific edge weight function was considered only in one prior work - [85]. In it, weight of an edge in the *client-domain* bipartite graph is defined as a maximal anomaly of any flow originating from given client and targeting the given domain. Anomaly score can be calculated by an arbitrary anomaly-based NIDS. In this case, the edge weight is not used to enumerate similarity of nodes in the graph, that is done by the bipartite form of the graph. Rather, it is used to limit the spread of threat along the edges that are deemed legitimate. The obvious drawback of this approach is the need to have anomaly values available, which many of the data sets may not have.

In graph D, threat can pass from one node in S to another node from S only through their neighbors in C. Also, no threat is passed between two nodes from S if they do not share a mutual neighbor. These two observations inspire us to consider only edge weight functions that:

- depend on the neighborhoods of nodes from S to calculate their similarity,
- assign positive edge weight only to pairs of nodes from S that share a mutual neighbor.

5.3.1 Edge weight functions

The following discussion of similarity functions extends the previous works [96, 131] and puts them into context of network security. All edge weight functions are explained on an example of the *client-domain* bipartite graph and can be calculated solely based on the information contained in that graph. One can therefore say that we are converting the bipartite graph into an unipartite graph.

Please note that not all edge weight functions presented are true similarities in the sense of its proper mathematical definition. Nevertheless, the PTP does not pose such requirement on edge weights of the underlying graph.

Basic edge weight function

The simplest edge weight function can be defined as:

$$w_b(s,t) = \begin{cases} 1 & \text{if any client connects to both } s \text{ and } t, \\ 0 & \text{otherwise.} \end{cases}$$

By using this function one claims that two servers visited by the same user are related thus the weight of an edge between them is 1.

Jaccard index

Jaccard index [60] is a statistic used for comparing the similarity of sample sets. It is defined as the size of the intersection divided by the size of the union of the sample sets:

$$\mathrm{JI}(A,B) = \frac{|A \cap B|}{|A \cup B|}, \text{ where A, B are sets.} \tag{5.3}$$

In turn, the edge weight function is defined as:

$$w_{ji}(s,t) = \mathrm{JI}(\mathcal{N}(s), \mathcal{N}(t)) = \frac{|\mathcal{N}(s) \cap \mathcal{N}(t)|}{|\mathcal{N}(s) \cup \mathcal{N}(t)|}, \tag{5.4}$$

with $\mathcal{N}(s)$ is the neighborhood function defined in Equation 2.4.

The w_{ji} can also be calculated using vector formalism:

$$w_{ji}(s,t) = \frac{x_s^\intercal \cdot x_t}{\|x_s\|_2^2 + \|x_t\|_2^2 - x_s^\intercal \cdot x_t}, s,t \in S, \tag{5.5}$$

where $x_s \in \{0,1\}^{|C|}$, each dimension of the vector representing one client in C and

$$x_s^{(i)} = \begin{cases} 1 & \text{if client represented by dimension } i \text{ visited server } s, \\ 0 & \text{otherwise,} \end{cases} \tag{5.6}$$

Jaccard index attains values in $[0,1]$ thus offering a finer resolution of “similarity”, compared to the basic edge weight function.

Meet/min

Meet/Min [47] is defined similarly to Jaccard index, with the exception that the denominator is defined as the size of the smaller of the two sets:

$$\mathrm{MM}(A,B) = \frac{|A \cap B|}{\min(|A|,|B|)}, \text{ where A, B are sets.,} \tag{5.7}$$

and the edge weight function itself is defined as

$$w_{mm}(s,t) = \mathrm{MM}(\mathcal{N}(s),\mathcal{N}(t)) = \frac{|\mathcal{N}(s) \cap \mathcal{N}(t)|}{\min(|\mathcal{N}(s)|,|\mathcal{N}(t)|)}, \tag{5.8}$$

and w_{mm} attains values in $[0,1]$.

Jaccard index and Meet/Min have a different behavior, especially when the sizes of two sets are highly imbalanced. Clearly, Meet/Min does not punish the the set size imbalance nearly as much as Jaccard index, which is to be expected based on their definition.

Weighted Jaccard similarity

We might not want all clients to have the same impact when determining similarity of domains. More importance should be given to clients that connect only to a few domains, in contrast to those that connect to a large number of domains. This is a key distinction, since a client that connects to almost all domains can be hardly used to infer relationships among those domains.

By decreasing importance of very active users, we simulate the effect of threat diffusion in the bipartite graph, where less threat is allowed to pass through nodes from C with high degree. Any path between two domains passing through a high degree client has a lower probability than a path passing through a low degree client. Such paths with high degree clients thus allow less threat to pass through and therefore contribute less to the overall threat passed between two domains.

Similar intuition motivates Term frequency - Inverse Document Frequency (TF-IDF) [113] used in text mining. We propose the *weighted jaccard edge weight function*, which can be defined using inner product and TF-IDF framework. The definition of the edge weight function remains the same as with the Jaccard index

$$w_{wji}(s,t) = \frac{\bar{x}_s^\intercal \cdot \bar{x}_t}{\|\bar{x}_s\|_2^2 + \|\bar{x}_t\|_2^2 - \bar{x}_s^\intercal \cdot \bar{x}_t} \tag{5.9}$$

however, the definition of the vector changes. As with TF-IDF, each element of $\bar{x}_s$ is defined as

$$\bar{x}_s^{(i)} = tf(s,i) \cdot idf(s,i,S). \tag{5.10}$$

Term frequency and inverse document frequency are defined as:

$$tf(s,i) = x_s^{(i)}, \tag{5.11}$$

$$idf(s,i,S) = \frac{1}{|\mathcal{N}(c_i)|}. \tag{5.12}$$

where c_i is a client represented by the i-th dimension of the vector.

Alternatively, in the set notation weighted jaccard edge weight function can be defined as:

$$w_{wji}(s,t) = \frac{\sum\limits_{c \in \mathcal{N}(s) \cap \mathcal{N}(t)} \frac{1}{|\mathcal{N}(c)|}}{\sum\limits_{c \in \mathcal{N}(s) \cup \mathcal{N}(t)} \frac{1}{|\mathcal{N}(c)|}}, s,t \in S. \tag{5.13}$$

Please note that we define weighted version of the edge weight function, and not weighted version of Jaccard index itself. Jaccard index does not have information about the popularity of elements in the sets. We draw this information from bipartite graph D.

Weighted meet/min

We use the same logic as in the case of weighted jaccard edge weight function to propose weighted meet/min edge weight function, which is defines as

$$w_{wmm}(s,t) = \frac{\sum\limits_{c \in \mathcal{N}(s) \cap \mathcal{N}(t)} \frac{1}{|\mathcal{N}(c)|}}{min(\sum\limits_{c \in \mathcal{N}(s)} \frac{1}{|\mathcal{N}(c)|}, \sum\limits_{c \in \mathcal{N}(t)} \frac{1}{|\mathcal{N}(c)|})}, s,t \in S. \tag{5.14}$$

Binomial test

Instead of a continuous similarity value, one might want to create only edges between servers that have some statistically significant overlap of clients.

A client connecting to both servers in question can be considered a *success* outcome of a Bernoulli trial. Hence the number of clients in the network connecting to both servers can be modeled using a binomial distribution. Number of trials is equal to the number of clients in the network. Existence of an edge is then conditioned on the rejection of the null hypothesis, that *two servers are unrelated*, using the binomial statistical test. The probability of a client connecting to two unrelated servers is a product of probabilities of the client connecting to either.

We denote probability of a user connecting to server s as p_s and probability of a client connecting to two servers s and t is denoted p_{st}. If the servers are unrelated, then

$$p_{st} = p_s \cdot p_t. \tag{5.15}$$

Binomial statistical test computes p value as

$$pval_{(s,t)} = 1 - \mathrm{CDF}(|\mathcal{N}(s) \cap \mathcal{N}(t)| - 1, |C|, p_{st}). \tag{5.16}$$

and similarity function is defined as

$$w_{bt}(s,t) = \begin{cases} 1 & \text{if } pval_{(s,t)} < 0.05 \\ 0 & \text{otherwise.} \end{cases} \tag{5.17}$$

The probability of any user connecting to a specific server s, p_s, can be estimated from the observed connections. Simple maximum likelihood estimate can perform poorly for the rare events (i.e. servers to which only a few users connect) therefore we apply smoothing to estimate empirical probabilities of connecting to a server closer to true the probabilities.

Number of users connecting to a server can be also modeled by a binomial distribution. To estimate p_s we use Bayesian inference using Binomial distribution with Beta distribution prior. Beta distribution parameters are set to $\alpha = \beta = 1$. With this choice of Beta distribution parameters, the estimate of p_s has a closed form solution:

$$p_s = \frac{|\mathcal{N}(s)| + 1}{|C| + 2}. \tag{5.18}$$

This is equivalent to maximum likelihood estimate of p_s from Binomial distribution with Laplace Smoothing [4] (also known as additive smoothing or Lidstone smoothing) with $k = 1$.

5.3.2 Distributed construction of unipartite graphs

The proposed unipartite graphs contain no information about communication patterns of specific clients. Still, if such graph were to be constructed globally for a number of networks, communication patterns of individual users would need to be shared globally as well.

Instead, we propose to build the unipartite graph in a distributed matter, which is possible for all edge weight functions described in Section 5.3.1. In the following, we denote

$$D_n = (C_n \cup S_n, E_n)$$

to be a bipartite *client-domain* graph specific for network n. The global *client-domain* graph then would be an union of these network-specific *client-domain* graphs.

$$D = \bigcup_n D_n. \tag{5.19}$$

Distributed construction of the unipartite graph using basic edge weight function is the easiest. To do so, all network-specific graphs are first converted to unipartite graphs G_n. Final global unipartite graphs is simply an union of the network-specific unipartite graphs:

$$G = \bigcup_n G_n. \tag{5.20}$$

Next, we describe how to construct an global unipartite graph using *Jaccard index* as edge weight function, which is not as straightforward. As was noted in Section 5.3.1, edge weight based on Jaccard index is calculated as:

$$JI(s,t) = \frac{|\mathcal{N}(s) \cap \mathcal{N}(t)|}{|\mathcal{N}(s) \cup \mathcal{N}(t)|} = \frac{num(s,t)}{den(s,t)} \tag{5.21}$$

where $N(\cdot)$ is the neighborhood function defined in Equation 2.4. This formula assumes the global view of the data. Assuming that networks have disjoint client sets we can define

$$\mathcal{N}_n(s) = \mathcal{N}(s) \cap C_n. \tag{5.22}$$

Then the following holds:

$$num(s,t) = |\mathcal{N}(s) \cap \mathcal{N}(t)| = \sum_n |\mathcal{N}_n(s) \cap \mathcal{N}_n(t)| = \sum_n num_n(s,t) \tag{5.23}$$

where n goes over all networks. $num_n(\cdot,\cdot)$ can be calculated locally using solely data originating from network n. Accordingly for denominator:

$$\begin{aligned} den(s,t) &= |\mathcal{N}(s) \cup \mathcal{N}(t)| = |\mathcal{N}(s)| + |\mathcal{N}(t)| - num(s,t) \\ &= \sum_n \mathcal{N}_n(s) + \sum_n \mathcal{N}_n(t) - num(s,t). \end{aligned} \tag{5.24}$$

The final edge weight can be then calculated globally as

$$JI(s,t) = \frac{num(s,t)}{\sum_n \mathcal{N}_n(s) + \sum_n \mathcal{N}_n(t) - num(s,t)}. \tag{5.25}$$

Therefore, to calculate the proposed unipartite graph using weight function based on Jaccard index, one needs to first calculate $num_n(s,t)$, $|\mathcal{N}_n(s)|$ and $|\mathcal{N}_n(t)|$ for each network and send this information to a global processor. There the global graph can be reconstructed. None of the information sent to the global processor contains any personally identifiable information.

Similarly, we can show how to create unipartite graph in a distributed fashion for weighted jaccard edge weight function, meet/min edge weight function and weighted meet/min edge weight function.

In order to construct a unipartite graph using the edge weight function based on binomial test in a distributed fashion, each network needs to report its size, $|C_n|$, as well as all the information needed above. To construct the global graph, we need to determine global probability of a client connecting to domain s. This can be calculated as:

$$p_s = \frac{\sum_n |\mathcal{N}_n(s)| + 1}{\sum_n |C_n| + 2}. \tag{5.26}$$

Formula for calculation of p_{st} remains the same. Finally, p value of the binomial statistical test is calculated as

$$pval_{(s,t)} = 1 - \text{CDF}(\sum_n |\mathcal{N}_n(s) \cap \mathcal{N}_n(t)| - 1, \sum_n |C_n|, p_{st}). \tag{5.27}$$

5.4 Pruning of client-based graphs

The fact that two domains are visited by the same user gives less evidence of the relationship of the two domains, in comparison to the fact that the two domains are hosted on the same IP address. This can be explained by the fact that a client in the *client-domain* graph actually encompasses several entities – user himself, operating system, any installed applications and malicious binary, if present. All of these initiate network connections, often independently of each other. As a result, probability of being malicious calculated for domains that are visited by only a single user, or by a small group of users are not reliable, and in turn can cause many false positives.

Therefore, *we propose to remove domains with low popularity from the graph.* By popularity of a domain we understand number of clients connecting to it. We investigate impact of the graph pruning in Section 5.5.5 and Section 5.5.6.

Table 5.1: Examples of second-level domain and public suffix of a fully qualified domain name (FQDN) for various FQDNs.

FQDN	second-level domain	public suffix of FQDN
www.google.com	google.com	google.com
www.amazon.co.uk	amazon.co.uk	amazon.co.uk
africlz26.dyndns.info	dyndns.info	africlz26.dyndns.info

Table 5.2: A summary of properties of data sets used in evaluation.

	networks	domains	clients	IP addresses	malicious domains
2016-10	336	$5,971,233$	$3,301,905$	$3,926,293$	$3,253$
2016-11	739	$6,651,490$	$4,148,226$	$4,177,737$	$3,340$
2016-12	615	$6,250,316$	$3,728,613$	$3,993,840$	$3,567$

5.5 Experiments & evaluation

In this section, we demonstrate the performance of the Probabilistic Threat Propagation applied to the proposed graphs.

Throughout the text we were using the term *domain* without specific explanation what is meant by that. In fact, the graphs can be used with various specific definitions of a domain, i.e. second level domain or fully qualified domain name. In this evaluation, by term domain we understand the public suffix of the fully qualified domain name. Such choice avoids graph size explosion due to the extensive number of fully qualified domain names, and at the same time it acknowledges that some second level domains, e.g. dyndns[.]info are public domains, where anyone can register a sub-domain. Differences in mentioned exact domain definitions are demonstrated on real examples in Table 5.1.

5.5.1 Data set description

In the evaluation we rely on three data sets, denoted *2016-10*, *2016-11* and *2016-12*. The *2016-10* data set was collected in October 2016 and contains proxy logs of $3,301,905$ users from 336 networks. There is $5,971,233$ domains and $3,926,293$ server IP addresses present in the data set. The *2016-11* data set was collected in November 2016, contains proxy logs of $4,148,226$ users from 739 networks. It contains $6,651,490$ domains and $4,177,737$ server IP addresses. The *2016-12* data set was collected in December 2016, contains proxy logs of $3,728,613$ users from 615 networks. It contains $6,250,316$ domains an $3,993,840$ server IP addresses. The properties of the data sets are also summarized in Table 5.2.

The *ip-sld* and *user-sld* graphs are constructed from these data sets by identifying unique pairs of ip addresses/users and second level domains and storing the information in the form of a bipartite graph. Unipartite graphs are extracted from the *client-domain* graph.

5.5.2 Source of labels

As a source of labels, we use the knowledge of malicious campaigns currently being tracked by Cisco CTA [135]. Malicious campaign is represented by a collection of second level domains

Table 5.3: Precision@K achieved by PTP using ip-domain *and* client-domain *graphs in two considered use cases.*

	general	*specific*
ip-domain bipartite	0.88	0
user-domain bipartite	0.54	0.33

that participate to achieve a common goal, or generally serve the same purpose. At the time of the experiment, 279 malicious campaigns were tracked, representing various threat types, including ransomware, information stealers, banking trojans, exploit kits, ad injectors, click frauds and others. Not all of the known malicious campaigns exhibit themselves in the data sets, since the intelligence used as a source of labels is more recent than the used data sets. We allow for a longer time gap between the data capture and intelligence version (date at which it is valid) to let security community catch up; that is give them time to identify as many malicious domains as possible.

Malicious domains from these campaigns are the sole domains used to seed PTP; no legitimate domains were used to seed the PTP.

An important distinction of our experimental setup, as compared to the experimental setup used in related work is the amount and form of the labeled data available. We consider a use case in which we only have labels for a few malicious domains and no legitimate labels whatsoever. For comparison, for the *2016-10* data set we only have labels for 3,258 (0.055%) domains (all of them malicious), which is in stark contrast to [89], in which 1.45% of domains were labeled.

5.5.3 Evaluation criteria

The output of PTP can be considered a classifier score in a binary class decision problem, and thus can be evaluated as such. As an evaluation criteria, we are using precision@K which measures precision for the K elements with the highest classification score. It aligns well with our use case where domains identified as malicious by PTP are further investigated by an analyst. An analyst usually has a limited time frame to go over the results of PTP. We choose K to be 100, which in our experience is a realistic estimate of the number of domains an analyst is able to evaluate within a single session.

Evaluation of domains found by PTP was done manually by domain experts. No hard criteria were used to determine whether a specific domain is malicious, it was solely at the discretion of the analysts. We recognize the possible bias in the labeling introduced by the domain experts. On the other hand, all domains were evaluated together, without any indication as to which graph was used to detect any particular domain. Therefore, there is no bias towards any specific evaluated graph.

5.5.4 Bipartite graphs

In this section, we aim to compare the detection performance of *ip-domain* and *client-domain* graphs. Evaluation is performend on the *2016-10* data set. Precision achieved using PTP with the *client-domain* and *ip-domain* is compared for two uses cases - *general* and *specific*. We use *specific* use case to demonstrate how a specific evasion technique can render the

Table 5.4: Achieved precision@K for various edge weight function and domain popularity thresholds.

	1	2	3	4	5
ip-domain bipartite	0.88	**0.90**	0.86	0.86	0.81
user-domain bipartite	0.54	0.57	0.69	**0.77**	**0.77**
basic	**0.54**	0.37	0.35	0.47	0.44
jaccard index	0.44	0.48	0.63	0.69	**0.78**
meet/min	0.55	0.50	0.51	0.69	**0.76**
weighted jaccard	0.45	0.54	0.72	0.72	**0.79**
weighted meet/min	0.55	0.60	0.73	0.81	**0.83**
statistical test	**0.48**	0.31	0.23	0.32	0.36

ip-client graph useless while *client-domain* graph is still able to find additional malicious domains. In this use case, we seed PTP only with domains from the malicious campaigns that intentionally avoid hosting two or more domains on the same IP address. In contrast to that, with the *general* use case we want to show how the two graphs behave in general, when all known malicious domains are used as seeds by PTP.

Results for all combinations of the used graphs and use cases can be found in Table 5.3. In the *specific* case, no additional malicious domains were found by PTP when using *ip-domain* graph, while the *client-domain* graph still achieved a reasonable precision. In the *general* case, the *ip-domain* graph offers superior performance compared to the *client-domain* graph.

The results suggest that monitoring connections between clients and domains is harder to evade. On the other hand, it offers inferior performance to the *ip-domain* graph, at least when represented by a bipartite graph. Therefore, to make client-domain-based graphs a viable alternative to the *ip-domain* graph, increasing precision is crucial. The precision can be improved by either pruning of the graph or using an unipartite graph, as shown in Section 5.5.5.

5.5.5 Improving precision of client-domain-based graphs

In Section 5.4 we hypothesized that precision of PTP run on graphs based on monitoring connections between clients and domains can be improved by removing domains with low popularity.

In order to verify this hypothesis, we create five versions of all bipartite and unipartite graphs. Each version is pruned using a different domain popularity threshold. Used popularity thresholds are 1, 2, 3, 4 or 5. Note that for the sake of fairness of comparison, in the experiment we also used five pruned versions of the *ip-domain* graph. In this specific case, the criteria for removing domains from the graph was number of IP addresses on which the domains are hosted. In the experiment, we utilize the *2016-10* data set, and use all known malicious domains to seed PTP.

The precision achieved with various graphs and domain popularity thresholds can be found in Table 5.4. The results confirm that the precision achieved by all graphs based on client - domain connections depend heavily on pruning. The impact of the domain popularity threshold is however not consistent across all proposed unipartite graphs. For example, graph using binomial statistical test as an edge weight function offers better precision when no

Table 5.5: Precision@K achieved by edge weight function and their optimized domain popularity thresholds on testing data sets.

	2016-11	*2016-12*
ip-domain bipartite	**0.95**	**0.95**
user-domain bipartite	0.78	0.80
basic	0.70	0.75
jaccard index	0.88	0.82
meet/min	0.87	0.84
weighted jaccard	0.90	0.89
weighted meet/min	0.85	0.90
statistical test	0.46	0.42

threshold is used, while graph using *Jaccard index* as an edge weight function performs best with the highest domain popularity threshold tested.

It is therefore appropriate to consider domain popularity threshold to be a hyper-parameter of the algorithm that needs to be optimized for each specific unipartite graph. To that end, in the following we consider the *2016-10* data set to be a training data set on which we optimize the value of the hyper-parameter. The optimal domain popularity threshold choice for each proposed edge weight function, which is further used in the evaluation, is clear from Table 5.4.

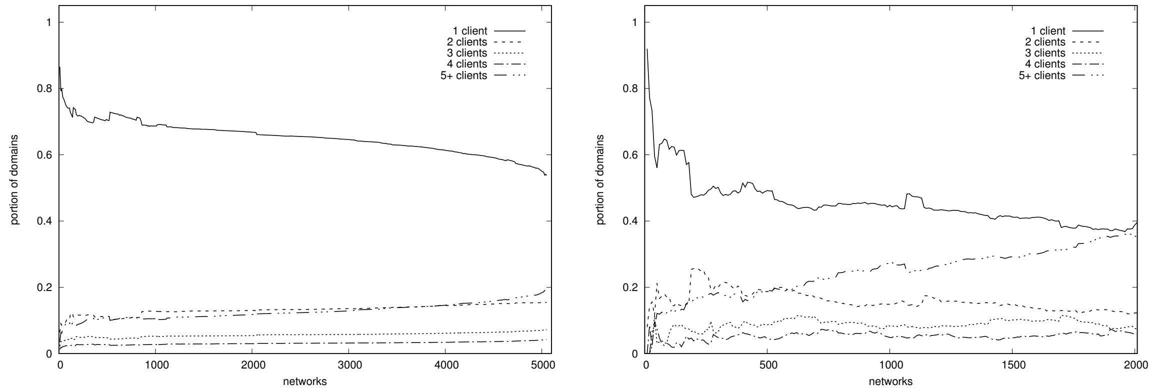
Precision achieved by PTP run on the proposed pruned graphs is then compared on two testing sets, *2016-11* and *2016-12*. Results of experiments on testing data sets can be found in Table 5.5. Results suggest that both weighted and unweighted versions of Jaccard index and meet/min edge weight functions outperform the bipartite graph, basic edge weight function and the edge weight function based on the binomial statistical test. While the precision achieved by the *ip-domain* graph is still the best, precision of the best performing proposed unipartite graphs is not far behind.

5.5.6 Impact of graph pruning

Graph pruning removes domains and thus possibly reduces recall. We cannot evaluate impact on recall directly, because we do not have ground truth for all domains in the data set. Instead, we measure how many domains are removed by pruning given various thresholds of domain popularity and then what is the impact on the number of detected users if malicious domains in the data set were used as an indicator of compromise (IOC).

Each domain has a true, global popularity which cannot be observed directly, unless one has access to the global network logs. Popularity of a domain within a data set is always lower than the global popularity, increases with the increase of size of the data set, and eventually should converge to the global popularity.

In order to provide insight into proportions of domains with various popularities and its dependence on the number of networks (and hence users) in the data set we use data from $5,000$ separate networks. We determine the portion of domains with popularity equal to 1, 2, 3, 4 and 5 or more for increasing number of networks. Results can be found in Figure 5.2a for *all* domains and in Figure 5.2b for *malicious* domains only. The plot in Figure 5.2a suggests that overwhelming number of domains is visited by only a single user. Portion of such domains decreases with increasing size of the data set, nevertheless even with $5,000$ networks in the



(a) all domains

(b) malicious domains

Figure 5.2: Portions domains with selected probabilities displayed as a function of size of the data set. Portion of domains with low popularity decreases with the increase of the data set size. Interestingly, even on data sat containing thousands of networks and millions of users, around 60% *is still visited by only a single client (figure left, solid line). This is less pronounced for malicious domains, where on the same full data set only around* 30% *of domains are visited by only a single user (figure right, solid line). The difference in the number of networks in the two plots is caused by the fact that 2000 networks had no hit on any of the malicious domains used in evaluation.*

data set, almost 60% of domains are unique to a single client.

The situation is slightly different with malicious domains, depicted in Figure 5.2b. The general trend is identical to the previous case – portion of domains visited by only a single user decreases with increase in the data set size. However, portion of domains visited by only a single client drops to around 30%. Note that plot in Figure 5.2b ends for $3,000$ networks. This is caused by the fact that in the remaining networks there is no communication to any of the malicious domain used in this evaluation.

If malicious domains are used as indicators of compromise (IOCs) to detect infected users, pruning of graph can have adverse effect on the number of detected infected users. Therefore, we investigate how does the number of detected users change, if we ignore known malicious domains with low popularity. The impact is depicted in Figure 5.3.

The drop in number of detected users is negligible – ignoring malicious domains with only one client, 99.46% of infected users would be still detected. If malicious domains with five or less clients are ignored, 98.19% of infected users is still detected. Whether such drop in recall (of infected users) is acceptable depends on the specific application, nevertheless drop in recall is negligible compared to the drop in numbed of considered domains and thus drop in computation requirements.

In conclusion, it has been shown earlier that graph pruning improves precision of client-domain-based graphs. In this section, we have shown that the impact on number of detected users is marginal and since at least half of the domains are removed (in the available data set), also the graph is considerably smaller and runtime of PTP shorter. Graph pruning is therefore a viable solution to low-precision problem of client-domain-based graphs.

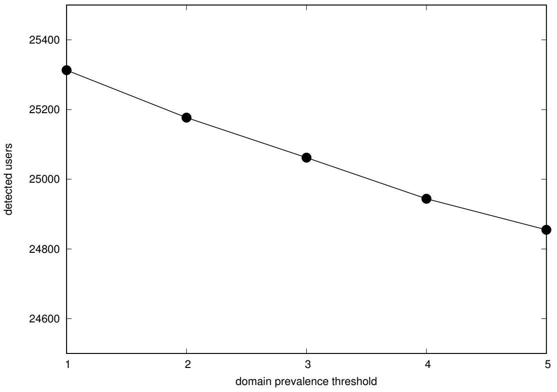
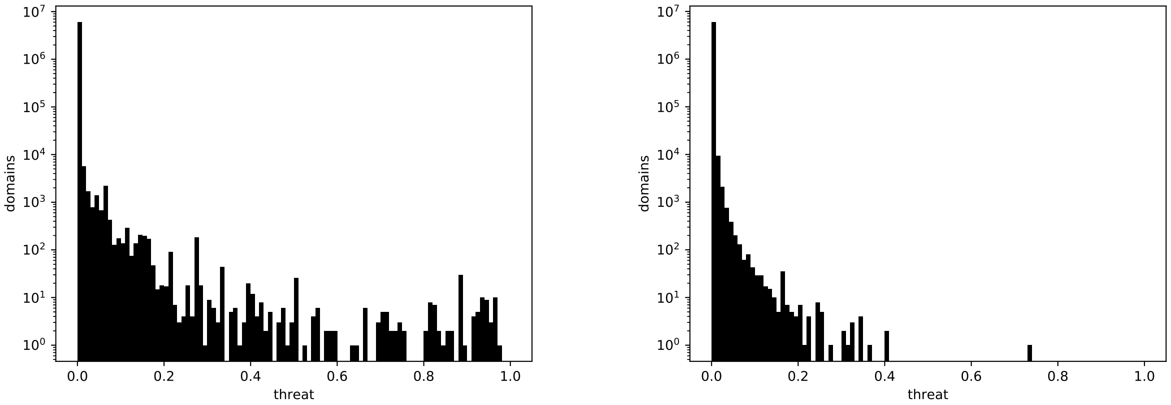


Figure 5.3: Number of detected users if malicious domains with low popularity are ignored. If malicious domains were used as an IOC to detect infected users, pruning graph would only have a marginal impact on the number of detected infected users.



(a) ip-domain graph

(b) user-domain graph

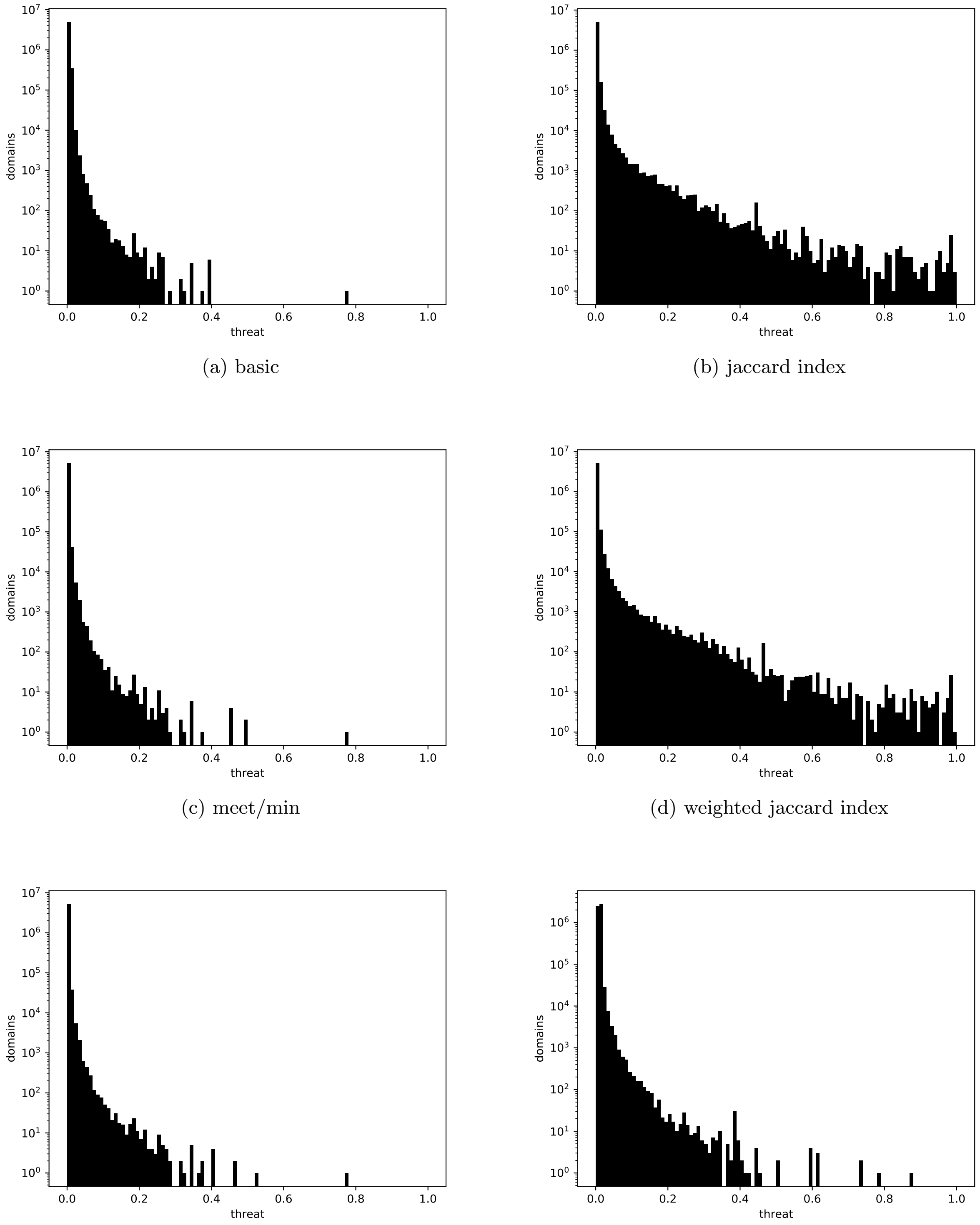
Figure 5.4: Histogram of achieved probabilities of being malicious using two graph representations. Using the user-domain *graph one gets probabilities that are very low, which is caused by the structure of the data. Please note the logarithmic scale.*

5.5.7 Distribution of calculated probabilities

Next, we analyze the distribution of probabilities calculated by PTP to confirm skew of these probabilities towards 0 caused by the use of a bipartite graph. We then analyze probabilities calculated by PTP when run on the proposed unipartite graphs to determine whether the change in the underlying graph improved the distribution.

Distribution of calculated threats for bipartite graphs can be found in Figure 5.4. The distribution is skewed especially for the *client-domain* graph. In fact, from the 100 domains with the highest threat score from the general experiment, of which 52 were true positives, only one had score higher than 0.4. On the other hand, distribution of the *ip-domain* graph does not seem to be ill-formed.

Skew towards 0 is more pronounced in the *user-domain* graphs due to the higher degree



(a) basic

(b) jaccard index

(c) meet/min

(d) weighted jaccard index

(e) weighted meet/min

(f) statistical test

Figure 5.5: Histogram of achieved probabilities of being malicious for various edge weight functions. Please note the logarithmic scale.

]

of users, as compared to the degree of IP addresses in the *ip-domain* graph. In the iterative calculation of PTP, threat (probability of being malicious) of each node is approximately calculated as an average of threats of its neighbors. Given the high number of domains visited by a client, and the fact that only negligible number of them are known malicious, threat of clients is generally very low. Consequently, only small threat is calculated for unlabeled domains, which is again calculated as an average threat of its neighbors. This skew towards 0 can be also explained using the random walk formalism of PTP - high degree nodes in the path between an unlabeled and a labeled domain considerably decrease the probability of this path and consequently also the calculated threat.

However, skew towards 0 can be also observed for unipartite graphs, with the exception of those based on Jaccard index, as can be seen in Figure 5.5. The difference in the distribution is the consequence of a fundamental difference of how the edge weight is calculated for two domains with considerably different popularities. There are dozens of domains with very high popularity – they are visited by almost every client and their probability of being malicious is low due to their degree. In the case of (weighted) Jaccard index, an edge weight between a domain with low popularity (which malicious domains typically are) and a domain with high popularity is very low. For the remaining edge weight functions the similarity is 1 or very close to 1. Since almost all infected clients frequently visit the legitimate domains as well, malicious domains are joined by an edge to the legitimate and popular domains. If weights of these edges are high, threat of the (unknown) malicious domains is also kept low. On the other hand, if edge weights are low, such as in case of (weigted) Jaccard index, the threat of a node is not impacted much by the threat level of such popular domains.

5.5.8 Differences in findings

In this section, we analyze the similarity of sets of domains detected by the PTP when using the bipartite graphs and the proposed unipartite graphs. Table 5.6 contains Jaccard index of detected domains for each combination of graphs used with PTP. The most notable is the observation that PTP using *ip-domain* graph found different malicious domains than PTP using any other graph. This suggests that different concepts behind the graphs lead to considerably different findings. The *ip-domain* graph excels in identification of malware that heavily relies on frequent migration between domains on a limited set of server hosts. This case is typical, as it avoids domain-based blocking on host and network prevention devices and does not force the attackers to laboriously migrate the infrastructure. However, a growing proportion of diverse malware deviates from this scheme and avoids the detection by the *ip-domain* graph completely. Such evasion techniques is ineffective with the proposed graphs.

There are also differences in sets of detected domains between the proposed graphs. These are not as pronounced as in the previous case and are caused by the behavior of edge weight functions that also impacts the distribution of calculated probabilities.

5.5.9 Correlation

In order to determine whether the proposed edge weight functions capture the same important properties of network communication, we analyze correlation between edge weight functions. Table 5.7 contains Person's correlation coefficients for the described edge weight functions. We can see that Jaccard Index and weighted Jaccard index are highly correlated, and so are Meet/Min and weighted Meet/Min. On the other hand, there is only a weak correlation

Table 5.6: JI of detected domains for the bipartite graphs and unipartite graphs using proposed edge weight functions.

	ip-domain	*user-domain*	basic	jaccard index	meet/min	weighted jaccard	weighted meet/min	statistical test
ip-domain bipartite	1	0	0.0081	0	0.0129	0	0.0123	0
user-domain bipartite	0	1	0.1635	0.3478	0.3784	0.4579	0.7391	0.2021
basic	0.0081	0.1635	1	0.1296	0.3043	0.1081	0.1545	0.6327
jaccard index	0	0.3478	0.1296	1	0.2727	0.7253	0.3306	0.1515
meet/min	0.0129	0.3784	0.3043	0.2727	1	0.2917	0.4722	0.3333
weighted jaccard	0	0.4579	0.1081	0.7253	0.2917	1	0.4336	0.1386
weighted meet/min	0.0123	0.7391	0.1545	0.3306	0.4722	0.4336	1	0.807018
statistical test	0	0.2021	0.6327	0.1515	0.3333	0.1386	0.807018	1

Table 5.7: Pearson's correlation coefficient between edge weights calculated by the proposed functions.

	w_{ji}	w_{mm}	w_{wji}	w_{wmm}	w_{bt}
w_{ji}	1	0.24	0.98	0.24	0.21
w_{mm}	0.24	1	0.23	0.98	0.05
w_{wji}	0.98	0.23	1	0.24	0.18
w_{wmm}	0.24	0.98	0.24	1	0.03
w_{bt}	0.21	0.05	0.18	0.03	1

between Jaccard index and Meet/Min, either weighted or unweighted. Similarly, there is a weak correlation between the statistical test and the (weighted) Jaccard index. In contrast, there is no correlation between statistical test and (weighted) Meet/Min.

Additionally, to show relationships between selected edge weight functions, in Figure 5.6 we present a scatter plot of edges in the coordinates calculated by the edge wight functions. Unsurprisingly, Meet/Min assigns higher edge weights than Jaccard index; and weighted Meet/Min generally assigns higher edge weights compared to weighted Jaccard index.

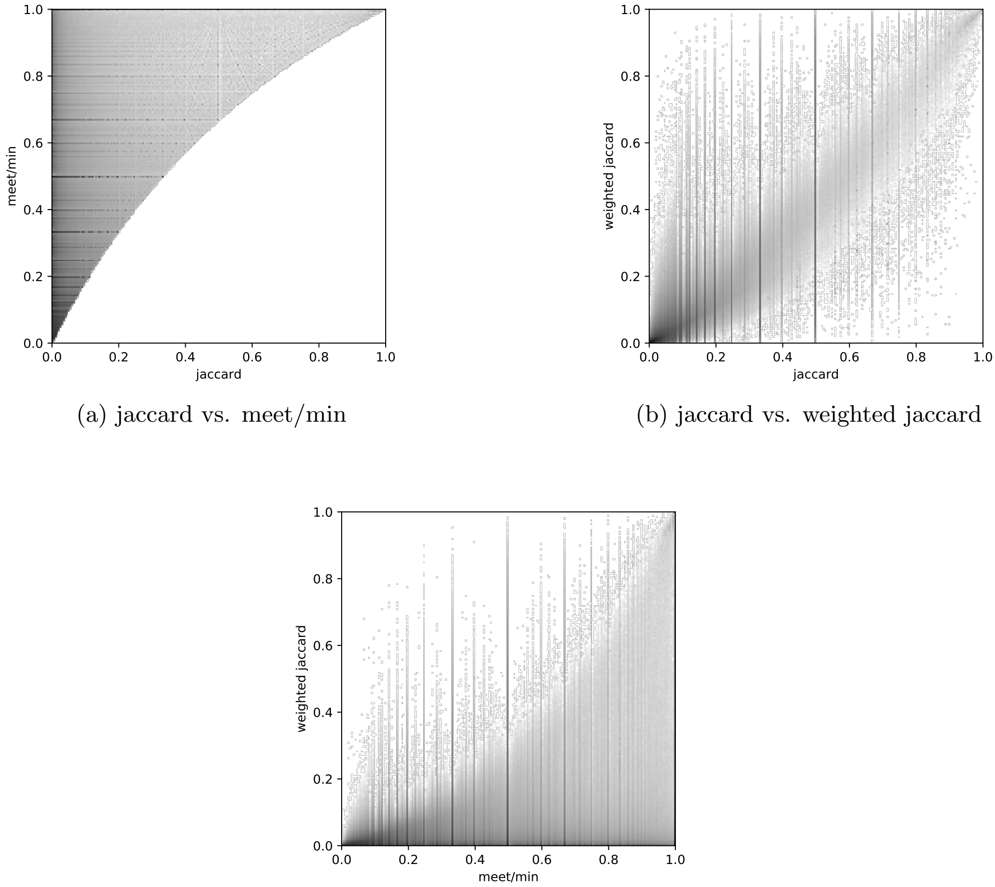
5.6 Discussion & remarks

Results in Table 5.5 suggest that four edge weight functions - w_{ji}, w_{wji}, w_{mm} and w_{wmm} outperform the two remaining edge weight functions – w_b and w_{bt} as well as the *client-domain* graph. Of the four, neither is clearly the best across all available data sets. Based on the degree distribution of the calculated probabilities we lean towards choosing either w_{ji} or w_{wji} as the best option.

The reason why we do not use cross-validation to evaluate the proposed edge weight functions lies in the labels we have available. In our experiments, we use relatively small list of malicious domains and have no legitimate labels available. Altogether, we have labels for only 0.055% of domains that are present in the data set. An option would be to consider all unlabeled domains to be legitimate. This, however, goes against the claim that blacklists have small recall which implies that there are many malicious domains that would be considered legitimate in the evaluation. In such case, the method would identify many domains that are truly malicious but due to the lack of labels would be marked as *false positives*. For comparison, evaluation in [89] was done using cross-validation. However, only labeled domains were used to determine true positive and false positive rates. Having labels for only 1.45% of domains means that the true performance of the evaluated method can be considerably different.

We assume that the reason for adoption of bipartite graphs instead of unipartite graphs is their lower memory complexity. Bipartite graphs have more nodes, but considerably less edges than their unipartite counterparts, at least when converted using the proposed edge weight functions. This was also observed in [136] and stated as a reason to use the bipartite graph. In fact, all domains visited by an user form a clique, and thus there are as many cliques in the unipartite graph as there are clients in the data used to construct the graph.

A natural solution would be to use the thresholding. Due to the properties of the edge weight functions, thresholding is a reasonable choice only for the w_{ji} and w_{wji}. Other edge



(a) jaccard vs. meet/min

(b) jaccard vs. weighted jaccard

(c) meet/min vs. weighted jaccard

Figure 5.6: Scatter plot of edges, each point in the plot is an edge with coordinates determined by two selected edge weight function. Color intensity is proportional to the number of edges with the given coordinates and is log scaled. As can be seen in (a), w_{mm} consistently attains higher values than w_{ji}, as is expected. (b) shows a visual clue that w_{ji} and w_{wji} are correlated. Finally, in (c) one can see that w_{mm} usually attains higher values than w_{wji}.

weight functions have the tendency to keep the edge weight high or even 1, which is also the reason for the skew in the distribution probabilities towards 0. In our approach we avoided thresholding, since we have already pruned the tree – removed domains removed domains with low popularity. Proposed pruning strategy not only improves precision of the graphs based on monitoring client - domains connections but also solves the issues with memory requirements.

Finally, we would like to address the question of the quality of data. Data used for construction of the *ip-domain* graph are less prone to containing unwanted artifacts. The only artifact we were able to identify was the presence of blackholing IP addresses. These were easily identified in the data, due to their high degree and the fact that only malicious domains are hosted on them. In contrast, data we used to construct the *client-domain* graph have many artifacts. Since we have no control over the networks, network administrators are free to use any network setup they wish. Therefore, in our data we see devices posing as clients that are in fact internal proxy servers. These can appear to visit overwhelming number of domains, which also has implications for the memory required to store the graph – size of the edges in the clique rises quadratically with the number of nodes in the clique. Another artifact commonly seen in the data is a single device changing identity, for example an IP address. In our experiments, we removed 1500 most active clients from the data. This number was chosen based on the degree distribution of clients in the *client-domain* graph. We are certain many proxy servers posing as clients remained in the data, therefore we suggest to use a proper proxy detector, such as [74] to identify proxies posing as clients. We did not attempt to merge clients that represented a single physical device. As a consequence, the precision achieved using the client-domain based graph representation can be likely improved, given the data can be properly sanitized.

5.7 Conclusion

In this chapter, we addressed the task of identifying additional malicious domains given the knowledge of some of them, specifically using Probabilistic Threat Propagation. The contribution of this chapter is three-fold:

1. We use data about client-domain connections to infer maliciousness of domains, instead ip-domain hosting information used in prior art [19]. Using two complementary views of the network data leads to identification of different malicious domains in each case. The data about client-domain connections should be therefore used alongside the original ip-domain hosting information to achieve better coverage of detected threats. It is also harder for attackers to avoid detection by PTP used with the graph based on the client-domain connections, as opposed to the graph based on ip-domain hosting data.

2. We propose to use unipartite graph format to use instead of a bipartite graph format. We show in evaluation that using unipartite graphs leads to better precision of PTP. Unipartite graph is a compressed representation of network communication which better protects privacy of the clients whose connections were used to construct the graph. Additionally, we propose three novel edge weight function – weighted versions of jaccard index and meet/min and edge weight based on binomial statistical test.

3. We propose graph pruning that considerably improves precision achieved by PTP and at the same time has only marginal impact on the number of detected users, if the identified malicious domains were to be used as indicators of compromise.

All proposed graphs and methods were evaluated on a large corpus of data, containing network traffic from hundreds of separate networks and millions of clients.

Chapter 6

Behavioral modeling of domains

In the previous chapter, we have shown the power of guilt by association approaches to discover cooperating malicious domains in a graph structure, modeled on top of HTTP telemetry data. In this chapter, we introduce an orthogonal technique allowing discovery of malicious campaigns using similarity of patterns in HTTP communication.

HTTP communication to various targets (in telemetry identified either by domain names or IPs) can be analyzed on multiple levels of granularity. In search of patterns in HTTP communication, it is possible to analyze HTTP communication at the level of individual HTTP requests, or alternatively analyze group of thereof. In both cases the idea is to discover and utilize potentially discriminative characteristics of such patterns w.r.t. individual targets. However, our goal here will not be the discovery of specific patterns, instead we will utilize their existence implicitly to consequently evaluate behavioral similarity of communication targets, such as domains.

The core of this chapter therefore lies in the definition of two kernel function suitable for analysis of HTTP telemetry. Kernel function can be considered as a measure of similarity and can be also used to define distance. With the kernel, similarity function and distance function define, we enable three principal use cases, all of which have been now implemented in Cognitive Threat Analytics [135] and verified on large industrial scale:

1. Classification (or anomaly detection). Classifying HTTP telemetry using standard techniques is principally difficult due to data structure that makes transformation to vector form hard. Definition of kernels allows to bypass the representation problem; the proposed kernels can be used in kernelized classifiers for a straightforward solution of the classification problem.

2. Clustering, i.e., campaign discovery in unsupervised setting. One of principal network security problems is the discovery of previously unseen infections. Assuming a big enough snapshot of telemetry is available, we can make use of the fact that malicious campaigns need to actively communicate and that their communication often exhibits similar patterns. Measure of similarity can thus be used to discover groups of similarly communicating nework nodes. There is no guarantee that each discovered group/cluster is malicious, but in concert with additional techniques allowing prioritization of clustering results, the proposed metrics become the basis of a powerful technique for malicious campaign discovery.

3. Finally, the impact of our proposed measures of similarity can be multiplied by combin-

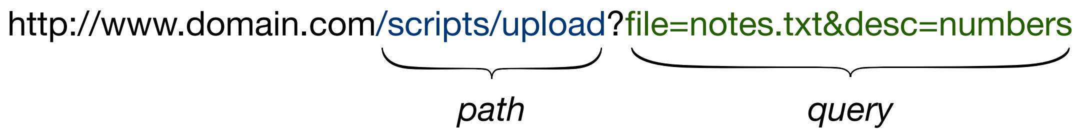

http://www.domain.com/scripts/upload?file=notes.txt&desc=numbers
path
query

Figure 6.1: Visualization of different parts of URL.

ing the clustering technique (use case 2) with the guilt by association approach described in the previous chapter. A recurrent loop of two repeating steps - new cluster discovery followed by threat propagation - leads to solid and extensive discovery of malicious campaigns whose performance has been verified on industrial scale. The key advantage here is the combination of multiple principally different ways to discover concealed relations between infected network nodes.

6.1 Behavioral models & kernels

Behavior of a server can be characterized in many ways. We propose two distinctive "views" on behavior of servers based on features extracted from HTTP traffic (further denoted as $\mathbb{W}$), namely

1. behavior represented by the HTTP GET query parameters sent to the server (query-based similarity),
2. behavior represented by the paths visited on the server (path-based similarity).

Query parameter and path can be both parsed from the URL. Example of an URL with highlighted *path* and *query parameters* can be found in Figure 6.1.

These behavioral models capture two different aspects of the behavior of domains. It is therefore advantageous to use them in combination to get a broader description of a domains behavior. Some domains may exhibit *no* behavior from the perspective of the individual behavioral models. These can be domains that accept only encrypted connections and for which we do not see neither path not query parameters. For such domains, the kernel and consequently similarity to any other domain under the given model is 0.

6.1.1 Query-based behavioral modeling

A query string is a part of the URL that carries information from client to a specific function or resource on the server in a form of key-value pairs without hierarchical structure or specific order. We assume that servers serving the same application also receive the same parameters via the query string. Values themselves are not that important, since they can differ from client to client but their types (number, string, e-mail address, etc.) are typically bundled with the parameters. Therefore, we extract all query parameter keys from $\mathbb{W}$ and augment every key k with the type of corresponding value (see Table 6.1).

To define similarity between two servers, we adopt bag-of-words [90] model, which is widely used in text mining. The vocabulary V is formed by all keys extracted from $\mathbb{W}$ enriched with

Table 6.1: An example of URL with query string and extracted keys augmented with type of corresponding types of values.

`http://www.abc.com/av?sv=1&v=3.0.5&e=651af&u=http%3A%2F%2Fxyz.com%2F&if=0`
`sv_number  v_version  e_string  u_url  if_number`

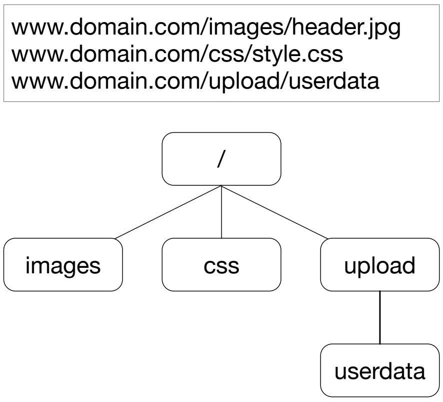

www.domain.com/images/header.jpg
www.domain.com/css/style.css
www.domain.com/upload/userdata
/
images
css
upload
userdata

Figure 6.2: Three URLs and a path tree constructed from them. Each path is first separated to path fragments and tree is built from top to bottom, representing the directory structure.

the value type. The server s is then represented as sparse vector defined as

$$q_s = (k_1, k_2, \dots, k_v) \tag{6.1}$$

where k_j is equal to the number of occurrences of key k_j in set of flows from $\mathbb{W}$ targeting server s, denoted $\mathbb{W}_s$. However, some query parameter names are very common and do not discriminate between servers (*id*, *ver*, etc.). To address this, we use *term frequency–inverse document frequency* scaling (TF-IDF) [91].

Finally, we employ cosine similarity on vectors q_s to determine similarity of two servers. Only servers that receive at least one query parameter are considered in the similarity calculation. It can be shown that cosine similarity is a valid kernel function.

6.1.2 Path-based behavioral modeling

The idea behind using paths in HTTP requests to determine similarity between servers is based on the assumption that two servers providing the same service, likely provide it at the same location specified by path.

Representation To reflect this assumption, we propose a representation that captures the directory structure of all paths ever visited on servers. Specifically, each server is represented by a tree, whose root is denoted "/" and each path from root to a leaf represents one particular path observed in URL where each node in the tree specifies one directory in the hierarchy.

Construction of path trees is simple. Paths from each URL are split by the directory separator and a tree is built from top to bottom, representing the directory structure. Figure 6.2 shows an example of several paths and the associated tree.

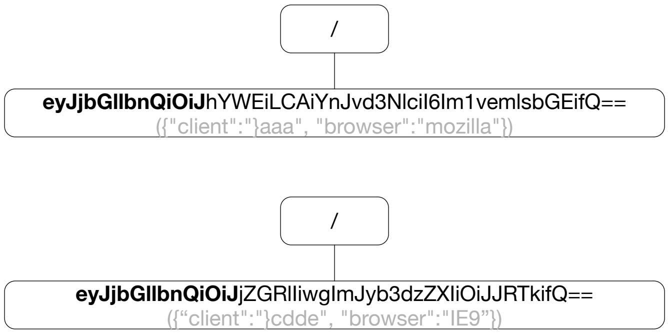

/
eyJjbGllbnQiOiJhYWEiLCAiYnJvd3NlciI6Im1vemlsbGEifQ==
({"client":"}aaa", "browser":"mozilla"})
/
eyJjbGllbnQiOiJjZGRlIiwgImJyb3dzZXIiOiJJRTkifQ==
({“client":"}cdde", "browser":"IE9”})

Figure 6.3: One of the options to tunnel data from client to server is to send base64 encoded json files. Figure shows an example of the path and data being sent.

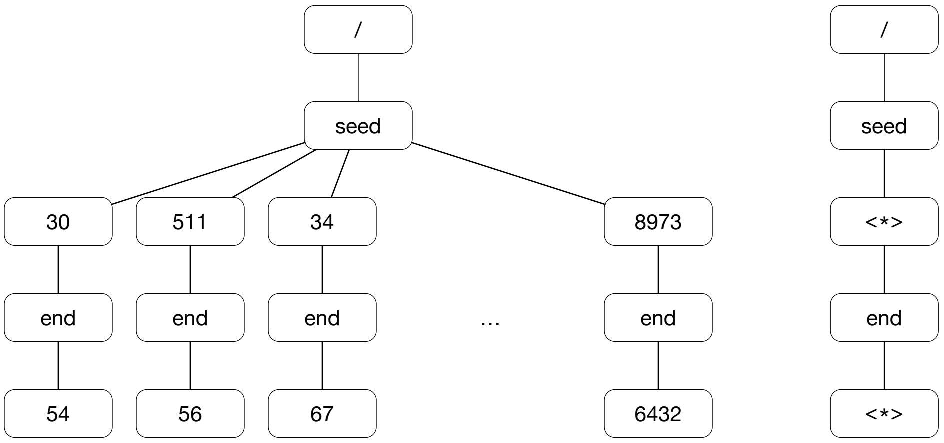
The importance of ordering path elements is also the reason why we do not use the bag-of-words representation in this case.

Path in the URL does not always serve only as a pointer to the resource requested by the client. Both legitimate and malicious applications may use URL to tunnel (or exfiltrate) data from client to the server. Figure 6.3 shows an example of an URL tunneling of a Base64-encoded JSON. In this case, the corresponding part of the path cannot be considered a pointer to the resource, but rather looked at as data. To capture information about the tunneled data, we define new type of a node in the tree – the *data node*. In the rest of the text we refer to the nodes representing directories as *simple nodes*. Data node keeps description of the data being tunneled. The exfiltrated information is not constant and changes in time. Therefore, the encoded strings are not exactly the same. On the other hand, because the structure of the data is always the same for the same malware, we can see some regularities in the encoded strings. The data node keeps track of parts of the data strings that are constant across all requests. In the case of URLs in Figure 6.3, the constant positions are emphasized in bold. Specifically, each data node is represented by a set of such positions. In the case depicted in Figure 6.3, the respective data node contains numbers between 1 and 15.

There is one more typical URL pattern that poses challenges to the proposed representation. Usually, parameters are specified in the query string, e.g. after the “?” symbol. However, in some applications the parameters are transferred using the path, for example it is often used in streaming media. An example of the parameter tunneling is depicted in Figure 6.4a. As you can see, the trees grow to large sizes and children nodes of the parameter values are redundant because most of them have the same value. Therefore, when we detect that parameters are being sent via the path, we join nodes representing parameter values into a single node with special value `<*>`. Figure 6.4b shows the same tree after compression.

Kernel function While there are measures of (dis)similarity for trees [54, 29], such as edit distance [13], we propose a similarity function tailored to our specific domain where ordering of the path elements is crucial but at the same time, ordering of children of any node is not. We take advantage of the kernel formalism [125] that allows us to define dot product of two trees in a transformed space. We define kernel

$$\mathrm{K}(n,m) = \mathrm{L}(n,m) \cdot (1 + C \cdot \sum_{\substack{u \in \mathrm{sub}(n) \\ v \in \mathrm{sub}(m)}} \mathrm{K}(u,v)), \tag{6.2}$$



(a) original

(b) compressed

Figure 6.4: Original and compressed tree that represents domain that uses path to tunnel parameters.

where n and m are two nodes, sub($\cdot$) returns all children of the node, C determines how quickly the importance of deeper tree levels increases or decreases, and $\mathrm{L}(n, m)$ is defined as

$$\mathrm{L}(n, m) = \begin{cases} \mathbb{I}_{\{n=m\}} & \text{if n,m are both ordinary nodes,} \\ \mathrm{JI}(n, m) & \text{if n,m are both data nodes,} \\ 0 & \text{otherwise,} \end{cases} \tag{6.3}$$

where $\mathbb{I}_{\{n=m\}}$ is equal to 1 if m and n represent the same directory (name), otherwise it is equal to 0. Data nodes are represented as sets of integers, and $\mathrm{JI}(n, m)$ of two data nodes is Jaccard index of these sets.

It can be shown that K is a valid kernel function and can be used to quantify similarity of two trees.

Not all HTTP request specify path, part of them can target the root directory itself. Such domains are not considered in the model, and their similarity to other domains from the point of view of the path-based model is 0.

6.2 Evaluation

In this section we compare the proposed similarity measures to the distance measure introduced in [106] and also described in Section 2.6. It is advantageous to use the proposed similarity measures in combination, since they capture different aspects of behavior. Therefore, for the purpose of evaluation we define similarity of two domains s_p to be an average of the two proposed similarities. We denote baseline distance function defined in [106] as d_b. Performance of similarity/distance functions is compared in a real-world scenario – unsupervised clustering of domains, which should group together malicious domains from the same malware campaigns.

6.2.1 Data Set Description

The comparison is done on a data set collected for 30 days in June and July 2017. It contains all HTTP requests that targeted a domain that participates in any of the monitored malicious campaigns described in Section 6.2.2. Altogether, it contains HTTP requests targeting 1267 unique malicious domains belonging to 150 malicious campaigns.

6.2.2 Source of Labels

As a source of labels we use the knowledge of malicious campaigns currently being tracked by Cisco CTA [135]. Malicious campaign is represented by a collection of second level domains that participate to achieve a common goal, or generally serve the same purpose. At the time of the experiment, 279 malicious campaigns were tracked, representing various threat types, including ransomware, information stealers, banking trojans, exploit kits, ad injectors, click frauds and others. Not all malicious campaigns are necessarily present in the data set.

6.2.3 Evaluation methodology

We compare the measures of similarity/distance based on the quality of clustering provided by a clustering algorithm. Various clustering algorithms offer different quality of results, often depending on the data. Therefore we do not evaluate similarity/distance functions in isolation, but rather in combination with a clustering algorithm. We chose five well established clustering algorithms to be used in this evaluation:

- hierarchical clustering with complete linkage (CLINK) [30],
- hierarchical clustering with single linkage (SLINK) [126],
- generalized density-based clustering (GDBSCAN) [120],
- K-Medoids [70],
- greedy modularity based (Louvian) [14] .

Clustering algorithms have various parameters that need to be tuned. Specifically, CLINK and SLINK are parametrized by distance threshold, GDBscan is parametrized by distance threshold and minimal cluster size, K-medoids is parametrized by number of clusters. Louvain does not have any tunable parameters, but we noticed strong dependence of the quality of produced clustering on the used edge weight threshold (edges with weight lower than the threshold are removed), therefore we optimize it as an algorithm parameter.

In order to optimize parameters of the clustering algorithms, we split the data into a training and testing sets. The split is done at the granularity of domains; therefore no domain is present in both the training and testing data set and HTTP requests targeting a single domain are all either in the training or in testing set. No other rules were enforced while splitting the data, therefore it is possible that some clusters might be missing in either training or testing set. Information about the training and testing sets are summarized in Table 6.2 Parameters of the clustering algorithms are optimized for each pair of metric and clustering algorithm.

Table 6.2: Description of the training and testing data set.

	domains	campaigns
training set	790	124
testing set	477	108
total	1267	150

While Louvain is a similarity-based algorithm, all remaining algorithms are distance-based. Therefore, we define the following conversion function to convert the similarity and distance functions in the required form.

$$s_b = 1 - \frac{d_b}{22} \tag{6.4}$$

$$d_p = 1 - s_p \tag{6.5}$$

where 22 is the maximal value of d_b. s_b is thus the similarity function defined based on the baseline distance function and d_p is distance function based on the proposed similarity function.

The baseline similarity does not use a single entity to represent a domain. Distance of two domains is calculated using an aggregation function over pairwise similarities of all HTTP requests targeting those domains. Calculating the distance between two domains is therefore expensive. Because of that, in the comparison we use only version of the baseline where at most 1, 2 and 5 HTTP requests per domain are used to calculate pairwise distances. When the actual number of HTTP requests targeting a domain is higher than that, the requests used in distance calculation are chosen randomly.

6.2.4 Quality of clustering

Quality of clustering is evaluated using Adjusted Rand Index [114] (ARI), a well-known measure of similarity between two clusterings. ARI can attain values in the interval $[-1, 1]$ – value of 1 suggesting that the two clusterings are identical and negative value of ARI suggesting that agreement between two clusterings is less than what is expected from a random result. For each clustering produced by any clustering method and similarity or distance measure we calculate ARI to the *true* clustering of the data. The *true* clustering is constructed based on the malicious campaigns contained in the ground truth. Domains from single malicious campaign always form a cluster. True clustering in this case contains 150 clusters. Louvain and K-medoids clustering algorithms depend either on a random initialization or random order of calculation. Both are therefore re-run 5 times and an average ARI is calculated for them.

Measured ARIs can be found in Table 6.3. The overall best clustering is produced by the Louvain method using the proposed similarity functions. The second-best result is again attained by the proposed similarity functions, used with CLINK clustering algorithm.

6.2.5 Computational Performance

We further compare computational requirements of the proposed similarity functions and the baseline. Time needed to construct a graph encoding pairwise similarities of domains is measured. The baseline distance function does not use a single entity to represent a domain.

Table 6.3: ARI for the compared metrics using 5 well established clustering algorithms.

	CLINK	SLINK	GDBSCAN	K-Medoids	Louvain
Perdisci (n=1)	0.350	*0.562*	*0.562*	0.534	0.599
Perdisci (n=2)	0.524	0.497	0.497	0.604	0.603
Perdisci (n=5)	0.462	0.424	0.472	*0.605*	0.610
proposed	*0.668*	0.495	0.417	0.412	**0.708**

Table 6.4: Time required to calculate pairwise similarities of domains present in the experiment (in seconds).

	avg	std
Perdisci (n=1)	26.85	0.62
Perdisci (n=2)	78.59	0.35
Perdisci (n=5)	323.86	1.57
Perdisci (n=7)	556.10	11.53
Perdisci (n=9)	795.95	6.57
Perdisci (n=11)	1,053.46	7.36
proposed	29.18	1.53

Distance of two domains is calculated using an aggregation function over pairwise similarities of all HTTP requests targeting those domains. Calculating the distance between two domains is therefore expensive, and the actual computation time depends on the number of HTTP requests targeting domains in question. To capture such dependence, we evaluate several version of the baseline distance, where the maximal number of HTTP requests per domain used to calculate the distance is denoted in the parentheses. The dependence on the number of HTTP requests can be clearly seen in Table 6.4. It is also clear from the table, that the proposed representations and calculation of similarities can be computed in a comparable time to the baseline distance using only a single HTTP request. Despite the similar run time, the proposed metric uses information from **all** HTTP requests and can thus better determine the similarity/distance of/between domains.

6.3 Conclusion

In this chapter we proposed a novel behavioral representation of domains and related kernel function and similarity measures s that can be used

- in a supervised setting to detect known threats,
- in an unsupervised setting to help analysts find hidden patterns of network traffic,
- in combination with guilt-by-association approach to progressively find new classes malicious behaviors and enumerate all domains participating in it.

In evaluation, we demonstrated that the proposed representation used together with the proposed kernel function offer superior performance compared to the state of the art. Using the proposed kernel functions and associated similarity functions we were able to achieve better clustering of domains – ARI of the resulting clustering was improved by 0.1, which is an increase by 17%. At the same time, calculation of pairwise similarities/distances of domains using the proposed similarity function is much faster than state of the art.

Chapter 7

Conclusion

This thesis has answered two very specific research questions presented in the introduction. The first question was: *"How to detect all P2P networks in a monitored physical network?"*.

To answer this question, we have designed a method for peer-to-peer network discovery presented in Chapter 3. The method recorded remote peers of known local hosts participating in a P2P network. The remote hosts were then used to further identify other internal hosts that are participating in the same P2P network, thus iteratively building a full picture of the peer-to-peer infrastructure. The method works with near-zero false positives (except for a single BitTorrent case) and nearly a 100% recall for most networks with a particular exception of Gnutella. This result has been presented as a journal paper [65] and constitutes one of the main contributions of the thesis.

To answer the second question: *"How can we use connection information to reveal C&C structures of malicious botnets?"*, we had to rely on a combination of techniques. This was due to a broader variety of malware command & control techniques and approaches. On the other hand, this problem has been increasingly important, as a majority of malware has opted for the use of HTTP as the C&C protocol.

In Chapter 5, we have laid the groundwork needed for the discovery of C&C servers by means of extended Probabilistic Threat Propagation. Instead of relying on the relationship between the domain names and their hosting servers, we have decided to exploit the communication patterns between client hosts and domains, some of which were used as C&C servers. Information about the communication patterns was represented in a form of unipartite graph. This graph, while still huge, significantly compressed the volume of data needed to represent the structure of communication. The application of the PTP algorithm on this graph has yielded completely new malware servers, previously undetectable by the PTP. The proposed graph used with PTP is more resilient to evasion attempts by malware authors. In fact, if they wanted to evade detection by the proposed method they would be to abandon the already infected botnet hosts together with all the C&C servers and build a new botnet from scratch.

We recognize that malicious domains can be also identified by behavior which tends to be similar for all domains participating in a single malicious campaign, and different across the malicious campaigns. In Chapter 6 we therefore propose similarity measures based on behavioral modeling of domains. Similarity measures are based on kernel functions, therefore can be used with ease with any kernelized classifier or clustering algorithm.

Individual contributions from Chapter 5 and Chapter 6 reinforce each other. Two methods can be used together with various goals in mind. First, behavioral modeling can be used to cluster together domains that exhibit the same behavior and results of PTP can be used to score such clusters. High-score clusters are likely to represent a set of malicious domains with similar behavior. Second, behavioral modeling can be used to extend results of PTP. We showed that pruning the unipartite graph based on client-domain connections improves the precision of PTP. On the other hand, it also lowers the recall. Behavioral modeling can be thus used to find domains pruned from the graph that have behavior similar to those of found malicious domains. Third, behavioral modeling can be used to extend the set of seeds used by PTP, which was the subject of a journal paper [66].

Besides scholarly publications that include two articles in impacted journals, the work on the thesis also resulted in 3 US patents already issued by USPTO and 2 patent submissions still pending USPTO approval.

All proposed methods are critical components of CTA, an on-line malware detection security-as-a-service product delivered by Cisco Systems [135]. CTA analyzes more than 10 billion web requests and other network communication generated by millions of users from a number of large enterprise networks. The system finds daily tens of thousands of network threats that evaded previously installed security measures, which positions the system as the last line of network defense.

7.1 List of author's publications

Articles in Journals with Impact Factor (2)

1. **Ján Jusko**, Martin Rehák, Jan Stiborek, Jan Kohout, Tomáš Pevný. Using Behavioral Similarity for Botnet Command-and-Control Discovery. In: *IEEE Intelligent Systems.* 2016, 31(5), pages 16–22. Impact factor 2.37 (**50%**)

2. **Ján Jusko**, Martin Rehák, Identifying peer-to-peer communities in the network by connection graph analysis. In: *International Journal of Network Management.* 2014, 24, pages 235–252. Impact factor 1.118 (**80%**)

Peer-reviewed Journal Articles (1)

1. Jan Stiborek, Martin Grill, Martin Rehák, Karel Bartoš and **Ján Jusko**. Game Theoretical Model for Adaptive Intrusion Detection System. In: *Lecture Notes in Computer Science: Transactions on Computational Collective Intelligence*, vol. 15, pages 133–163, 2014. (**20%**)

Patents (3)

1. Jan Kohout, **Ján Jusko**, Martin Rehák, Tomáš Pevný. Detection of malicious network connections. US Patent 9,531,742, 2016. (Issued)

2. Ivan Nikolaev, Martin Grill, **Ján Jusko**. Detecting Network Services Based On Network Flow Data. US Patent 20,160,080,236, 2016. (Issued)

3. **Ján Jusko**, Tomáš Pevný, Martin Rehák. Server grouping system. US Patent 9,596,321, 2017. (Issued)

In ISI Proceedings (3)

1. Jan Stiborek, Martin Grill, Martin Rehák, Karel Bartoš and **Ján Jusko**. Game Theoretical Adaptation Model for Intrusion Detection System. In: *Proceedings of the 10th Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, pages 201–210, Springer Berlin, 2012. (**20%**)

2. **Ján Jusko**, Martin Rehák, Tomáš Pevný. A memory efficient privacy preserving representation of connection graphs. In: *Proceedings of the 1st International Workshop on Agents and CyberSecurity.* 2014, Article 4 , 8 pages. (**60%**)

3. **Ján Jusko**, Martin Rehák. Identifying Skype nodes by NetFlow-based graph analysis. In: *8th International Wireless Communications and Mobile Computing Conference (IWCMC).* 2012, pages 636–641. (**80%**)

In Other Proceedings (2)

1. **Ján Jusko**, Martin Rehák. Revealing Cooperating Hosts by Connection Graph Analysis. In: *Security and Privacy in Communication Networks, SecureComm 2012.* 2012, pages 241–255. (**80%**)

2. **Ján Jusko**, Martin Rehák. Identifying skype nodes in the network exploiting mutual contacts. In: *Traffic Monitoring and Analysis, TMA 2012.* 2012. (**80%**)

Bibliography

[1] Bitcoin - open source p2p money. http://bitcoin.org. Accessed: 2013-10-18.

[2] Waste protocol. http://en.wikipedia.org/wiki/WASTE.

[3] William Acosta and Surendar Chandra. Trace driven analysis of the long term evolution of gnutella peer-to-peer traffic. In *Proceedings of the 8th international conference on Passive and active network measurement*, PAM'07, pages 42–51, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] Alan Agresti and David B Hitchcock. Bayesian inference for categorical data analysis. *Statistical Methods & Applications*, 14(3):297–330, 2005.

[5] Muhammad Aurangzeb Ahmad, Brian Keegan, Atanu Roy, Dmitri Williams, Jaideep Srivastava, and Noshir Contractor. Guilt by association? network based propagation approaches for gold farmer detection. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 121–126. IEEE, 2013.

[6] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *arXiv preprint cond-mat/0008064*, 2000.

[7] Sumayah Alrwais, Xiaojing Liao, Xianghang Mi, Peng Wang, XiaoFeng Wang, Feng Qian, Raheem Beyah, and Damon McCoy. Under the shadow of sunshine: Understanding and detecting bulletproof hosting on legitimate service provider networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 805–823. IEEE, 2017.

[8] Brian Amento, Loren Terveen, and Will Hill. Does "authority" mean quality? predicting expert quality ratings of web documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 296–303. ACM, 2000.

[9] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *USENIX security symposium*, pages 273–290, 2010.

[10] G. Bartlett, J. Heidemann, and C. Papadopoulos. Inherent behaviors for on-line detection of peer-to-peer file sharing. In *IEEE Global Internet Symposium, 2007*, pages 55–60, may 2007.

[11] Daniel S Bernstein, Zhengzhu Feng, Brian Neil Levine, and Shlomo Zilberstein. Adaptive peer selection. In *International Workshop on Peer-to-Peer Systems*, pages 237–246. Springer, 2003.

[12] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. Poisoning behavioral malware clustering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 27–36. ACM, 2014.

[13] Philip Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1):217–239, 2005.

[14] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[15] Dario Bonfiglio, M. Mellia, M. Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. *ACM SIGCOMM Computer Communication Review*, 37(4):37–48, 2007.

[16] Giovanni Bottazzi and Gianluigi Me. The botnet revenue model. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 459. ACM, 2014.

[17] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998.

[18] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *Usenix security symposium*, page 15, 2011.

[19] Kevin M Carter, Nwokedi Idika, and William W Streilein. Probabilistic threat propagation for network security. *IEEE Transactions on Information Forensics and Security*, 9(9):1394–1405, 2014.

[20] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, December 2002.

[21] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2006.

[22] Duen Horng "Polo" Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 131–142. SIAM, 2011.

[23] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM, 2003.

[24] M Patrick Collins, Timothy J Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. Using uncleanliness to predict future botnet addresses. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 93–104. ACM, 2007.

[25] F. Constantinou and P. Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pages 93–102, july 2006.

[26] Baris Coskun, Sven Dietrich, and Nasir Memon. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 131–140, New York, NY, USA, 2010. ACM.

[27] David Dagon, Guofei Gu, Christopher P. Lee, and Wenke Lee. A taxonomy of botnet structures. In *In Proc. of the 23 Annual Computer Security Applications Conference (ACSAC'07*, 2007.

[28] Carlton R. Davis, Stephen Neville, José M. Fernandez, Jean-Marc Robert, and John McHugh. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures?. In *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 461–480. Springer, 2008.

[29] Damien M de Vienne, Tatiana Giraud, and Olivier C Martin. A congruence index for testing topological similarity between trees. *Bioinformatics*, 23(23):3119–3124, 2007.

[30] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[31] David Dittrich and Sven Dietrich. P2p as botnet command and control: a deeper insight. In *In Proceedings of the 3rd International Conference On Malicious and Unwanted Software (Malware 2008*, pages 46–63, 2008.

[32] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, UK, 2002. Springer-Verlag.

[33] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley, 2001.

[34] Sven Ehlert, Sandrine Petgang, and T Magedanz. Analysis and signature of Skype VoIP session traffic. In *4th IASTED International*, 2006.

[35] P. Erdös and A. Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[36] P. Erdös and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[37] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a million user dht. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 129–134, New York, NY, USA, 2007. ACM.

[38] Nicolas Falliere. Sality: Story of a peer-to-peer viral network. *Rapport technique, Symantec Corporation*, 2011.

[39] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[40] Agner Fog. Biased urn theory. 2015.

[41] Santo Fortunato, Marián Boguñá, Alessandro Flammini, and Filippo Menczer. Approximating pagerank from in-degree. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–71. Springer, 2006.

[42] Jérôme François, Shaonan Wang, Thomas Engel, et al. Bottrack: tracking botnets using netflow and pagerank. In *International Conference on Research in Networking*, pages 1–14. Springer, 2011.

[43] Jason Franklin, Adrian Perrig, Vern Paxson, and Stefan Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *ACM conference on Computer and communications security*, pages 375–388, 2007.

[44] Lingyun Fu, Haitao Qu, Hui Chen, Huasong Wang, and Xiaohui Wang. A Hierarchical and Heterogeneous P2P-SIP Architecture. In *Pervasive Computing and Applications, 2008. ICPCA 2008. Third International Conference on*, volume 2, pages 995–998, 2008.

[45] Carrie Gates and Carol Taylor. Challenging the anomaly detection paradigm: a provocative discussion. In *Proceedings of the 2006 workshop on New security paradigms*, pages 21–29. ACM, 2006.

[46] Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, and Dina Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 326–345, Berlin, Heidelberg, 2009. Springer-Verlag.

[47] Debra S Goldberg and Frederick P Roth. Assessing experimentally derived interactions in a small world. *Proceedings of the National Academy of Sciences*, 100(8):4372–4376, 2003.

[48] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, Berkeley, CA, USA, 2007. USENIX Association.

[49] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS'06: The 5th International Workshop on Peer-to-Peer Systems*. Microsoft Research, 2006.

[50] Duc T. Ha, Guanhua Yan, Stephan Eidenbenz, and Hung Q. Ngo. On the effectiveness of structural detection and defense against p2p-based botnets. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'09)*, Estoril, Lisbon, Portugal, June 2009.

[51] Jiwoon Ha, Soon-Hyoung Kwon, Sang-Wook Kim, Christos Faloutsos, and Sunju Park. Top-n recommendation through belief propagation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2343–2346. ACM, 2012.

[52] Steffen Haas, Shankar Karuppayah, Selvakumar Manickam, Max Mühlhäuser, and Mathias Fischer. On the resilience of p2p-based botnet graphs. In *Communications and Network Security (CNS), 2016 IEEE Conference on*, pages 225–233. IEEE, 2016.

[53] Irfan Ul Haq, Sardar Ali, Hassan Khan, and Syed Ali Khayam. What is the impact of p2p traffic on anomaly detection? In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 1–17, Berlin, Heidelberg, 2010. Springer-Verlag.

[54] Stephen C Hirtle. Lattice-based similarity measures between ordered trees. *Journal of Mathematical Psychology*, 25(3):206–225, 1982.

[55] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical review E*, 65(5):056109, 2002.

[56] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, pages 9:1–9:9, Berkeley, CA, USA, 2008. USENIX Association.

[57] Marios Iliofotou, Hyun chul Kim, Michalis Faloutsos, Michael Mitzenmacher, Prashanth Pappu, and George Varghese. Graption: A graph-based p2p traffic classification framework for the internet backbone. *Computer Networks*, 55(8):1909 – 1920, 2011.

[58] Marios Iliofotou, Michalis Faloutsos, and Michael Mitzenmacher. Exploiting dynamicity in graph-based traffic analysis: techniques and applications. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 241–252, New York, NY, USA, 2009. ACM.

[59] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 315–320, New York, NY, USA, 2007. ACM.

[60] Paul Jaccard. *Distribution de la Flore Alpine: dans le Bassin des dranses et dans quelques régions voisines.* Rouge, 1901.

[61] Márk Jelasity and Vilmos Bilicki. Towards automated detection of peer-to-peer botnets: on the limits of local approaches. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'09, Berkeley, CA, USA, 2009. USENIX Association.

[62] Nan Jiang, Jin Cao, Yu Jin, Li Erran Li, and Zhi-Li Zhang. Identifying suspicious activities through dns failure graph analysis. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 144–153. IEEE, 2010.

[63] Raúl Jiménez, Flutra Osmani, and Björn Knutsson. Connectivity properties of mainline bittorrent dht nodes. In Henning Schulzrinne, Karl Aberer, and Anwitaman Datta, editors, *Peer-to-Peer Computing*, pages 262–270. IEEE, 2009.

[64] Yu Jin, Nick Duffield, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. Inferring applications at the network layer using collective traffic statistics. *SIGMETRICS Perform. Eval. Rev.*, 38(1):351–352, June 2010.

[65] Jan Jusko and Martin Rehak. Identifying peer-to-peer communities in the network by connection graph analysis. *International Journal of Network Management*, 24(4):235–252, 2014.

[66] Jan Jusko, Martin Rehak, Jan Stiborek, Jan Kohout, and Tomas Pevny. Using behavioral similarity for botnet command-and-control discovery. *IEEE Intelligent Systems*, 31(5):16–22, 2016.

[67] Alex T Kalinka. The probability of drawing intersections: extending the hypergeometric distribution. *arXiv preprint arXiv:1305.0717*, 2013.

[68] Brent Kang and Chris Nunnery. Decentralized peer-to-peer botnet architectures. In Zbigniew Ras and William Ribarsky, editors, *Advances in Information and Intelligent Systems*, volume 251 of *Studies in Computational Intelligence*, pages 251–264. Springer Berlin / Heidelberg, 2009.

[69] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM, 2004.

[70] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids.* North-Holland, 1987.

[71] Max Kerkers, José Jair Santanna, and Anna Sperotto. Characterisation of the kelihos. b botnet. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 79–91. Springer, 2014.

[72] Nizar Kheir, Gregory Blanc, Hervé Debar, Joaquin Garcia-Alfaro, and Dingqi Yang. Automated classification of c&c connections through malware url clustering. In *IFIP International Information Security Conference*, pages 252–266. Springer, 2015.

[73] Jonghyun Kim, Khushboo Shah, and Stephan Bohacek. Detecting p2p traffic from the p2p flow graph. In *IWCMC*, pages 1795–1800. IEEE, 2011.

[74] Tomáš Komárek, Martin Grill, and Tomáš Pevný. Passive nat detection using http access logs. In *Information Forensics and Security (WIFS), 2016 IEEE International Workshop on*, pages 1–6. IEEE, 2016.

[75] Yehuda Koren, Stephen C North, and Chris Volinsky. Measuring and extracting proximity in networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–255. ACM, 2006.

[76] Dimitris Koukis, Spyros Antonatos, and Kostas G Anagnostakis. On the privacy risks of publishing anonymized ip network traces. In *Communications and Multimedia Security*, volume 4237, pages 22–32. Springer, 2006.

[77] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 245–260. Springer, 2011.

[78] M. Kryczka, R. Cuevas, C. Guerrero, and A. Azcorra. Unrevealing the structure of live bittorrent swarms: Methodology and analysis. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 230–239, 31 2011-sept. 2 2011.

[79] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[80] Marc Kührer, Christian Rossow, and Thorsten Holz. Paint it black: Evaluating the effectiveness of malware blacklists. In *International Workshop on Recent Advances in Intrusion Detection*, pages 1–21. Springer, 2014.

[81] Y.K.R. Kwok. *Peer-To-Peer Computing: Applications, Architecture, Protocols, and Challenges*. Computational Science. Taylor & Francis, 2011.

[82] Laks VS Lakshmanan, Raymond T Ng, and Ganesh Ramesh. To do or not to do: the dilemma of disclosing anonymized data. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 61–72. ACM, 2005.

[83] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 203–216, New York, NY, USA, 2006. ACM.

[84] Chunxi Li and Changjia Chen. Topology analysis of gnutella by large scale mining. In *Communication Technology, 2006. ICCT '06. International Conference on*, pages 1–4, Nov. 2006.

[85] Lei Liu, Sabyasachi Saha, Ruben Torres, Jianpeng Xu, Pang-Ning Tan, Antonio Nucci, and Marco Mellia. Detecting malicious clients in isp networks using http connectivity graph and flow information. In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 150–157. IEEE, 2014.

[86] Xiangtao Liu, Yang Li, Zhezhong Li, and Xueqi Cheng. Social network analysis on kad and its application. In *Proceedings of the 13th Asia-Pacific web conference on Web technologies and applications*, APWeb'11, pages 327–332, Berlin, Heidelberg, 2011. Springer-Verlag.

[87] Xin Liu and Tsuyoshi Murata. Community detection in large-scale bipartite networks. *Information and Media Technologies*, 5(1):184–192, 2010.

[88] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.

[89] Pratyusa K Manadhata, Sandeep Yadav, Prasad Rao, and William Horne. Detecting malicious domains via graph inference. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2014.

[90] Christopher D. Manning, Prabhakar Raghavan, and Schütze. *An Introduction to Information Retrieval.* Number c. Cambridge University Press, Cambridge, England, 2009.

[91] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[92] Evangelos P. Markatos. Tracing a large-scale peer to peer system: An hour in the life of gnutella. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 65–75, Washington, DC, USA, 2002. IEEE Computer Society.

[93] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[94] Mary McGlohon, Stephen Bay, Markus G Anderle, David M Steier, and Christos Faloutsos. Snare: a link analytic system for graph labeling and risk detection. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1265–1274. ACM, 2009.

[95] Kevin McNamee. Malware analysis report - botnet: Zeroaccess/sirefef. `http://www.kindsight.net/sites/default/files/Kindsight_Malware_Analysis-ZeroAcess-Botnet-final.pdf`, February 2012.

[96] Guy Melançon and Arnaud Sallaberry. Edge metrics for visual graph analytics: A comparative study. In *Information Visualisation, 2008. IV'08. 12th International Conference*, pages 610–615. IEEE, 2008.

[97] Shicong Meng, Cong Shi, Dingyi Han, Xing Zhu, and Yong Yu. A statistical study of today's gnutella. In *Proceedings of the 8th Asia-Pacific Web conference on Frontiers of WWW Research and Development*, APWeb'06, pages 189–200, Berlin, Heidelberg, 2006. Springer-Verlag.

[98] Zoltán Móczár and Sándor Molnár. Characterization of bittorrent traffic in a broadband access network. In *AccessNets*, volume 63 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 176–183. Springer, 2010.

[99] Shishir Nagaraja. Botyacc: Unified p2p botnet detection using behavioural analysis and graph analysis. In *European Symposium on Research in Computer Security*, pages 439–456. Springer, 2014.

[100] Pratik Narang, Subhajit Ray, Chittaranjan Hota, and Venkat Venkatakrishnan. Peershark: detecting peer-to-peer botnets by tracking conversations. In *Security and Privacy Workshops (SPW), 2014 IEEE*, pages 108–115. IEEE, 2014.

[101] Kei Ohnishi, Satoshi Nagamatsu, Toshiya Okamura, and Yuji Oie. Autonomously reconstructable semi-structured p2p networks for file sharing. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, ICAS '07, pages 10–, Washington, DC, USA, 2007. IEEE Computer Society.

[102] Zhonghong Ou, Jiehan Zhou, Erkki Harjula, and Mika Ylianttila. Truncated pyramid peer-to-peer architecture with vertical tunneling model. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, pages 1227–1231, Piscataway, NJ, USA, 2009. IEEE Press.

[103] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, pages 201–210. ACM, 2007.

[104] Judea Pearl. *Reverend Bayes on inference engines: A distributed hierarchical approach.* Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[105] Roberto Perdisci, Davide Ariu, and Giorgio Giacinto. Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks*, 57(2):487–500, 2013.

[106] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, volume 10, page 14, 2010.

[107] Daniel Plohmann and Elmar Gerhards-Padilla. Case study of the miner botnet. In *Cyber Conflict (CYCON), 2012 4th International Conference on*, pages 1–16. IEEE, 2012.

[108] Jiayin Qi, Hongli Zhang, Zhenzhou Ji, and Liu Yun. Analyzing bittorrent traffic across large network. In *Cyberworlds, 2008 International Conference on*, pages 759–764, sept. 2008.

[109] Aviv Raf. Kelihos.b is still live and social. `https://www.seculert.com/blog/2012/03/kelihosb-is-still-live-and-social.html`, March 2012.

[110] M Zubair Rafique and Juan Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. In *International Workshop on Recent Advances in Intrusion Detection*, pages 144–163. Springer, 2013.

[111] S. Ragan. Millions of home networks infected by zeroaccess botnet. `http://www.securityweek.com/millions-home-networks-infected-zeroaccess-botnet`, 2012.

[112] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. Segugio: Efficient behavior-based tracking of malware-control domains in large isp networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 403–414. IEEE, 2015.

[113] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets.* Cambridge University Press, 2011.

[114] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[115] Amir H. Rasti, Daniel Stutzbach, and Reza Rejaie. On the long-term evolution of the two-tier gnutella overlay. In *INFOCOM.* IEEE, 2006.

[116] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31(4):161–172, August 2001.

[117] Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J Dietrich, and Herbert Bos. Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 97–111. IEEE, 2013.

[118] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.

[119] Ramin Sadre, Anna Sperotto, and Aiko Pras. The effects of ddos attacks on flow monitoring applications. In *NOMS*, pages 269–277. IEEE, 2012.

[120] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2(2):169–194, 1998.

[121] P.M. Santiago del Rio, J. Ramos, J.L. Garcia-Dorado, J. Aracil, A. Cuadra-Sanchez, and M. Cutanda-Rodriguez. On the processing time for detection of Skype traffic. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1784–1788, 2011.

[122] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94, 2007.

[123] Jan Seedorf, Sebastian Kiesel, and Martin Stiemerling. Traffic localization for p2p-applications: The alto approach. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 171–177. IEEE, 2009.

[124] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. Netw.*, 12(2):219–232, April 2004.

[125] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004.

[126] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

[127] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A global view of kad. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 117–122, New York, NY, USA, 2007. ACM.

[128] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Long term study of peer behavior in the kad dht. *IEEE/ACM Trans. Netw.*, 17(5):1371–1384, October 2009.

[129] Ben Stock, Jan Göbel, Markus Engelberth, Felix C Freiling, and Thorsten Holz. Walowdac-analysis of a peer-to-peer botnet. In *Computer Network Defense (EC2ND), 2009 European Conference on*, pages 13–20. IEEE, 2009.

[130] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[131] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. Impact of similarity measures on web-page clustering. In *Workshop on artificial intelligence for web search (AAAI 2000)*, volume 58, page 64, 2000.

[132] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. *IEEE/ACM Trans. Netw.*, 16(2):267–280, April 2008.

[133] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[134] Philipp Svoboda, E. Hyytia, F. Ricciato, M. Rupp, and M. Karner. Detection and tracking of Skype by exploiting cross layer information in a live 3G network. *Traffic Monitoring and Analysis*, pages 93–100, 2009.

[135] Cisco Systems. Cta cisco cognitive threat analytics on cisco cloud web security. `https://cognitive.cisco.com/`, 2014–2015.

[136] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1524–1533. ACM, 2014.

[137] Menno Tammens. Comparing peer selection mechanisms in p2p systems. 2011.

[138] B. Trammell and D. Schatzmann. A tale of two outages: A study of the skype network in distress. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1282 –1286, july 2011.

[139] Brian Trammell, Elisa Boschi, Gregorio Procissi, Christian Callegari, Peter Dorfinger, and Dominik Schatzmann. Identifying Skype Traffic in a Large-Scale Flow Data Repository. *Traffic Monitoring and Analysis*, pages 72–85, 2011.

[140] Andrei Venzhega, Polina Zhinalieva, and Nikolay Suboch. Graph-based malware distributors detection. In *Proceedings of the 22Nd International Conference on World Wide Web*, pages 1141–1144. ACM, 2013.

[141] Graeme Wearden. Us army aims to take p2p into battle. `http://www.zdnet.com/us-army-aims-to-take-p2p-into-battle-3002094181`. Accessed: 2013-10-20.

[142] T. Werner. The miner botnet: Bitcoin mining goes peer-to-peer. `http://www.securelist.com/en/blog/208193084/`, 2011.

[143] Chuan Wu and Baochun Li. Optimal peer selection for minimum-delay peer-to-peer streaming with rateless codes. In *Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*, pages 69–78. ACM, 2005.

[144] James Wyke. The zeroaccess botnet: Mining and fraud for massive financial gain. *Sophos Technical Paper*, 2012.

[145] Haiyong Xie, Y Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4p: Provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362, 2008.

[146] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 51(7):2282–2312, 2005.

[147] Tamara Yu, Richard Lippmann, James Riordan, and Stephen Boyer. Ember: a global perspective on extreme malicious behavior. In *Proceedings of the seventh international symposium on visualization for cyber security*, pages 1–12. ACM, 2010.

[148] Dongyan Zhang, Chao Zheng, Hongli Zhang, and Hongliang Yu. Identification and analysis of skype peer-to-peer traffic. In *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, pages 200 –206, may 2010.

[149] Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz, and Xiapu Luo. Detecting stealthy p2p botnets using statistical traffic fingerprints. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 121–132. IEEE, 2011.

[150] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

[151] David Zhao, Issa Traore, Ali Ghorbani, Bassam Sayed, Sherif Saad, and Wei Lu. Peer to peer botnet detection based on flow intervals. *Information Security and Privacy Research*, pages 87–102, 2012.

[152] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[153] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, volume 3, pages 912–919, 2003.

[154] Jianwei Zhuge, Thorsten Holz, Chengyu Song, Jinpeng Guo, Xinhui Han, and Wei Zou. Studying malicious websites and the underground economy on the chinese web. In *Managing Information Risk and the Economics of Security*, pages 225–244. Springer, 2009.