MASTERTHESIS

Valentin Brandl

Collaborative Monitoring of P2P Botnets

19th April 2022

Faculty:	Informatik und Mathematik
Study Programme:	Master Informatik
Supervisor:	Prof. Dr. Christoph Skornia
Secondary Supervisor:	Prof. Dr. Thomas Waas

Botnets pose a huge risk to general internet infrastructure and services. Distributed P2P topologies make it harder to detect and take those botnets offline. To take a Peer-to-Peer (P2P) botnet down, it has to be monitored to estimate the size and learn about the network topology. With the growing damage and monetary value produced by such botnets, ideas emerged on how to detect and prevent monitoring activity in the network. This work explores ways to make monitoring of fully distributed botnets more efficient, resilient, and harder to detect, by using a collaborative, coordinated approach. Further, we show how the coordinated approach helps in circumventing anti-monitoring techniques deployed by botnets.

do me

Keywords— P2P, botnet, monitoring, collaboration

Contents

Todo list

1 Introduction

The Internet has become an irreplaceable part of our day-to-day lives. We are always connected via numerous "smart" and Internet of Things (IoT) devices. We use the Internet to communicate, shop, handle financial transactions, and much more. Many personal and professional workflows are so dependent on the Internet, that they won't work when being offline, and with the pandemic, we are living through, this dependency grew even stronger.

In 2021, there were around 10 billion Internet-connected IoT devices and this number is estimated to more than double over the next years up to 25 billion in 2030 [21]. Many of these devices run on outdated software, don't receive regular updates, and don't follow general security best practices. While in 2016 only $77\,\%$ of German households had a broadband connection with a bandwidth of $50\,\mathrm{MBit/s}$ or more, in 2020 it was already $95\,\%$ with more than $50\,\mathrm{MBit/s}$ and $59\,\%$ with at least $1000\,\mathrm{MBit/s}$ [4]. Their nature as small, always online devices—often without any direct user interaction—behind Internet connections that are getting faster and faster makes them a desirable target for *botnet* operators.

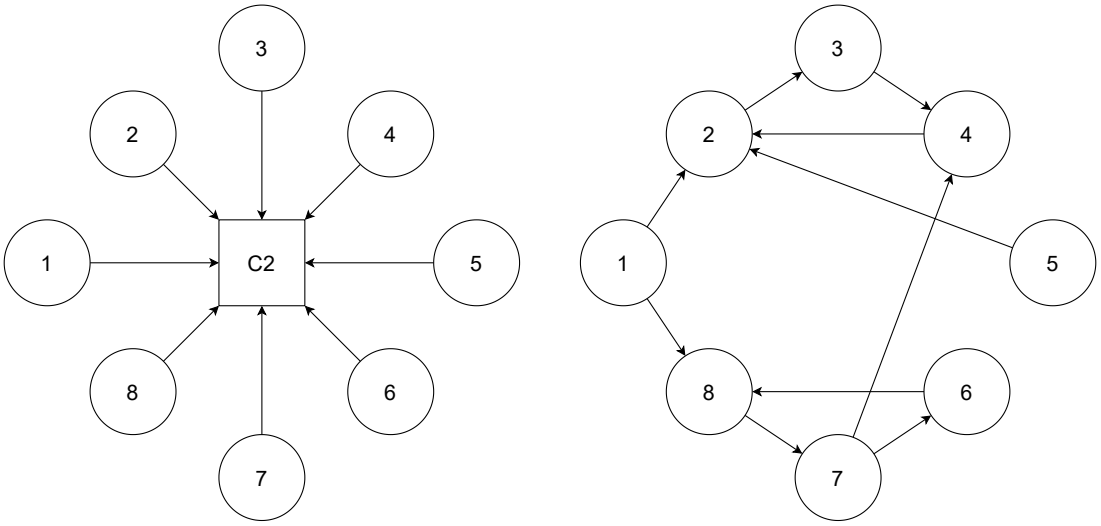
A *botnet* is a network of malware-infected computers, called *bots*, controlled by a *botmaster*. Botnets are controlled via a *Command and Control (C2) channel*. The communication patterns of a C2 channel can be *centralized*, *decentralized*, or *distributed*. Centralized or decentralized botnets use one or more coordinating hosts to contact and receive new commands. Distributed botnets create a *P2P* network as their communication layer. The C2 channel for centralized and decentralized botnets can use anything from Internet Relay Chat (IRC) over HTTP to Twitter [23].

In recent years, IoT botnets have been responsible for some of the biggest Distributed Denial of Service (DDoS) attacks ever recorded—creating up to $1\,\mathrm{TBit/s}$ of traffic [10]. Other malicious use of bots includes several activities—DDoS attacks, banking fraud, proxies to hide the attacker's identity, and sending of spam emails, just to name a few.

The constantly growing damage produced by botnets has many researchers and law enforcement agencies trying to shut down these operations [18, 17, 8, 7]. A coordinated

operation with help from law enforcement, hosting providers, domain registrars, and platform providers could shut down or take over the operation by changing how requests are routed or simply shutting down the controlling servers/accounts.

The monetary value of these botnets directly correlates with the amount of effort botmasters are willing to put into implementing defense mechanisms against take-down attempts. Botnet operators came up with a number of ideas: Domain Generation Algorithms use pseudorandomly generated domain names to render simple domain blacklist-based approaches ineffective [3] or fast-flux DNS entries, where a large pool of IP addresses is randomly assigned to the C2 domains to prevent IP based blacklisting and hide the actual C2 servers [20]. Analyzing and shutting down a centralized or decentralized botnet is comparatively easy since the central means of communication (the C2 IP addresses or domain names, Twitter handles or IRC channels), can be extracted from the malicious binaries or determined by analyzing network traffic and can therefore be considered publicly known. A number of botnet operations were taken down by shutting down the C2 channel [18, 12] and as the defenders upped their game, so did attackers—the concept of P2P botnets emerged. The idea is to build a distributed network without Single Points of Failure (SPoF) in the form of C2 servers as shown in Figure 1b.



(a) Topology of a C2 controlled botnet (b) Topology of a P2P botnet

Figure 1: Communication paths in different types of botnets

This lack of a SPoF makes P2P botnets more resilient to take-down attempts since there is no easy way to stop the communication and botmasters can easily rejoin the network and send new commands.

Taking down a P2P botnet requires intricate knowledge of the botnet's characteristics, e.g. size, risk, distribution over IP subnets or geolocations, network topology, participating peers, and protocol characteristics. Just like for centralized and decentralized botnets, to take down a P2P botnet, the C2 channel needs to be identified and disrupted. By *monitoring* peer activity of known participants in the botnet, this knowledge can be obtained and used to find attack vectors in the botnet protocol.

few words about monitoring

In this work, we will show how a collaborative system of crawlers and sensors can make the monitoring and information gathering phase of a P2P botnet more efficient, resilient to detection and how collaborative monitoring can help circumvent anti-monitoring techniques.

2 Background

In a P2P botnet, each node in the network knows a number of its neighbors and connects to those. Each of these neighbors has a list of neighbors on its own, and so on. The botmaster only needs to join the network to send new commands or receive stolen data but there is no need for a coordinating host, that is always connected to the network. Any of the nodes in Figure 1b could be the botmaster but they don't even have to be online all the time since the peers will stay connected autonomously. In fact, there have been arrests of P2P botnet operators but due to the autonomy offered by the distributed approach, the botnet keeps intact and continues operating [27]. Especially worm-like botnets, where each peer tries to actively find and infect other systems, can keep lingering for many years.

Bots in a P2P botnet can be split into two distinct groups according to their reachability: peers that are not publicly reachable (e.g. because they are behind a Network Access Translation (NAT) router or firewall) and those, that are publicly reachable, also known as *superpeers*. In contrast to centralized botnets with a fixed set of C2 servers, in a P2P botnet, every superpeer might take the role of a C2 server and *non-superpeers* will connect to those superpeers when joining the network.

As there is no well-known server in a P2P botnet, they have to coordinate autonomously. This is achieved by connecting the bots among each other. Bot B is considered a *neighbor* of bot A, if A knows and connects to B. Since bots can go offline can become unavailable (e.g. because the system was shut down or the malware infection was detected and removed), they have to consistently update their neighbor lists to avoid losing their connection into the botnet. This is achieved by periodically querying their neighbor's neighbors in a process known as *Membership Management (MM)*.

MM can be distinguished into two categories: *structured* and *unstructured* [5]. Structured P2P botnets have strict rules for a bot's neighbors based on its unique ID and often use a Distributed Hash Table (DHT), which allows persisting data in a distributed network. The DHT could contain a ordered ring structure of IDs and neighborhood in the structure also means neighborhood in the network, as is the case in the Kademila botnet [16]. In P2P botnets that employ unstructured MM on the other hand, bots ask any peer they know

for new peers to connect to, in a process called *peer discovery*. To enable peers to join a unstructured P2P botnets, the malware binaries include hardcoded lists of superpeers for the newly infected systems to connect to.

The concept of *churn* describes when a bot becomes unavailable. There are two types of churn:

- *IP churn*: A bot becomes unreachable because it got assigned a new IP address. The bot is still available but under another IP address.
- *Device churn*: The device is actually offline, e.g. because the infection was cleaned, it got shut down or lost its Internet connection.

2.1 Formal Model of P2P Botnets

A P2P botnet can be modelled as a digraph

$$G = (V, E)$$

With the set of nodes V describing the peers in the network and the set of edges E describing the communication flow between bots.

G is not required to be a connected graph but might consist of multiple disjoint components [24]. Components consisting of peers, that are infected by the same malware, are considered part of the same graph.

For a bot $v \in V$, the *predecessors* pred(v) and *successors* (neighbors) succ(v) are defined as:

$$\mathsf{succ}(v) = \{u \in V \mid (v, u) \in E\}$$
$$\mathsf{pred}(v) = \{u \in V \mid (u, v) \in E\}$$

The set of nodes succ(v) is also called the *peer list* of v. Those are the nodes, a peer will connect to, to request new commands and other peers.

For a node $v \in V$, the in and out degree $\deg^+$ and $\deg^-$ describe how many bots know v or are known by v respectively.

$$\deg^+(v) = |\mathsf{pred}(v)|$$
$$\deg^-(v) = |\mathsf{succ}(v)|$$

2.2 Monitoring Techniques for P2P Botnets

There are two distinct methods to map and get an overview of the network topology of a P2P botnet:

2.2.1 Passive Monitoring

For passive detection, traffic flows in large amounts of collected network traffic often obtained from Internet Service Providers or network telescopes [15] are analyzed. This has some advantages: e.g. it is not possible for botmasters to detect or prevent data-collection of that kind, though it is not trivial to distinguish valid P2P application traffic (e.g. BitTorrent, Skype, cryptocurrencies, ...) from P2P bots. Zhang et al. propose a system of statistical analysis to solve some of these problems in "Building a Scalable System for Stealthy P2P-Botnet Detection" [28]. Also getting access to the required datasets might not be possible for everyone.

Like most botnet detection mechanisms, also the passive ones work by building communication graphs and finding tightly coupled subgraphs that might be indicative of a botnet [19]. An advantage of passive detection is, that it is independent of protocol details, specific binaries, or the structure of the network (P2P vs. centralized/decentralized) [11].

Passive monitoring is only mentioned for completeness and not further discussed in this thesis.

2.2.2 Active Monitoring

For active detection, a subset of the botnet protocol and behavior is reimplemented to participate in the network. To do so, samples of the malware are reverse engineered to understand and recreate the protocol. This partial implementation includes the communication part of the botnet but ignores the malicious functionality to not support and take part in illicit activity.

There are two subtypes of active detection: *sensors* wait to be contacted by other peers, while *crawlers* actively query known bots and recursively ask for their neighbors [14]. Crawlers can only detect superpeers and therefore only see a small subset of the network, while sensors are also contacted by peers in private networks and behind firewalls. To accurately monitor a P2P botnet, a hybrid approach of crawlers and sensors is required.

A crawler starts with a predefined list of known bots, connects to those, and uses the peer exchange mechanism to request other bots. Each found bot is crawled again, slowly building the graph of superpeers on the way. Every entry E in the peer exchange response received from bot A represents an edge from A to E in the graph. "Reliable Recon in Adversarial Peer-to-Peer Botnets" describes disinformation attacks, in which bots will include invalid entries in their peer list replies [1]. Therefore, edges should only be considered valid, if at least one crawler or sensor was able to contact or contacted by peer E, thereby confirming, that E is an existing participant in the botnet.

A sensor implements the passive part of the botnet's MM. It is populated into the network by crawlers or other sensors and waits for other peers to contact them. They cannot be

used to create the botnet graph (only edges into the sensor node) or find new peers, but are required to enumerate the whole network, including non-superpeers.

2.2.3 Anti-Monitoring Techniques

Andriesse, Rossow, and Bos explore some monitoring countermeasures in "Reliable Recon in Adversarial Peer-to-Peer Botnets". These include deterrence, which limits the number of bots per IP address or subnet; blacklisting, where known crawlers and sensors are blocked from communicating with other bots in the network (mostly IP based); disinformation, when fake bots are placed in the peer lists, to invalidate the data collected by crawlers; and active retaliation like DDoS attacks against sensors or crawlers [1].

2.3 Related Work

In "BoobyTrap" [13], the authors evaluate criteria to detect monitoring attempts in a P2P botnet:

Defiance Peers that don't abide by the MM protocol rules are most likely crawlers or sensors, e.g. peers that query other peers that shouldn't be in their neighborhood according to geolocation or IP subnet rules.

Abuse Higher MM frequency as an indicator for a sensor or crawler

Avoidance Peers that avoid aiding the botnet, e.g. by returning empty replies on MM requests are potential monitoring nodes

"SensorBuster" explores graph ranking algorithms to detect monitoring activity in a P2P botnet. They depend on suspicious graph properties to enumerate candidate peers [14].

PageRank The algorithm used by Google to rank their search results uses the ratio of $\deg^+$ and $\deg^-$ to detect sensors since they have many incoming but few outgoing edges [22]

SensorRank A deviation of PageRank that normalizes the result, to better account for churn and valid peers with few high-ranked predecessors but only a few successors

SensorBuster evaluates Weakly Connected Component (WCC) in a graph since sensors have only incoming but no outgoing edges, thereby creating a disconnected graph component

Botnet Monitoring System (BMS)[1]is a monitoring platform for P2P botnets described by Böck et al. in "Challenges of Accurately Measuring Churn in P2P Botnets". BMS is intended for a hybrid active approach of crawlers and sensors (reimplementations of the P2P protocol of a botnet, that won't perform malicious actions) to collect live data from active botnets.

In an earlier project, we implemented different graph ranking algorithms—among others *PageRank* [22] and *SensorRank*—to detect sensor candidates in a botnet, as described in "SensorBuster". In an earlier project, we implemented the ranking algorithms described in "SensorBuster" for BMS.

[1] `https://github.com/Telecooperation/BMS`

3 Methodology

The implementation of the concepts of this work will be done as part of BMS. The goal of this work is to complicate detection and anti-monitoring mechanisms for botmasters by coordinating the work of the system's crawlers and sensors. The coordinated work distribution also helps in efficiently monitoring large botnets where one crawler is not enough to track all peers. The changes should allow the current BMS crawlers and sensors to use the new implementation with as few changes as possible to the existing code.

We will explore how cooperative monitoring of a P2P botnet can help with the following problems:

- Impede detection of monitoring attempts by reducing the impact of aforementioned graph metrics
- Circumvent anti-monitoring techniques
- Make crawling more efficient

The final results should be as general as possible and not depend on any botnet's specific behavior (except for the mentioned anti-monitoring techniques which might be unique to some botnets), but we assume that every P2P botnet has some way of querying a bot's neighbors for the concept of crawlers and sensors to be applicable.

In the current implementation, each crawler will itself visit and monitor each new node it finds. The general idea for the implementation of the concepts in this thesis is to report newfound nodes back to the BMS backend first, where the graph of the known network is created, and a fitting worker is selected to achieve the goal of the according coordination strategy. That worker will be responsible to monitor the new node.

If it is not possible to select a sensor so that the monitoring activity stays inconspicuous, the coordinator can do a complete shuffle of all nodes between the sensors to restore the wanted graph properties or warn if more sensors are required to fulfill the goal defined by the strategy.

The improved crawler system should allow new crawlers to register themselves and their capabilities (e.g. bandwidth, geolocation), so the amount of work can be scaled accordingly between hosts.

3.1 Protocol Primitives

The coordination protocol must allow the following operations:

Register Worker Register a new worker with capabilities (which botnet, the available bandwidth and processing power, if it is a crawler or sensor, ...). This is called periodically and used to determine which worker is still active when assigning new tasks.

Report Peer Report found peers. Both successful and failed attempts are reported, to detect churned peers, and blacklisted crawlers as soon as possible.

Report Edge Report found edges. Edges are created by querying the peer list of a bot. This is how new peers are detected.

Request Tasks Receive a batch of crawl tasks from the coordinator. The tasks consist of the target peer, if the worker should start or stop monitoring the peer, when the monitoring should start and stop and at which frequency the peer should be contacted.

Request Neighbors Sensors can request a list of candidate peers to return when their peer list is queried.

```
type Peer struct {
  BotID string
  IP    string
  Port  uint16
}
type PeerTask struct {
  Peer          Peer
  StartAt       *Time
  StopAt        *Time
  Frequency     uint
  StopCrawling bool
}
```

Listing 1: Relevant Fields for Peers and Tasks

Listing 1 shows the Go structures used for crawl tasks.

4 Coordination Strategies

Let C be the set of available crawlers. Without loss of generality, if not stated otherwise, we assume that C is known when BMS is started and will not change afterward. There will be no joining or leaving crawlers. This assumption greatly simplifies the implementation due to the lack of changing state that has to be tracked while still exploring the described strategies. A production-ready implementation of the described techniques can drop this assumption but might have to recalculate the work distribution once a crawler joins or leaves. The protocol primitives described in Section 3.1 already allow for this to be implemented by first creating tasks with the `StopCrawling` flag set to true for all active tasks, running the strategy again, and creating the according tasks to start crawling again.

4.1 Load Balancing

Depending on a botnet's size, a single crawler is not enough to monitor all superpeers. While it is possible to run multiple, uncoordinated crawlers, two or more of them can find and monitor the same peer, making the approach inefficient with regard to the computing resources at hand.

The load balancing strategy solves this problem by systematically splitting the crawl tasks into chunks and distributing them among the available crawlers. The following load balancing strategies will be investigated:

Round Robin Evenly distribute the peers between crawlers in the order they are found

IP-based partitioning Use the uniform distribution of cryptographic hash functions to assign peers to crawlers in a random manner but still evenly distributed

Load balancing in itself does not help prevent the detection of crawlers but it allows better usage of available resources. It prevents unintentionally crawling the same peer with multiple crawlers and allows crawling of bigger botnets where the uncoordinated approach would reach its limit and could only be worked around by scaling up the machine where the crawler is executed. Load balancing allows scaling out, which can be more cost-effective.

4.1.1 Round Robin Distribution

This strategy distributes work evenly among crawlers by either naively assigning tasks to the crawlers rotationally or weighted according to their capabilities. To keep the distribution as even as possible, we keep track of the last crawler a task was assigned to and start with the next in line in the subsequent round of assignments. For the sake of simplicity only the bandwidth will be considered as a capability but it can be extended by any shared property between the crawlers, e.g. available memory or processing power. For a given crawler $c_i \in C$ let $cap(c_i)$ be the capability of the crawler. The total available capability is $B = \sum_{c \in C} cap(c)$. With G being the greatest common divisor of all the crawler's capabilities, the weight $W(c_i) = \frac{cap(c_i)}{G}$. $\frac{cap(c_i)}{B}$ gives us the percentage of the work a crawler is assigned. The algorithm in Listing 2 distributes the work according to the crawler's capabilities.

1 – 2 sentences about naive rr?

```
func WeightCrawlers(crawlers ...Crawler) map[string]uint {
  weights := []int{}
  totalWeight := 0
  for _, crawler := range crawlers {
    totalWeight += crawler.Bandwith
    weights = append(weights, crawler.Bandwith)
  }
  gcd := Fold(Gcd, weights...)
  weightMap := map[string]uint{}
  for _, crawler := range crawlers {
    weightMap[crawler.ID] = uint(crawler.Bandwith / gcd)
  }
  return weightMap
}

func WeightedCrawlerList(crawlers ...Crawler) []string {
  weightMap := WeightCrawlers(crawlers...)
  didSomething := true
  crawlerIds := []string{}
  for didSomething {
    didSomething = false
    for k, v := range weightMap {
      if v != 0 {
        didSomething = true
        crawlerIds = append(crawlerIds, k)
        weightMap[k] -= 1
      }
    }
  }
  return crawlerIds
}
```

Listing 2: Pseudocode for weighted round-robin

This creates a list of crawlers where a crawler can occur more than once, depending on its capabilities. To ensure better distribution, first, every crawler is assigned one task,

then, according to the capabilities, every crawler with a weight of 2 or more is assigned a task, repeating this process until all tasks are assigned. The set of crawlers $\{a, b, c\}$ with the capabilities $cap(a) = 3$, $cap(b) = 2$, $cap(c) = 1$ would produce $< a, b, c, a, b, a >$, allocating two and three times the work to crawlers b and a respectively.

4.1.2 IP-based Partitioning

The output of cryptographic hash functions is uniformly distributed. Given the hash function H, calculating the hash of an IP address and distributing the work with regard to $H(\text{IP}) \mod |C|$ creates almost evenly sized buckets for each worker to handle. This gives us the mapping $m(i) = H(i) \mod |C|$ to sort peers into buckets.

Any hash function can be used but since it must be calculated often, a fast function should be used. While the Message-Digest Algorithm 5 (MD5) hash function must be considered broken for cryptographic use [25], it is faster to calculate than hash functions with longer output. For the use case at hand, only the uniform distribution property is required so MD5 can be used without scarifying any kind of security.

md5 crypto broken, distribution not?

This strategy can also be weighted using the crawlers' capabilities by modifying the list of available workers so that a worker can appear multiple times according to its weight. The weighting algorithm from Listing 2 is used to create the weighted multiset of crawlers C_W and the mapping changes to $m(i) = H(i) \mod |C_W|$.

MD5 returns a 128 Bit hash value. The Go standard library includes helpers for arbitrarily sized integers[2]. This helps us in implementing the mapping m from above.

By exploiting the even distribution offered by hashing, the work of each crawler is also about evenly distributed over all IP subnets, Autonomous Systems (ASs), and geolocations. This ensures neighboring peers (e.g. in the same AS, geolocation, or IP subnet) get visited by different crawlers. It also allows us to get rid of the state in our strategy since we don't have to keep track of the last crawler we assigned a task to, making it easier to implement and reason about.

[2] `https://pkg.go.dev/math/big#Int`

4.2 Reduction of Request Frequency

The GameOver Zeus botnet limited the number of requests a peer was allowed to perform and blacklisted peers, that exceeded the limit, as an anti-monitoring mechanism [2]. In an uncoordinated crawler approach, the crawl frequency has to be limited to prevent hitting the request limit.

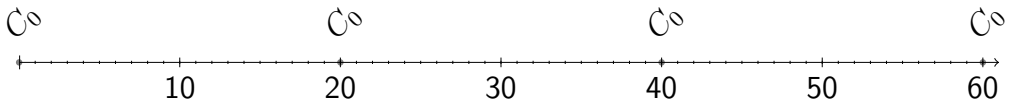


Figure 2: Timeline of requests as received by a peer when crawled by a single crawler

Using collaborative crawlers, an arbitrarily fast frequency can be achieved without being blacklisted. With $l \in \mathbb{N}$ being the maximum allowed frequency as defined by the botnet's protocol, $f \in \mathbb{N}$ being the crawl frequency that should be achieved. The number of crawlers n required to achieve the frequency f without being blacklisted and the offset o between crawlers are defined as

$$n = \left\lceil \frac{f}{l} \right\rceil$$
$$o = \frac{1\,\text{req}}{f}$$

Taking advantage of the `StartAt` field from the `PeerTask` returned by the `requestTasks` primitive above, the crawlers can be scheduled offset by o at a frequency l to ensure, the overall requests to each peer are evenly distributed over time.

Given a limit $l = 6\,\text{req}/\text{min}$, crawling a botnet at $f = 24\,\text{req}/\text{min}$ requires $n = \left\lceil \frac{24\,\text{req}/\text{min}}{6\,\text{req}/\text{min}} \right\rceil = 4$ crawlers. Those crawlers must be scheduled $o = \frac{1\,\text{req}}{24\,\text{req}/\text{min}} = 2.5\,\text{s}$ apart at a frequency of l to evenly distribute the requests over time.

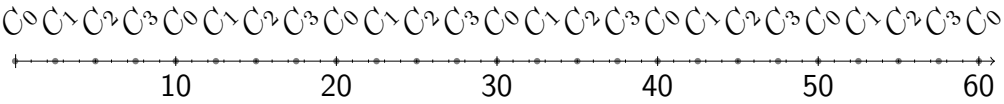


Figure 3: Timeline of crawler events when optimized for effective frequency

As can be seen in Figure 3, each crawler C_0 to C_3 performs only $6\,\text{req}/\text{min}$ while overall achieving $24\,\text{req}/\text{min}$.

Vice versa given an amount of crawlers n and a request limit l, the effective frequency f can be maximized to $f = n \times l$ without hitting the limit l and being blocked.

Using the example from above with $l = 6\,\text{req}/\text{min}$ but now only two crawlers $n = 2$, it is still possible to achieve an effective frequency of $f = 2 \times 6\,\text{req}/\text{min} = 12\,\text{req}/\text{min}$ with $o = \frac{1\,\text{req}}{12\,\text{req}/\text{min}} = 5\,\text{s}$:

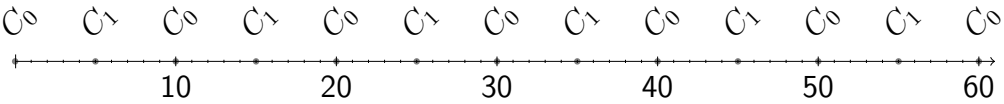


Figure 4: Timeline of crawler events when optimized over the number of crawlers

While the effective frequency of the whole system is halved compared to Figure 3, it is still possible to double the effective frequency over the limit.

4.3 Creating Edges for Crawlers and Sensors

"SensorBuster: On Identifying Sensor Nodes in P2P Botnets" describes different graph metrics to find sensors in P2P botnets. These metrics depend on the uneven ratio between incoming and outgoing edges for crawlers. The *SensorBuster* metric uses WCCs since naive sensors don't have any edges back to the main network in the graph.

Building a complete graph $G_C = K_{|C|}$ between the sensors and crawlers by making them return the other known worker on peer list requests would still produce a disconnected component and while being bigger and maybe not as obvious at first glance, it is still

easily detectable since there is no path from G_C back to the main network (see Figure 13b and Table 3).

With $v \in V$, succ(v) being the set of successors of v and pred(v) being the set of predecessors of v, *PageRank* is recursively defined as [22]:

$$\begin{aligned} \mathsf{PR}_0(v) &= \mathsf{initialRank} \\ \mathsf{PR}_{n+1}(v) &= \mathsf{dampingFactor} \times \sum_{p \in \mathsf{pred}(v)} \frac{\mathsf{PR}_n(p)}{|\mathsf{succ}(p)|} + \frac{1 - \mathsf{dampingFactor}}{|V|} \end{aligned}$$

For the first iteration, the PageRank of all nodes is set to the same initial value. Page et al. argue that when iterating often enough, any value can be chosen [22].

The dampingFactor describes the probability of a person visiting links on the web to continue doing so, when using PageRank to rank websites in search results. For simplicity—and since it is not required to model human behavior for automated crawling and ranking—a dampingFactor of 1.0 will be used, which simplifies the formula to

$$\mathsf{PR}_{n+1}(v) = \sum_{p \in \mathsf{pred}(v)} \frac{\mathsf{PR}_n(p)}{|\mathsf{succ}(p)|}$$

Based on this, *SensorRank* is defined as

$$\mathsf{SR}(v) = \frac{\mathsf{PR}(v)}{|\mathsf{succ}(v)|} \times \frac{|\mathsf{pred}(v)|}{|V|}$$

Since crawlers never respond to peer list requests, they will always be detectable by the described approach but sensors might benefit from the following technique.

By responding to peer list requests with plausible data and thereby producing valid outgoing edges from the sensors, we will try to make those metrics less suspicious. The challenge

here is deciding which peers can be returned without actually supporting the network. The following candidates to place on the neighbor list will be investigated:

- Return the other known sensors, effectively building a complete graph $K_{|C|}$ containing all sensors
- Detect churned peers from AS with dynamic IP allocation
- Detect peers behind carrier-grade NAT that rotate IP addresses very often and pick random IP addresses from the IP range

4.3.1 Other Sensors or Crawlers

Returning all the other sensors when responding to peer list requests, thereby effectively creating a complete graph $K_{|C|}$ among the workers, creates valid outgoing edges. The resulting graph will still form a WCC with now edges back into the main network.

PageRank is the sum of a node's predecessors ranks divided by the amount of successors each predecessor's successors. Predecessors with many successors should therefore reduce the rank. By their nature, crawlers have many successors, so they are good candidates to reduce the PageRank of a sensor.

crawlers as predecessors

4.3.2 Churned Peers After IP Rotation

Churn describes the dynamics of peer participation in P2P systems, e.g. join and leave events [26]. Detecting if a peer just left the system, in combination with knowledge about ASs, peers that just left and came from an AS with dynamic IP allocation (e.g. many consumer broadband providers in the US and Europe), can be placed into the crawler's peer list. If the timing of the churn event correlates with IP rotation in the AS, it can be assumed, that the peer left due to being assigned a new IP address—not due to connectivity issues or going offline—and will not return using the same IP address. These peers, when placed in the peer list of the crawlers, will introduce paths back into the main network and defeat the WCC metric. It also helps with the PageRank and SensorRank metrics since the

übergang

what is an AS

crawlers start to look like regular peers without actually supporting the network by relaying messages or propagating active peers.

4.3.3 Peers Behind Carrier-Grade NAT

Some peers show behavior, where their IP address changes almost after every request. Those peers can be used as fake neighbors and create valid-looking outgoing edges for the sensor.

5 Evaluation

To evaluate the strategies from above, we took a snapshot of the Sality [9] botnet obtained from BMS throughout of 21st to 28th April 2021, if not stated otherwise.

5.1 Load Balancing

To evaluate the real-world applicability of IP based partitioning, we will partition the dataset containing 1595 distinct IP addresses among 2, 4, 6, and 10 crawlers and verify if the work is about evenly distributed between crawlers.

We will compare the variance σ^2 and standard derivation σ to evaluate the applicability of this method.

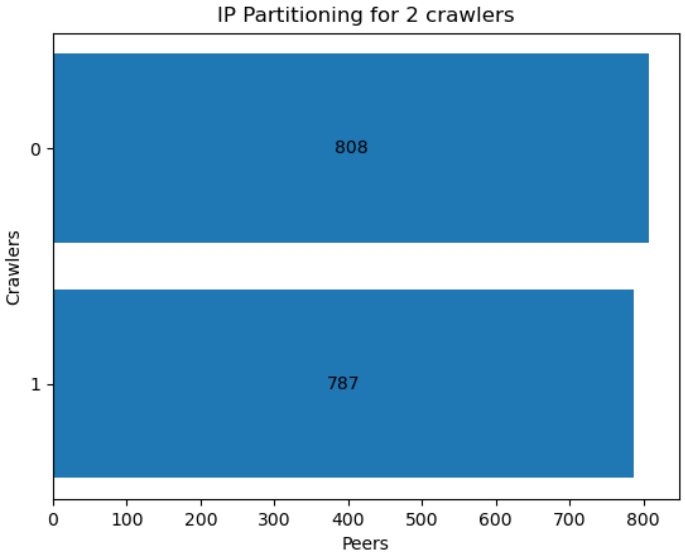


Figure 5: IP based partitioning for 2 crawlers

$$
\begin{aligned}
n &= 2 \\
\mu &= \frac{1595}{n} = 797.5 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (808 - 797.5)^2 + (787 - 797.5)^2 \\
&= 220.5 \\
\sigma^2 &= \frac{s}{n} = 110.2 \\
\sigma &= \sqrt{\sigma^2} = 10.5
\end{aligned}
$$

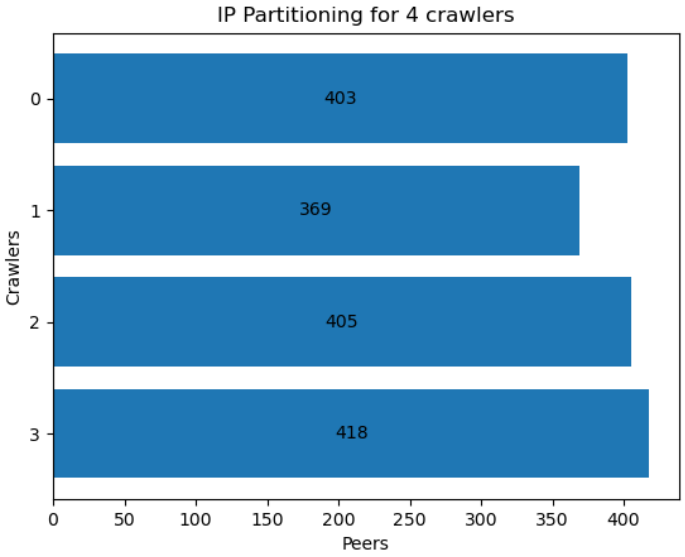


Figure 6: IP based partitioning for 4 crawlers

$$
\begin{aligned}
n &= 4 \\
\mu &= \frac{1595}{n} = 398.8 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (403 - 398.8)^2 + (369 - 398.8)^2 + (405 - 398.8)^2 \\
&\quad + (418 - 398.8)^2 \\
&= 1312.8 \\
\sigma^2 &= \frac{s}{n} = 328.2 \\
\sigma &= \sqrt{\sigma^2} = 18.1
\end{aligned}
$$

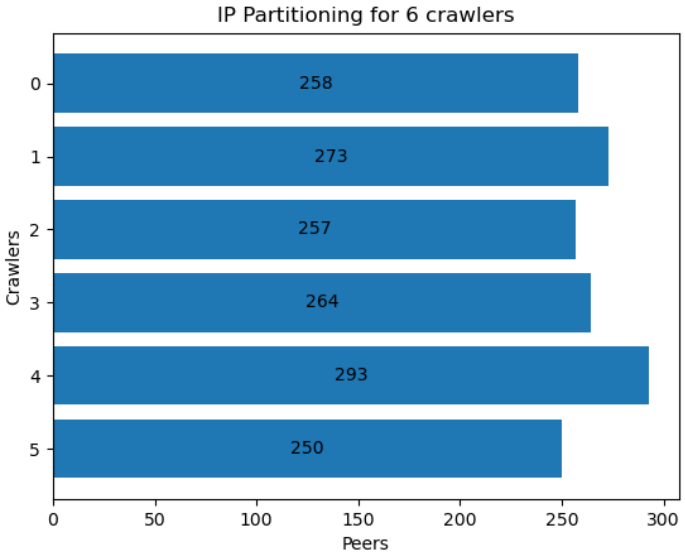


Figure 7: IP based partitioning for 6 crawlers

$$
\begin{aligned}
n &= 6 \\
\mu &= \frac{1595}{n} = 265.8 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (258 - 265.8)^2 + (273 - 265.8)^2 + (257 - 265.8)^2 \\
&\quad + (264 - 265.8)^2 + (293 - 265.8)^2 + (250 - 265.8)^2 \\
&= 1182.8 \\
\sigma^2 &= \frac{s}{n} = 197.1 \\
\sigma &= \sqrt{\sigma^2} = 14.0
\end{aligned}
$$

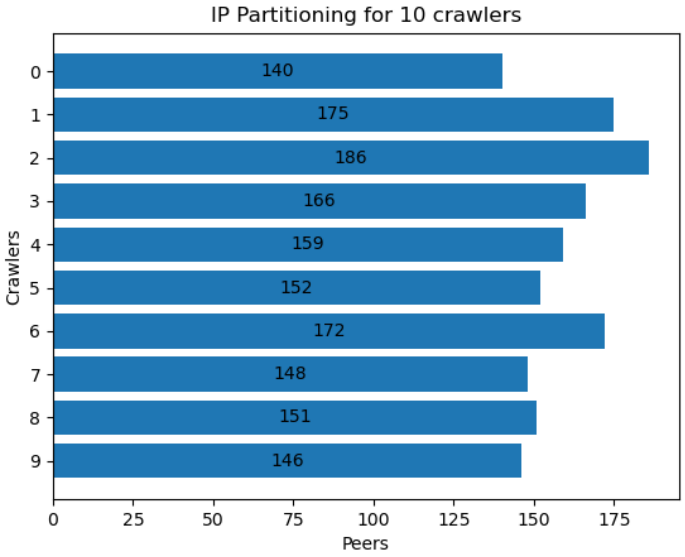


Figure 8: IP based partitioning for 10 crawlers

$$
\begin{aligned}
n &= 10 \\
\mu &= \frac{1595}{n} = 159.5 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (140 - 159.5)^2 + (175 - 159.5)^2 + (186 - 159.5)^2 \\
&\quad + (166 - 159.5)^2 + (159 - 159.5)^2 + (152 - 159.5)^2 \\
&\quad + (172 - 159.5)^2 + (148 - 159.5)^2 + (151 - 159.5)^2 \\
&\quad + (146 - 159.5)^2 \\
&= 1964.5 \\
\sigma^2 &= \frac{s}{n} = 196.4 \\
\sigma &= \sqrt{\sigma^2} = 14.0
\end{aligned}
$$

n	μ	σ^2	σ	$\frac{\sigma}{\mu}$
2	797.5	110.2	10.5	1.3%
4	398.8	328.2	18.1	4.5%
6	265.8	197.1	14.0	5.3%
10	159.5	196.4	14.0	8.8%

Table 1: Variance and standard derivation for IP-based partitioning on 1595 IP addresses

Table 1 shows that the derivation from the expected even distribution is within $10\,\%$. Since the used sample is not very big, according to the law of big numbers we would expect the derivation to get smaller, the bigger the sample gets. Therefore, we simulate the partitioning on a bigger sample of $1{,}000{,}000$ random IP addresses.

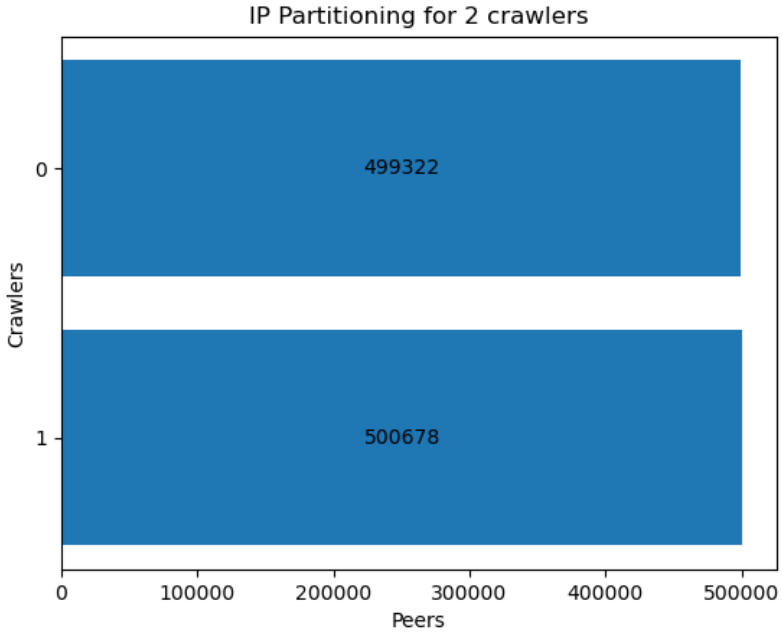


Figure 9: IP based partitioning for 2 crawlers on generated dataset

$$
\begin{aligned}
n &= 2 \\
\mu &= \frac{1000000}{n} = 500000 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (499322 - 500000)^2 + (500678 - 500000)^2 \\
&= 919368 \\
\sigma^2 &= \frac{s}{n} = 459684 \\
\sigma &= \sqrt{\sigma^2} = 678
\end{aligned}
$$

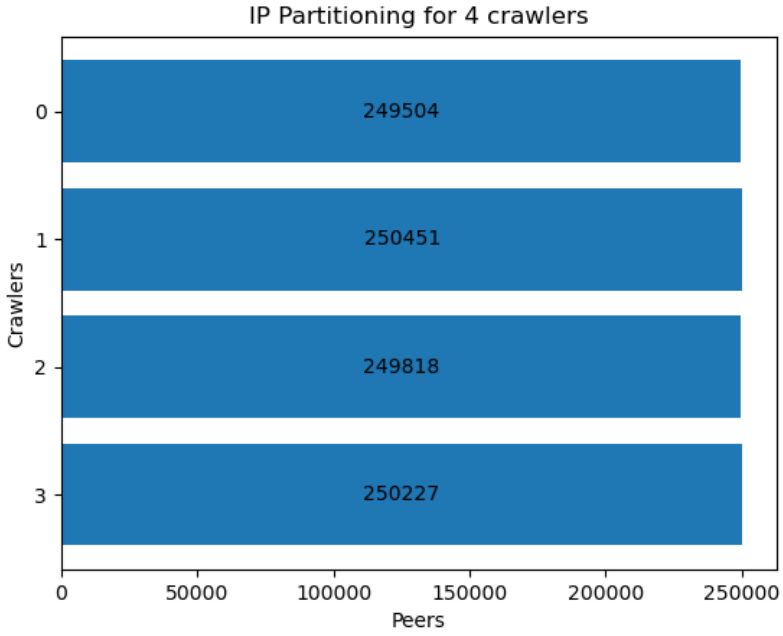


Figure 10: IP based partitioning for 4 crawlers on generated dataset

$$
\begin{aligned}
n &= 4 \\
\mu &= \frac{1000000}{n} = 250000 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (249504 - 250000)^2 + (250451 - 250000)^2 + (249818 - 250000)^2 \\
&\quad + (250227 - 250000)^2 \\
&= 534070 \\
\sigma^2 &= \frac{s}{n} = 133517.5 \\
\sigma &= \sqrt{\sigma^2} = 365.4
\end{aligned}
$$

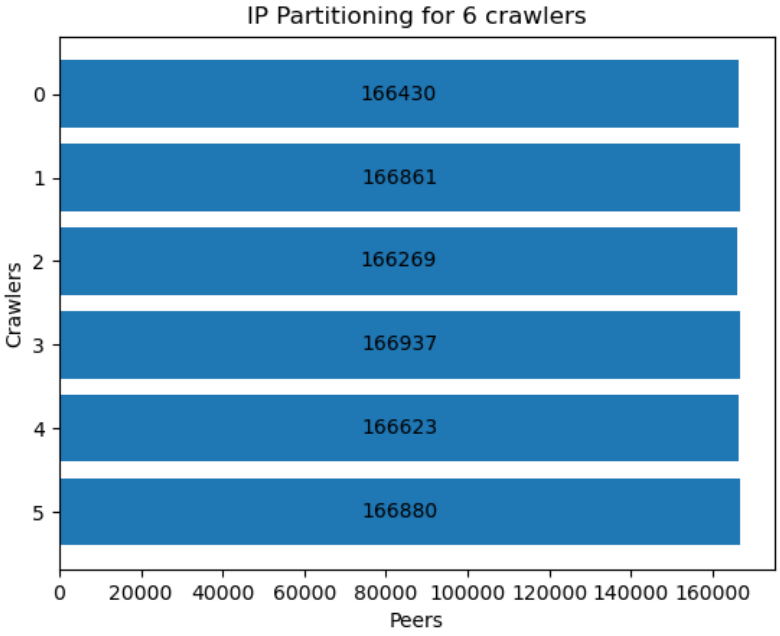


Figure 11: IP based partitioning for 6 crawlers on generated dataset

$$
\begin{aligned}
n &= 6 \\
\mu &= \frac{1000000}{n} = 166666.7 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (166430 - 166666.7)^2 + (166861 - 166666.7)^2 + (166269 - 166666.7)^2 \\
&\quad + (166937 - 166666.7)^2 + (166623 - 166666.7)^2 + (166880 - 166666.7)^2 \\
&= 372413.3 \\
\sigma^2 &= \frac{s}{n} = 62068.9 \\
\sigma &= \sqrt{\sigma^2} = 249.1
\end{aligned}
$$

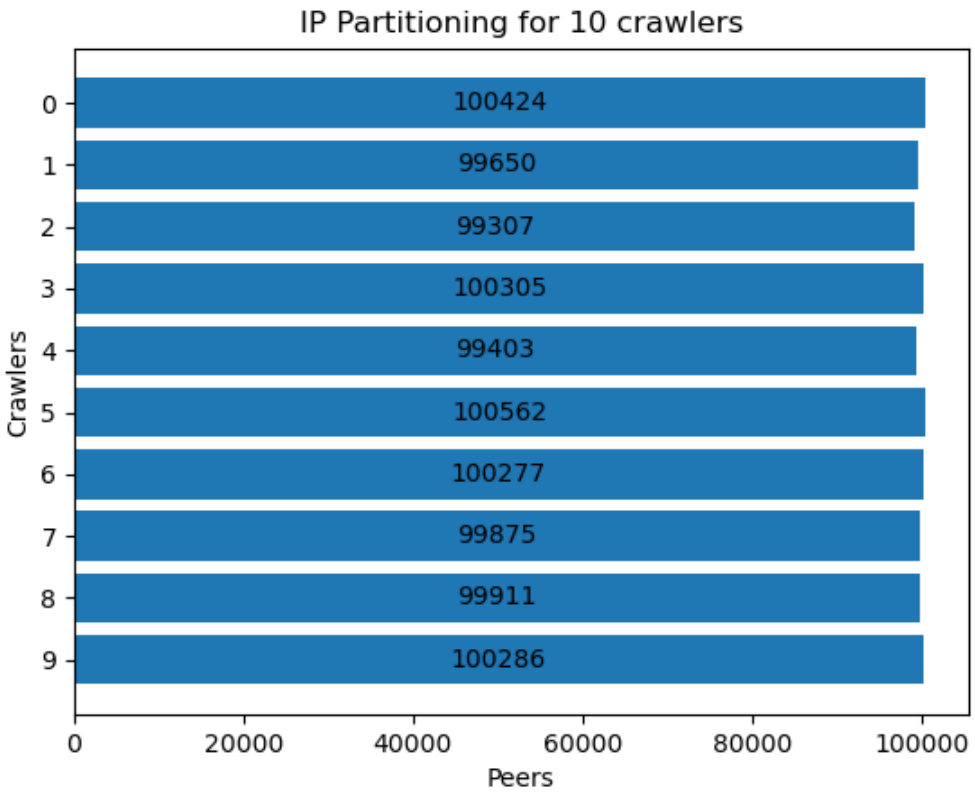


Figure 12: IP based partitioning for 10 crawlers on generated dataset

$$
\begin{aligned}
n &= 10 \\
\mu &= \frac{1000000}{n} = 100000 \\
s &= \sum_{i=1}^{n} (x_i - \mu)^2 \\
&= (100424 - 100000)^2 + (99650 - 100000)^2 + (99307 - 100000)^2 \\
&\quad + (100305 - 100000)^2 + (99403 - 100000)^2 + (100562 - 100000)^2 \\
&\quad + (100277 - 100000)^2 + (99875 - 100000)^2 + (99911 - 100000)^2 \\
&\quad + (100286 - 100000)^2 \\
&= 1729874 \\
\sigma^2 &= \frac{s}{n} = 172987.4 \\
\sigma &= \sqrt{\sigma^2} = 415.9
\end{aligned}
$$

n	μ	σ^2	σ	$\frac{\sigma}{\mu}$
2	500,000.0	459,684.0	678.0	0.14%
4	250,000.0	133,517.5	365.4	0.15%
6	166,666.7	62,069.9	249.1	0.15%
10	100,000.0	172,987.4	415.9	0.42%

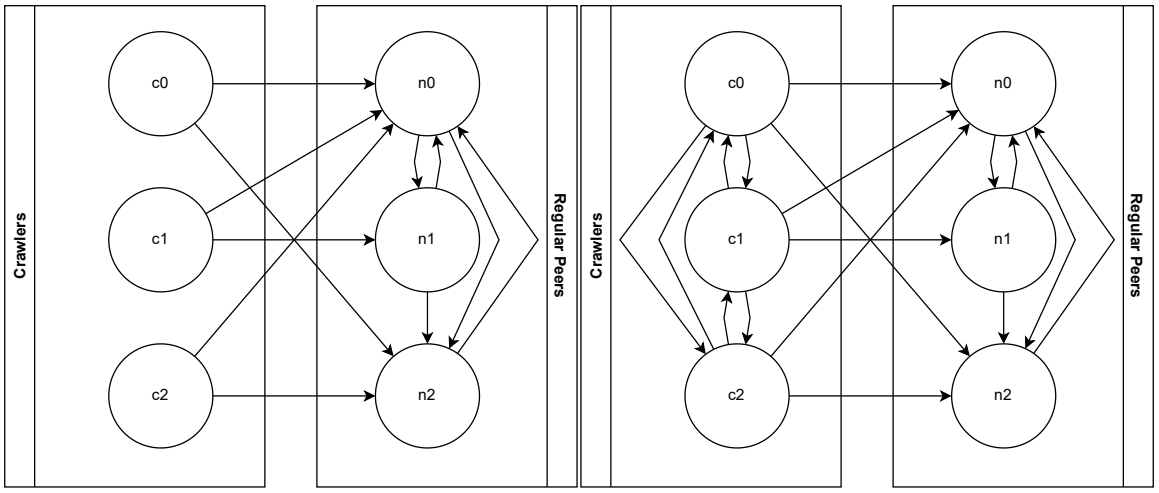
Table 2: Variance and standard derivation for IP-based partitioning on $1{,}000{,}000$ IP addresses

As expected, the work is still not perfectly distributed among the crawlers but evenly enough for our use case. The derivation for larger botnets is within $0.5\,\%$ of the even distribution. This is good enough for balancing the work among workers.

5.2 Impact of Additional Edges on Graph Metrics

5.2.1 Use Other Known Sensors

By connecting the known sensors and effectively building a complete graph $K_{|C|}$ between them creates $|C| - 1$ outgoing edges per sensor. In most cases this won't be enough to reach the amount of edges that would be needed. Also this does not help against the WCC metric since this would create a bigger but still disconnected component.



(a) WCCs for independent crawlers (b) WCCs for collaborated crawlers

Figure 13: Differences in graph metrics

Applying PageRank with an initial rank of 0.25 once on the example graphs in Figure 13 results in:

Node	$\deg^+$	$\deg^-$	In WCC?	PageRank	SensorRank
n0	0/0	4/4	no	0.75/0.5625	0.3125/0.2344
n1	1/1	3/3	no	0.25/0.1875	0.0417/0.0313
n2	2/2	2/2	no	0.5/0.375	0.3333/0.25
c0	3/5	0/2	yes (1/3)	0.0/0.125	0.0/0.0104
c1	1/3	0/2	yes (1/3)	0.0/0.125	0.0/0.0104
c2	2/4	0/2	yes (1/3)	0.0/0.125	0.0/0.0104

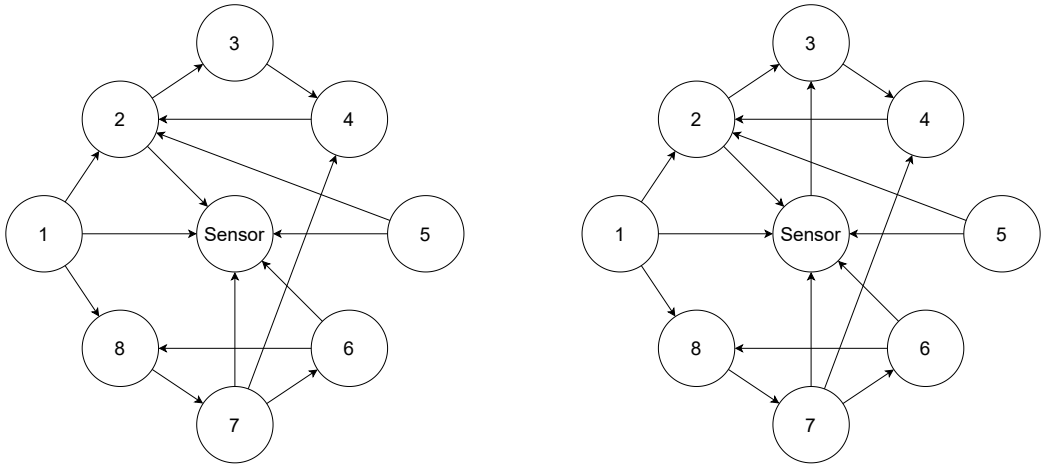
Table 3: Values for metrics from Figure 13 (a/b)

While this works for small networks, the crawlers must account for a significant amount of peers in the network for this change to be noticeable. The generated K_n needs to be at least as big as the smallest regular component in the botnet, which is not feasible. Also, if detected, this would leak the information about all known sensors to the botmasters. The limited scalability, and potential information leak, which might be used by botmasters

to retaliate against the sensors or the whole monitoring operation, make this approach unusable in real-world scenarios.

5.2.2 Effectiveness against SensorBuster

SensorBuster relies on the assumption that sensors don't have any outgoing edges, thereby creating a disconnected graph component.



(a) Sensor without outgoing edge creates disconnected graph component

(b) Single outgoing edge connects sensor back to the main component

Figure 14b shows how a single valid edge back into the network (from *Sensor* to peer 3 in the example) renders the SensorBuster metric ineffective by making the sensor part of the main graph component.

For the WCC metric, it is obvious that even a single edge back into the main network is enough to connect the sensor back to the main graph and therefore beat this metric. formulieren

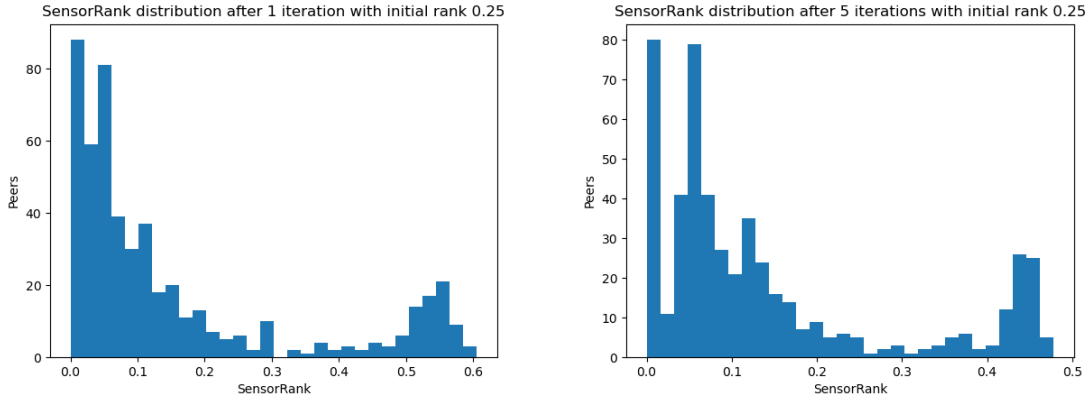
5.2.3 Effectiveness against Page- and SensorRank

In this section we will evaluate how adding outgoing edges to a sensor impacts it's PageRank and SensorRank values. Before doing so, we will check the impact of the initial rank

by calculating it with different initial values and comparing the value distribution of the result.

Iteration	Avg. PR	Crawler PR	Avg. SR	Crawler SR
1	0.24854932	0.63277194	0.15393478	0.56545578
2	0.24854932	0.63277194	0.15393478	0.56545578
3	0.24501068	0.46486353	0.13810930	0.41540997
4	0.24501068	0.46486353	0.13810930	0.41540997
5	0.24233737	0.50602884	0.14101354	0.45219598

Table 4: Values for PageRank iterations with initial rank $\forall v \in V : \mathrm{PR}_0(v) = 0.25$



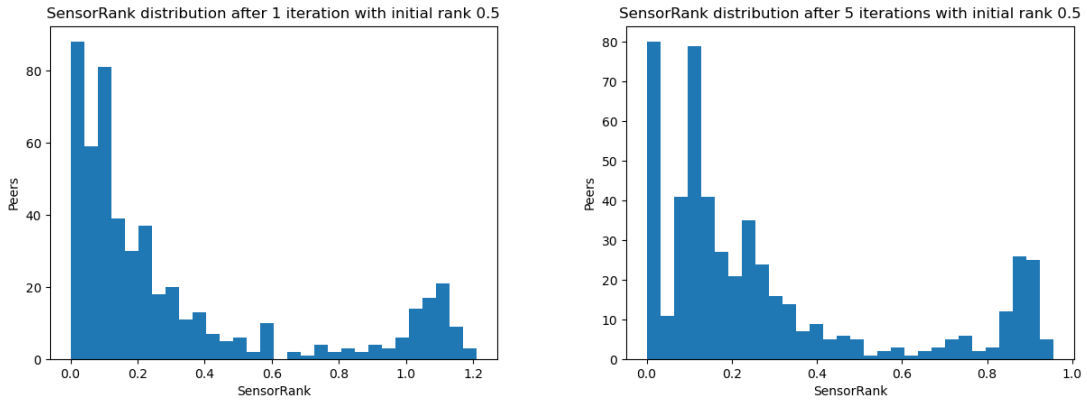
(a) Distribution after 1 iteration

(b) Distribution after 5 iterations

Figure 15: SensorRank distribution with initial rank $\forall v \in V : \mathrm{PR}_0(v) = 0.25$

Iteration	Avg. PR	Crawler PR	Avg. SR	Crawler SR
1	0.49709865	1.26554389	0.30786955	1.13091156
2	0.49709865	1.26554389	0.30786955	1.13091156
3	0.49002136	0.92972707	0.27621861	0.83081993
4	0.49002136	0.92972707	0.27621861	0.83081993
5	0.48467474	1.01205767	0.28202708	0.90439196

Table 5: Values for PageRank iterations with initial rank $\forall v \in V : \mathrm{PR}_0(v) = 0.5$

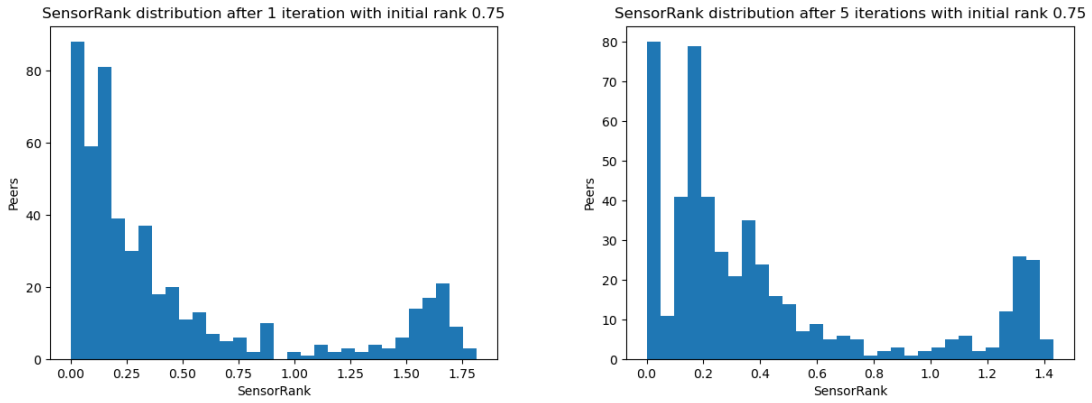


(a) Distribution after 1 iteration (b) Distribution after 5 iterations

Figure 16: SensorRank distribution with initial rank $\forall v \in V : \mathrm{PR}_0(v) = 0.5$

Iteration	**Avg. PR**	**Crawler PR**	**Avg. SR**	**Crawler SR**
1	0.74564797	1.89831583	0.46180433	1.69636734
2	0.74564797	1.89831583	0.46180433	1.69636734
3	0.73503203	1.39459060	0.41432791	1.24622990
4	0.73503203	1.39459060	0.41432791	1.24622990
5	0.72701212	1.51808651	0.42304062	1.35658794

Table 6: Values for PageRank iterations with initial rank $\forall v \in V : \mathrm{PR}_0(v) = 0.75$



(a) Distribution after 1 iteration

(b) Distribution after 5 iterations

Figure 17: SensorRank distribution with initial rank $\forall v \in V : \mathrm{PR}_0(v) = 0.75$

The distribution graphs in Figure 15, Figure 16 and Figure 17 show that the initial rank has no effect on the distribution, only on the actual numeric rank values and how far apart they are spread.

For all combinations of initial value and PageRank iterations, the rank for a well-known crawler is in the 95th percentile, so for our use case—detecting sensors due their high ranks—those parameters do not matter.

Looking at the data in smaller buckets of one hour each, the average number of successors per peer is 90.

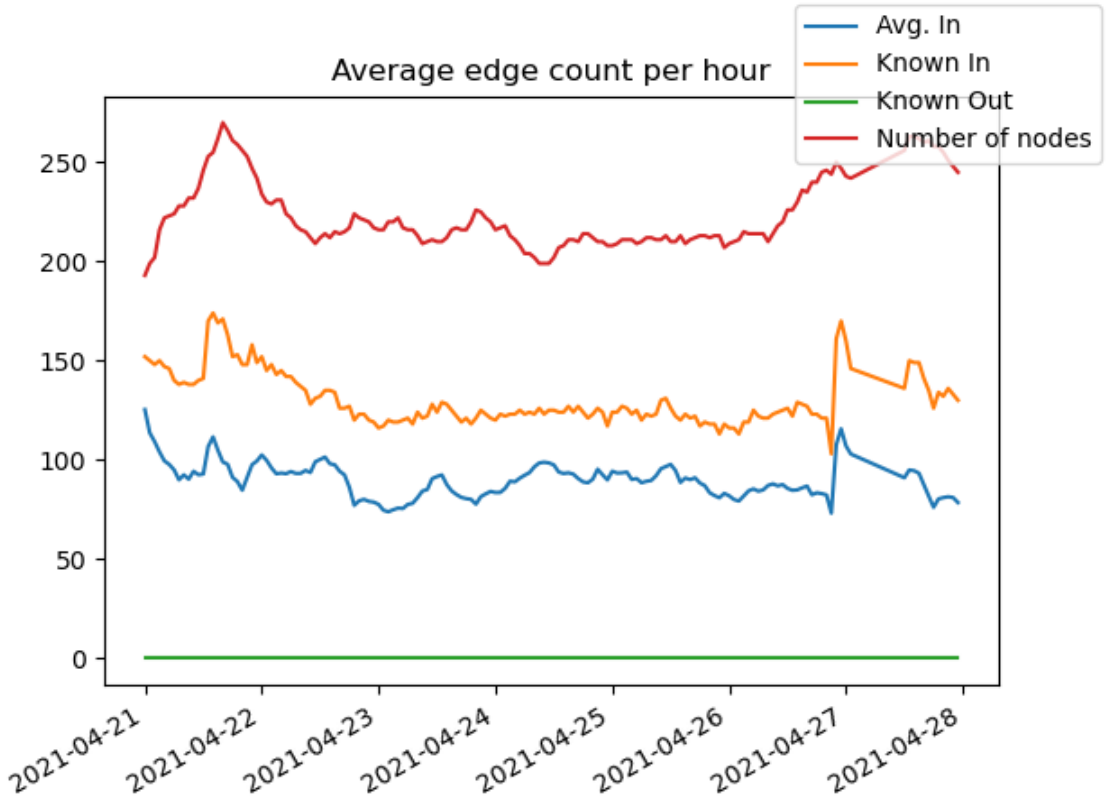
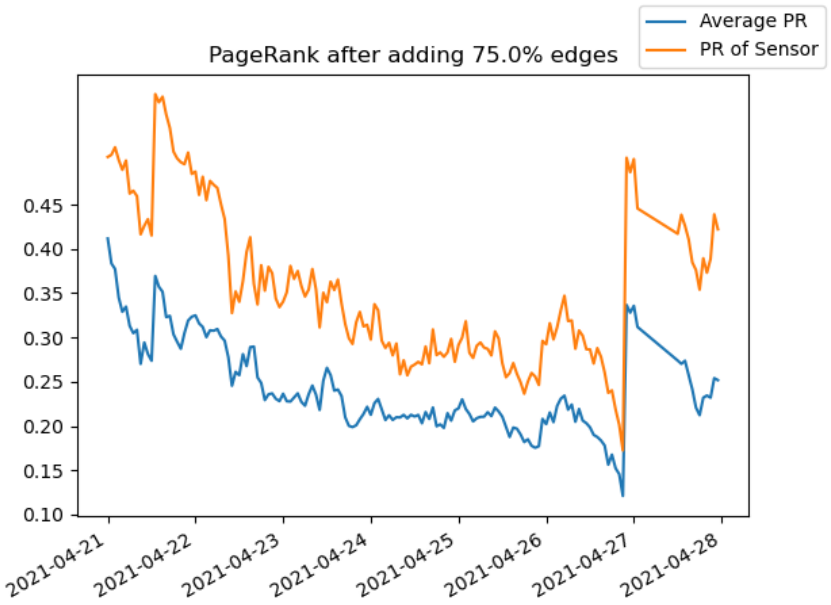
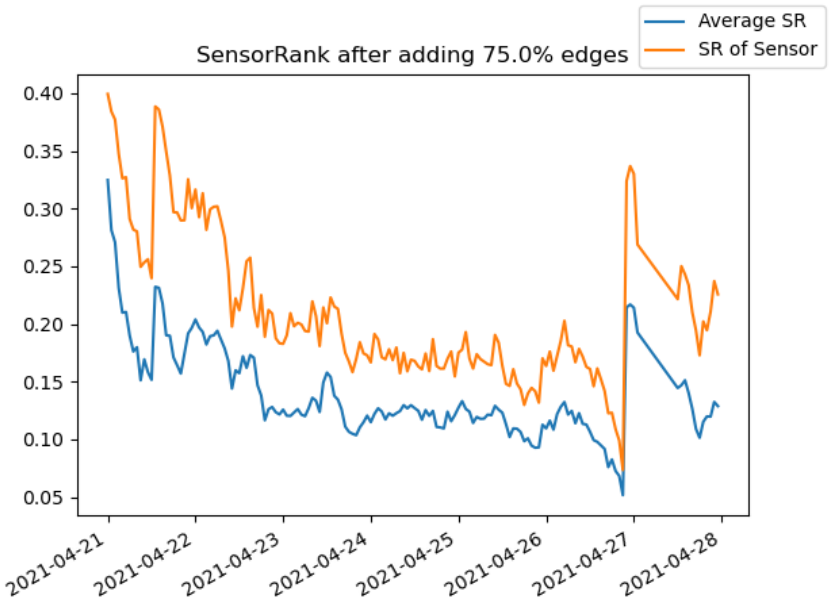


Figure 18: Average outgoing edges per peer per hour

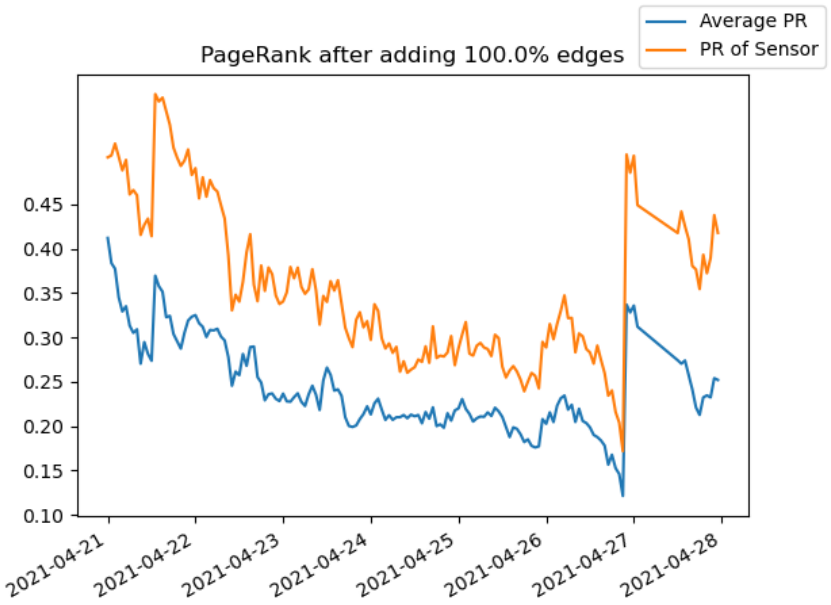
We evaluate the impact of outgoing edges by picking a percentage of random nodes in each bucket and creating edges from the sensor to each of the sampled peers, thereby evening the ratio between $\deg^+$ and $\deg^-$.



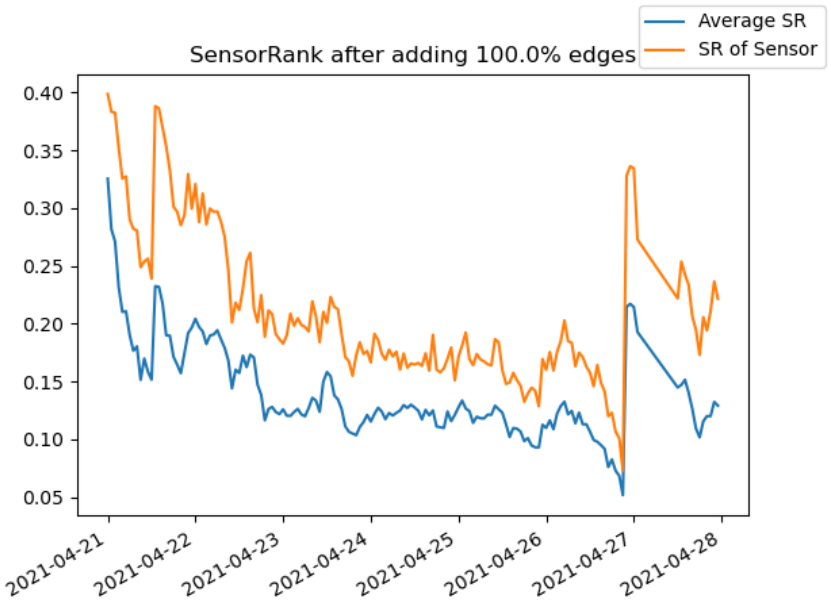
(a) PageRank after adding $0.75 \times |\text{pred}(v)|$ edges



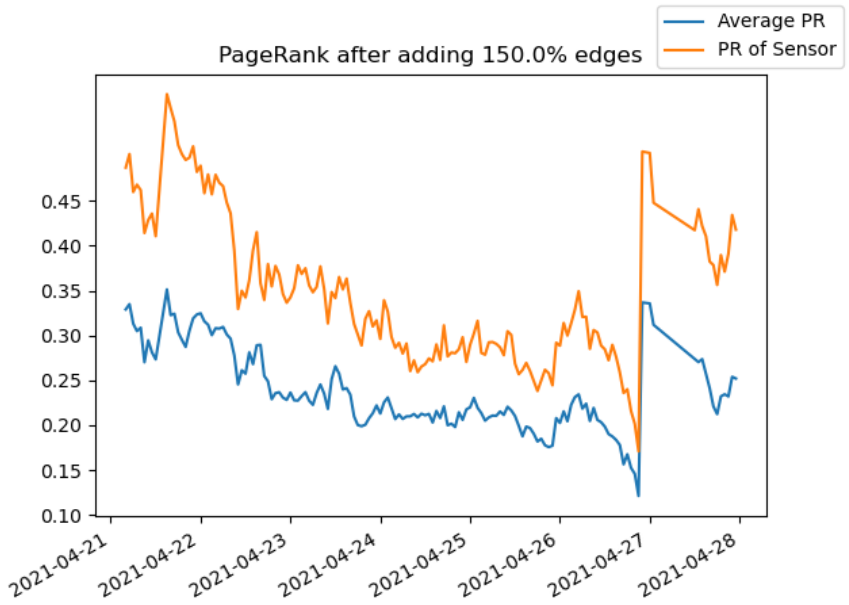
(b) SensorRank after adding $0.75 \times |\text{pred}(v)|$ edges



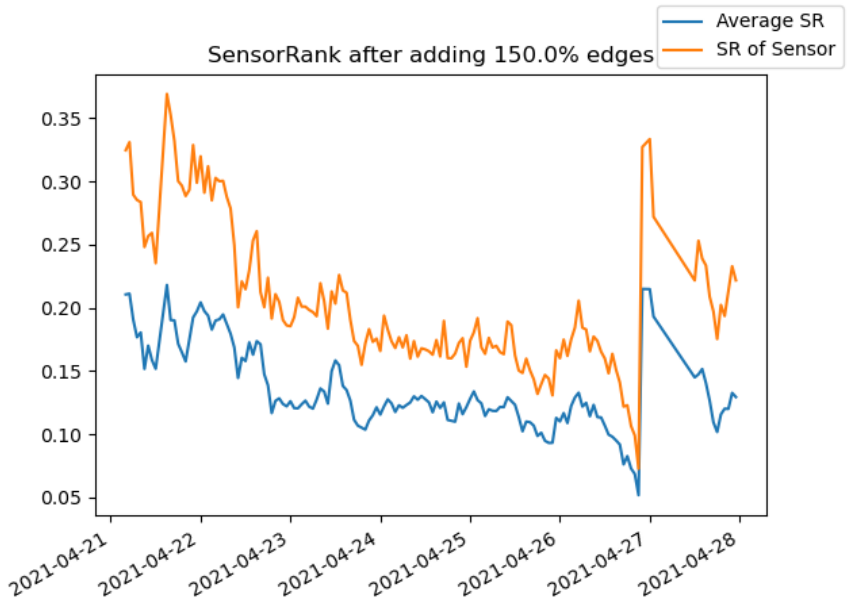
(a) PageRank after adding $1.0 \times |\text{pred}(v)|$ edges



(b) SensorRank after adding $1.0 \times |\text{pred}(v)|$ edges



(a) PageRank after adding $1.5 \times |\text{pred}(v)|$ edges



(b) SensorRank after adding $1.5 \times |\text{pred}(v)|$ edges

These results show, that simply adding new edges is not enough and we need to limit the

incoming edges to improve the Page- and SensorRank metrics.

6 Implementation

Crawlers in BMS report to the backend using gRPC Remote Procedure Calls (gRPCs)[3]. Both crawlers and the backend gRPC server are implemented using the Go[4] programming language, so to make use of existing know-how and to allow others to use the implementation in the future, the coordinator backend, and crawler abstraction were also implemented in Go.

BMS already has an existing abstraction for crawlers. This implementation is highly optimized but also tightly coupled and grown over time. The abstraction became leaky and extending it proved to be complicated. A new crawler abstraction was created with testability, extensibility, and most features of the existing implementation in mind, which can be ported back to be used by the existing crawlers.

The new implementation consists of three main interfaces:

- `FindPeer`, to receive new crawl tasks from any source
- `ReportPeer`, to report newly found peers
- `Protocol`, the actual botnet protocol implementation used to ping a peer and request its peer list

Currently, there are two sources `FindPeer` can use: read peers from a file on disk or request them from the gRPC BMS coordinator. The `ExactlyOnceFinder` delegate can wrap another `FindPeer` instance and ensures the source is only requested once. This is used to implement the bootstrapping mechanism of the old crawler, where once when the crawler is started, the list of bootstrap nodes is loaded from a text file. `CombinedFinder` can combine any amount of `FindPeer` instances and will return the sum of requesting all the sources.

The `PeerTask` instances returned by `FindPeer` contain the IP address and port of the peer, if the crawler should start or stop the operation, when to start and stop crawling, and

[3] `https://www.grpc.io`
[4] `https://go.dev/`

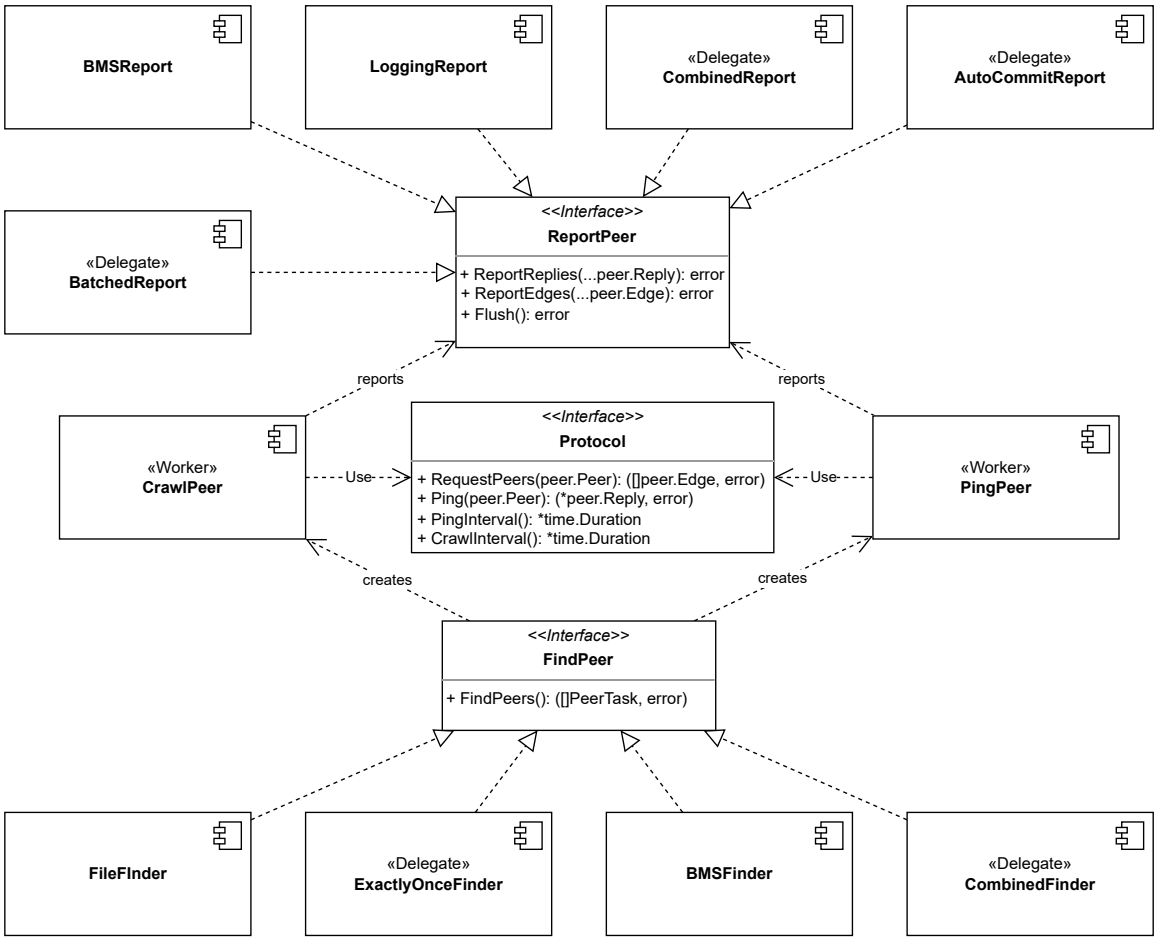


Figure 22: Architecture of the new crawler

in which interval the peer should be crawled. For each task, a `CrawlPeer` and `PingPeer` worker is started or stopped as specified in the received `PeerTask`. These tasks use the `ReportPeer` interface to report any new peer that is found.

Current report possibilities are `LoggingReport` to simply log new peers to get feedback from the crawler at runtime, and `BMSReport` which reports back to BMS. `BatchedReport` delegates a `ReportPeer` instance and batch newly found peers up to a specified batch size and only then flush and actually report. `AutoCommitReport` will automatically flush a delegated `ReportPeer` instance after a fixed amount of time and is used in combination with `BatchedReport` to ensure the batches are written regularly, even if the batch limit

is not reached yet. `CombinedReport` works analogous to `CombinedFinder` and combines many `ReportPeer` instances into one.

`PingPeer` and `CrawlPeer` use the implementation of the botnet `Protocol` to perform the actual crawling in predefined intervals, which can be overwritten on a per `PeerTask` basis.

The server-side part of the system consists of a gRPC server to handle the client requests, a scheduler to assign new peers, and a `Strategy` interface for modularity over how tasks are assigned to crawlers.

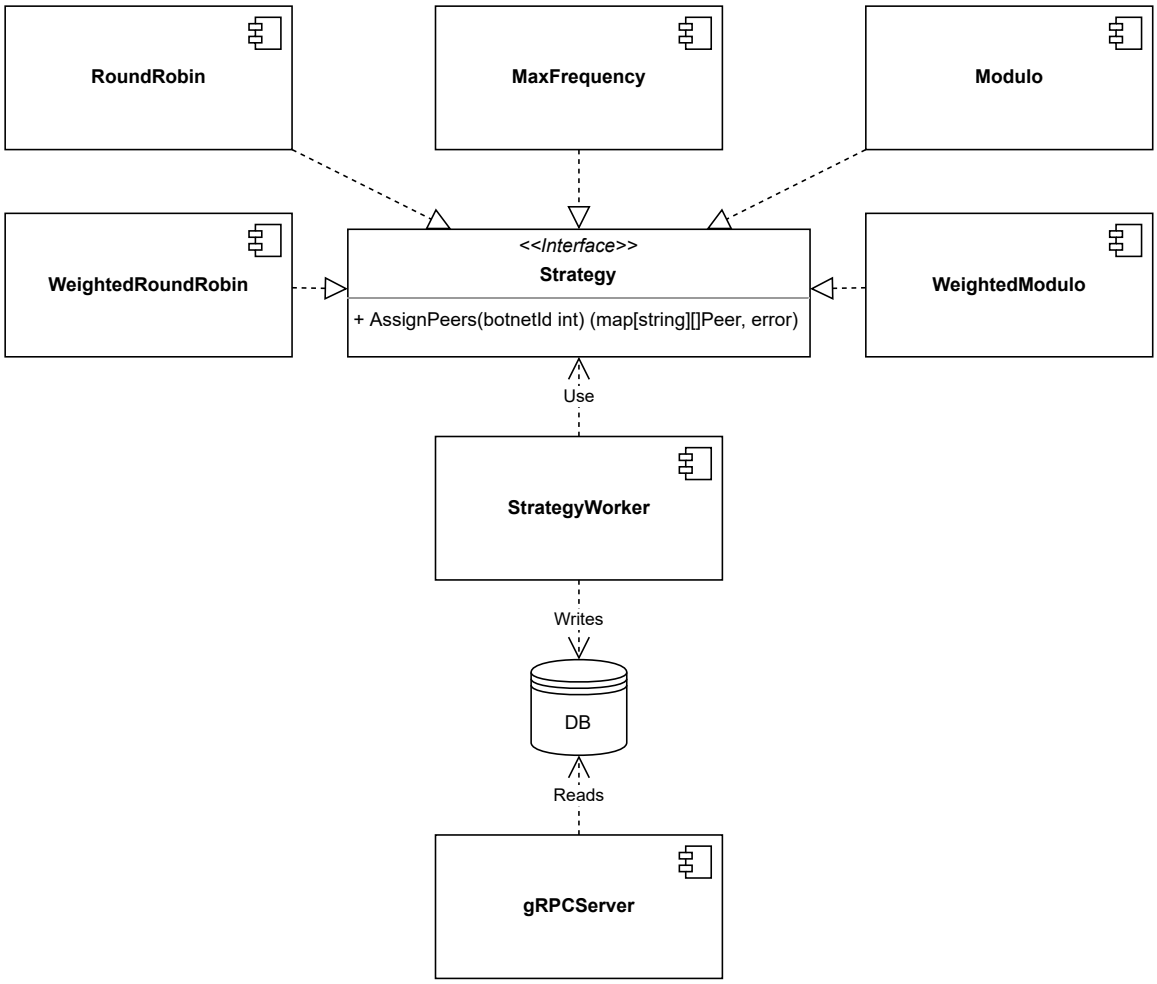


Figure 23: Architecture of the gRPC backend

7 Conclusion

Collaborative monitoring of P2P botnets allows circumventing some anti-monitoring efforts. It also enables more effective monitoring systems for larger botnets, since each peer can be visited by only one crawler. The current concept of independent crawlers in BMS can also use multiple workers but there is no way to ensure a peer is not watched by multiple crawlers thereby using unnecessary resources.

8 Further Work

Following this work, it should be possible to rewrite the existing crawlers using the new abstraction. This might bring some performance issues to light which can be solved by investigating the optimizations from the old implementation and applying them to the new one.

Another way to expand on this work is automatically scaling the available crawlers up and down, depending on the botnet size and the number of concurrently online peers. Doing so would allow a constant crawl interval for even highly volatile botnets.

Placing churned peers or peers with suspicious network activity (those behind carrier-grade NATs) might just offer another characteristic to flag sensors in a botnet. The feasibility of this approach should be investigated and maybe there are ways to mitigate this problem.

Autoscaling features offered by many cloud-computing providers can be evaluated to automatically add or remove crawlers based on the monitoring load, a botnet's size, and the number of active peers. This should also allow the creation of workers with new IP addresses in different geolocations in a fast, easy and automated way. The current implementation assumes an immutable set of crawlers. For autoscaling to work, efficient reassignment of peers has to be implemented to account for added or removed workers.

Acknowledgments

In the end, I would like to thank

- Prof. Dr. Christoph Skornia for being a helpful supervisor in this and many earlier works of mine
- Leon Böck for offering the possibility to work on this research project, regular feedback and technical expertise
- Valentin Sundermann for being available for insightful ad hoc discussions at any time of day for many years
- Friends and family who pushed me into continuing this path

List of Figures

List of Tables

List of Listings

List of Acronyms

AS Autonomous System 21, 25

BMS Botnet Monitoring System . . . . . 14 f., 18, 27, 48 f., 51

C2 Command and Control 6–9

DDoS Distributed Denial of Service 6, 13

DHT Distributed Hash Table 9

gRPC gRPC Remote Procedure Call 48, 50

IoT Internet of Things 6

IRC Internet Relay Chat . . . . . 6 f.

MD5 Message-Digest Algorithm 5 21

MM Membership Management . . . . . 9, 12 f.

NAT Network Access Translation 9, 25, 52

P2P Peer-to-Peer 2, 6–15, 23, 25, 51

SPoF Single Point of Failure 7

WCC Weakly Connected Component 14, 23, 25, 37, 39

References

[1] Dennis Andriesse, Christian Rossow, and Herbert Bos. “Reliable Recon in Adversarial Peer-to-Peer Botnets”. In: *Proceedings of the 2015 Internet Measurement Conference*. IMC '15: Internet Measurement Conference. Tokyo Japan: ACM, Oct. 28, 2015, pp. 129–140. ISBN: 978-1-4503-3848-6. DOI: 10.1145/2815675.2815682. URL: https://dl.acm.org/doi/10.1145/2815675.2815682 (visited on 11/16/2021).

[2] Dennis Andriesse et al. “Highly Resilient Peer-to-Peer Botnets Are Here: An Analysis of Gameover Zeus”. In: *2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*. 2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE). Fajardo, PR, USA: IEEE, Oct. 2013, pp. 116–123. ISBN: 978-1-4799-2534-6 978-1-4799-2535-3. DOI: 10.1109/MALWARE.2013.6703693. URL: https://ieeexplore.ieee.org/document/6703693/ (visited on 02/27/2022).

[3] Manos Antonakakis et al. “From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware”. In: *21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX Association, Aug. 2012, pp. 491–506. ISBN: 978-931971-95-9. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/antonakakis.

[4] *Availability of broadband internet to households in Germany from 2017 to 2020, by bandwidth class*. Statista Inc. Aug. 16, 2021. URL: https://www.statista.com/statistics/460180/broadband-availability-by-bandwidth-class-germany/ (visited on 11/11/2021), archived at https://web.archive.org/web/20210309010747/https://www.statista.com/statistics/460180/broadband-availability-by-bandwidth-class-germany/ on Mar. 9, 2021.

[5] Leon Böck et al. "Next Generation P2P Botnets: Monitoring Under Adverse Conditions". In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Michael Bailey et al. Vol. 11050. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 511–531. ISBN: 978-3-030-00469-9 978-3-030-00470-5. DOI: 10.1007/978-3-030-00470-5_24. URL: http://link.springer.com/10.1007/978-3-030-00470-5%5C_24 (visited on 04/08/2022).

[6] Leon Böck et al. "Poster: Challenges of Accurately Measuring Churn in P2P Botnets". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19: 2019 ACM SIGSAC Conference on Computer and Communications Security. London United Kingdom: ACM, Nov. 6, 2019, pp. 2661–2663. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3363281. URL: https://dl.acm.org/doi/10.1145/3319535.3363281 (visited on 11/12/2021).

[7] Joseph Demarest. *Taking Down Botnets*. Federal Bureau of Investigation. July 15, 2014. URL: https://www.fbi.gov/news/testimony/taking-down-botnets (visited on 03/23/2022), archived at https://web.archive.org/web/20220318082034/https://www.fbi.gov/news/testimony/taking-down-botnets.

[8] David Dittrich. "So You Want to Take over a Botnet". In: *Proceedings of the 5th USENIX Conference on Large-Scale Exploits and Emergent Threats*. LEET'12. San Jose, CA: USENIX Association, 2012, p. 6. DOI: 10.5555/2228340.2228349.

[9] Falliere, Nicolas. *Sality: Story of a Peer-to-Peer Viral Network*. July 2011. URL: https://papers.vx-underground.org/archive/Symantec/sality-story-of-peer-to-peer-11-en.pdf (visited on 03/16/2022), archived at https://web.archive.org/web/20161223003320/http://www.symantec.com/content/en/us/enterprise/media/security_

response/whitepapers/sality_peer_to_peer_viral_network.pdf on Dec. 23, 2016.

[10] Dan Goodin. *Brace yourselves — source code powering potent IoT DDoSes just went public*. Ars Technica. Oct. 2, 2016. URL: https://arstechnica.com/information-technology/2016/10/brace-yourselves-source-code-powering-potent-iot-ddoses-just-went-public/ (visited on 11/11/2021), archived at https://web.archive.org/web/20211022032617/https://arstechnica.com/information-technology/2016/10/brace-yourselves-source-code-powering-potent-iot-ddoses-just-went-public/ on Oct. 22, 2021.

[11] Guofei Gu et al. "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection". In: *Proceedings of the 17th Conference on Security Symposium*. SS'08. San Jose, CA: USENIX Association, 2008, pp. 139–154.

[12] Amy Hogan-Burney. *Notorious cybercrime gang's botnet disrupted*. Microsoft. URL: https://blogs.microsoft.com/on-the-issues/2022/04/13/zloader-botnet-disrupted-malware-ukraine/ (visited on 04/15/2022), archived at https://web.archive.org/web/20220413210653/https://blogs.microsoft.com/on-the-issues/2022/04/13/zloader-botnet-disrupted-malware-ukraine/ on Apr. 13, 2022.

[13] Shankar Karuppayah et al. "BoobyTrap: On autonomously detecting and characterizing crawlers in P2P botnets". In: *2016 IEEE International Conference on Communications (ICC)*. ICC 2016 - 2016 IEEE International Conference on Communications. Kuala Lumpur, Malaysia: IEEE, May 2016, pp. 1–7. ISBN: 978-1-4799-6664-6. DOI: 10.1109/ICC.2016.7510885. URL: http://ieeexplore.ieee.org/document/7510885/ (visited on 11/12/2021).

[14] Shankar Karuppayah et al. "SensorBuster: On Identifying Sensor Nodes in P2P Botnets". In: *Proceedings of the 12th International Conference on Avail-*

ability, Reliability and Security. ARES '17. New York, NY, USA: Association for Computing Machinery, Aug. 29, 2017, pp. 1–6. ISBN: 978-1-4503-5257-4. DOI: 10.1145/3098954.3098991. URL: https://doi.org/10.1145/3098954.3098991 (visited on 03/23/2021).

[15] Erwan Le Malécot and Daisuke Inoue. "The Carna Botnet Through the Lens of a Network Telescope". In: *Foundations and Practice of Security*. Ed. by Jean Luc Danger et al. Vol. 8352. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 426–441. ISBN: 978-3-319-05301-1 978-3-319-05302-8. DOI: 10.1007/978-3-319-05302-8_26. URL: http://link.springer.com/10.1007/978-3-319-05302-8%5C_26 (visited on 04/16/2022).

[16] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Red. by Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen. Vol. 2429. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-44179-3 978-3-540-45748-0. DOI: 10.1007/3-540-45748-8_5. URL: http://link.springer.com/10.1007/3-540-45748-8_5 (visited on 04/16/2022).

[17] Yacin Nadji, Roberto Perdisci, and Manos Antonakakis. "Still Beheading Hydras: Botnet Takedowns Then and Now". In: *IEEE Transactions on Dependable and Secure Computing* 14.5 (Sept. 1, 2017), pp. 535–549. ISSN: 1545-5971. DOI: 10.1109/TDSC.2015.2496176. URL: http://ieeexplore.ieee.org/document/7312442/ (visited on 03/17/2022).

[18] Yacin Nadji et al. "Beheading hydras: performing effective botnet takedowns". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. the 2013 ACM SIGSAC conference. Berlin, Germany: ACM Press, 2013, pp. 121–132. ISBN: 978-1-4503-2477-9.

DOI: 10.1145/2508859.2516749. URL: http://dl.acm.org/citation.cfm?doid=2508859.2516749 (visited on 03/15/2022).

[19] Shishir Nagaraja et al. "BotGrep: Finding P2P Bots with Structured Graph Analysis". In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security'10. Washington, DC: USENIX Association, 2010, p. 7. ISBN: 8887666655554.

[20] Jose Nazario and Thorsten Holz. "As the net churns: Fast-flux botnet observations". In: *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*. 2008 3rd International Conference on Malicious and Unwanted Software (MALWARE). Fairfax, VI: IEEE, Oct. 2008, pp. 24–31. ISBN: 978-1-4244-3288-2. DOI: 10.1109/MALWARE.2008.4690854. URL: https://ieeexplore.ieee.org/document/4690854/ (visited on 03/15/2022).

[21] *Number of Internet of Things (IoT) Connected Devices Worldwide from 2019 to 2030*. Statista Inc. Dec. 2020. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (visited on 11/11/2021), archived at https://web.archive.org/web/20211025185804/https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ on Oct. 25, 2021.

[22] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web*. Jan. 29, 1998. URL: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf (visited on 11/30/2021).

[23] Nick Pantic and Mohammad I. Husain. "Covert Botnet Command and Control Using Twitter". In: *Proceedings of the 31st Annual Computer Security Applications Conference on - ACSAC 2015*. the 31st Annual Computer Security Applications Conference. Los Angeles, CA, USA: ACM Press, 2015, pp. 171–180. ISBN: 978-1-4503-3682-6. DOI: 10.1145/2818000.2818047.

URL: http://dl.acm.org/citation.cfm?doid=2818000.2818047 (visited on 03/15/2022).

[24] Christian Rossow et al. "SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets". In: *2013 IEEE Symposium on Security and Privacy*. 2013 IEEE Symposium on Security and Privacy (SP) Conference dates subject to change. Berkeley, CA, USA: IEEE, May 2013, pp. 97–111. ISBN: 978-1-4673-6166-8 978-0-7695-4977-4. DOI: 10.1109/SP.2013.17. URL: https://ieeexplore.ieee.org/document/6547104/ (visited on 03/15/2022).

[25] Marc Stevens. "Fast Collision Attack on MD5". In: (2006). https://ia.cr/2006/104.

[26] Daniel Stutzbach and Reza Rejaie. "Understanding Churn in Peer-to-Peer Networks". In: *Proceedings of the 6th ACM SIGCOMM on Internet Measurement - IMC '06*. The 6th ACM SIGCOMM. Rio de Janeriro, Brazil: ACM Press, 2006, p. 189. ISBN: 978-1-59593-561-8. DOI: 10.1145/1177080.1177105. URL: http://portal.acm.org/citation.cfm?doid=1177080.[redacted]1177105 (visited on 03/08/2022).

[27] Alex Turing, Hui Wang, and Genshen Ye. *The Mostly Dead Mozi and Its' Lingering Bots*. 360 Netlab. Aug. 30, 2021. URL: https://blog.netlab.360.com/the-mostly-dead-mozi-and-its-lingering-bots/ (visited on 04/07/2022), archived at https://web.archive.org/web/20220130162722/https://blog.netlab.360.com/the-mostly-dead-mozi-and-its-lingering-bots/ on Jan. 30, 2022.

[28] Junjie Zhang et al. "Building a Scalable System for Stealthy P2P-Botnet Detection". In: *IEEE Transactions on Information Forensics and Security* 9.1 (Jan. 2014), pp. 27–38. ISSN: 1556-6013, 1556-6021. DOI: 10.1109/TIFS.2013.2290197. URL: http://ieeexplore.ieee.org/document/6661360/ (visited on 11/09/2021).

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Masterthesis als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.

2. Ich erkläre hiermit, dass ich diese Masterthesis selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum und Unterschrift

Presented by:	Valentin Brandl
Student ID:	3220018
Study Programme:	Master Informatik
Supervisor:	Prof. Dr. Christoph Skornia
Secondary Supervisor:	Prof. Dr. Thomas Waas