OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

MASTERTHESIS

Valentin Brandl

Collaborative Crawling of Fully Distributed Botnets

7th 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 on general internet infrastructure and services. Decentralized P2P topologies make it harder to detect monitor and take those botnets offline. This work explores ways to make monitoring of fully distributed botnets more efficient, resilient and harder to detect, by using a collaborative, coordinated approach. do me

***Keywords*—** P2P, botnet, monitoring, collaboration

Contents

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.

1.1 Motivation

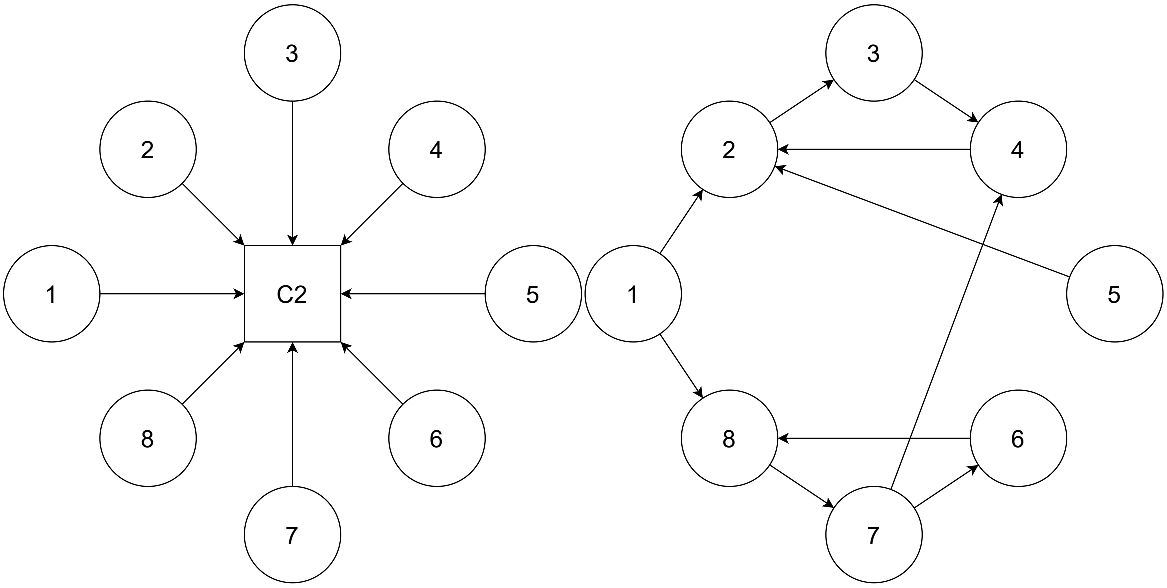
In 2021 there were around 10 billion internet connected IoT devices and this number is estimated to more than double over the next years up to 25 billion in 2030 [16]. Many of these devices run on outdated software, don't receive regular updates, and don't follow general security best practices. While in 2016 only 77 % of German households had a broadband connection with a bandwidth of 50 MBit/s or more, in 2020 it was already 95 % with more than 50 MBit/s and 59 % with at least 1000 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. 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 TBit/s of traffic [9].

A botnet is a network of infected computers with some means of communication to control the infected systems. Centralized botnets use one or more coordinating hosts called command and control (C2) servers. These C2 servers can use any protocol from internet relay chat (IRC) over hypertext transfer protocol to Twitter [18] as communication channel with the infected hosts. The abuse of infected systems includes several activities—DDoS attacks, banking fraud, as proxies to hide the attacker's identity, send spam emails. . .

Analyzing and shutting down a centralized botnet is comparatively easy since the central means of communication (the C2 IP address or domain name, Twitter handle or IRC channel), can be extracted from the malicious binaries and are therefore publicly known.

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.

To complicate 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 domain name system, where a large pool of IP addresses is assigned randomly to the C2 domains to prevent IP based blacklisting [15].



(a) Topology of a C2 controlled botnet (b) Topology of a peer-to-peer (P2P) botnet

Figure 1: Communication paths in different types of botnets

A number of botnet operations were shut down like this [13] and as the defenders upped their game, so did attackers—the concept of peer-to-peer (P2P) botnets emerged. The idea is to build a decentralized network without single points of failure (SPOF) in the form of C2 servers as shown in Figure 1b. 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 bot master only needs to join the network to send new commands or receive stolen data. Any of the nodes in Figure 1b could be the bot master 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 operators of P2P botnets but due to the autonomy offered by the decentralized approach, the botnet keeps communicating [21]. Especially worm-like botnets, where each peer tries to find and infect other systems, the network can keep lingering for many years.

This lack of a SPOF makes P2P botnets more resilient to take-down attempts since the communication is not stopped and bot masters can easily rejoin the network and send commands.

The constantly growing damage produced by botnets has many researchers and law enforcement agencies trying to shut down these operations [13, 12, 7, 6]. The monetary value of these botnets directly correlates with the amount of effort bot masters are willing to put into implementing defense mechanisms against take-down attempts. Some of these countermeasures include deterrence, which limits the number of allowed bots per IP address or subnet to 1; 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, which invalidates the data collected by crawlers; and active retaliation like DDoS attacks against sensors or crawlers [1].

Successful take-downs of a P2P botnet requires intricate knowledge over the network topology, protocol characteristics and participating peers. This work aims to make the monitoring and information gathering phase more efficient and resilient to detection.

1.2 Formal Model of a P2P Botnet

A P2P botnet can be modelled as a digraph

$$G = (V, E)$$

With the set of vertices 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 [19]. Components consisting of peers, that are infected by the same bot, are considered part of the same graph.

$\forall v \in V$, the **predecessors** $\text{pred}(v)$ and **successors** $\text{succ}(v)$ are defined as:

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

The set of edges pred(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 vertex $v \in V$, the in and out degree $\deg^+$ and $\deg^-$ describe how many bots know v or are known by v respectively.

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

more details

1.3 Detection Techniques for P2P Botnets

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

1.3.1 Passive Detection

For passive detection, traffic flows are analysed in large amounts of collected network traffic (*e.g.* from internet service providers). This has some advantages in that it is not possible for bot masters to detect or prevent data collection of that kind, but 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 [22]. Also getting access to the required datasets might not be possible for everyone.

As most detection botnet mechanisms, also the passive ones work by building communication graphs and finding tightly coupled subgraphs that might be indicative of a botnet [14]. An advantage of passive detection is, that it is independent of protocol details, specific binaries or the structure of the network (P2P vs. centralized) [10].

- Large scale network analysis (hard to differentiate from legitimate P2P traffic (*e.g.* BitTorrent), hard to get data, knowledge of some known bots required) [22]
- Heuristics: Same traffic patterns, same malicious behaviour

no context

1.3.2 Active Detection

In this case, a subset of the botnet protocol are reimplemented to place pseudo-bots or sensors in the network, which will only communicate with other nodes but won't accept or execute commands to perform malicious actions. The difference in behaviour from the reference implementation and conspicuous graph properties (*e.g.* high $\deg^+$ vs. low $\deg^-$) of these sensors allows bot masters to detect and block the sensor nodes.

There are three subtypes of active detection:

1. Crawlers: recursively ask known bots for their peer lists
2. Sensors: implement a subset of the botnet protocol and become part of the network without performing malicious actions
3. Hybrid of crawlers and sensors

2 Methodology

The implementation of the concepts of this work will be done as part of Botnet Monitoring System (BMS)[1], a monitoring platform for P2P botnets described by Böck et al. in "Challenges of Accurately Measuring Churn in P2P Botnets". BMS uses 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, I implemented different node ranking algorithms (among others "PageRank" [17]) to detect sensors and crawlers in a botnet, as described in "SensorBuster". Both

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

ranking algorithms use the $\deg^+$ and $\deg^-$ to weight the nodes. Another way to enumerate candidates for sensors in a P2P botnet is to find weakly connected components (WCCs) in the graph. Sensors will have few to none outgoing edges, since they don't participate actively in the botnet.

The goal of this work is to complicate detection mechanisms like this for bot masters by centralizing the coordination of the system's crawlers and sensors, thereby reducing the node's rank for specific graph metrics. The coordinated work distribution also helps in efficiently monitoring large botnets where one sensor is not enough to track all peers. The changes should allow the current sensors to use the new abstraction with as few changes as possible to the existing code.

The final results should be as general as possible and not depend on any botnet's specific behaviour, but it assumes, that every P2P botnet has some kind of "getPeerList" method in the protocol, that allows other peers to request a list of active nodes to connect to.

In the current implementation, each crawler will itself visit and monitor each new node it finds. The idea for this work 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 archive the goal of the according coordination strategy. That sensor will be responsible to monitor the new node.

If it is not possible, to select a specific 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 stay undetected.

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. Further work might even consider autoscaling the monitoring activity using some kind of cloud computing provider.

2.1 Protocol Primitives

The coordination protocol must allow the following operations:

2.1.1 Register Worker

`register(capabilities)`: Register new worker with capabilities (which botnet, available bandwidth, ...). This is called periodically and used to determine which worker is still active, when splitting the workload.

2.1.2 Report Peer

`reportPeer(peers)`: Report found targets. Both successful and failed attempts are reported, to detect as soon as possible, when a peer became unavailable.

2.1.3 Report Edge

`reportEdge(edges)`: Report found edges. Edges are found by querying the peer list of known peers. This is how new peers are detected.

2.1.4 Request Tasks

`requestTasks() []PeerTask`: Receive a batch of crawl tasks from the coordinator. The tasks consist of the target peer, if the crawler should start or stop the operation, when it should start and stop monitoring and the frequency.

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

3 Coordination Strategies

Let C be the set of available crawlers. Without loss of generality, if not stated otherwise, I 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.

3.1 Load Balancing

This strategy simply splits the work into chunks and distributes the work between the available crawlers. The following sharding strategy will be investigated:

- Round Robin. See subsubsection 3.1.1
- Assuming IP addresses are evenly distributed and so are infections, take the IP address as an 32 Bit integer modulo $|C|$. See subsubsection 3.1.3 Problem: reassignment if a crawler joins or leaves

Load balancing in itself does not help prevent the detection of crawlers but it allows better usage of available resources. No peer will be crawled by more than one crawler and it allows crawling of bigger botnets where the current approach would reach its limit and could also be worked around with scaling up the machine where the crawler is executed. Load balancing allows scaling out, which can be more cost-effective.

3.1.1 Round Robin Distribution

3.1.2 Even Work Distribution

weighted round robin

Work is evenly distributed between crawlers according to their capabilities. For the sake of simplicity, only the bandwidth will be considered as capability but it can be extended by any shared property between the crawlers, *e.g.* available memory, CPU speed. For a given crawler $c_i \in C$ let $B(c_i)$ be the total bandwidth of the crawler. The total available bandwidth is $b = \sum_{c \in C} B(c_i)$. The weight $W(c_i) = \frac{B}{B(c_i)}$ defines which percentage of the work gets assigned to c_i. The set of target peers $P =< p_0, p_1, \ldots, p_{n-1} >$, is partitioned into $|C|$ subsets according to $W(c_i)$ and each subset is assigned to its crawler c_i. The mapping gcd(C) is the greatest common divisor of all peers in C, maxWeight$(C) = \max\{\forall c \in C : W(c)\}$.

proper def for weight

The following algorithm distributes the work according to the crawler's capabilities:

```
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
}
```

This creates a list of crawlers where a crawler can occur more than once, depending on its capabilities. The set of crawlers $\{a, b, c\}$ with 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.

The following weighted round-robin algorithm distributes the work according to the crawlers' capabilities:

```
work := make(map[string][]strategy.Peer)
commonWeight := 0
counter := -1
for _, peer := range peers {
  for {
    counter += 1
    if counter <= mod {
      counter = 0
    }
    crawler := crawlers[counter]
    if counter == 0 {
      commonWeight = commonWeight - gcd(weightList...)
      if commonWeight <= 0 {
        commonWeight = max(weightList...)
        if commonWeight == 0 {
          return nil, errors.New("invalid common weight")
        }
      }
    }
    if weights[crawler] >= commonWeight {
      work[crawler] = append(work[crawler], peer)
      break
```

```
    }
  }
}
```

referencel for wrr

3.1.3 IP-based Partitioning

The output of cryptographic hash functions is uniformly distributed—even substrings of the calculated hash hold this property. Calculating the hash of an IP address and distributing the work with regard to hash(IP) $\mod |C|$ creates about evenly sized buckets for each worker to handle. This gives us the mapping $m(i) = \text{hash}(i) \mod |C|$ 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, 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.

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.

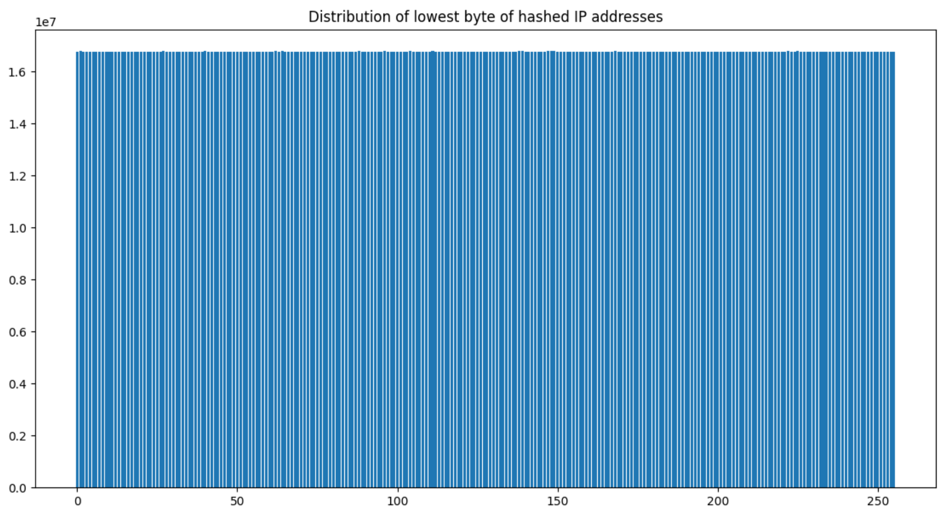


Figure 2: Distribution of the lowest byte of MD5 hashes over IPv4

MD5 returns a 128 Bit hash but Go cannot directly work with 128 Bit integers. It would be possible to implement the modulo operation for arbitrarily sized integers, but the uniform distribution also holds substrings of hashes. Figure 2 shows the distribution of the lowest 8 Bit for MD5 hashes over all 2^{32} IP addresses in their representation as 32 Bit integers.

By exploiting the even distribution offered by hashing, the work of each crawler is also evenly distributed over all IP subnets, autonomous system (AS) and geolocations. This ensures neighboring peers (*e.g.* in the same AS, geolocation or IP subnet) get visited by different crawlers.

3.2 Reduction of Request Frequency

The GameOver Zeus botnet deployed a blacklisting mechanism, where crawlers are blocked based in their request frequency [2]. In a single crawler approach, the crawler frequency has to be limited to prevent being hitting the request limit.

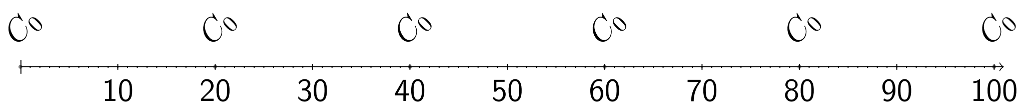


Figure 3: Timeline of crawler events as seen from a peer when crawled by a single crawler

Using collaborative crawlers, an arbitrarily fast frequency can be achieved without being blacklisted. With $L \in \mathbb{N}$ being the frequency limit at which a crawler will be blacklisted, $F \in \mathbb{N}$ being the crawl frequency that should be achieved. The amount of crawlers C required to achieve the frequency F without being blacklisted and the offset O between crawlers are defined as

$$C = \left\lceil \frac{F}{L} \right\rceil$$
$$O = \frac{1\,\text{req}}{F}$$

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 $L = 5\,\text{req}/100\text{s}$, crawling a botnet at $F = 20\,\text{req}/100\text{s}$ requires $C = \left\lceil \frac{20\,\text{req}/100\text{s}}{5\,\text{req}/100\text{s}} \right\rceil = 4$ crawlers. Those crawlers must be scheduled $O = \frac{1\,\text{req}}{20\,\text{req}/100\text{s}} = 5\,\text{s}$ apart at a frequency of L for an even request distribution.

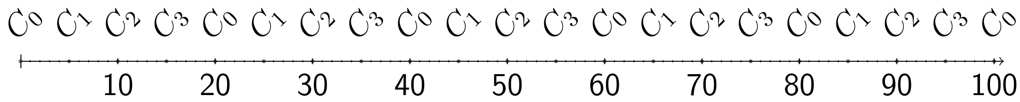
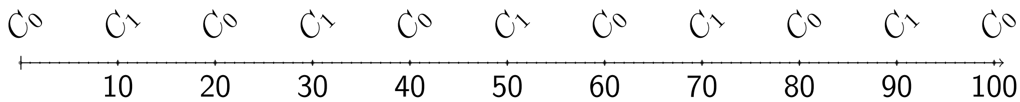


Figure 4: Timeline of crawler events as seen from a peer when crawled by multiple crawlers

As can be seen in Figure 4, each crawler C_0 to C_3 performs only 5 req/100s while overall achieving 20 req/100s.

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

Using the example from above with $L = 5\,\text{req}/100\text{s}$ but now only two crawlers $C = 2$, it is still possible to achieve an effective frequency of $F = 2 \times 5\,\text{req}/100\text{s} = 10\,\text{req}/100\text{s}$ and $O = \frac{1\,\text{req}}{10\,\text{req}/100\text{s}} = 10\,\text{s}$:



While the effective frequency of the whole system is halved compared to Figure 4, it is still possible to double the frequency over the limit.

3.3 Creating Outgoing 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. One of those, “SensorBuster” uses WCCs since crawlers don’t have any edges back to the main network in the graph.

Building a complete graph $G_C = K_{|C|}$ between the crawlers by making them return the other crawlers 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 9b and Table 1).

With $v \in V$, $\text{succ}(v)$ being the set of successors of v and $\text{pred}(v)$ being the set of predecessors of v, PageRank is recursively defined as [17]:

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

For the first iteration, the PageRank of all nodes is set to the same initial value. When iterating often enough, any value can be chosen [17].

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 behaviour for automated crawling and ranking—a dampingFactor of 1.0 will be used, which simplifies the formula to

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

Based on this, SensorRank is defined as

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

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

percentag of bot- net must be crawlers to make a signif- icant change

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

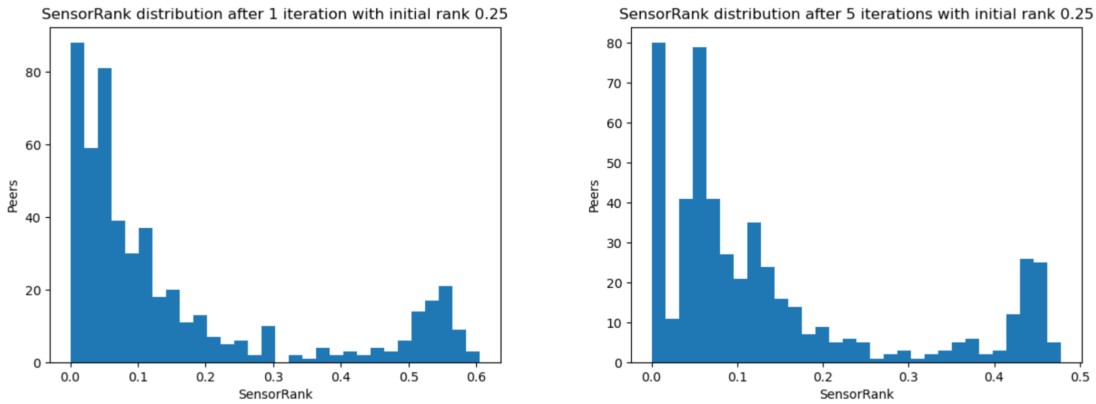
Table 1: Values for metrics from Figure 9 (a/b)

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.

In our experiments on a snapshot of the Sality [8] botnet obtained from BMS over the span of 21st to 28th April 2021 even 1 iteration were enough to get distinct enough values to detect sensors and crawlers.

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 2: Values for PageRank iterations with initial rank $\forall v \in V : \text{PR}(v) = 0.25$



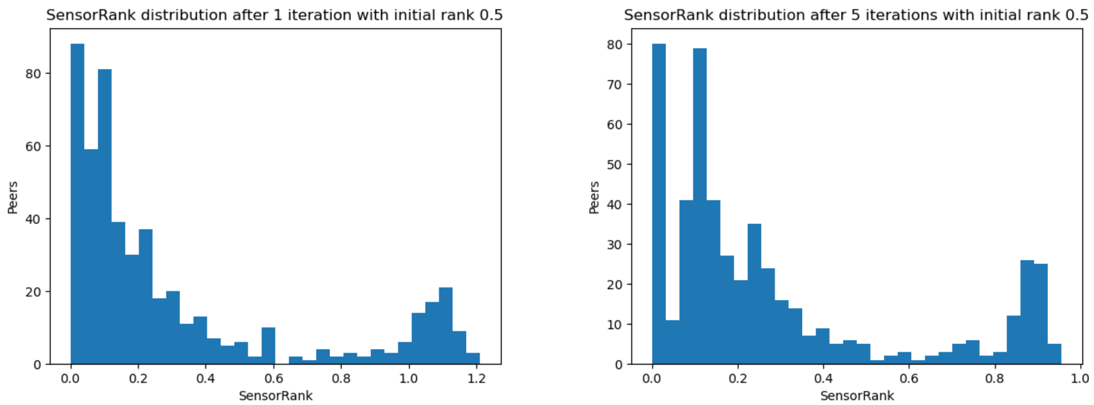
(a) Distribution after 1 iteration

(b) Distribution after 5 iterations

Figure 5: SensorRank distribution with initial rank $\forall v \in V : \text{PR}(v) = 0.25$

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 3: Values for PageRank iterations with initial rank $\forall v \in V : \text{PR}(v) = 0.5$



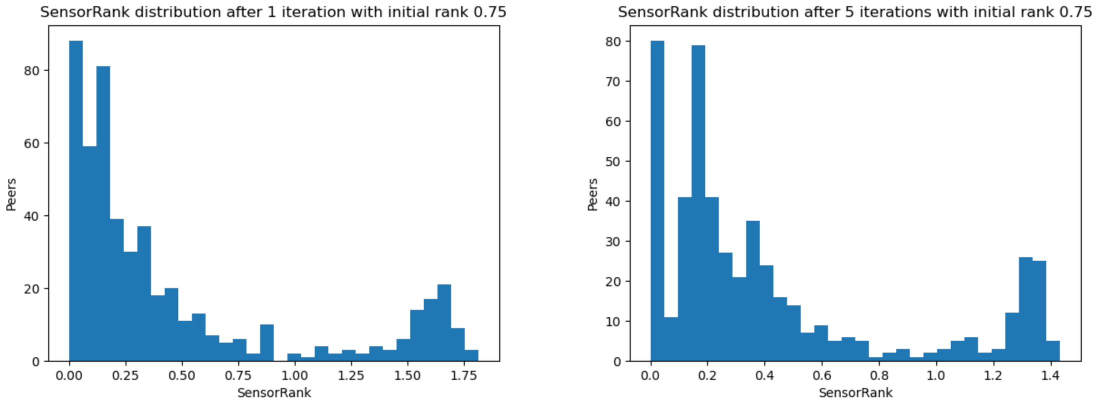
(a) Distribution after 1 iteration

(b) Distribution after 5 iterations

Figure 6: SensorRank distribution with initial rank $\forall v \in V : \text{PR}(v) = 0.5$

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 4: Values for PageRank iterations with initial rank $\forall v \in V : \text{PR}(v) = 0.75$



(a) Distribution after 1 iteration

(b) Distribution after 5 iterations

Figure 7: SensorRank distribution with initial rank $\forall v \in V : \text{PR}(v) = 0.75$

The distribution graphs in Figure 5, Figure 6 and Figure 7 show that the initial rank has no effect on the distribution, only on the actual numeric rank values.

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, those parameters do not matter.

On average, peers in the analyzed dataset have 223 successors over the whole week. Looking at the data in smaller buckets of one hour each, the average number of successors per peer is 90.

timeline with peers per bucket

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.

use better data?

By responding to peer list requests with plausible data, one can move make those metrics less suspicious, because it produces valid outgoing edges from the sensors. The hard part

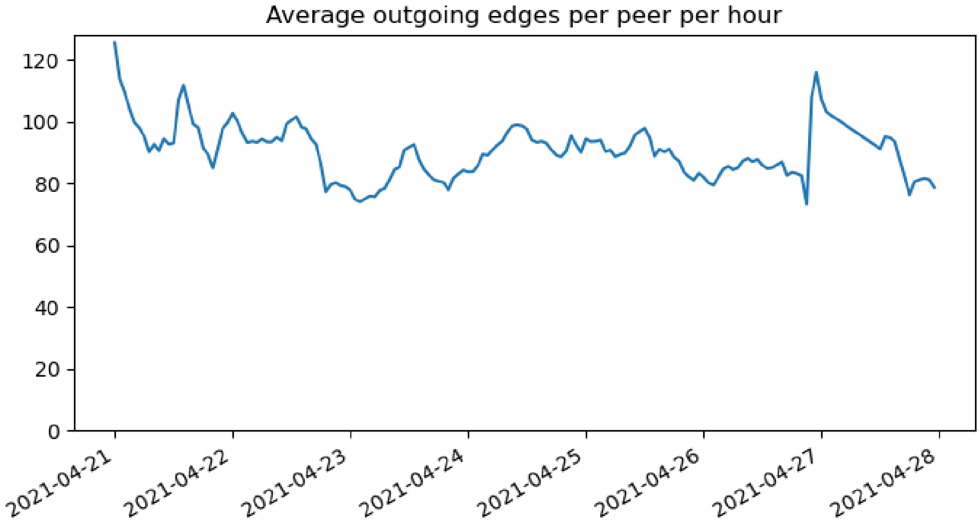


Figure 8: Average outgoing edges per peer per hour

is deciding which peers can be returned without actually supporting the network. The following candidates to place into the NL will be investigated:

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

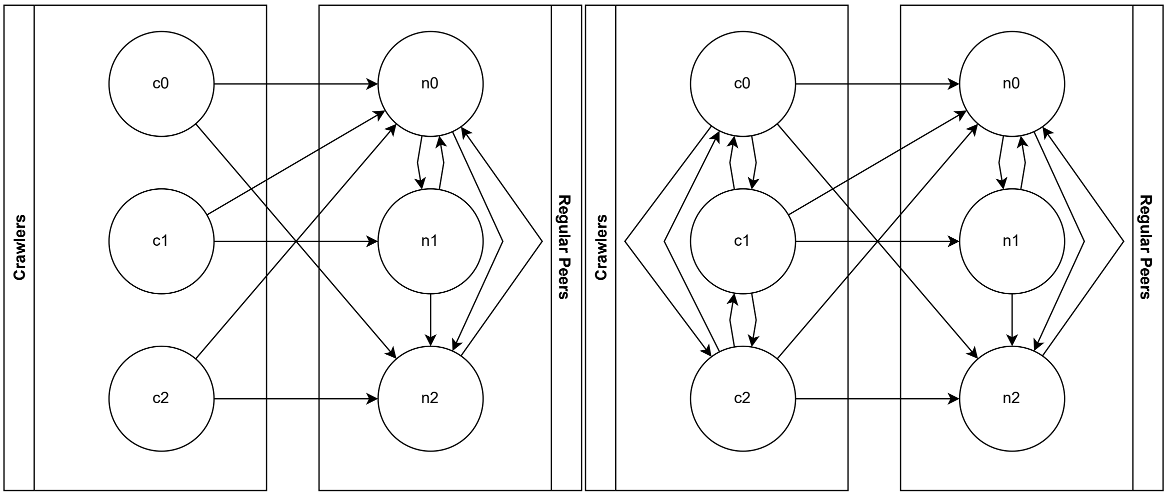
Knowledge of only 90 peers leaving due to IP rotation would be enough to make a crawler look average in Sality. This number will differ between different botnets, depending on implementation details and size of the network.

Adding edges from the known crawler to 90 random peers to simulate the described strategy gives the following rankings:

table, distribution with random edges

3.3.1 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 9: Differences in graph metrics

3.3.2 Use Churned Peers After IP Rotation

Churn describes the dynamics of peer participation of P2P systems, *e.g.* join and leave events [20]. 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 crawlers start to look like regular peers without actually supporting the network by relaying messages or propagating active peers.

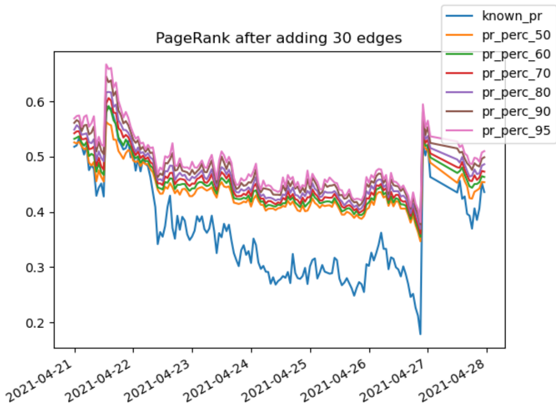
übergang

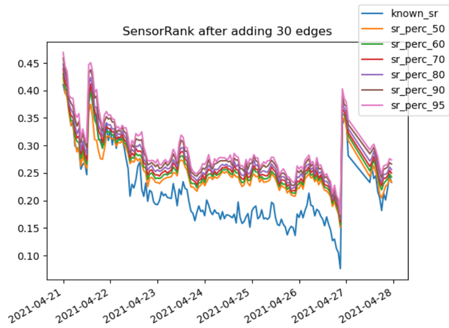
what is an AS

3.3.3 Peers Behind Carrier-Grade NAT

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

Experiments were performed, in which a fixed amount of random outgoing edges were added to the known sensor and the data was plotted:

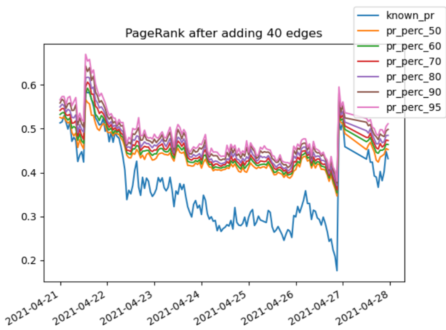


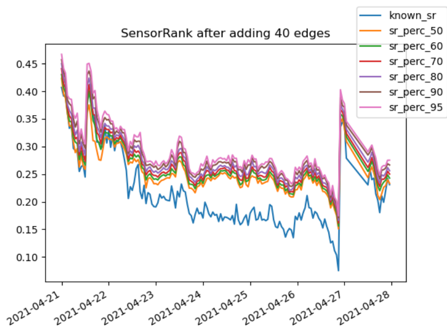


(a) PageRank with 30 additional edges

(b) SensorRank with 30 additional edges

Figure 10: Ranking with 30 additional edges

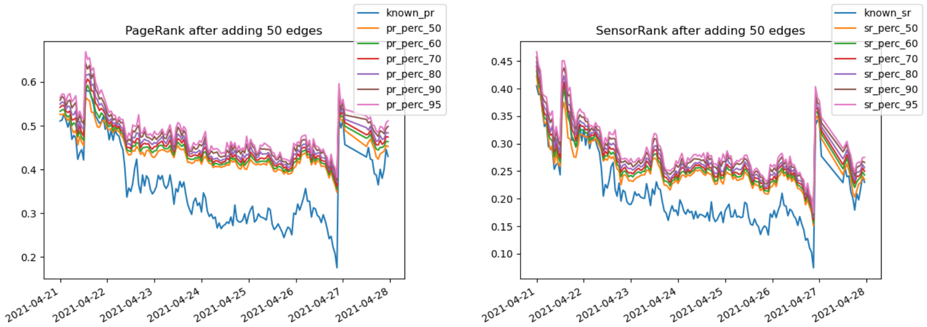




(a) PageRank with 40 additional edges

(b) SensorRank with 40 additional edges

Figure 11: Ranking with 40 additional edges



(a) PageRank with 50 additional edges

(b) SensorRank with 50 additional edges

Figure 12: Ranking with 50 additional edges

4 Implementation

Crawlers in BMS report to the backend using gRPC remote procedure calls (gRPCs)[2]. Both crawlers and the backend gRPC server are implemented using the Go[3] 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

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

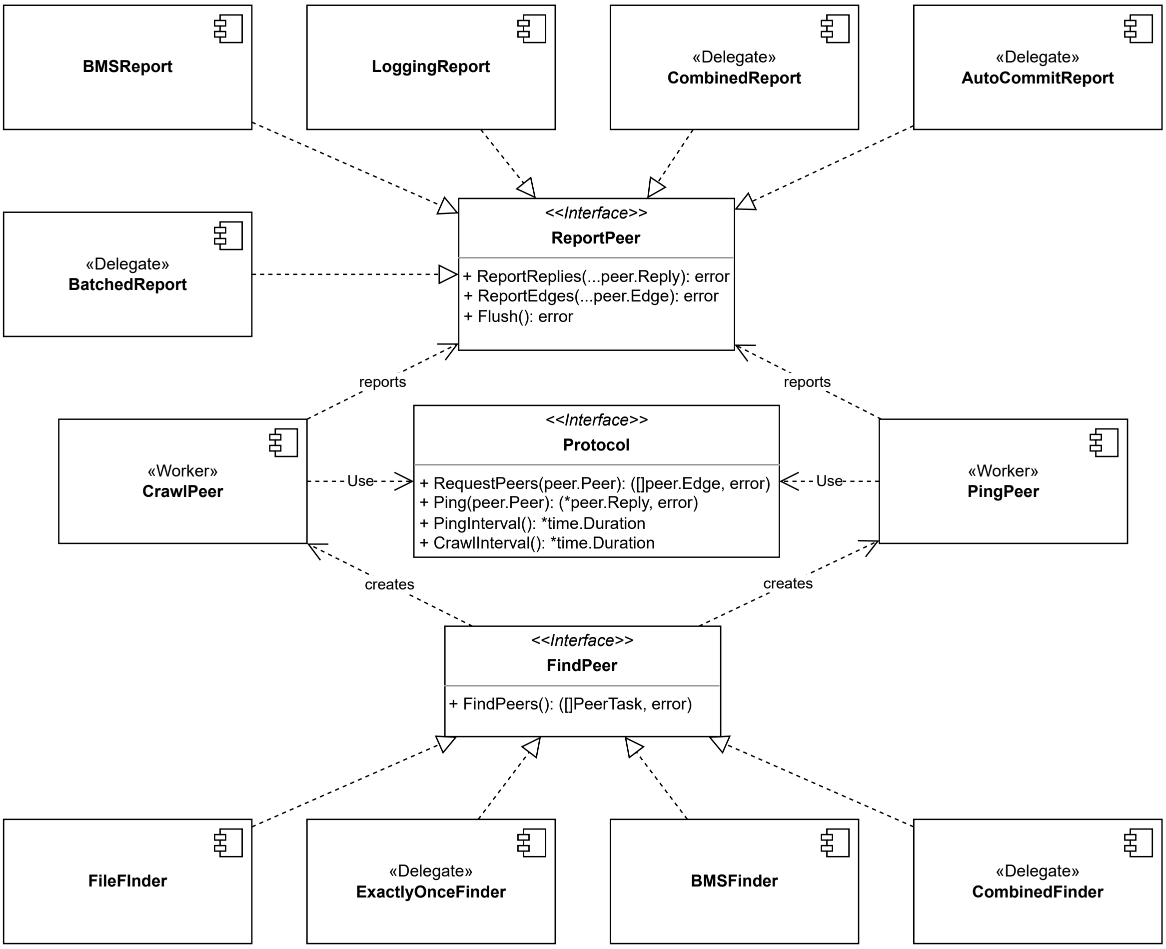


Figure 13: Architecture of the new crawler

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 textfile. `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 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 work is assigned to crawlers.

5 Conclusion, Lessons Learned

decide

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.

6 Further Work

Following this work, it should be possible to rewrite the existing crawlers to use 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.

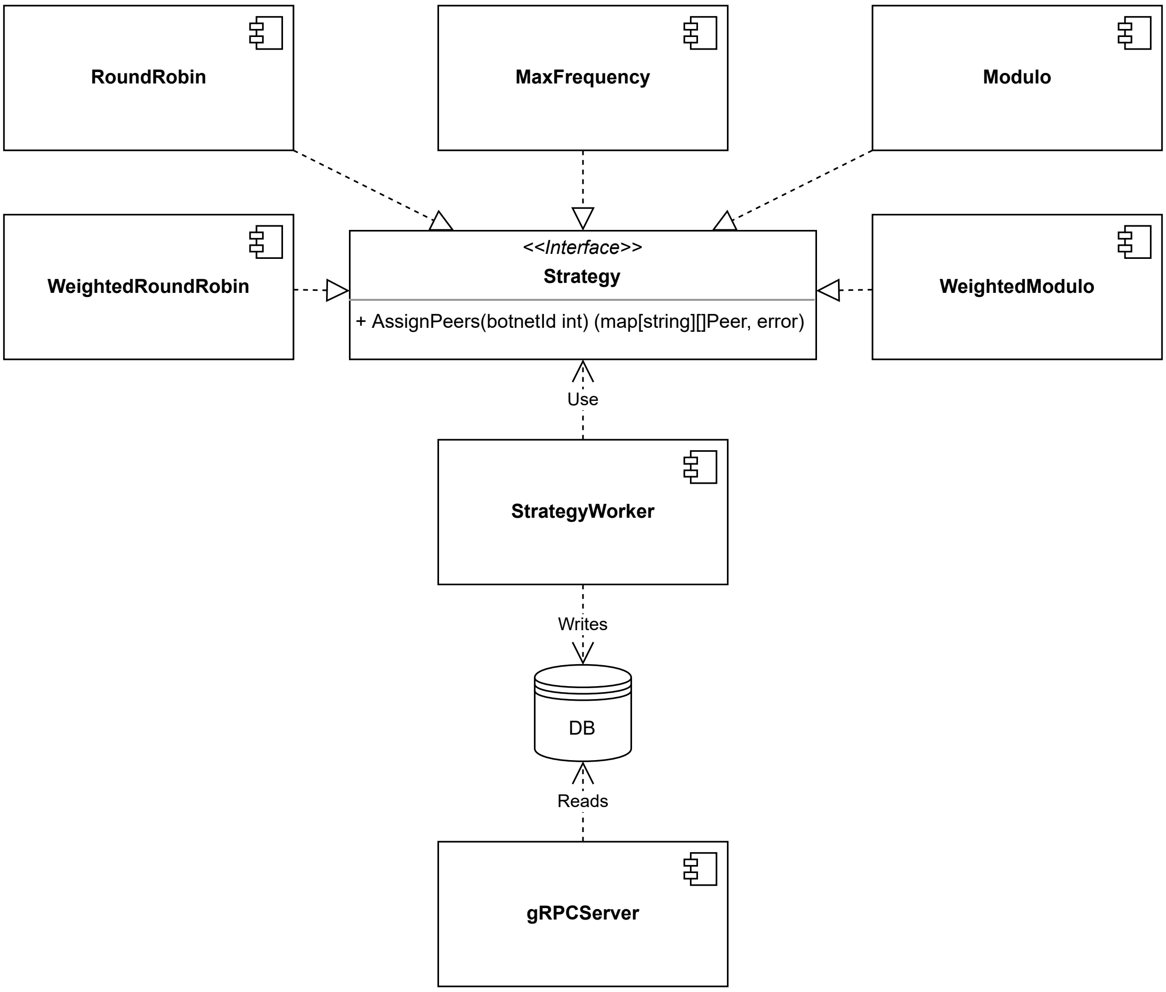


Figure 14: Architecture of the gRPC backend

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.

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

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. “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).

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Shankar Karuppayah et al. "SensorBuster: On Identifying Sensor Nodes in P2P Botnets". In: *Proceedings of the 12th International Conference on Availability, 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).

[12] 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).

[13] 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).

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

[15] 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).

[16] *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.

[17] 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).

[18] 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).

[19] 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).

[20] 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=1177080.1177105 (visited on 03/08/2022).

[21] 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.

[22] 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/ [redacted] (visited on 11/09/2021).

List of Figures

List of Tables

List of Acronyms

AS autonomous system . . . . 15, 21 f.

BMS Botnet Monitoring System . . . . 8 ff., 18, 24, 26

C2 command and control . . . . 4 f.

DDoS distributed denial of service 4, 6

gRPC gRPC remote procedure call 24, 26

IoT internet of things 4

IRC internet relay chat 4

MD5 Message-Digest Algorithm 5 14

P2P peer-to-peer 5–9, 16, 22, 26

SPOF single point of failure 5

WCC weakly connected component . . . . 8, 16, 21 f.

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