
OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

# MASTERTHESIS

Valentin Brandl

## Collaborative Crawling of Fully Distributed Botnets

March 15, 2022

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

TODO: abstract

# 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 bigger.

## 1.1 Motivation

The number of connected IoT devices is around 10 billion in 2021 and estimated to be constantly growing over the next years up to 25 billion in 2030 [10]. Many of these devices run on outdated software, don't receive any updates and don't follow general security best practices.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 $^{\text{MBit}}/_{\text{s}}$ or more, in 2020 it were already 95 % with more than 50 Mbit/s and 59 % with at least 1000 $^{\text{MBit}}/_{\text{s}}$ [4]. This makes them an attractive target for botmasters since they are easy to infect, always online, behind internet connections that are getting faster and faster, and due to their nature as small devices, often without any direct user interaction, an infection can go unnoticed for a long time. 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 $^{\text{TBit}}/_{\text{s}}$ of traffic [6].

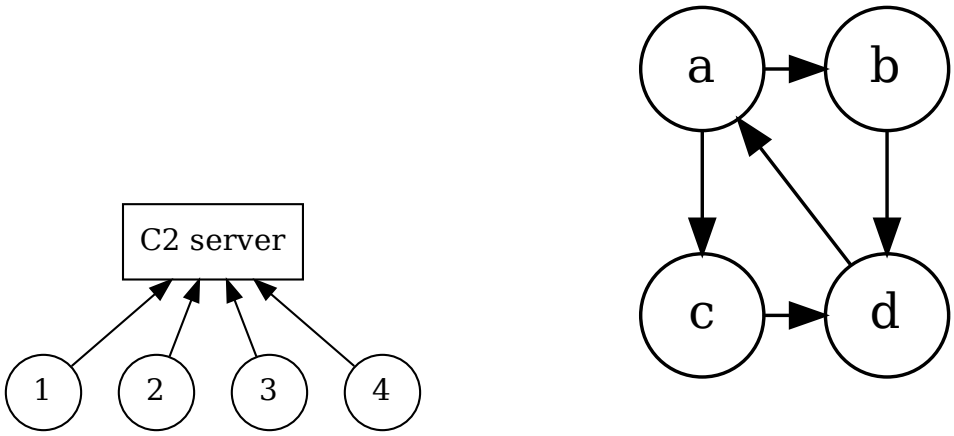
A botnet is a network of infected computers with some means of communication to control the infected systems. Classic botnets use one or more central coordinating hosts called command and control (C2) servers. These C2 servers could use any protocol from internet relay chat (IRC) over hypertext transfer protocol to Twitter [12] as communication channel with the infected hosts. The infected systems can be abused for a number of things, *e.g.* DDoS attacks, banking fraud, as proxies to hide the attacker's identity, send spam emails...

Analysing and shutting down a centralized botnet is comparatively easily since every bot knows the IP address, domain name, Twitter handle or IRC channel the C2 servers are using.

what is a bot? Infected systems. Malware. DGA, beispiele, tree vs graph

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 rooted 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 (DNS), where a large pool of IP addresses is used assigned randomly to the C2 domains to prevent IP based blacklisting [9].




(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

better image for p2p, really needed?

A number of botnet operations were shut down like this [8] and as the defenders upped their game, so did attackers—the idea of peer-to-peer (P2P) botnets came up. The idea is to build a decentralized network without single points of failure (SPOF) where the C2 servers are as shown in Figure 1b. In a P2P botnet, each node in the network knows a number of its neighbours and connects to those, each of these neighbours has a list of neighbours on his own, and so on.

too informal?

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

commands.

The damage produced by botnets has been constantly growing and there are many researchers and law enforcement agencies trying to shut down these operations. 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. Some of these countermeasures include deterrence, which limits the amount 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 neighbourhood lists, which invalidates the data collected by crawlers; and active retaliation like DDoS attacks against sensors or crawlers [1].

source for constantly growing, position in text

take-down? take down?

## 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 bots in the network and the set of edges $E$ describing the communication flow between bots.

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

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

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.

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

## 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 botmasters 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 [14]. Also getting access to the required datasets might not be possible for everyone.

no context

- Large scale network analysis (hard to differentiate from legitimate P2P traffic (*e.g.* BitTorrent), hard to get data, knowledge of some known bots required) [14]
- Heuristics: Same traffic patterns, same malicious behaviour

BotGrep (in zhang_bu

BotMiner (in zhang_bu

### 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 botmasters to detect and block the sensor nodes.

There are three subtypes of active detection:

1. Crawlers: recursively ask known bots for their neighbourhood 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

### 1.4 Detection Criteria

- P2P online time vs host online time
- neighbourhood lists
- no/few DNS lookups; instead direct lookups from routing tables

## 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 [5]. 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" [11]) to detect sensors and crawlers in a botnet, as described in "Sensor-Buster". Both 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 botmasters, by centralizing the coordination of the system's crawlers and sensors, thereby reducing the node's rank for specific graph metrics. The changes should allow the current sensors to use the new abstraction with as few changes as possible to the existing code.

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

The final result 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 "getNeighbourList" method in the protocol, that allows other peers to request a list of active nodes to connect to.

In the current implementation, each sensor 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 sensor is selected, so that the specific ranking algorithm doesn't calculate to a suspiciously high or low value. 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 sensor system should allow new sensors 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.

To validate the result, the old sensor implementation will be compared to the new system using different graph metrics.

maybe?

If time allows, Botnet Simulation Framework[2] will be used to simulate a botnet place sensors in the simulated network and measure the improvement achieved by the coordinated monitoring effort.

which botnet?

As a proof of concept, the coordinated monitoring approach will be implemented and deployed in the (Sality, Mirai, ...)? botnet.

## 2.1 Protocol Primitives

The coordination protocol must allow the following operations:

Testnet + testnet crawler erweitern um mit complete knowl-

### 2.1.1 Sensor to Backend

- `registerSensor(capabilities)`: Register new sensor with capabilities (which botnet, available bandwidth, ...). This is called periodically and used to determine which crawler is still active, when splitting the workload.
- `unreachable(targets)`:
- `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
}
```

bestehend session Mechanik verwenden/erwei

failedTries im backend statt eigenem nachrichtentyp: remove?

### 2.1.2 Backend to Sensor

# 3 Coordination Strategies

Let $C$ be the set of available crawlers.

[2] https://github.com/tklab-tud/BSF

## 3.1 Load Balancing

This strategy simply splits the work into even chunks and split it between the available crawlers. The following sharding conditions come to mind:

- Assuming IP addresses are evenly distributed and so are infections, take the IP address as an 32 Bit integer modulo $|C|$. Problem: reassignment if a crawler joins or leaves
- Maintain an internal counter/list of tasks for each available crawler and assign to the crawler with the most available resources. Easy reassignment
- Round Robin

## 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.

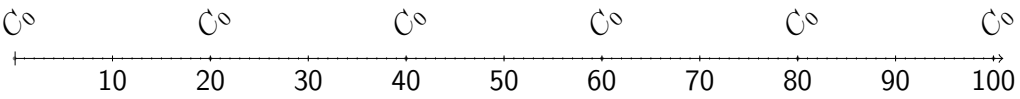



Figure 2: Timeline of crawler events as seen from a peer when crawled by a single crawler

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.

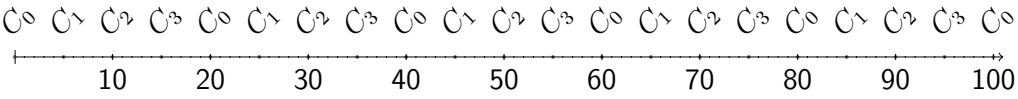
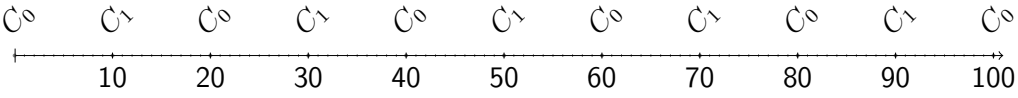



Figure 3: Timeline of crawler events as seen from a peer when crawled by multiple crawlers

As can be seen in Figure 3, each crawler $C_0$ to $C_3$ performs only $5\,\text{req}/100\text{s}$ while overall achieving $20\,\text{req}/100\text{s}$.

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 3, it is still possible to double the frequency over the limit.

sinnvoll?

## 3.3 Working Against Suspicious Graph Metrics

"SensorBuster: On Identifying Sensor Nodes in P2P Botnets" describes different graph metrics to find sensors in P2P botnets. 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 7b and Figure 8).

rank? deg+ - deg-?

With $v \in V$, succ($v$) being the set of successors of $v$ and pred($v$) being the set of predecessors of $v$, PageRank recursively is defined as [11]:

$$\mathrm{PR}(v) = \mathrm{dampingFactor} \times \sum_{p \in \mathrm{pred}(v)} \frac{\mathrm{PR}(p)}{|\mathrm{succ}(p)|} + \frac{1 - \mathrm{dampingFactor}}{|V|}$$

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 [11]. In our experiments on a snapshot of the Sality botnet exported from BMS over the span of, 3 iterations were enough to get distinct enough values to detect sensors and crawlers.

how often? experiments!

export timespan

| Iteration | Avg. PR | Crawler PR | Avg. SR | Crawler SR |
|---|---|---|---|---|
| 1 | wat? | wut? | wit? | wot? |
| 2 | wat? | wut? | wit? | wot? |
| 3 | wat? | wut? | wit? | wot? |
| 4 | wat? | wut? | wit? | wot? |
| 5 | wat? | wut? | wit? | wot? |

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

proper table formatting

| Iteration | Avg. PR | Crawler PR | Avg. SR | Crawler SR |
|---|---|---|---|---|
| 1 | wat? | wut? | wit? | wot? |
| 2 | wat? | wut? | wit? | wot? |
| 3 | wat? | wut? | wit? | wot? |
| 4 | wat? | wut? | wit? | wot? |
| 5 | wat? | wut? | wit? | wot? |

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

proper table formatting

| Iteration | Avg. PR | Crawler PR | Avg. SR | Crawler SR |
|---|---|---|---|---|
| 1 | wat? | wut? | wit? | wot? |
| 2 | wat? | wut? | wit? | wot? |
| 3 | wat? | wut? | wit? | wot? |
| 4 | wat? | wut? | wit? | wot? |
| 5 | wat? | wut? | wit? | wot? |

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

proper table for-mat-ting

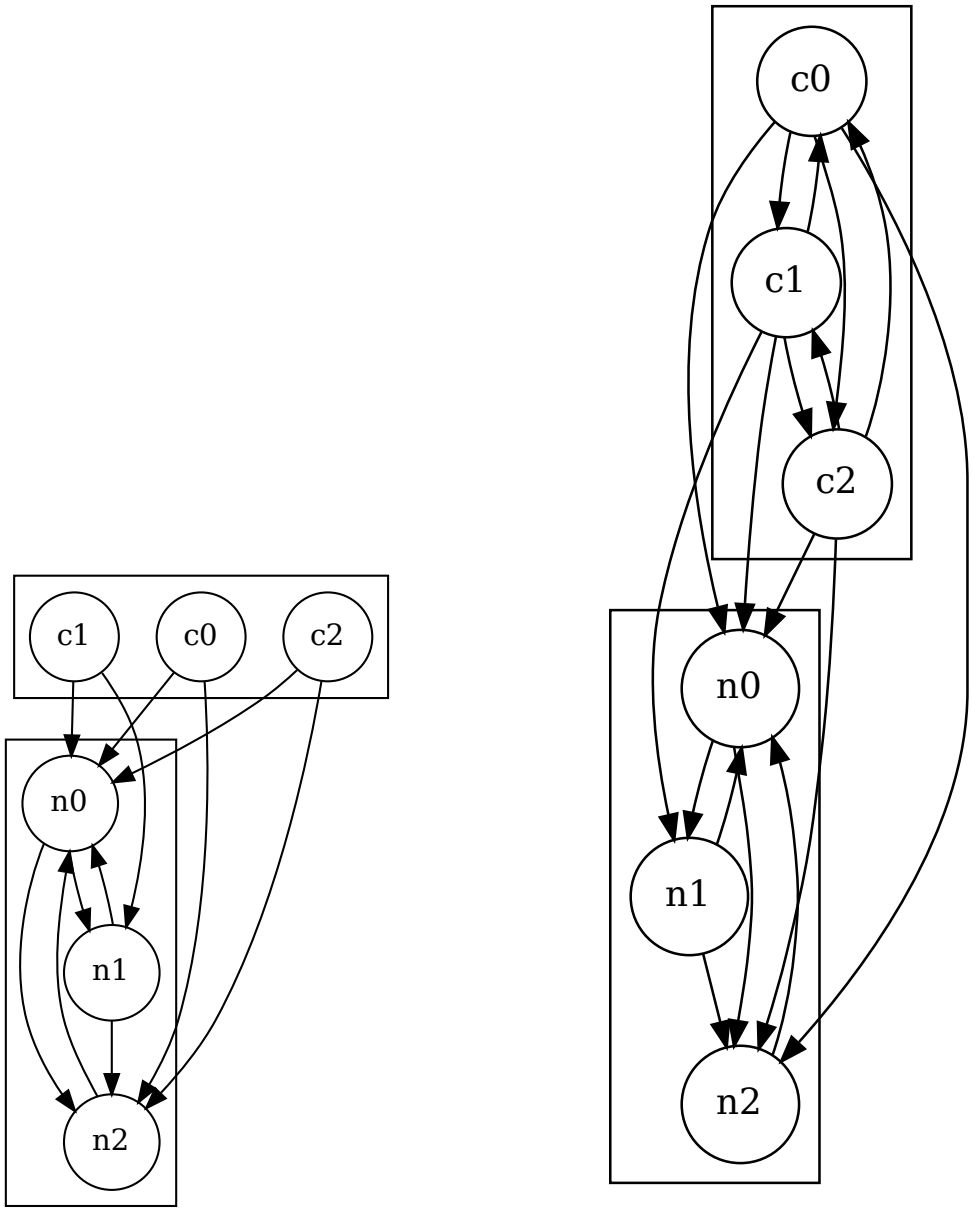
The dampingFactor describes the probability of a person visiting links on the web to continue doing so, when using PageRank to rank websites in search results. For simplicity—and since it is not required to model human behaviour for automated crawling and ranking—a dampingFactor of $1.0$ will be used,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

$$\mathrm{PR}(v) = \sum_{p \in \mathrm{pred}(v)} \frac{\mathrm{rank}(p)}{|\mathrm{succ}(p)|}$$

Based on this, SensorRank is defined as

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

percentag of botnet must be crawlers to make a signifi-cant change




(a) WCCs for independent crawlers

(b) WCCs for collaborated crawlers

Figure 7: Differences in graph metrics

these examples suck; chose better examples

Applying SensorRank PageRank once with an initial rank of $0.25$ once on the example

graphs above 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 |

Figure 8: Values for metrics from Figure 7 (a/b)

pagerank, sensorrank calculations, proper example graphs, proper table formatting

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.

### 3.3.1 Excurs: Churn

Churn describes the dynamics of peer participation of P2P systems, *e.g.* join and leave events [13]. Detecting if a peer just left the system, in combination with knowledge about autonomous systems (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 neighbourhood 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 neighbourhood 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.

big graphs, how many Kn to get significant?

for bigger (generated) graphs?

# 4 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 neighbourhood 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 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.

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

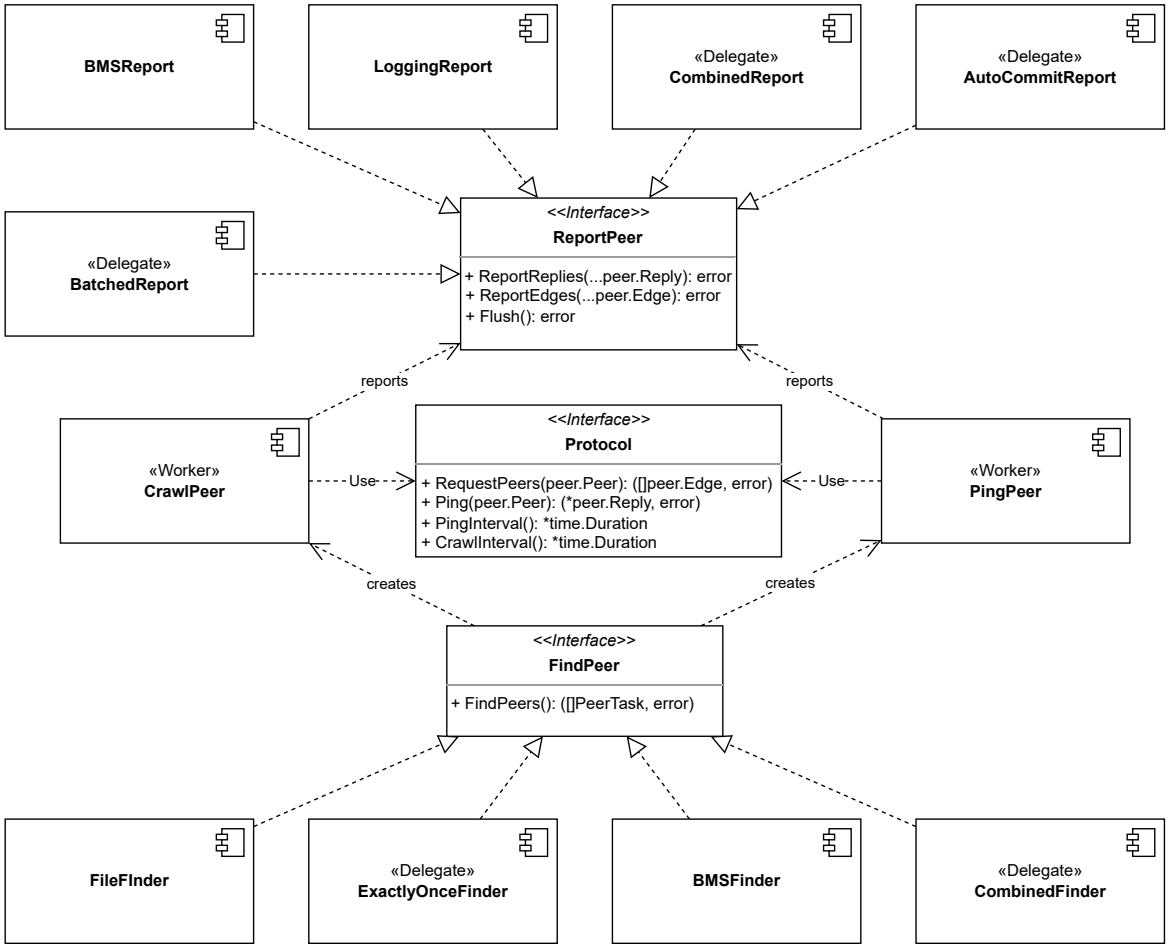



Figure 9: Architecture of the new crawler

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.

# 5 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 apply them to the new one.

# Acknowledgments

In the end, I would like to thank

- Prof. Dr. Christoph Skornia for being a helpful supervisor in this and 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 helpful ad-hoc discussions at any time of day for many years

# 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] 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.

[7] 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).

[8] 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).

[9] 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).

[10] *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.

[11] 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).

[12] 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).

[13] 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).

[14] 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).

# List of Figures

# Acronyms

**AS** autonomous system

**BMS** Botnet Monitoring System

**C2** command and control

**DDoS** distributed denial of service

**DNS** domain name system

**gRPC** gRPC remote procedure call

**IoT** internet of things

**IRC** internet relay chat

**P2P** peer-to-peer

**SPOF** single point of failure

**WCC** weakly connected component

**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 |