# SIMBIoTA++: Improved Similarity-based IoT Malware Detection

Levente Buttyán
*CrySyS Lab, BME and ELKH-BME Information Systems Research Group*
Budapest, Hungary
0000-0003-4233-2559

Roland Nagy
*CrySyS Lab*
*Budapest University of Technology and Economics*
Budapest, Hungary
0000-0003-2305-3271

Dorottya Papp
*CrySyS Lab*
*Budapest University of Technology and Economics*
Budapest, Hungary
0000-0002-9976-614X

***Abstract*—The Internet of Things is quickly developing and it enables exciting new applications, but at the same time, it also brings new security risks. In particular, embedded IoT devices may be subject to malware infection, undermining the trustworthiness of IoT systems. Malware detection on IoT devices is challenging due to their resource constraints, and antivirus tools developed for desktop PCs and servers are not directly applicable for them. In an earlier paper, we proposed a lightweight antivirus solution for IoT devices, called SIMBIoTA. In this paper, we propose SIMBIoTA++, an improvement on SIMBIoTA in terms of resource requirements. We also present a graph theory and measurement-based argument for selecting an appropriate similarity threshold, which is a key parameter in both SIMBIoTA and SIMBIoTA++.**

***Index Terms*—Internet of Things, malware detection, similarity hashing, graph theory, dominating set algorithm**

## I. Introduction

In recent years, millions of embedded devices have been connected to the Internet, and this transformed it to the *Internet of Things*, or IoT for short. On the one hand, IoT technologies drive new applications in many domains, enabling, for instance, the development of smart homes, smart cities, smart factories, precision agriculture, intelligent transportation systems, and personalized healthcare. On the other hand, IoT also creates new security risks, mainly because embedded IoT devices have not always been designed with security in mind, and therefore, they are notoriously easy to compromise, leading to untrustworthy IoT systems and representing danger to the Internet infrastructure as a whole. A specific problem, falling into the scope of our work, is that IoT devices are subject to infection by malware, by which attackers can take remote control over them, and build large IoT botnets that can be used for attacking any Internet-based service that our society relies on. Probably, the best known example for this phenomenon is the Mirai botnet [1] that consisted of hundreds of thousands of infected IoT devices, such as web cameras and WiFi routers, and that was used to launch a massive distributed denial-of-service attack on multiple Internet-based services in 2016[1]. But, of course, the IoT threat landscape contains many other malware families as well, such as Gafgyt, Tsunami, and DnsAmp [2].

As a response to the threat, researchers have started to work on malware detection solutions suitable for IoT systems [3]–[6]. The main challenges in this new domain are the scarcity of resources of embedded IoT devices and their architectural heterogeneity that render traditional antivirus products, developed for desktop computers and mobile devices, not immediately applicable in IoT systems. One popular approach is to outsource malware detection into the *cloud* [7]. This essentially means that IoT devices submit new files that they encounter to a cloud-based malware detection service, where they are inspected and their fate is decided. This approach has obvious advantages: the burden of computation is removed from the resource-constrained IoT devices and all kinds of malware detection methods can easily be introduced, including new techniques, such as those based on machine learning [8]. But cloud-based malware detection also has disadvantages: IoT devices must be able to reach the malware detection service at any time, submitting files has communication cost, and detection is delayed. In some application domains, such as real-time cyber-physical systems, these disadvantages may not be tolerable, and there is a need for performing malware detection on the IoT devices themselves in an efficient and effective manner.

We have recently proposed a lightweight, yet effective, malware detection approach, called SIMBIoTA, in [9]. SIMBIoTA detects malware based on binary similarity, and it can be used on resource-constrained IoT devices. Its operation is similar to that of traditional signature-based antivirus solutions, but it uses TLSH hash values of known malware instead of raw binary signatures for detection purposes. TLSH [10] is a similarity hash algorithm, which means that similar inputs result in similar TLSH hash values, and SIMBIoTA takes advantage of this feature. When using SIMBIoTA, embedded IoT devices store only a few TLSH hash values of known

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

[1]https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet

malware, and they compare the TLSH hash value of a new file to these stored hashes. If the TLSH hash of an unknown file is similar to that of a known malware, the unknown file is detected as malware. We showed in [9] that SIMBIoTA has lightweight storage, computation, and bandwidth requirements, while it has remarkable detection capabilities: according to the experiments we performed, it achieved a true positive detection rate of around 90%, even for previously unseen malware, and its false positive detection rate was 0%.

A key design objective of SIMBIoTA was to keep the amount of storage needed to store TLSH hash values on IoT devices low. For this purpose, we proposed to compute a *dominating set* of the graph of TLSH values known to the antivirus provider, and send only the TLSH hashes in this dominating set to the IoT devices as a sort of "signature database" that can be used for malware detection. The size of the dominating set, hence, determines the storage requirements for the IoT devices, and we want to keep it small. In this paper, we propose SIMBIoTA++, a new dominating set computation algorithm, which improves upon the dominating set computation algorithm of SIMBIoTA: it produces smaller dominating sets, while it preserves almost the same computation time.

In addition, we report the results of our measurement for determining the TLSH difference threshold below which two files can be considered similar. This threshold was set to the specific value of 40 in SIMBIoTA, based on heuristics and manual verification of a sampled dataset. In this paper, we show stronger evidences than heuristics that the value 40 is indeed a good threshold. This validates the results of SIMBIoTA and puts SIMBIoTA++ on a solid basis.

In the following, we give an overview of the operation of SIMBIoTA and its dominating set computation algorithm in Section II, we present the new dominating set computation algorithm of SIMBIoTA++ and compare its performance to that of SIMBIoTA in Section III, we present the results of the measurement we performed for determining an appropriate TLSH difference threshold to be used in SIMBIoTA and SIMBIoTA++ in Section IV, and finally, we conclude the paper in Section V.

## II. SIMBIoTA

In this section, we first give a brief overview of the operation of SIMBIoTA and we present its iterative dominating set update algorithm. Then we make a few observations and present some measurement results that suggest that SIMBIoTA's algorithm produces larger dominating sets than an algorithm based on a well-known, greedy dominating set construction method. At the same time, this second algorithm has a larger computation time. These observations motivated us to look for a better iterative dominating set update algorithm that produces dominating sets as small as the algorithm based on the greedy construction method, while it preserves the fast computation time of SIMBIoTA.

### A. Operation

SIMBIoTA relies on a large malware database maintained by an antivirus service provider. This malware database is assumed to be continuously updated with samples obtained from a so called *intelligence network*, which collects malware samples from public and private feeds, honeypot farms, and cloud-based malware analysis platforms. The service provider computes the TLSH hash values of the samples in its database, and pushes a subset of these TLSH hashes to the client-side SIMBIoTA component running on the embedded IoT devices, where a light-weight algorithm uses them to detect malware based on binary similarity: as said before, if the TLSH hash of an unknown file is similar to any of the TLSH hashes pushed to the IoT device from the service provider, then the unknown file is detected as malware. As the database at the service provider is continuously updated, the subset of TLSH hashes to be pushed to the IoT devices needs to be regularly re-computed and re-sent too. We may assume, for instance, that IoT devices are updated with new hash values once every week.

On the one hand, due to the resource constraints of IoT devices, we want that the subset of TLSH hashes provided to them by the service provider be small, and on the other hand, it must be sufficiently large, such that it allows for the detection of *all* malware samples known to the service provider, and also potentially unknown ones. For this reason, in SIMBIoTA, the subset of TLSH values pushed to the IoT devices is computed as a *dominating set* of the *similarity graph* constructed from the TLSH hash values of the samples known to the service provider. More specifically, the vertices of this similarity graph are the TLSH hash values of the known samples, and two vertices are connected if the TLSH difference of the hash values corresponding to the vertices is below a given difference threshold. Furthermore, the dominating set $D$ of a graph $G(V, E)$ is a subset of $V$ such that every vertex of the graph is either in $D$ or a neighbor of a vertex in $D$. Hence, this construction ensures that all known samples are similar to those represented by the dominating set pushed to the IoT devices.

SIMBIoTA computes a dominating set of the similarity graph $G$ of the known samples at the service provider in an iterative manner by updating the current dominating set when a new sample is added to the graph using Algorithm 1.

Essentially, when a new node $u$ (i.e., the TLSH value of a new sample) is added to the graph $G$, the set of vertices of $G$ is extended with $u$ and the set of edges of $G$ is extended with the edges between $u$ and all other nodes in $G$ that represent samples similar to $u$. In addition, if the new node $u$ is already dominated by the current dominating set $D$, then $D$ is not changed; otherwise, $u$ is added to $D$.

### B. Observations

The iterative dominating set update algorithm of SIMBIoTA is very efficient, but it extends the current dominating set in each iteration too carelessly. To demonstrate this, let us consider a well-known greedy algorithm for computing a

**Algorithm 1** SIMBIoTA dominating set update algorithm

```
Input: current graph $G$, a dominating set $D$ of $G$, a new node
    $u$ to be added to $G$
Output: new graph $G'$ (i.e., $G$ extended with $u$), a dominating
    set $D'$ of $G'$
1: let $G'$ be $G$ extended with $u$
2: if $u$ is dominated by $D$ then
3:     let $D'$ be $D$
4: else
5:     let $D'$ be $D \cup \{u\}$
6: end if
7: return $G'$, $D'$
```

dominating set[2] presented as Algorithm 2. This algorithm starts with an empty set $D$, which is then extended step-by-step, by always adding the node with the largest number of neighbors not yet dominated by $D$. The algorithm stops when all nodes of $G$ become dominated by $D$.

**Algorithm 2** Greedy dominating set computing algorithm

```
Input: graph $G(V, E)$
Output: a dominating set $D$ of $G$
1: let $D$ be $\emptyset$
2: while there exist nodes in $G$ not dominated by $D$ do
3:     choose a node $v$ from $V \setminus D$ with the largest number
       of neighbors not yet dominated by $D$
4:     let $D$ be $D \cup \{v\}$
5: end while
6: return $D$
```

Algorithm 2 could be used by the antivirus service provider to re-compute the dominating set of the similarity graph $G$ of the known malware samples when a new node $u$ (i.e., a TLSH hash of a new sample) is added to $G$, as shown in Algorithm 3.

**Algorithm 3** Global dominating set update algorithm

```
Input: current graph $G$, a new node $u$ to be added to $G$
Output: new graph $G'$ (i.e., $G$ extended with $u$), a dominating
    set $D'$ of $G'$
1: let $G'$ be $G$ extended with $u$
2: call Algorithm 2 on $G'$ to compute $D'$
3: return $G'$, $D'$
```

We call Algorithm 3 "global", because it always considers the entire graph $G'$, hence global information, when computing $D'$, in contrast to Algorithm 1, which computes $D'$ by looking at only $D$. One may expect that using global information results in better performance in terms of the size of the computed dominating set, whereas running the computation on the entire graph should be much slower. These intuitive expectations are confirmed by our measurements.

[2]We note that finding the smallest dominating set of a graph $G$ is known to be an NP-complete problem. Hence, we are interested in efficient heuristic algorithms that find small, although maybe not the smallest, dominating sets. The greedy algorithm that we consider in this paper is known to compute an $\ln(\Delta + 2)$-approximation of the optimal dominating set of $G$, where $\Delta$ is the maximal degree of $G$ [11].

We performed all measurement using the same, publicly available[3] data set that we used for the evaluation of SIMBIoTA. This data set consists of 29,209 malicious ARM samples and 18,715 malicious MIPS samples with metadata, such as the dates the samples were first seen in the wild (i.e., submitted to VirusTotal[4]). We also followed the same experiment design as used for SIMBIoTA in [9]. The timeline of the experiment is between January 1st, 2018 and September 15th, 2019, divided into weeks. The malware samples are organized into weekly batches based on the date they were first seen, and each weekly batch is further divided into two groups. The first group, which contains 10% of that weekly batch's samples and is called the *intelligence part*, is made available to the antivirus service provider for processing (i.e., including in its similarity graph of the known samples). The second group, called the *wilderness part*, contains 90% of that weekly samples, and it is assumed to exist only in the wild and is never revealed to the service provider. (The wilderness parts of the weekly batches are used to evaluate the true positive rate of the detection algorithms, presented later).

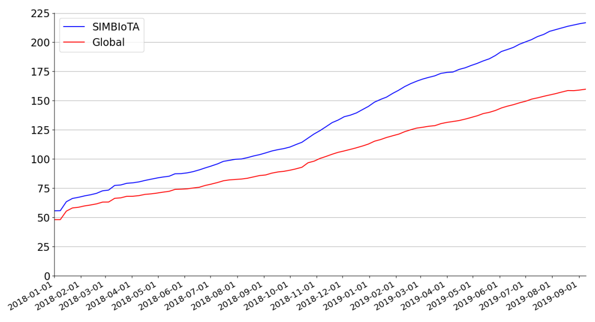

Fig. 1. Average size of the dominating set computed on each week by Algorithm 1 (SIMBIoTA) and Algorithm 3 (Global)

Figures 1 and 2 show a comparison between the dominating set sizes and the running times, respectively, of Algorithms 1 and 3. As it can be seen in the figures, Algorithm 3 (Global) produces smaller dominating sets than Algorithm 1 (SIMBIoTA), but its running time becomes much larger as the size of the graph at the antivirus service provider increases. Considering these results, one may ask the following question: *Can we design a dominating set update algorithm that produces as small dominating sets as Algortihm 3 does, while running as fast as Algorithm 1 runs?* Our affirmative answer to this question is SIMBIoTA++.

## III. SIMBIoTA++

In this section, we present SIMBIoTA++, an improved dominating set update algorithm, and its comparison – in

[3]https://github.com/CrySyS/cube-maliot-2021
[4]https://www.virustotal.com

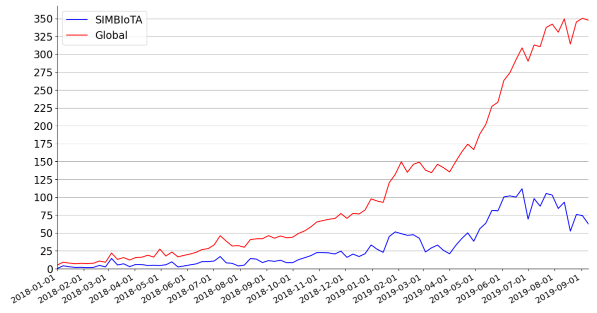

Fig. 2. Running time (in milliseconds) of Algorithm 1 (SIMBIoTA) and Algorithm 3 (Global) on each week

terms of dominating set size produced, running time, and true positive detection rate – to SIMBIoTA and the global update algorithm introduced in the previous section.

## A. A new dominating set update algorithm

Our design of the dominating set update algorithm of SIMBIoTA++ is based on the observation that in case of SIMBIoTA, if a node becomes part of the dominating set, then it remains part of it forever, whereas this is not the case for the global algorithm that always recomputes the dominating set from scratch. When adding a new node $u$ to the graph $G$, our new algorithm may remove a node from the current dominating set $D$, if it can be replaced by another node such that, with this replacement, $G$ extended with $u$ would be dominated. The algorithm is presented in more details as Algorithm 4.

As in case of SIMBIoTA, if the new node $u$ is already dominated by the current dominating set $D$, then $D$ is not changed (lines 2–4). Otherwise, we check for every node $w$ in $D$, if it can be replaced by another node $w'$, such that $w'$ would dominate all the nodes $F_w$ that may become non-dominated by the removal of $w$ from $D$, as well as the new node $u$ (lines 7–9). If such a node $w'$ can indeed be found, then we replace $w$ by $w'$ in $D$ (line 10) and stop (line 11). If such a node $w'$ cannot be found, we simply add $u$ to $D$ (line 14). Note that it is sufficient to look for $w'$ among the neighbors of $u$ in $G'$, because $w'$ must dominate $u$ as it is not dominated by $D$ itself, and hence by $D \setminus \{w\}$ neither.

## B. Evaluation

We compared the performance of Algorithm 4 to the previously discussed algorithms using the exact same experiment setup and methodology as before. The results can be seen in Figures 3, 4, and 5.

As it can be clearly seen, our new dominating set update algorithm produces dominating sets with sizes close to those produced by the global algorithm, while at the same time, its running times are extremely close to those of SIMBIoTA. In other words, SIMBIoTA++ represents the best of both worlds that we had before: it produces small dominating sets like an

**Algorithm 4** SIMBIoTA++ dominating set update algorithm

```
Input: current graph $G$, a dominating set $D$ of $G$, a new node
    $u$ to be added to $G$
Output: new graph $G'$ (i.e., $G$ extended with $u$), a dominating
    set $D'$ of $G'$
1:  let $G'$ be $G$ extended with $u$
2:  if $u$ is dominated by $D$ then
3:      let $D'$ be $D$
4:      return $G'$, $D'$
5:  else
6:      let $N_u$ be the set of neighbors of $u$ in $G'$
7:      for every node $w$ in $D$ do
8:          let $F_w$ be the set of nodes in $G'$ that would not be
            dominated if we removed $w$ from $D$
9:          if there exists a node $w'$ in $F_w \cap N_u$ that dominates
            all nodes in $F_w$ then
10:             let $D'$ be $D \cup \{w'\} \setminus \{w\}$
11:             return $G'$, $D'$
12:         end if
13:     end for
14:     let $D'$ be $D \cup \{u\}$
15:     return $G'$, $D'$
16: end if
```

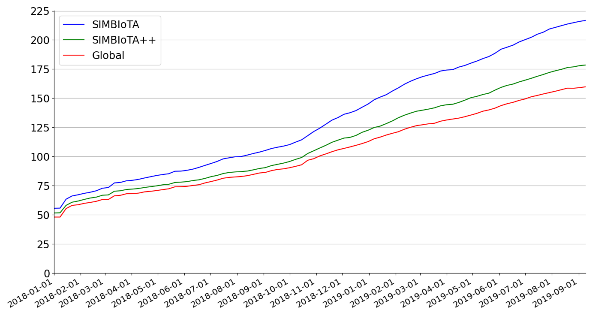

Fig. 3. Average size of the dominating set computed on each week by Algorithm 1 (SIMBIoTA), Algorithm 3 (Global), and Algorithm 4 (SIMBIoTA++)

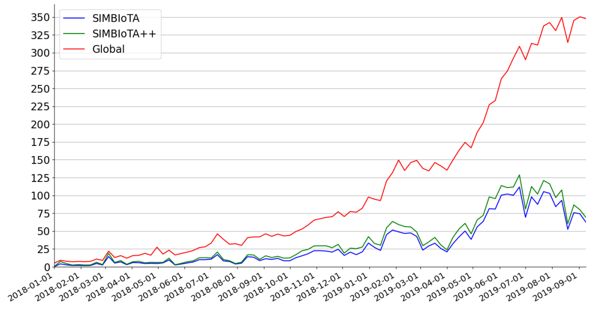

Fig. 4. Running time (in milliseconds) of Algorithm 1 (SIMBIoTA), Algorithm 3 (Global), and Algorithm 4 (SIMBIoTA++) on each week

algorithm that always takes the entire graph into account in the computation, but it has a much shorter running time, close to the simple update algorithm of SIMBIoTA. In addition, the true positive malware detection rates in case of using any of these algorithms to produce the dominating set are very close to each other, as shown in Figure 5. We also measured the false positive detection rate and obtained 0% just like in the case of SIMBIoTA. All these results clearly show the superiority of SIMBIoTA++ over SIMBIoTA.

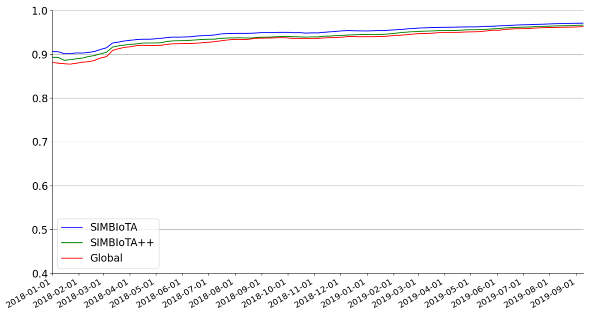

Fig. 5. Averge true positive detection rate on each week when the dominating set is computed by Algorithm 1 (SIMBIoTA), Algorithm 3 (Global), and Algorithm 4 (SIMBIoTA++)

## IV. Similarity threshold selection

Both SIMBIoTA and SIMBIoTA++ relies on computing a dominating set on the similarity graph that represents the malware samples known to the antivirus service provider and their similarity at the binary level (see Subsection II-A for more details). In [9] and in this paper, we used the similarity threshold value 40 to construct the similarity graphs. However, the selection of this value was based on heuristics and manual verification in [9], whereas now we show that the value 40 is indeed a reasonable threshold in a more trustworthy manner.

In order to gain some intuition, in Figures 6 and 7, we visualized the similarity graphs of 2000 ARM and 2000 MIPS malware samples, respectively, chosen randomly from our dataset, using the Gephi[5] graph visualization tool. We used TLSH difference 40 as the similarity threshold. We also colored the nodes of the graphs based on the malware labels obtained from VirusTotal and processed by the AVClass[6] tool. As it can be seen in the figures, the nodes of the graphs appear to be clustered, and the clusters resulting from the similarity relationship match pretty well the coloring representing the malware labels.

Now let us consider the effect of changing the similarity threshold. A lower value would mean that fewer pairs of nodes would be connected in the graph, as a smaller TLSH difference would be required between any two samples to be deemed similar. At extremely low (i.e., close to 0) similarity

[5]https://gephi.org

[6]https://github.com/malicialab/avclass

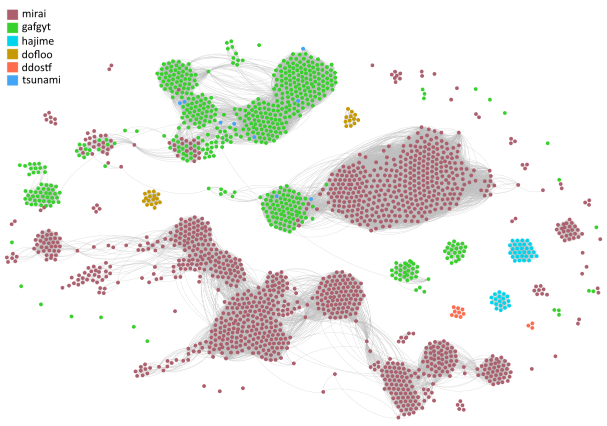

Fig. 6. Similarity graph of 2000 randomly chosen ARM malware samples with TLSH difference threshold 40

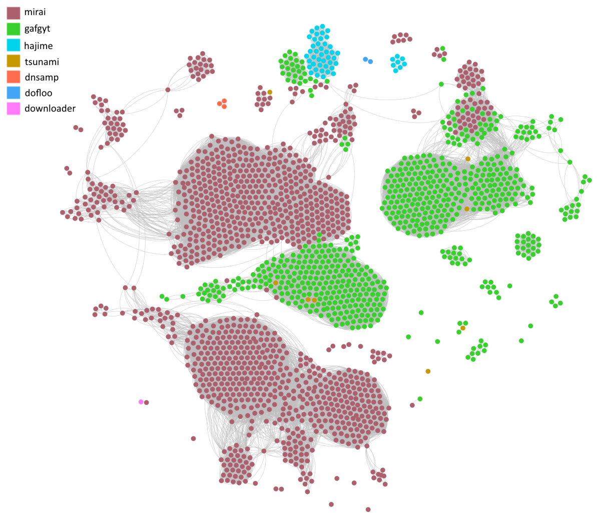

Fig. 7. Similarity graph of 2000 randomly chosen MIPS malware samples with TLSH difference threshold 40

threshold values, the graph would “fall apart” to disjoint nodes, and it would not exhibit any level of clusteredness at all. In contrast, increasing the similarity threshold value would result in more and more edges in the graph. For very high values (e.g., close to and above 100) even non-similar samples would be considered similar, and eventually, the graph would become an almost fully connected click. Such a graph does not exhibit any clusteredness either. Hence, we may expect that if we can somehow measure the level of clusteredness, then we can observe a maximum between the two extremes of TLSH difference 0 and 100.

Fortunately, there exist graph metrics that can be used to characterize the clusteredness of a graph [12]. A well-known such metric is the *clustering coefficient* of the graph, which is defined as follows. For unweighted graphs, the clustering of a node $u$ is defined as the fraction of possible triangles through node $u$ that exist:

$$c_u = \frac{2\,T(u)}{deg(u)(deg(u)-1)} \tag{1}$$

where $T(u)$ is the number of existing triangles through node $u$ and $deg(u)$ is the degree of $u$. The clustering coefficient of a graph $G(V,E)$ is then defined as the average clustering of its nodes:

$$C = \frac{1}{|V|}\sum_{v\in V} c_v \tag{2}$$

Using this definition, we measured the clustering coefficient of the similarity graphs resulting from our dataset when using different similarity thresholds. The result is plotted in Figure 8. It can be clearly seen that the curve representing the clustering coefficient as a function of the similarity threshold has a maximum around the threshold value 40. This clearly shows that TLSH difference 40 is indeed a meaningful choice for the similarity threshold.

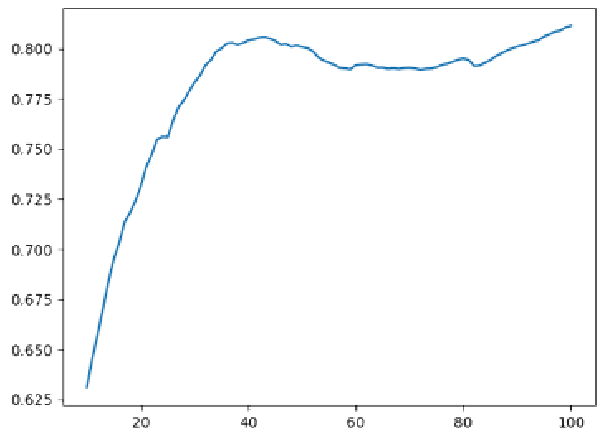

Fig. 8. Clustering coefficient as a function of the similarity threshold.

## V. Conclusion

In this paper, we presented SIMBIoTA++, an improvement on SIMBIoTA, which is a lightweight antivirus solution designed for resource-constrained IoT devices. Both SIMBIoTA and SIMBIoTA++ relies on measuring similarity between binary files for malware detection. They both use TLSH, which is a similarity hash function, and TLSH difference, which is a similarity metric computed from TLSH hash values. To detect malware, IoT devices are provided with a set of TLSH hash values of known malware, and they compare the TLSH hash of a new file to these hashes. An unknown file is detected as malware if its TLSH hash is similar to any of the pre-loaded TLSH hashes of known malware.

The size of the set of the TLSH hashes pushed to the IoT devices determines their storage requirements, therefore, keeping that size small is important. The set of TLSH hashes to be pushed to the IoT devices is actually computed as a dominating set of a similarity graph representing all malware known to an antivirus service provider. The improvement we proposed in this paper is a new dominating set update algorithm that results in smaller sets of TLSH hash values to be pushed to the IoT devices than in the case of SIMBIoTA, while the computation time remains almost the same.

In addition, we reported on a measurement experiment that proved that the TLSH difference threshold of 40, which we used as a similarity threshold when constructing the similarity graphs in case of both SIMBIoTA and SIMBIoTA++, is indeed a meaningful threshold value, because it maximizes the clustering coefficient of the similarity graph of the malware samples.

## Acknowledgment

The authors are thankful to Iuhos Zoltán for assisting in implementing the experiments presented in this paper and in visualizing the results, including the similarity graphs. In addition, the authors are grateful to Máté Horváth for his observations on an earlier version of the SIMBIoTA++ dominating set update algorithm that helped to improve it further.

## References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Aug. 2017, pp. 1093–1110.

[2] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bigle, and D. Balzarotti, "The tangled genealogy of IoT malware," in *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[3] M. F. B. Abbas and T. Srikanthan, "Low-complexity signature-based malware detection for IoT devices," in *Applications and Techniques in Information Security*, L. Batten, D. S. Kim, X. Zhang, and G. Li, Eds. Springer Singapore, 2017, pp. 181–189.

[4] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, "Analyzing and detecting emerging Internet of Things malware: A graph-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8977–8988, 2019.

[5] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, "HADES-IoT: A practical host-based anomaly detection system for IoT devices," in *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. New York, NY, USA: Association for Computing Machinery, 2019, p. 479–484.

[6] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of IoT malware and detection methods based on static features," *ICT Express*, vol. 6, no. 4, pp. 280–286, Dec. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2405959520300503

[7] H. Sun, X. Wang, R. Buyya, and J. Su, "CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained Internet of Things (IoT) devices," *Software: Practice and Experience*, vol. 47, no. 3, pp. 421–441, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2420

[8] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123 – 147, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404818303808

[9] C. Tamás, D. Papp, and L. Buttyán, "SIMBIoTA: Similarity-based malware detection on IoT devices," in *Proceedings of the 6th International Conference on Internet of Things, Big Data and Security (IoTBDS)*, INSTICC. SciTePress, 2021, pp. 58–69.

[10] J. Oliver, C. Cheng, and Y. Chen, "TLSH – a locality sensitive hash," in *Fourth IEEE Cybercrime and Trustworthy Computing Workshop*. IEEE, Nov. 2013, pp. 7–13. [Online]. Available: http://ieeexplore.ieee.org/document/6754635/

[11] T. W. Haynes, S. Hedetniemi, and P. Slater, *Fundamentals of Domination in Graphs*. CRC Press, 1988.

[12] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, pp. 75–174, 2010.