LiMNet: Early-Stage Detection of IoT Botnets with Lightweight Memory Networks[⋆]

Lodovico Giaretta[1][0000−0002−0223−8907], Ahmed Lekssays[2][0000−0001−5783−8638], Barbara Carminati[2][0000−0002−7502−4731], Elena Ferrari[2][0000−0002−7312−6769], and Šarūnas Girdzijauskas[1][0000−0003−4516−7317]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
{lodovico,sarunasg}@kth.se
[2] University of Insubria, Varese, Italy
{alekssays,barbara.carminati,elena.ferrari}@uninsubria.it

Abstract. IoT devices have been growing exponentially in the last few years. This growth makes them an attractive target for attackers due to their low computational power and limited security features. Attackers use IoT botnets as an instrument to perform DDoS attacks which caused major disruptions of Internet services in the last decade. While many works have tackled the task of detecting botnet attacks, only a few have considered early-stage detection of these botnets during their propagation phase.
While previous approaches analyze each network packet individually to predict its maliciousness, we propose a novel deep learning model called LiMNet (**Li**ghtweight **M**emory **Net**work), which uses an internal memory component to capture the behaviour of each IoT device over time. This memory incorporates both packet features and behaviour of the peer devices. With this information, LiMNet achieves almost maximum AUROC classification scores, between 98.8% and 99.7%, with a 14% improvement over state of the art. LiMNet is also lightweight, performing inference almost 8 times faster than previous approaches.

Keywords: IoT · Botnet Detection · Memory Networks · Recurrent Networks.

1 Introduction

IoT devices are gaining popularity thanks to their usefulness in gathering and processing data. They have become an important pillar in Industry 4.0, which led to an exponential growth in terms of IoT deployments worldwide. As a result, the number of IoT connections is expected to reach 83 billions by 2024.[3]

⋆ This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813162. The content of this paper reflects the views only of their author(s). The European Commission/ Research Executive Agency are not responsible for any use that may be made of the information it contains.

[3] https://www.juniperresearch.com/press/iot-connections-to-reach-83-bn-by-2024

Their growing number is one reason IoT devices have become an attractive target for attackers. Another reason is that cybersecurity practices for IoT deployments are still not well understood and often not applied. For instance, IoT devices often use weak passwords and unencrypted network traffic[4]. Moreover, their low computational power limits their ability to run sophisticated security solutions. Thus, vulnerable IoT devices are exploited by attackers by injecting malicious software (malware) to perform various attacks (e.g., Distributed Denial of Service (DDoS)) on different targets. For example, the DNS provider Dyn faced one of the largest known DDoS attacks that reached 1.2 Tbps [2]. The attack was performed by Mirai, a type of malware that spreads across IoT devices to form a network of compromised devices referred to as a botnet.

There are several contributions in the literature focusing on detecting IoT botnet attacks based on network traffic patterns [19,12,3,9], using different machine learning (ML) techniques (e.g., Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), etc.). Yet, few works focus on detecting IoT botnets during their spreading phase, before any attacks. These approaches typically use shallow ML techniques [7] or recurrent neural networks [1] to analyze network packet headers. Unfortunately, while these models provide good accuracy, their architectures exhibit several limitations, such as low inference speed and the inability to account for temporal and topological information.

In this paper, we expose the issues and limitations of these state-of-the-art models for early-stage botnet detection, propose changes to mitigate these issues, and propose an alternative model called LiMNet (**Li**ghtweight **M**emory **Net**work). LiMNet is a novel device-centric model for early-stage botnet detection. It uses an internal memory to understand the behaviour of each IoT device and employs mutually-recurrent units to capture the causal interactions among the devices over time. This allows it to classify infected and under-attack devices, in addition to malicious packets. LiMNet is designed to be a lightweight model suitable for large-scale IoT deployments. It achieves better results than state-of-the-art recurrent models, while being smaller in memory footprint and faster during inference.

Contributions. The main contributions of this work can therefore be summarized as follows, in order of importance:

1. A a novel lightweight model for early-stage IoT botnet detection based on memory networks and mutually-recurrent units, which achieves near maximum scores ($\sim$99% AUROC), 14% better than state of the art, while being almost 8 times faster in inference;
2. novel device classification tasks to aid the deployment of targeted countermeasures in infected Iot networks;
3. a critical analysis of the issues and limitations of state-of-the art recurrent models for early-stage IoT botnet detection;
4. a modification to the input representations of existing recurrent models, mitigating some generalization issues while providing 4 times faster inference.

[4] https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot

The rest of the paper is organized as follows. Section 2 provides background on IoT botnets and ML techniques. Section 3 introduces the state of the art for early-stage botnet detection. Section 4 presents LiMNet, while Section 5 analyzes the limitations of recurrent models for botnet detection. Sections 6 and 7 present the evaluation methodology and experimental results, respectively. Sections 8 and 9 provide insights and conclusions. Finally, Appendix A further analyses the datasets used, while Appendices B and C present additional results.

2 Background

To understand the limitations of existing recurrent models for botnet detection and the architecture of LiMNet, we first introduce some background information on IoT botnets (Section 2.1) and several concepts from the ML field, namely Recurrent Neural Networks (Section 2.2), Memory Networks (Section 2.3) and Graph Representation Learning (Section 2.4).

2.1 IoT Botnets

Botnets present four main components: the bot, the C&C server, the loader, and the report server [12]. The bot is a malicious executable that infects IoT devices and is responsible for executing commands issued by the botmaster (i.e., the owner of the botnet). The C&C server is a dashboard that communicates with all compromised devices. It is managed by the botmaster and allow him/her to issue commands to the bot to, for example, orchestrate an attack. The loader is the component that helps in disseminating the malware in different computer architectures (e.g., ARM, x86, etc.) by communicating with potential new victims. Finally, the report server stores information about the bots.

Botnets are designed with different architectures. While some botnets like Torii, Mirai and its variants are based on a centralized architecture where there is only one central C&C server, other botnets, such as Hajime, employ decentralized communication patterns [12].

While IoT botnets present similar ways of attacking and infecting devices, they often differ in how they identify potential victims. Certain botnets can random Internet devices (e.g. Bashlite/gafgyt), while others prioritize scanning the local network. There are also botnets with unknown or non-trivial behaviour, such as Torii, for which the source code is not available, and Hajime, for which no attacks have been observed yet.

Mirai is one of the largest botnets currently active in IoT environments, with over 600k bots and attacks surpassing 1 Tbps in traffic volume, making it an important target for analysis. Infected devices scan random IPv4 addresses by sending TCP SYN probes on Telnet using TCP/23 or TCP/2323, ignoring a list of hard-coded IP addresses. If a device responds, its credentials are brute-forced using a hard-coded list of 62 username and password combinations, which are extracted from default configurations of IoT devices. Upon a successful login attempt, the credentials and IP address of the device are sent to the report server.

Then, the loader can log into the new victim, determine its architecture and download the appropriate build of the bot. After that, the malware hides itself by changing its process name and kills all processes using TCP/23 or TCP/2323, or processes of other competing bots. In parallel, it listens for commands from the C&C server while scanning for new victims [2].

The spreading phase and the attack phase of a botnet are independent. The spreading phase starts when a device is infected and tries to recruit new victims. On the other hand, the attack phase begins when the botmaster issues commands via the C&C to the bots to perform an orchestrated attack on a target, usually in the form of a distributed denial of service (DDoS).

2.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) [20] are a type of deep learning model that takes sequential data as input. RNNs can be used to process a variety of sequential inputs, such as sentences in language translation, sounds in speech recognition, or time-series in financial analysis. RNN models typically have small sizes, as the same operations and internal weights (referred to as a *cell*) are reused to process each entry in the sequence. The entries are processed one at a time, as each cell takes as an additional input a *memory* produced by the previous cell and outputs an updated memory for the next cell. This allows RNNs to model the evolution of the sequence and interpret later entries based on earlier ones, but it also makes RNNs slower than non-sequential models.

Several RNN cell types have been introduced over the years. For instance, LSTMs (Long Short Term Memory) use two memory vectors to remember important facts over longer sequences, compared to the original RNN cells [4]. GRUs (Gated Recurrent Units) achieve similar results but use a single memory vector and less internal weights [4]. Finally, FastGRNN cells [14] are even smaller and faster, while maintaining good output quality.

RNNs are often used in conjunction with *embedding layers*. These are responsible for converting arbitrary tokens into numerical vectors that can then be input to the RNN, or to other deep learning models, using a lookup table. For example, in a language translation application, a word embedding layer maps each word to a low-dimensional space, where its position with respect to other words encodes the meaning and usage of the word itself. This position is trained jointly with the rest of the ML model. Embedding layers are often very large, as a separate vector is needed for each possible input token.

2.3 Memory Networks

Memory networks [25] were introduced to address the limited capability of deep learning models to store and organize long-term memories. They introduce a novel long-term memory component that can be dynamically updated based on new input facts, which are stored and then retrieved to respond to queries.

In addition to the internal memory, memory networks have four components. First, an *input feature map* converts the input to an internal representation, which is then used by a *generalization layer* to update the internal memory of the model. The updated memory and the input representation are then combined by an *output feature map* into an output representation, which is finally used by a *response layer* to produce the actual output.

Memory networks have been effectively applied in several fields, including question answering [25] and recommender systems [6].

2.4 Graph Representation Learning on Temporal Interaction Networks

Graph Representation Learning (GRL) is the field of extracting low-dimensional representations from graph-structured data, in order to apply ML techniques on them [8]. Temporal Interaction Networks are graphs where an edge between two nodes indicates that they interacted at a specific time. These networks naturally evolve over time to incorporate new interactions. Several techniques have been proposed to capture both short-term and long-term behaviour of these graphs, with a focus on recommender systems. In particular, JODIE [13] and DGNN [16] employ mutually-recurrent neural networks to update the short-term representations of the nodes involved after each interaction. DeepRed[11] is conceptually similar, but forgoes recurrent networks in favor of dynamically generating short-term representations from long-term embeddings.

While the literature on GRL for temporal interaction networks and that of memory models are mostly separate, we note that the described GRL techniques can be seen as particular instances of the broader memory model concept, as they dynamically update their internal knowledge in response to incoming information, and subsequently use this updated knowledge to perform predictions.

3 Related Work

Many works have tackled the issue of IoT botnet detection using ML models. One popular direction, which also applies to other types of malware, consists in performing static or dynamic analysis of the executable file on the infected device itself. For instance, in [22], a Convolutional Neural Network is used to detect printable string information stored in the executable code of IoT botnets.

A different direction explored in the broader domain of malware detection consists of analyzing traffic at the network level to identify which nodes in a network are infected. In particular, a number of features are extracted from each network packet and fed to an ML model which classifies the packet as benign or malicious. In the context of IoT botnets specifically, several works used network-level analysis to detect botnets in the attack phase, with the goal of mitigating these attacks by filtering malicious traffic. Different ML techniques have been employed, including deep autoencoders fed with statistical packet features [18] and recurrent models fed with packet headers [17,9].

Another research direction focuses on exploiting graph theory to detect botnets, not only in IoT but also in more traditional settings. As P2P bots frequently communicate with each other, community detection techniques have proven effective in detecting them [5,27,23]. Statistical approaches have also been used to identify correlations between nodes and thus infer botnet affiliations [26,15].

Yet, few work have focused on detecting botnets during their propagation phase, before any large-scale attacks. MedBIoT[7] is a dataset explicitly designed for this task. Its authors evaluate the performance of simple techniques including k-Nearest Neighbours classifiers, decision trees and random forests. Recurrent models have also been shown to be very effective for early-stage detection on MedBIoT [1]. These models, like the ones for detection of botnet attacks by which they are inspired [9], treat packet headers as sequences of fields, such as IP address, source port, packet length, etc. The low-dimensional representations of these fields, obtained from an embedding layer, are passed to layers of recurrent cells, with the last output being fed to a classifier for botnet detection.

Our approach, LiMNet, focuses on IoT botnet spreading, but differs from the discussed works [7,1] in several aspects. First, LiMNet is a memory network which learns device representations, rather than a recurrent network focused on packet representations. This allows it to more easily identify not only malicious packets, but also devices that have been, or will soon be, infected. Furthermore, it uses few key packet features, rather than all header fields.

4 LiMNet: A Lightweight Memory Network for Early-Stage Botnet Detection

LiMNet is a novel Lightweight Memory Network that can extract causal relationships from a stream of interactions between nodes in a graph, store relevant node-level information in an internal memory and use this information to solve node- and interaction-level tasks.

While LiMNet should be able to generalize to temporal interaction networks from various domains, here we focus solely on early-stage detection of IoT botnets, where the interactions are network packets and the nodes are IoT devices.

In this context, LiMNet substantially differs from previous works in that it is device-centric, rather than packet-centric. It builds and tunes over time an internal representation of the behaviour of each device and then uses this, combined with packet features, to identify bots. This provides it with important additional information compared to state-of-the-art models that analyze each packet in isolation.

4.1 LiMNet Architecture

Figure 1 provides an overview of LiMNet. We subdivide its description following the four main components of any memory model (cfr. Section 2.3): input feature map, generalization layer, output feature map and response.

Input Feature Map Given an incoming packet, `tshark`[5] is used to extract the packet length and the IP addresses of source and destination devices, and

[5] https://www.wireshark.org/docs/man-pages/tshark.html

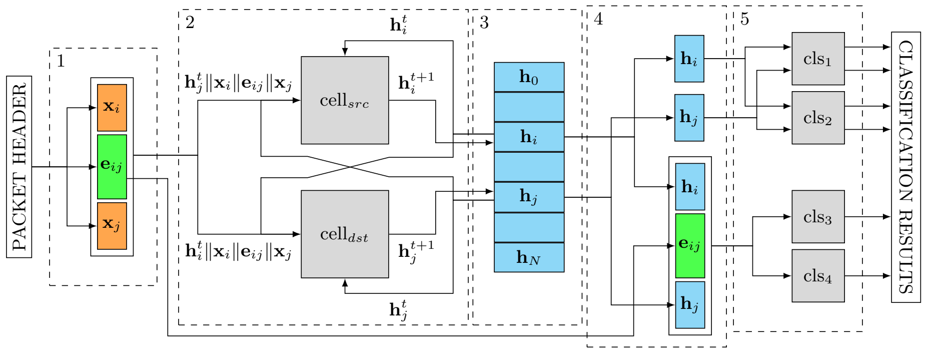

Fig. 1. Structure of LiMNet with two device classifiers and two packet classifiers. Colored boxes are: blue) memory representations green) packet features orange) device features grey) trainable networks. Dashed numbered boxes are: 1) input feature map 2) generalization layer 3) memory 4) output feature map 5) response layer.

also to infer the application-level communication protocol, or the transport-level protocol if the former cannot be detected. A feature vector for the packet is then built by concatenating its normalized length with the one-hot encoding of the detected protocol. For each of the source and destination addresses, a feature vector is built containing two binary features: whether they are unicast or multicast and whether they are public or private addresses.

Generalization Layer The generalization layer consists of two mutually-recurrent cells, inspired by JODIE [13] and DGNN [16]. These can be LSTM, GRU or FastGRNN cells, as introduced in Section 2.2. The memory of the model is a dictionary mapping IP addresses to the memory of the recurrent cells. The source cell (resp. destination cell) updates the memory representation of the source node (resp. destination node), using as input the concatenation of the edge features, address features of both devices and previous memory of the destination node (resp. source node). More precisely, when a new packet is sent from IP address i to IP address j, the following update is performed:

$$\mathbf{h}_i^{t+1} = \text{cell}_{\text{src}} \left(\mathbf{h}_j^t \| \mathbf{x}_i \| \mathbf{e}_{ij} \| \mathbf{x}_j \,,\, \mathbf{h}_i^t \right) \tag{1}$$

$$\mathbf{h}_j^{t+1} = \text{cell}_{\text{dst}} \left(\mathbf{h}_i^t \| \mathbf{x}_i \| \mathbf{e}_{ij} \| \mathbf{x}_j \,,\, \mathbf{h}_j^t \right) \tag{2}$$

where $\mathbf{e}_{ij}$, $\mathbf{x}_i$ and $\mathbf{x}_j$ are the feature vectors for, respectively, the packet, the device i and device j, and $\mathbf{h}_i^t$ (resp. $\mathbf{h}_j^t$) is the memory representation of device i (resp. j) after t packets have been processed. This representation is the memory of the recurrent cell, and thus consists of a single vector for FastGRNN and GRU cells, or two vectors (short and long term memories) for LSTM cells. For IP addresses not seen before, the memory representation is initialized with zeros.

To prevent excessive memory usage in large deployments, if an IP address is not seen for a long period of time, its representation can be removed from the memory dictionary. This is a sensible decision, as external public servers may change IP address or stop being accessed by the local IoT devices. Furthermore, if an internal IP address remains silent for a long time, it is likely that the IoT device was removed from the network and the address reassigned to a new IoT device, therefore rendering the previous memory useless and even detrimental.

Output Feature Map The output feature map is responsible for extracting the new packet and device representations to be used to generate the model responses. For the device representation, the updated contents of the memory are used directly. For cell types that produce multiple memory vectors, such as the LSTM, only the short-term representation is passed forward to the response layer. The packet representation is obtained by concatenating the new representations of the source and destination devices with the original features of the packet from the input feature map.

Response Layer The response layer is responsible for computing the desired model outputs, in this case, device and packet classifications. To achieve device classifications, the device representations from the output feature map are passed to a single-layer feed-forward classifier. A separate classifier is trained for each classification task. The same process is used for packet-level classification, using the packet representation from the output feature map.

4.2 Training LiMNet

The trivial approach to train LiMNet would be to order all packets in the network trace and then feed the entire sequence to the model, packet by packet. However, this leads to two issues: 1) scalability, as the training hardware parallelism cannot be exploited, and 2) vanishing gradients, as the contribution of early packets is lost in the backpropagation process.

We overcome these issues using truncated backpropagation through time (p-BPTT) [10], a well-known RNN training technique in which the input trace is split into shorter, partially-overlapping subtraces, which are treated as independent inputs. Batches of subtraces can be trained in parallel, while each of them is processed sequentially to ensure causal consistency. In general, p-BPTT can hinder the ability of a model to capture long-term relations [24]. However our analysis, reported in Appendix B, does not show generalization issues and training can be performed efficiently on very long subtraces, up to $\sim 10^5$ packets.

5 Limitations of Existing Recurrent Models for Early-Stage Botnet Detection

While existing recurrent models from [1] and [9] have been shown to perform well, they present several issues that make their deployment in real-world IoT scenarios challenging. This motivates the contributions of this work: a review of these recurrent models to analyze their limitations and the proposal of the novel LiMNet architecture. Here we provide an overview of these issues.

Issue 1: packet headers contain specific device IDs, such as MAC and IP addresses, that should not be used to classify traffic in local networks. A model trained with these features might, for example, learn that the packets sent from a certain local IP are malicious based on the training dataset, while in the real IoT network that local IP might not be malicious. Therefore, existing recurrent models might fail to generalize to different or evolving networks. LiMNetdoes not use any device IDs in its training and so is not affected by this issue.

Issue 2: by employing an input embedding layer, recurrent models assume that the same token appearing in different input positions represents the same, or similar, concept. However, the number 22 appearing as source port, destination port or packet length carries very different meanings. Therefore, these models might fail to capture complex scenarios where these distinctions become fundamental. Furthermore, the embedding layer treats the packet headers as a categorical variable, with each value mapped to an independent embedding. This prevents it from capturing patterns in numerical fields, such as the similarity between packet length 22 and 23. By not using an embedding layer to parse the headers, LiMNet avoids these pitfalls.

Issue 3: recurrent models are well-known to incur high inference latency. This is because the recurrent cell must be applied sequentially to each element of the input sequence, which in this case is a sequence of packet header fields. High throughput can be achieved by increasing the batch size used for inference, but only at the cost of additional latency introduced by the batching process. By contrast, the source and destination cells in LiMNet are independent and can be processed in parallel, speeding up inference.

Issue 4: by ignoring the temporal and topological relations among packets, focusing on a single header at a time, these models miss the overall evolution of the network. This may hinder their ability to accurately classify traffic flows, especially when malicious traffic looks similar to benign traffic at the level of individual packets. LiMNet is designed specifically to exploit these relations.

6 Evaluation Methodology

6.1 Deployment Environment

The use case targeted in this work is that of early detection of botnet propagation at the network level. Therefore, a suitable solution should fulfill a number of requirements. First, inference should be performed on a stream of packets with both low-latency and high throughput, in order to scale to large IoT deployments while still being able to react quickly to incoming threats.

Second, inference should not require any specialized hardware to achieve the desired performance level. This is because the inference should be able to run on simple, low-power hardware such as smart routers and smart switches, as close as possible to the actual IoT devices, to have real-time access to all packets being exchanged in the network. These network devices typically provide simple x86 or ARM CPUs with low core counts and small amounts of RAM. On the other hand, training can be done offline, by replaying captured network traffic on GPU-accelerated training servers.

Therefore, unlike previous work [1], we focus our performance testing on single-core CPU inference, and use a batch size of 1. In these conditions, lightweight models that can fit in the small private cache of the core and only require small matrix multiplications gain a substantial performance advantage.

6.2 Datasets

In order to evaluate the effectiveness of the recurrent models and of LiMNet, we use two different datasets: MedBIoT and Kitsune. Both datasets, summarized in Table 1, focus on the spreading phase of IoT botnets, rather than on the attacks driven by these botnets, and are therefore the most suitable to evaluate early-stage detection models.

The MedBIoT dataset [7] is gathered from a medium-size network with 83 devices. These devices are a combination of real and emulated devices, spanning categories such as smart locks, switches, fans, and light bulbs. Real, working botnets, namely Mirai, Bashlite/Gafgyt, and Torii, are injected in the controlled environment at different times. The network traffic of the spreading phases is captured, including communication between bots and controlled C&C servers.

The Kitsune dataset [21] is gathered in a small network of 3 PCs and 9 IoT devices, including a thermostat, a baby monitor, a webcam, low-cost security cameras and doorbells. One of these is infected with the Mirai malware and the network traffic is subsequently captured.

Table 1. Datasets Summary

Dataset	# of Devices	# of Packets	Botnets
MedBIoT	83	17,845,567	Mirai, Bashlite, Torii
Kitsune	12	764,137	Mirai

6.3 Tasks

Previous works [7,1] consider the classification of malicious packets as the only task for botnet detection. However, device-level tasks may play an important role as well. Classifying a device as infected or under attack allows administrators to identify vulnerable devices and deploy more targeted countermeasures.

Therefore, in addition to the binary classification of malicious packets based on ground-truth labels, we consider the classification of infected and under attack devices, with device labels generated from the packet labels and headers using simple heuristics. While the datasets do not present any overlap of multiple malware types, this is a possibility in real networks, and a device may therefore be infected and under attack at the same time. Thus, we consider these as two independent binary classification tasks. The metric used is the area under the ROC curve (AUROC), as it provides a good indication of the possible tradeoffs between sensitivity and fallout achievable by the models.

6.4 State-of-the-Art Recurrent Models for Early-Stage Botnet Detection

We test state-of-the-art recurrent models from [9] and [1] with two goals: to evaluate the impact of the issues discussed in Section 5 and to establish baseline accuracy and inference speed when evaluating LiMNet.

However, these recurrent models are only designed to perform packet-level classification, while LiMNet also performs the device-level tasks described above. In the original models, the output representation of the last recurrent cell is fed to a shallow feed-forward decoder which classifies the input packet. We extend this by adding a pair of decoders which take the same representation as input and are trained to classify the source and destination node of the packet, respectively. As for the memory model, we allow multiple packet classification tasks by training multiple packet classifiers and we allow multiple device classification tasks by training multiple pairs of source and destination classifiers.

The implementation of LiMNet, the modified baselines and the complete evaluation code used in this paper are freely available online[6].

7 Experimental Results

7.1 Experiment 1: Parameter Selection for Recurrent Models

In their original formulation, the recurrent models from [9] and [1] take all packet header fields as inputs. As discussed in Section 5, this may lead to overfitting due to the presence of device identifiers (Issue 1). Furthermore, due to the sheer length of the header (33 fields in [9]), inference latency is high (Issue 3).

To alleviate these issues, we test the recurrent models with smaller subsets of the packet headers. In particular, we test them with all headers expect for MAC and IP addresses, to prevent overfitting, and with only the UDP/TCP ports and packet length, providing a level of input information similar to LiMNet.

Table 2. Combinations of datasets and hyperparameters employed in the evaluation

Datasets	MedBIoT (Torii subset), MedBIoT (Mirai subset), MedBIoT (BashLite subset), MedBIoT (complete), Kitsune
Cell Types	LSTM, GRU, FastGRNN
Layers	3, 1
Layer Sizes	64, 32
Input Features	Full header, Full header except MACs and IPs, Only ports and length

[6] https://github.com/lodo1995/LiMNet

Motivated by the results in [1], we test both deep models with 3 recurrent layers and shallow models with a single layer, as well as three different recurrent cells, namely LSTM, GRU and FastGRNN. According to [1], both using shallower models and using smaller recurrent cells, such as GRU and FastGRNN, can achieve similar performance to the deep LSTM-based model in [9], while being much faster. Finally, in the same spirit of minimizing size without loss of quality, we test smaller layers of size 32, in addition to the size 64 used in previous works. Considering the different datasets, the total number of parameter combinations to test is 180, summarized in Table 2.

Unfortunately, the results on the MedBIoT dataset do not allow for meaningful comparisons, as all combinations lead to over 99.9% AUROC scores on all three tasks. In contrast, on the Kitsune dataset, the results are much lower and present a meaningful spread. Particularly interesting is the fact that feeding the recurrent model with only port and length information provides consistently better results than using all features as done in previous work, with up to 2% better AUROC scores, as seen in Table 3. On the contrary, providing all features except the IDs leads to the worst results.

Table 3. AUROC scores of recurrent models with different hyperparameter combinations, fed with ports and length features on the Kitsune dataset. "best of ..." indicates the best results obtained by models fed with more features (not necessary the same model in all columns). All results are averages over 5 runs. Bold results are the best, underlined results are within 0.5%.

Layers	Layers size	Cell	Device malicious	Device attacked	Packet malicious
best with all headers except IDs			84.27	95.77	75.88
best with all headers			84.99	96.6	78.02
1	32	FastGRNN	85.66	96.91	80.63
1	32	GRU	85.72	97.11	80.76
1	32	LSTM	85.75	97.2	80.66
1	64	FastGRNN	85.71	97.09	80.86
1	64	GRU	85.74	97.44	80.86
1	64	LSTM	**85.83**	97.38	81.04
3	32	FastGRNN	85.48	96.83	80.79
3	32	GRU	85.56	97.26	80.83
3	32	LSTM	85.62	97.28	80.9
3	64	FastGRNN	85.8	97.22	**81.37**
3	64	GRU	85.82	**97.52**	81.23
3	64	LSTM	85.7	97.45	81.12

We therefore focus on the comparison of these low-features models, summarized in Table 3. A larger layer size of 64 provides consistently better results than 32. Using a 3 layers network as in [9] provides a slight advantage compared to the single-layer network from [1], although the margin is slim. On the other hand, the choice of recurrent cell seems to have almost no impact, with the best per-task results achieved by three different cell types.

In summary, the key takeaway of this experiment is that reducing the amount of input features provides measurably better results and should be considered even if inference speed is not an issue. However, further time and space savings by reducing number of layers and layer sizes may have a slight negative effect on model quality.

7.2 Experiment 2: Recurrent vs Memory Models

The next experiment aims at comparing the best recurrent models identified above with LiMNet. Our model is always fed with minimal features extracted from port and length headers and always has a single layer, as explained in Section 4. Thus, the only parameters that need to be evaluated are layer size and cell type, for which the same values presented in Table 2 are used.

As was the case for the recurrent models, LiMNet also easily achieves over 99.9% AUROC score on all tasks in MedBIoT, and we therefore focus our attention on the harder and more indicative Kitsune dataset.

As can be seen in Table 4, LiMNet with layer size of 64 and GRU recurrent units achieves an almost maximum score of ∼99%, outperforming the best low-features recurrent models by a large margin of over 12% on average on the three tasks. When compared to the original recurrent models that use all packet headers, the advantage of LiMNet grows to over 14%. Similarly to recurrent models, LiMNet also provides better results with larger layers. However, while the former were not significantly affected by the choice of recurrent cell, LiMNet appears to strongly favor GRU units. In general, the results present a wider spread, indicating a higher sensitivity of LiMNet to its hyperparameters.

Table 4. AUROC scores of different configurations of recurrent models and of LiMNet. All results are averaged over 5 runs. Bold results are the best, underlined results are within 0.5% of the best.

Type	Layers	Layers size	Cell	Device malicious	Device attacked	Packet malicious
best recurrent with all headers				84.99	96.6	78.02
recurrent	1	64	LSTM	85.83	97.38	81.04
recurrent	3	64	FastGRNN	85.8	97.22	81.37
recurrent	3	64	GRU	85.82	97.52	81.23
LiMNet	1	32	FastGRNN	85.52	95.97	92.8
LiMNet	1	32	GRU	<u>98.73</u>	<u>98.72</u>	<u>99.72</u>
LiMNet	1	32	LSTM	85.91	96.7	88.82
LiMNet	1	64	FastGRNN	98.21	97.87	<u>99.48</u>
LiMNet	1	64	GRU	**99.13**	**98.84**	**99.75**
LiMNet	1	64	LSTM	84.54	97.09	86.36

7.3 Experiment 3: Inference Speed

The third and final experiment aims at comparing the inference speed of the best configuration of both recurrent models and LiMNet.

Given the characteristics of the deployment environment described in Section 6.1, the inference is performed on a single x86 CPU core[7], as low-power smart network equipment typically lacks more powerful resources. Furthermore, the batch size is set to 1, effectively disabling batching, as it is undesirable for two reasons. First, it substantially increases latency by queuing messages, while providing little throughput improvements due to the low level of data parallelism available on a single x86 core. Second, batching puts additional pressure on the RAM, which might be limited in these low-power devices.

Table 5 summarizes the results obtained by the recurrent and LiMNet configurations that ranked best in previous experiments. We report the numbers from the MedBIoT dataset, as it includes a more diverse (and more realistic) range of devices, ports and protocols, therefore slightly increasing the size and reducing the speed of both recurrent and LiMNet models. However, the same conclusions could be drawn on the Kitsune dataset, albeit at a lower scale.

Table 5. Inference speed and model size of different model configurations on the MedBIoT dataset. Models marked with * are trained on all packet headers, while the others are trained with the minimal amounts of features as described in the text.

Type	Layers	Layer size	Cell	Model size [kiB]				Infer. speed [packets/s]
				embed.	cells	classif.	total	
recurrent*	1	64	LSTM	16384	130	1.3	16515	429
recurrent	1	64	LSTM	9178	130	1.3	9309	1814
recurrent	3	64	FastGRNN	9178	98	1.3	9277	948
recurrent	3	64	GRU	9178	293	1.3	9472	972
LiMNet	1	32	GRU	-	64	0.6	**65**	**3381**
LiMNet	1	64	FastGRNN	-	75	1.0	76	3067
LiMNet	1	64	GRU	-	225	1.0	226	3037

The fastest LiMNet model (GRU cell, layer size 32) achieves over 3300 packets per second. However, it is not the best performing model in terms of quality. Based on Table 4. a layer size of 64 would achieve marginally better average scores. However, this comes at a 10% speed reduction. Depending on the specific deployment setting, this may be an important tradeoff to consider.

The fastest recurrent model is around 46% slower. This difference has three causes. First, the LSTM recurrent unit is slightly larger and slower than the GRU unit. Second, LiMNet requires 2 recurrent units (for source and target devices), while the recurrent models require one unit for each feature, and thus 3 units when considering ports and packet length. Finally, the large size of the

[7] More precisely, a single core of an Intel Cascade Lake-SP CPU with 2.2 GHz base clock, 3.2 GHz max. turbo clock, 32 KB L1d cache and 1 MB L2 private cache.

embedding layer (over 95% of the total) means that the recurrent model cannot fit in the private L2 cache of a CPU core and must instead reside in the slower and shared L3 cache, or in main memory. This introduces a memory bottleneck compared to LiMNet, as large embedding representations need to be fetched from distant memory locations and, due to the random access patterns, speculative pre-fetching is not effective.

Furthermore, moving from the fastest to the highest-quality recurrent model brings an additional 46% speed reduction, from 1814 to 972 packets per second. This is due to the switch from 1 layer to 3 layers, which triples the number of recurrent cells that need to be evaluated during inference.

For completeness, we also take the fastest hyperparameter combination for the recurrent model and train it with all features instead of just ports and packet length. The results is a model that is 76% slower, as due to the much larger number of input headers the recurrent units grow in number from 3 to 33.

To summarize, LiMNet not only achieves much better scores than the original recurrent models from [9] and [1], but can do so at an almost 8 times higher speed. On the other hand, the speedup that these models can achieve with less input features stops at 4 times their original speed.

8 Discussion and Limitations

8.1 Dataset Limitations

Unfortunately, the number of open-access datasets that focus on the spreading phase (rather than the attack phase) of IoT botnets and that provide enough devices to extract network-level interactions is very limited. Each of the two datasets used in this paper presents its limitations.

MedBIoT is a large dataset with multiple devices and malware types. However, as discussed in Section 7 and as already hinted by previous results [7,1], the distinction between benign and malicious traffic is too obvious, with any deep learning-based approach achieving near-perfect scores, making comparisons difficult. Kitsune, on the other hand, appears to present a harder challenge, making it a useful benchmark for comparisons. However, it is several orders of magnitude smaller and lacks diverse malware and IoT devices. It may thus not provide a complete picture of a real IoT deployment.

We further analyze these two datasets in Appendix A and find that Kitsune provides a more balanced, and thus more challenging, mix of network protocols. However, more work is necessary to fully understand these dataset dynamics and to develop more open-access datasets suitable for advanced IoT botnet detection.

8.2 Issues of Recurrent Models for Early-Stage Botnet Detection

Section 5 discussed a number of potential issues in the architecture of state-of-the-art recurrent models for IoT botnet detection. While the limitations of the datasets do not allow for a complete, in-depth analysis of each issue, the results presented in Section 7 provide some hints to the extent of these issues.

Issue 1, that is, the incorrect use of device IDs that may prevent model generalization, seems to be confirmed by the results in Table 3. Models trained without those IDs perform worse than those trained with them, indicating that the model is exploiting these dataset-specific information in its embeddings.

However, that issue is rendered moot by the fact that removing almost all features achieves even better performance. This may be related to **Issue 2**. As most packet headers reuse the same numerical values with different meaning (e.g. port 22 vs packet length 22 vs packet checksum 22), reducing the number of input headers reduces the number of different meanings that a single numerical value can have. This allows the embedding layer to better capture the semantics of these numerical values. Intuitively, the model is able to better focus on the few relevant fields selected, without being confused by insignificant headers such as checksums and reserved fields.

Issue 3 is also partially mitigated by the reduced number of features. However, the inference speed of recurrent models is still inferior to that of LiMNet, as shown in Section 7.3.

Finally, the substantially better results obtained by LiMNet confirm the relevance of **Issue 4**, indicating the presence of useful information in the temporal and topological relationships between different packet flows.

9 Conclusion

Securing IoT networks from malicious botnets is an important step towards the widespread deployment of IoT devices in the Industry 4.0 and in smart homes.

In this work, we analyzed the issues and limitations of state-of-the art recurrent models for early-stage detection of devices infected and under attack. Our proposed modifications to mitigate these issues slightly improved classification performance, while increasing inference speed by 4 times.

However, we have shown that an increase of 14% in classification scores, up to a nearly maximum value of $\sim$99%, and an almost 8 times speedup can be achieved by switching to a completely different model. Based on memory networks and mutually-recurrent units, our LiMNet architecture can understand and exploit the crucial temporal and topological relationships in the network, while being incredibly lightweight in both size and computational requirements.

Furthermore, the LiMNet architecture proposed in this work is not specific to malware detection and could be applied in any temporal interaction network that presents strong causal relationships. Therefore, the potential use of LiMNet in other areas of security should be explored in future works. One such area could be the detection of fraudulent behaviour in financial transactions on cryptocurrency networks such as Bitcoin and Ethereum.

Acknowledgements The authors would like to thank Stefanos Antaris (affiliated with KTH and Hive Streaming AB) for the insightful discussions and literature recommendations that helped shape the direction of this work.

References

1. Alzahrani, H., Abulkhair, M., Alkayal, E.: A multi-class neural network model for rapid detection of iot botnet attacks. International Journal of Advanced Computer Science and Applications **11** (01 2020)
2. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al.: Understanding the mirai botnet. In: 26th {USENIX} security symposium ({USENIX} Security 17). pp. 1093–1110 (2017)
3. Bahşi, H., Nõmm, S., La Torre, F.B.: Dimensionality reduction for machine learning based iot botnet detection. In: 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV). pp. 1857–1862. IEEE (2018)
4. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
5. Coskun, B., et al.: Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts. In: Proceedings of the 26th Annual Computer Security Applications Conference. p. 131–140. ACSAC '10, Association for Computing Machinery, New York, NY, USA (2010)
6. Ebesu, T., Shen, B., Fang, Y.: Collaborative memory network for recommendation systems. In: The 41st international ACM SIGIR conference on research & development in information retrieval. pp. 515–524 (2018)
7. Guerra-Manzanares, A., Medina-Galindo, J., Bahsi, H., Nomm, S.: Medbiot: Generation of an iot botnet dataset in a medium-sized iot network. In: ICISSP (2020)
8. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 (2017)
9. Hwang, R.H., Peng, M.C., Nguyen, V.L., Chang, Y.L.: An lstm-based deep learning approach for classifying malicious traffic at the packet level. Applied Sciences **9**(16) (2019). https://doi.org/10.3390/app9163414
10. Jaeger, H.: Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach, vol. 5. GMD-Forschungszentrum Informationstechnik Bonn (2002)
11. Kefato, Z.T., Girdzijauskas, S., Sheikh, N., Montresor, A.: Dynamic embeddings for interaction prediction. In: Proceedings of The Web Conference 2021 (2021)
12. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: Ddos in the iot: Mirai and other botnets. Computer **50**(7), 80–84 (2017)
13. Kumar, S., Zhang, X., Leskovec, J.: Predicting dynamic embedding trajectory in temporal interaction networks. In: Proceedings of the 25th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM (2019)
14. Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., Varma, M.: Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. p. 9031–9042. NIPS'18, Curran Associates Inc., Red Hook, NY, USA (2018)
15. Li, J., et al.: Distributed threat intelligence sharing system: A new sight of p2p botnet detection. In: 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS). pp. 1–6. Riyadh, Saudi Arabia (2019)
16. Ma, Y., Guo, Z., Ren, Z., Tang, J., Yin, D.: Streaming graph neural networks. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 719–728 (2020)
17. McDermott, C.D., Majdani, F., Petrovski, A.V.: Botnet detection in the internet of things using deep learning approaches. In: 2018 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2018). https://doi.org/10.1109/IJCNN.2018.8489489

18. Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y.: N-baiot—network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Computing **17**(3), 12–22 (2018). https://doi.org/10.1109/MPRV.2018.03367731
19. Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., Elovici, Y.: N-baiot—network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Computing **17**(3), 12–22 (2018)
20. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh annual conference of the international speech communication association (2010)
21. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. Network and Distributed System Security Symposium (NDSS) (2018)
22. Nguyen, H., Ngo, Q., Le, V.: Iot botnet detection approach based on psi graph and dgcnn classifier. In: 2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP). pp. 118–122 (2018). https://doi.org/10.1109/ICICSP.2018.8549713
23. Sagirlar, G., Carminati, B., Ferrari, E.: Autobotcatcher: blockchain-based p2p botnet detection for the internet of things. In: 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC). pp. 1–8. IEEE (2018)
24. Tallec, C., Ollivier, Y.: Unbiasing truncated backpropagation through time. CoRR **abs/1705.08209** (2017), http://arxiv.org/abs/1705.08209
25. Weston, J., Chopra, S., Bordes, A.: Memory networks. arXiv preprint arXiv:1410.3916 (2014)
26. Yang, Z., et al.: P2p botnet detection based on nodes correlation by the mahalanobis distance. Information **10**(5), 160 (2019)
27. Zhuang, D., et al.: Peerhunter: Detecting peer-to-peer botnets through community behavior analysis. In: 2017 IEEE Conference on Dependable and Secure Computing. pp. 493–500. Taipei (2017)

A Kitsune vs MedBIoT: challenges for ML models

To effectively and fairly compare early-stage botnet detection models, large, realistic and challenging datasets are required. We therefore analyze Kitsune and MedBIoT, to understand why the former is more challenging than the latter.

In [1], the authors hypothesize that their balancing of the different malware classes in MedBIoT may cause their very high scores. However, in our work, we do not perform any balancing and still achieve near-perfect scores for both our approach and the baseline from [1], thus disproving this hypothesis.

The breakdown of the protocol distributions of the datasets, reported in Table 6, shows that MedBIoT is dominated by a single protocol and that within most protocols legitimate packets are one order of magnitude more (or less) than malicious ones, with the overall dataset being skewed towards legitimate traffic. A model can easily achieve high scores by focusing on the dominating protocol, and by providing simple majority answers for the others. Kitsune, on the other hand, is fairly well-split across three dominant protocols and fairly balanced between legitimate and malicious traffic, both overall and within each protocol. Any model therefore needs to capture multiple legitimate behaviours and learn

Table 6. Distribution of protocols in the datasets, as identified by `tshark`. When the application-level protocol is not identified, the transport-level protocol is reported.

MedBIoT				**Kitsune**			
protocol	legitimate	malicious	% of total packets	protocol	legitimate	malicious	% of total packets
TCP	14971443	4747941	89.1	UDP	39650	23904	41.2
MQTT	1732790	2210	7.8	TCP	20669	30048	32.9
TELNET	0	350431	1.6	DNS	23608	14305	24.6
HTTP	266377	4769	1.2	SSDP	636	416	0.7
DNS	17994	34016	0.2	TELNET	0	435	0.3
10 others	2848	3616	<0.1	7 others	282	144	<0.3

to discern malicious traffic within each protocol based on additional signals. It is thus unsurprising that Kitsune proved more challenging in our experiments, as it better tests the modelling capabilities of botnet detection approaches.

B Effect of Truncated Backpropagation Through Time

Truncated Backpropagation Through Time (p-BPTT) [10] has gained traction in the RNN field as a simple technique to quickly and efficiently train model on very long sequences. However, this technique is known to reduce the ability of a model to capture long-range relations, as inputs that are very far in the original sequence never co-appear in the same subsequence after splitting [24].

To ensure that this issue is not affecting LiMNet, we train it with different combinations of length and stride for the subsequences. For a fair comparison, it is important to consider the length/stride ratio. A higher ratio indicates more overlaps between the subsequences and thus leads to more training data points per epoch. The results are reported in Table 7.

Table 7. AUROC scores of LiMNet with GRU units and layer size 32 on the Kitsune dataset, with varying subsequence length and stride for p-BPTT.

Sequence length	Sequence stride	length/stride ratio	Device malicious	Device attacked	Packet malicious
1000	200	5	98.68	<u>98.93</u>	98.97
5000	1000	5	98.38	98.07	<u>99.59</u>
10000	2000	5	96.37	97.37	98.64
20000	4000	5	64.77	90.89	86.26
10000	1000	10	**99.42**	**99.26**	**99.93**
20000	1000	20	<u>99.2</u>	<u>98.99</u>	<u>99.84</u>

Keeping the ratio fixed at 5, Table 7 shows that longer subsequences lead to worse results, not better. If there are any gains from modelling long-term relations, they are offset by the larger strides, which cause most packets to never appear close to the end of any subsequence, where the backpropagation gradients are stronger. This issue can be mitigated by reducing the stride and thus increasing the ratio. This bring performance up, but at much higher computational costs. Furthermore, even with high ratios, increasing sequence length over 10k packets does not seem to provide any benefit, indicating that, at this length p-BPTT does not negatively impact LiMNet performance.

C Cross-dataset Model Generalization

As an additional experiment, in Table 8 we consider the performance of LiMNet when trained on one dataset and tested on another. The results show once more how different the scenarios presented by Kitsune and MedBIoT are.

As the datasets present different protocol mixes (as shown in Appendix A), the one-hot protocol encoding of the testing dataset needs to be modified to match that of the training dataset, which is the one the model expects. For protocols present in both datasets, this "alignment" amounts to a simple reshuffle of the features. For application-level protocols that are present in the testing dataset but not in the training one, we consider two options: 1) replacing them with their transport-level protocol, as TCP and UDP are present in both datasets, or 2) setting all protocol features to zero, effectively marking the packet as having no protocol. Our results show no substantial differences between these two options.

A model trained on Kitsune has no knowledge of the Torii and Bashlite malware present in MedBIoT, while one trained on the latter is aware of the Mirai malware in Kitsune. This explains why training on MedBIoT and testing on Kitsune provide better results than the opposite in the malicious packet detection task. However, the results on this task are still extremely low, probably because the model also faces different patterns of legitimate traffic, which it cannot recognize. The device-level tasks, present much better (although still low) results. This may be due to the memory component of LiMNet: while a single packet may be hard to judge in these conditions, the model still memorizes enough knowledge over time to correctly flag at least part of the devices.

Table 8. AUROC scores of LiMNet with GRU units and layer size 32, trained and tested on different combinations of datasets

Training dataset	Testing dataset	Protocol alignment	Device malicious	Device attacked	Packet malicious
Kitsune	Kitsune	-	98.73	98.72	99.72
Kitsune	MedBIoT	transport	58.92	60.39	28.64
Kitsune	MedBIoT	no proto	58.83	59.98	28.52
MedBIoT	MedBIoT	-	99.79	99.79	99.84
MedBIoT	Kitsune	transport	42.87	60.99	35.62
MedBIoT	Kitsune	no proto	42.87	60.99	35.63