DeepSniffer: An Attention-based Neural Network for Real-time Schedule Interception

Ashkan Farhangi, Zhishan Guo
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, USA
{ashkan.farhangi,zsguo}@ucf.edu

Abstract—Security breaches in real-time systems can cause serious harm to the system, environment, or humans. To avoid missing deadlines, real-time systems are designed to provide guarantees and behave in a deterministic manner. Hence, it is important to discover the potential information that can be intercepted to prepare for external attacks. In this study, we have used the busy interval of two popular real-time schedulers (EDF, RM) to intercept the scheduler type. Specifically, we trained an attention-based framework. Next, our framework extracts the schedule type of busy interval. Compared to baseline methods (LSTM), our method converges faster and provides greater precision in detection of scheduler when provided only 10 seconds of the busy interval. The results demonstrate that our approach provides more than 83% accuracy when tested on unseen busy intervals.

Index Terms—Real-time Scheduling, Time Series Analysis

I. INTRODUCTION

With the rise of common-off-the-shelf components, real-time systems are driving further toward remote monitoring and control. Real-time systems are a crucial component of our daily lives where are apparent in cars, medical devices, and smart cities. Many of these systems are designed to have a safety-critical to avoid damaging the environment or posing a threat to human safety Chen et al. (2015). For example, pacemakers are medical devices that are implemented in the heart of the patient to simulate regular pulses of the heartbeat. Recently, these real-time devices are designed to be remotely monitored by the doctor or the clinic Alkady et al. (2018). The security of such a remote connection can potentially damage or cause fatality to the human heart.

Generally, real-time systems are designed to have deterministic characteristics. For example, they are designed to provide guarantees of the task execution at determined points in time Chen et al. (2015). Hence, such care in the design process provides a set of important information about the design, purpose, and exploits of a system. Such information can then be exploited by various types of external attacks such as primary side-channel attack Kocher (1996) or secondary covert-channel attacks VölP et al. (2008). Both category of attacks often requires precise knowledge of systems so that their attack guarantees a high chance of success Chen et al. (2015).

Previous works such as Chen et al. (2015), showed that under a particular real-time scheduling protocol (rate-monotonic scheduling), the exact scheduling behavior of a real-time system could be reconstructed by only knowing few properties of a task-set. Yoon et al. (2016) also showed the practicality of a side-channel attack on a real-time system by developing a simulated attack on a UAV in a FreeRTOS environment. For the task of processor sniffing, storing a complete set of information about running real-time schedules of a processor and its environment (IO, application, memory, disk,...) is hard to gather Yoon et al. (2016); Zhang et al. (2018). But in previous studies, the interception of the schedule requires the number of tasks, their periods and execution times on top of the busy interval of the system. In practice, none of this information is available other than the busy interval that can be accessed through dozen ways such as sound, heat, or electromagnetic field emitted through the environment and can be intercepted by a sensor.

Hence, we aim to develop an attention-based neural network, namely DeepSniffer, to intercept real-time schedulers without complete a priori knowledge. In this study, we present a novel DNN sniffing model (DeepSniffer) that is designed to extract the exact RM schedules of real-time systems. The leaked schedules are then can be used to launch a side-channel attack against a specific victim task. Particularly, we aim to generate random tasks on a single processor with respect to a real-world processor. The DeepSniffer framework is designed to find the recurring and nonrecurring patterns by using the information it learns along the timeline as a uniprocessor. Furthermore, our model is robust enough when a random scheduler was applied. In the next section, we briefly discuss our system design.

Contribution. To overcome the aforementioned challenges, we make the following contributions:

- To best of our knowledge we are the first study that evaluates interceptability of various schedules.
- We provided a novel attention-based network that provides better accuracy and interpretable results when compared to LSTM networks.
- We evaluated our method on more than 1100 busy intervals scheduled on randomly generated tasks. Our framework provides up to 83% accuracy in scheduler determination. We believe that such accuracy can be translated to further interception evaluation such as number of task.
### Table I: Glossary of real-time system notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$</td>
<td>a set of $n$ real-time tasks ${\tau_1, \ldots, \tau_n}$</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>a real-time task</td>
</tr>
<tr>
<td>$p_i$</td>
<td>period of $\tau_i$</td>
</tr>
<tr>
<td>$d_i$</td>
<td>deadline of $\tau_i$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>execution time (computation time) of $\tau_i$</td>
</tr>
<tr>
<td>$pri(\tau_i)$</td>
<td>priority of $\tau_i$, smaller value represents higher priority</td>
</tr>
<tr>
<td>$V$</td>
<td>an interval set representing schedules, $V = ID + W$</td>
</tr>
<tr>
<td>ID</td>
<td>idle interval set</td>
</tr>
<tr>
<td>W</td>
<td>busy interval set ${\omega_1, \ldots, \omega_m}$</td>
</tr>
<tr>
<td>$\omega_k$</td>
<td>a busy interval</td>
</tr>
<tr>
<td>$T_s(\omega_k)$</td>
<td>start time of $\omega_k$</td>
</tr>
<tr>
<td>$C(\omega_k)$</td>
<td>duration of $\omega_k$</td>
</tr>
<tr>
<td>$N_k(\tau_i)$</td>
<td>the number of jobs of $\tau_i$ enclosed in $\omega_k$</td>
</tr>
<tr>
<td>$S_k(\tau_i)$</td>
<td>an array storing the start times for each job of $\tau_i$ in $\omega_k$</td>
</tr>
<tr>
<td>$A_k(\tau_i)$</td>
<td>an array storing arrival times for each job of $\tau_i$ in $\omega_k$</td>
</tr>
</tbody>
</table>

Table I disclose the notations in our paper.

**Organization.** The remainder of this paper is organized as follows. Section II summarizes related work and provide a description of drawbacks. Section III defines the DeepSniffer framework in details and provides the generated taskset. Section IV presents the dataset, metrics, and discusses the experimental results. Section VI concludes this paper and proposes our future research directions.

## II. Related Works

In recent years, several studies have revealed that the use of Deep Learning techniques for the task of RTS is attainable. A simple feed-forward neural network was first proposed by Nomani and Szefer (2015) for defending against side-channel attacks. Particularly, the study focused on detecting and predicting program phases. To interpret their process, the scheduler can ensure that the programs in a same program phase, are not scheduled on the same processor core, thus helping to mitigate potential side-channel attacks.

Kim (2018) proposed a feed-forward neural network for task size inference and sequence prediction. Specifically, the study leverages a feed-forward neural network for task size inference and a bidirectional LSTM architecture for sequence prediction. In Chen et al. (2015), the authors showed that under RMS, the exact scheduling behavior of a real-time system could be reconstructed by only knowing few properties of a task-set. In addition, the Stuxnet worm showed that knowing the context of task execution and knowing the occurrence of certain events was crucial in carrying out a successful attack Kim (2018). Particularly, the worm gained foothold of a target system and remained dormant until it detected that a SCADA controller is attempting to control certain equipment and thus attack. Yoon et al. (2016) also showed the practicality of a side-channel attack on RTS by developing a simulated attack on a UAV in a FreeRTOS environment.

Völp et al. (2008) demonstrated that a feed-forward neural net could be used to predict the behavior of application context switching. Such prediction then can be used to reduce the side-channel attack in multi-core processors using hardware counters. Traditional shallow methods, which contain only a small number of nonlinear operations, do not have the capacity to model such complex data accurately Hasan et al. (2017).

In Li et al. (2019), Transformer has been proposed as a brand new architecture which leverages attention mechanism to process a sequence of time series data. Unlike the RNN-based methods, Transformer allows the model to access any part of the history regardless of distance, making it potentially more suitable for grasping the recurring patterns with long-term dependencies. In addition, the authors used a Log-sparse self-attention mechanism to allow each cell of the network to only attend to its previous cells with an exponential step size and itself Zhou (2020).

## III. DeepSniffer Framework

The proposed DeepSniffer framework consists of three main components, as shown in Figure 1. In the first two components, we generate a set of randomly generated task and feed them into an Earliest Deadline First (EDF) and Rate-Monotonic (RM) scheduler. These two components are described in Subsections III-A and III-B respectively. Then we extract the busy interval of the scheduler and feed them into the DeepSniffer framework (the third component, described in Subsection III-C),
which produces the schedule type. The following subsections describe the approach of each block in detail.

A. Random Task-Set Generator

As shown in Figure 1, we first generate a set of random tasks, including high and low criticality nature. However, we are only feeding the sequence of busy and idle intervals. The DeepSniffer then attempts to extract the busy intervals’ patterns and estimate the task set size and the future sequence with its DeepSniffer predictor.

To better model the busy intervals’ tasks, one approach is to develop robust features that capture the relevant information Farhangi et al. (2019). However, in this study, we keep the information about the system minimal and look for the existing deep and shallow patterns found in the processor’s state (busy or idle) Chen et al. (2015). Lastly, we deep sniffer can leverage the learned patterns to aid in its prediction of task size and future sequence of the tasks, as shown in Figure 2. For future work, we aim to use a shuffling task algorithm to produce a random busy interval and evaluate the DeepSniffer on such a problem. As shown in Figure 2, the distribution of the task-set follows a uniform distribution of utilization.

B. RM and EDF Scheduler

The RM and EDF scheduler are programmed using the Simso library. We intend to increase the number of the scheduler to multiprocessors to further improve our study. The busy intervals are observed through the preemptions of an observer task (the compromised task with the lowest priority). The attack demonstration showed that the time cost of measuring the busy interval is small enough to be practical, and the cache footprint it leaves behind is small enough to use the cache as a side-channel. This is assuming that the adversary has a priori knowledge about the task set Kim (2018).

C. Busy interval interception

In previous works, it was shown that under RMS, the exact scheduling behavior of a real-time system could be reconstructed by knowing the number of tasks, their execution time and period Chen et al. (2015). However, such information is hard to derive even for primary and secondary attacks described in Section II. In this study, we leverage the busy interval of the system to further to derive the type of the scheduler under 10 seconds of observation. As shown in figure ??, the DeepSniffer follows a bi-directional LSTM network coupled with attention mechanism to further improve the accuracy and interpretability of the network. As there are different types of Attention, we adopt the local attention mechanism that can be developed on top of Bidirectional LSTM as an encoder and a decoder that emulates searching through a idle time of the processor. The main idea is that, given an input sequence of idle time \( x = (x_1, \ldots, x_T) \), embeds the sequence into a context vector \( c \) and then, based on this context vector, it produces the the prediciton targets such as \( y = (y_1, \ldots, y_T) \).

In our case, we the target of prediction is the scheduler type but such system can be used to derive the sequence of future tasks as well. The Bi-LSTM layer consists of two LSTM layers where the first layer reads the input sequence as it is ordered (from \( x_1 \) to \( x_T \)) and the second layer reads the input sequence in reverse order Lim et al. (2019). As a result of these two layers, for each element \( x_j \) in the sequence \( x \), we obtain two hidden states \( (\hat{h}_j, h_j) \) which we need to concatenate such that \( h_j = [\hat{h}_j \leftarrow h_j] \) Laptev et al. (2017). In our particular case, we use LSTM in the baseline method as well to provide the effect of attention on the LSTM mechanism directly.

Regarding to the context vector, it is computed as a weighted
Busy interval interception is made by observing the Idle time of the processor. Such idle time is fed into a bi-directional LSTM where the attention mechanism provides greater inference and accuracy when evaluated.

\[
c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j
\]

where \( \alpha_{ij} \) are called attention weights. The weights are computed using an alignment model which scores how well the inputs around position \( j \) and the output at position \( i \) match:

\[
\alpha_{t,i} = \text{align} (y_t, x_i) = \frac{\exp \left( \text{score} (s_{t-1}, h_i) \right)}{\sum_{k=1}^{T_x} \exp \left( \text{score} (s_{t-1}, h_k) \right)}
\]

The score function is the non-linear function \( \text{score} (s_{t-1}, h_i) = v^\top \tanh (W [s_{t-1}; h_i]) \) where both \( v \) and \( W \) are weight matrices to be learned in the alignment model. In short, it implies that the index \( t \) in the previous equations is always one as we have the type of schedule as our output (Makridakis et al. 2020).

### IV. Experiments

In this subsection, we discuss the parameters of the task set generator. Then, we provide the metrics of the evaluations shortly and provide an evaluation of the baseline methods such as LSTM compared to DeepSniffer. Specifically, we first generate a set of tasks with equal utilization but different parameters. Furthermore, we mix the utilization between tasks by randomly selecting two tasks and moving one percent of the total utilization from one to another. Lastly, the algorithm selects the periods and offsets and calculates the execution time. The evaluation metrics are detailed as below:

\[
\text{Accuracy} = \frac{T_p + T_n}{T_p + T_n + F_p + F_n},
\]

where \( T_p \) denotes the true positive predictions.

\[
\text{Precision} = \frac{T_p}{T_p + F_p},
\]

\[
\text{Recall} = \frac{T_p}{T_p + T_n},
\]

\[
F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]

As shown in Table III, the DeepSniffer provides greater accuracy when compared to the LSTM method. To our surprise the method is capable of providing over 99% accuracy in the detection of EDF schedule. Figure 5 also shows the attention weights for the first 256 Bi-directional LSTM cells.

![Attention Weights](attachment:attention_weights.png)

**Fig. 5**: Attention weights are distributed more on the earlier part of the Bi-directional LSTM neurons.
weights of the bi-directional LSTM neurons, where the earlier neurons are holding more weights when compared to the later parts.

V. Conclusion

In this work, we developed an attention-based neural network namely DeepSniffer to intercept real-time schedulers without complete a priori knowledge. We showed that the using only the busy interval of the system (which can be gathered by noise, electromagnetic field or heat) one can use our framework to derive the schedule type of the system. Specifically, we showed that under EDF schedule, the DeepSniffer has over 99% accuracy when tested on unseen busy intervals. In future, we aim to extend our study to other important information such as the number of tasks, the schedule of the system and memory utilization. All of these informations can be can be used to launch a side-channel attack that needs to be addressed in future.

References


