Cloud-based cyber-physical intrusion detection for vehicles using Deep Learning

George Loukas, Tuan Vuong, Ryan Heartfield, Georgia Sakellari, Yongpil Yoon, and Diane Gan

***Abstract*—Detection of cyber attacks against vehicles is of growing interest. As vehicles typically afford limited processing resources, proposed solutions are rule-based or lightweight machine learning techniques. We argue that this limitation can be lifted with computational offloading commonly used for resource-constrained mobile devices. The increased processing resources available in this manner allow access to more advanced techniques. Using as case study a small four-wheel robotic land vehicle, we demonstrate the practicality and benefits of offloading the continuous task of intrusion detection that is based on deep learning. This approach achieves high accuracy much more consistently than with standard machine learning techniques and is not limited to a single type of attack or the in-vehicle CAN bus as previous work. As input, it uses data captured in real-time that relate to both cyber and physical processes, which it feeds as time series data to a neural network architecture. We use both a deep multilayer perceptron and a recurrent neural network architecture, with the latter benefitting from a long-short term memory hidden layer, which proves very useful for learning the temporal context of different attacks. We employ denial of service, command injection and malware as examples of cyber attacks that are meaningful for a robotic vehicle. The practicality of the latter depends on the resources afforded onboard and remotely, as well as the reliability of the communication means between them. Using detection latency as the criterion, we have developed a mathematical model to determine when computation offloading is beneficial given parameters related to the operation of the network and the processing demands of the deep learning model. The more reliable the network and the greater the processing demands, the greater the reduction in detection latency achieved through offloading.**

I. Introduction

Over the last few years, there have been multiple examples of both proof of concept and real-world attacks against vehicles. While in the past, these would be focusing primarily on unlocking automobiles and defeating their immobilisers' cryptographic protection, they have now progressed into being increasingly cyber-physical [1], affecting the integrity and availability of core physical functions, including steering, accelerating and braking. In 2010 and 2011, researchers from the University of Washington and University of California San Diego [2], [3] were the first to demonstrate highly practical wireless attacks on a common production automobile, able to affect several of its core functions, including disengaging the brakes or selectively engaging them on only one side of the vehicle while driven at high speed. Since then, several other researchers have showcased attacks on a variety of automobiles, and even time-constrained automotive hacking competitions have taken place within security conferences. As a result, automotive cyber security is now considered a primary concern in the industry [4] and an effort has been made to document a variety of exploitable automotive security vulnerabilities [5]. At the same time, reports of attacks on U.S. drones have emerged during the wars in Afghanistan and Iraq [6], as well as in relation to the alleged hijacking of a drone by the electronic warfare unit of the Iranian army [7]. Since then, researchers have turned their attention to the cyber security of unmanned vehicles too, including GPS spoofing [8] as well as sensory channel attacks exploiting the physical limitations of the vehicles' LIDAR systems [9] and gyroscopes [10]. Security becomes even more important as driverless automobiles are becoming a reality, because ensuring their safe and uninterrupted operation is key for their acceptability by the public. In the meantime, there is also increasing interest in the security of industrial, reconnaissance, rescue and other types of robotic vehicles, each with their own vulnerabilities and scale of physical impact of cyber attacks against them [11], [12].

While vehicles may differ enormously in terms of their type, size, operation and how safety-critical they are, most tend to share the following characteristics which make them challenging to secure: (a) Any cyber security functions on them are resource-constrained, either because of lack of processing power or because minimising energy consumption has priority; (b) most cyber-physical operations that an attacker would target are time-critical, especially if they affect mobility; and (c) unlike cyber threats to conventional computer systems, which have been meticulously observed and statistically analysed for decades, threats here are largely unknown, and consequently there are no meaningful datasets to be used for benchmarks.

In combination, the above challenges mean that one cannot rely on purely preventive security measures, such as cryptography, but needs to assume that some attacks will go through and will need to be detected by the vehicle or its operator quickly and accurately, using only the very limited resources that can be afforded for an intrusion detection system (IDS). One approach for this is to include an onboard intrusion detection module, which would be trained offline to learn simple rules in relation to its own normal behaviour [13], [14] or to the signatures of different types of known attacks based on different monitored features. When in actual operation, the vehicle would then monitor these features and apply the simple rules, which would have a relatively low processing load. This approach can work well for simple and previously seen attacks. It can have low detection latency and relatively high accuracy [15], [16], but can be considerably less effective when it encounters unusual conditions or complex attacks.

An alternative approach, which we evaluate here is to run the intrusion detection process not onboard, but offloaded

to a powerful external processing system, such as a cloud infrastructure. This can reduce the processing load on the actual vehicle, but importantly, it can also allow leveraging much more complex intrusion detection techniques, for instance involving deep learning. In the following sections, we provide the related work in terms of vehicular intrusion detection (Section II), our deep learning based approach to intrusion detection (Section III), and our testbed, experimental methodology and results for different configurations (Section IV). We evaluate the practicality of offloading detection based on a mathematical model, which we validate experimentally (Section V-C). We conclude in section VI with a summary of our findings and recommendations for future work.

II. Related work

Although relatively recent as a scientific problem, there have already been some first attempts to provide intrusion detection for vehicles, primarily for manned and unmanned aircraft, robotic vehicles and driverless automobiles. Some intrusion detection approaches are highly specialised, focusing on specific aspects of a vehicle's communication, actuation or sensing, while others take a more holistic view of its health across all three.

In terms of in-vehicle communication, recent work by Cho and Shin [17] has shown that it is possible to infer the origin of an attack on an in-vehicle CAN bus network (e.g., the particular Electronic Control Unit inside a car) from its voltage profile, the fingerprint of which has been learned by their detection system. Although relatively narrow in scope, performance evaluation in two real cars has shown that there is great merit in learning the normal behaviour of different aspects of a vehicle, so as to detect and pinpoint attacks. Along similar lines, Moore et al. [18] have developed an algorithm to detect anomalies in the CAN bus network traffic by monitoring the refresh rates of certain commands, which are shown to be indicative of signal injection attacks. Tackling the same problem, Martinelli et al. [19] have argued that normal CAN messages are triggered by human action, and as such can be modelled by fuzzy techniques. So, they have developed a technique based on fuzzy-rough nearest neigbour classification to distinguish between legitimate CAN messages generated by the human driver and injected ones generated by an attacker.

In terms of external communication, Lauf et al. [20] and Strohmeier et al. [21] have focused on the automatic dependent surveillance-broadcast (ADS-B) protocol used by aircraft to periodically broadcast their position and other situational data to other aircraft and air traffic control ground stations. An ADS-B spoofing attack could severely compromise the safety of the aircraft. To spot signs of such spoofing attacks, the method proposed by Lauf et al. looks for suspicious peaks in the probability density functions of types of data requests between nodes, as well as any behaviour correlations that would indicate cooperation between intruders. However, their assumption is that direct data requests between aircraft are possible, which is not currently the case in ADS-B. In [21], Strohmeier et al. have used as input features only statistics related to the received signal strength (RSS), assuming that the RSS of spoofed ADS-B signals coming from an attacker on the ground would differ considerably to that of legitimate signals coming from aircraft.

Other approaches focus on the actuation of a vehicle and especially what makes it a vehicle, which is its movement control. For example, Birnbaum et al. [22] have proposed a prototype monitoring system geared specifically towards detecting hardware failures, tampered hardware and suspicious behaviour of the flight control computer. It captures data on roll, pitch, yaw, and servo motor control parameters, such as elevator, throttle, rudder and aileron. Their approach adapts the recursive least squares method as an estimator of airframe and controller parameters. By establishing a set of parameter estimations for each drone's control law, the monitoring system can compare the parameters in between flights using normalised root mean square deviation. A large value for the latter indicates a significant difference in parameters and as such can be used as the basis of an anomaly detection system. For experimental verification, the approach has been tested on open-source flight simulation platforms.

Beyond communication and actuation, a vehicle's safe operation also depends on sensing. Especially autonomous vehicles are almost entirely dependent on the robustness of their sensing processes. This makes them particularly attractive targets to sensory channel attacks and network-based false data injection attacks that affect the integrity or availability of a vehicle's sensor data, for instance, to disrupt its collision avoidance subsystem. One approach that is commonly used to detect attacks on sensors is to treat them as standard sensor failure events and utilise statistical anomaly detection methods. For instance, if it can be assumed that the rate of change of a sensor's data cannot exceed a particular value, then the recursive least-square filter can be used to discard data that do. Gwak et al. have demonstrated this approach on a small robotic vehicle whose obstacle avoidance does not have the luxury of cross-checking between different types of sensing and is limited to only ultrasonic sensors [23]. The simple approach followed is that if a sensor's data are deemed to be unreliable, the particular sensor is excluded from the vehicle's collision avoidance procedures.

Other researchers have addressed intrusion detection more holistically, looking at the wider picture of a vehicle's state. Vuong et al. [11], [15], [16] have focused on denial of service, false data injection and different types of malware attacks against a robotic vehicle. Their detection method is based on decision trees with a training phase that involves a range of attacks and measures their impact on a set of cyber and physical features. Unsurprisingly, it is the cyber features that are the most relevant, especially network-related ones, but it has also been shown that introducing physical features too, such as battery consumption and physical vibration of the chassis, can noticeably improve the accuracy of the specific detection method. In particular, physical vibration appears to be the result of continuously losing network connection to the remote controller of the vehicle, and as a result often entering fail-safe mode for brief durations. Spotting these physical manifestations early helps reduce detection latency. One of the most complete solutions for a drone's onboard

security monitoring framework is R2U2 by Schumann et al. [24], which monitors traffic on the flight computer and communication buses, including inputs from the GPS, the ground control station, sensor readings, actuator outputs, and flight software status. It looks for commands that should not be run because they are nonsensical, repeated, ill-formatted, illegal commands or otherwise dangerous (e.g., "Reset Flight Software" while in-flight), and also monitors system behaviour, including oscillations of the aircraft around any of its axes, deviation from flight path, sudden changes or consistent drifts of sensor readings, as well as memory leaks, real-time failures and other unusual software behaviour. Probabilistic security diagnosis is based on a bayesian network engine. For instance, if barometric measurements and laser altitude coincide, any transients in GPS signal strength would indicate a likely attack. Implementation on a reconfigurable field-programmable gate array and performance evaluation on a NASA DragonEye drone has provided promising results in detecting spoofed GPS signals and malicious commands sent to the aircraft. Bayesian networks have also been used by Bezemski et al. [25] to classify the nature of the source of an attack (cyber or physical) on a robotic vehicle that uses the United Kingdom's Generic Vehicle Architecture approach for military vehicles [26].

A very different family of intrusion detection techniques is behaviour specification, where it is an expert user that specifies the rules of what is normal. For instance, the rules used by Mitchell and Chen for drones [27] included that weapons need to be disarmed outside the target area, that minimum thrust is used when loitering, that information is only transmitted to whitelisted destinations etc. These are transformed into state machines, where the "attack state" is the result of violation of any of the specified behaviour rules. In their evaluation, their state machine consisted of 165 safe and 4443 unsafe states, with probabilities assigned for going from one state to another one, and using binary grading for each state (completely safe or completely unsafe). Then, the proportion of time a device is in safe states is a measure of the degree of compliance to the behaviour rules. This was extended in [28] where the authors showed more thoroughly that the approach offers the flexibility of adjusting the strength of detection to reduce false negatives at the expense of increasing the false positives. The main disadvantage of this approach is that it typically needs an impractically large number of states to be specified to accurately capture all safety specifications for all altitudes, environmental conditions etc.

An aspect of robotic vehicles that is of increasing interest, whether they are autonomous or not, is their participation in teams that coexist in the same physical space and share a set of common interaction rules. For example, when drones detect a possible collision, the interaction rule may be "turn right", but this assumes that a drone will always follow the rules, which is not the case in a cyber intrusion scenario. Martini et al. [29] have developed a distributed misbehaviour detection mechanism to be run onboard each drone, based on a Boolean consensus protocol on the events as they are observed by them. Each drone uses its own sensors and information from its neighbours to predict the allowed trajectories that another drone can follow if it abides by the interaction rules. If its actual trajectory does not match the predicted one, then that drone is deemed as uncooperative. The particular approach has been tested experimentally with success on a team of four real drones, including a misbehaving one. However, extending any conclusions to larger teams with several misbehaving ones requires rigorous analysis of the consensus mechanism based on the Byzantine Generals problem [30]. In the same space, but for land vehicles, Alheeti et al. [31] have looked into the use of vehicular ad hoc networks for the communication between driverless automobiles, raising the question of what happens when one of the vehicles misbehaves and launches a cyber attack on other vehicles on the same network. In this initial work, they have used NS-2 and mobility simulation tools to evaluate the performance of an intrusion detection system based on artificial neural networks. Its input features were some of the ones typically used in standard network intrusion detection systems, such as payload sizes, hop counts etc. The same authors have extended their work to take into account magnetometer sensors [32] and gyroscopes [33], and address grey hole attacks [34].

A common characteristic of all detection mechanisms provided above, whether local or distributed across multiple vehicles, is a focus on minimising the processing load, either by applying lightweight techniques from statistics or by predefining simple behavioural rules that are easy to monitor. This is because they are all limited by the onboard capabilities of the vehicle at hand. As a result, they usually cannot leverage modern classification techniques, such as those currently developed in the field of deep learning. In effect, the stronger the detection algorithm, the greater the energy consumption, and in turn the less attractive a solution is for a resource-constrained vehicle. To overcome this limitation, we turn to the emerging field of cloud robotics [35]. Our proposal is to offload the bulk of the processing required to benefit from deep learning to a more powerful infrastructure (whether a single server, cloudlet or cloud). By computation offloading, we refer to the process of executing certain computational tasks on more resourceful computers which are not in the user's immediate computing environment. The concept has similarities with the online forensics techniques used for cloud-based detection of malware and tainted data on Android smartphones [36], [37]. However, instead of crowd-sourcing detection, we focus on utilising computational offloading to allow access to deep learning based techniques without the processing and energy cost which would otherwise be prohibitive for a vehicle. Figure 1 illustrates the conceptual difference between onboard and offloaded intrusion detection for a vehicle. In both cases, data collection and aggregation occurs on the vehicle. In the onboard case, the reasoning (the analysis of the data to determine whether there is an attack or not) is also on the vehicle. In the offloaded case, the data aggregated onboard are sent via a network to a cloud infrastructure or equivalent to perform the reasoning.

Over the past couple of years, the growing maturity in deep learning algorithms has led to wider use outside of its traditional applications in image and natural language processing, e.g. in detecting malware [38] or rogue certificates

from trusted certificate authorities [39]. It has also been used to improve intrusion detection accuracy in traditional computer networks [40], [41], but not for cyber-physical systems, such as vehicles. An exception is the recent work by Kang et al. [42], which is geared towards the automotive industry and detection of attacks on CAN bus. While a promising start, the particular work is limited to a generic command injection attack, which is detected by monitoring a single type of data source and using a simple and generalist deep neural network architecture, which does not account for the overall state of the vehicle through multiple features or for temporal information (the fact that the impact of the attack changes over time during the attack). Also, it has been evaluated only in simulation.

Here, we progress considerably further with the following key contributions:

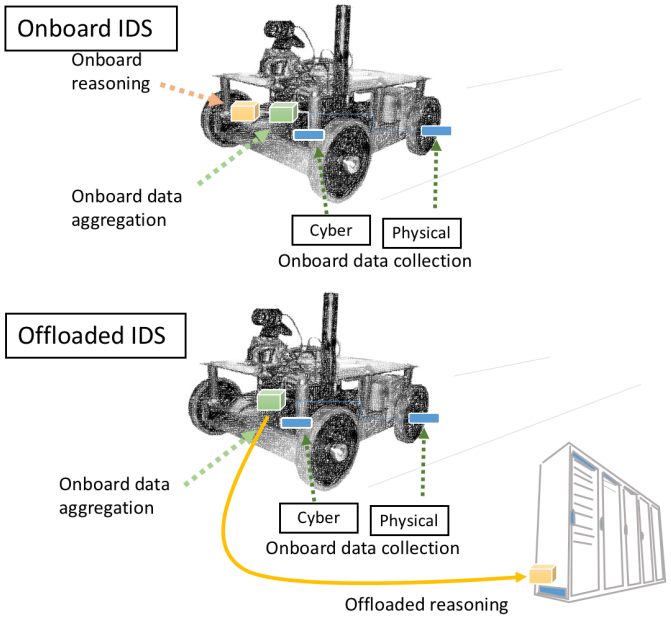
- We design and evaluate two neural network architectures for the real-time analysis of multiple sources of data collected periodically onboard the vehicle and representing both its cyber and physical processes.
- We produce a prototype implementation of deep learning based intrusion detection for cyber-physical attacks on a real robotic vehicle, tested for three different types of attacks and compared against the best-performing generalist machine learning techniques typically used in intrusion detection.
- We evaluate both experimentally and using a mathematical model the practicality of a computational offloading configuration for providing resource-constrained cyber-physical systems with access to high-end intrusion detection.

Note that an early version of this work has been included in the second author's PhD dissertation [43]. Here, we expand through evaluation of two deep learning approaches, a more practical setup with a single deep learning model for all attacks rather than individual for each attack, as well as evaluation on an unseen attack and an updated literature review.

III. Cyber-physical intrusion detection using deep learning

In real-world cyber-physical attacks, interactions between sensors, actuators and computational components often exhibit temporal correlations based on complex time dependencies, of arbitrary length. For example, in a cyber-physical system such as a robotic vehicle, a rogue operator executing remote command injection may command the vehicle to accelerate forward. As a result, this may cause a spike in network traffic leading to a change in vehicle wheel speed, which increases power consumption and current. Here, these feature interactions may occur one after the other in a specific sequence. The result of such sequential temporally-related behaviours leads to the generation of time series datasets with the potential for high-dimensional inputs that change over time. For feed-forward neural networks this type of temporal information can be lost, because they look for occurrences of the same patterns in the feature-space based on current state, irrespective of the prior input patterns that came before. By comparison, recurrent neural networks exhibit dynamic temporal behaviour

Fig. 1: Conceptual comparison between onboard and offloaded intrusion detection for a vehicle



by using internal memory to process sequences of inputs based on interconnected hidden layers from previous input states, feeding the hidden layers from the previous states as an additional input into the next state. In this manner, recurrent neural networks are trained based on historic and current input, where the likelihood of an attack occurring depends both on prior states of the features and the current states of the features at that point in time.

As cyber-physical attacks occur as a series of both cyber and physical events over time, we have chosen to evaluate the approach of using a recurrent neural network approach for the development of our cyber-physical IDS, which has proven highly appropriate for handling multivariate sequential time series data [44], [45]. The approach taken for construction of the deep learning IDS starts with launching different types of cyber attack against the robotic vehicle and collect data with regards to a series of features, appending the ground truth labels based on the timings of the attacks (whether an attack was really in action at each point in time or not). As the data from different features come at different times and at different sample rates, we synchronise them in a pre-processing phase. The output of pre-processing is a data stream with a data point sample interval τ. In the learning phase, the data is split into a training set and a validation set. The recurrent neural network algorithm is applied on the training data to produce a detection model, as defined by the weights of the connections between neurons. The model is then validated using the validation set before producing the final classifier, which is evaluated experimentally using real-time testing data. (The precise configuration parameters are summarised in Table I.)

Figure 2 shows the recurrent nature of the learning process, where $X^{(t)}$ is the vector of input features and $Y^{(t)}$ is the binary detection decision (0 if no attack, and 1 if attack) at

Fig. 2: learning process using recurrent neural network

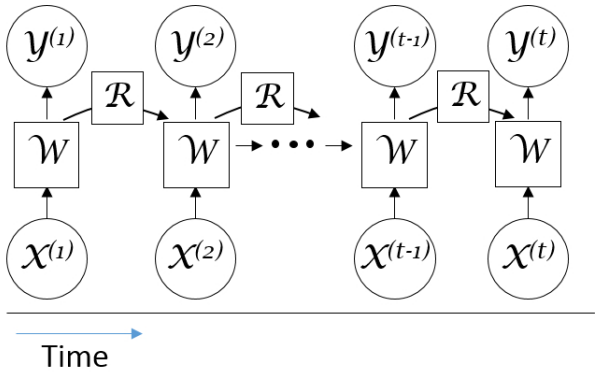
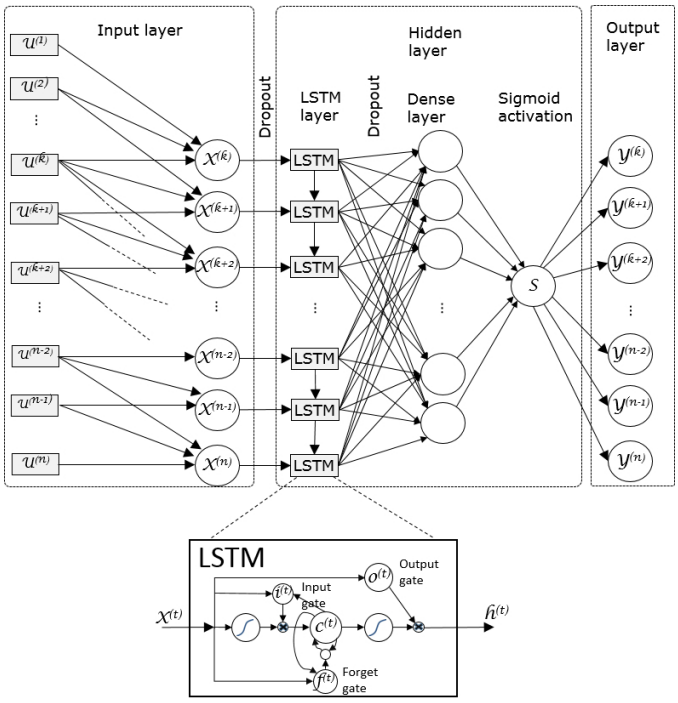


Fig. 3: RNN with LSTM deep learning architecture



time t, and $Y^{(t)} = WX^{(t)} + RX^{(t-1)}$, with W and R being the weight matrices in relation to X and the incorporation of the output of the previous step respectively.

In terms of the deep learning architecture designed for our intrusion detection methodology, in the input layer, $U^{(1)}$, $U^{(2)}$, ..., $U^{(n)}$ is the time series dataset in a period T, which corresponds to $n = \frac{T}{\tau}$ data points. We group k consecutive data points together into $X^{(k)}$, $X^{(k+1)}$, ..., $X^{(n)}$, as shown in Figure 3. The purpose of grouping is to help the algorithm have a picture of more than a single point in taking a detection decision, but without using the whole dataset (of n data points in one period T) either, which would increase considerably the detection latency. So, $1 \leq k \leq n$. The hidden layer includes a Long-Short Term Memory (LSTM) layer, a dense layer and Sigmoid activation. Conventional recurrent neural networks find it difficult to train with long step sizes due to the"vanishing gradient" problem in gradient-based activation functions (such as sigmoid or tanh). The vanishing gradient relates to the exponential decrease in the size of the gradient (from which the network learns changes in the input parameters which effect the expected output) by iteratively mapping large input regions into smaller output regions through sequential layers or long inputs sequences in the neural network [46]. As a result, when the gradient reaches a value near zero, the recurrent nature of the neural network produces small outputs even for large changes in input and as a result affects the ability of the neural network to learn from early layers or inputs in long training sequences or over many hidden layers.

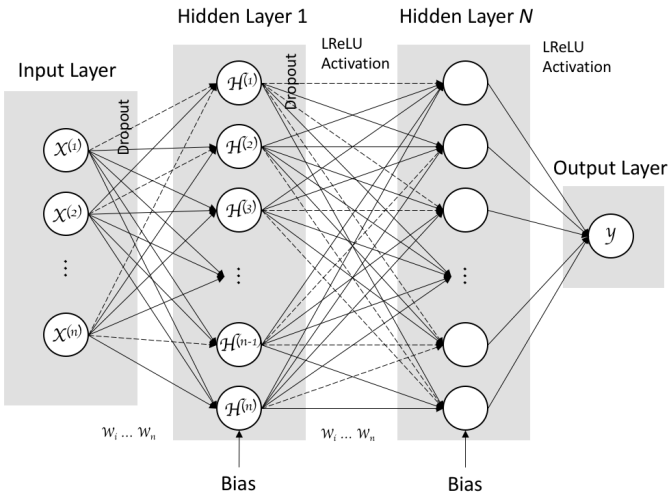
LSTM helps solve this problem by employing a "gating" function (1 to remember the input and pass it to the next hidden node/layer or 0 to forget the input) that replaces the activation function. The network is trained on the combination of the gates in the network and as long as the gates are 1 along the input sequence or across all hidden layers in the network, the network can remember the values of early input to identify how it affects the expected output. We use a standard LSTM architecture, with each block containing gates that determine the significance of the input and whether it should continue to remember its value or forget it, and when it should output it. The LSTM layer is followed by a dense layer, where the number of hidden nodes serves as our main tuning parameter. Then, a sigmoid activation function converts the values produced by the dense layer into real values between 0 and 1. Finally, a binary detection decision (0 or 1) is taken based on a predefined threshold. Below this threshold, the detection decision is that there is no attack, and above it, that there is an attack.

Alongside our RNN deep learning approach, we also develop a deep multi-layer perceptron (MLP) classifier as an example feed-forward neural network to directly compare the detection performance between a deep learning architecture which benefits from learning time-based sequences and one that does not (RNN with LSTM *Vs.* MLP). Unlike RNNs, feed-forward networks send signals in one-direction from input to output with no feedback loops; the output of any layer in the MLP classifier does not affect that same layer, as connections do not form as a cycle as in RNNs.

Figure 4 shows the MLP architecture adopted here. It consists of a input layer, one or more hidden layers with Leaky Rectified Linear Units (LReLU) as an activation function and an output layer. For MLP neural networks, ReLU activation is a preferred method over sigmoid to avoid the vanishing gradient (in logistic gradient descent activation) occurring over multiple hidden layers in the MLP network. A ReLU function output is either 0 for input than is smaller than 0 (e.g., negative) or 1 for input that is larger than 0 (e.g., positive), which allows a MLP network to employ multiple layers without neuron gradients being saturated for input.

However, one problem with ReLU activation, known as "dying ReLU", results when ReLUs (neurons) are activated with a value of 0 which forces their gradient to be set to zero (i.e., inactive) in back-propagation (e.g., all inputs). This means that potentially large numbers of neurons in a network "die" as they are stuck in an in active state, between layers, which results

Fig. 4: MLP deep learning architecture



in decreasing model capacity as they are no longer used. To avoid this, one method is to adjust the network learning rate through methods such as dropout, or by utilising LReLUs. Instead of setting ReLU neurons to 0, LReLUs assign non-zero gradient (e.g., 0.01) when a neuron is not active, which allows the neuron to remain active for negative input between each layer of the MLP. Inline with our RNN approach, for the MLP classifier, we also introduce input and hidden layer dropout to reduce overfitting during network training. During training of the MLP, we have used the same number of neurons tested within the RNN LSTM modelling, and test up to three hidden layers to establish whether increasing the network depth and non-linearity of the deep learning architecture improves attack detection performance over RNN, which is designed with a single hidden layer using LSTM gates.

For the deep learning development and implementation, we have used Keras [47] to run on top of TensorFlow/Theano library. Keras is an extensible Python neural network library that supports fast prototyping.

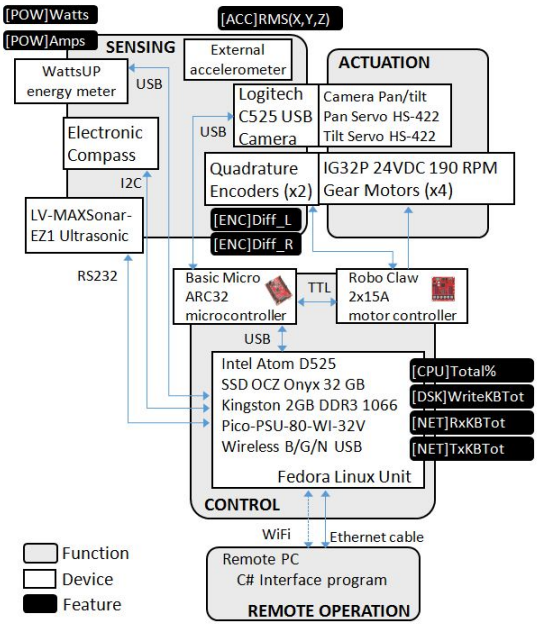
Attack-specific supervised signature-based IDS models' can be impractical and inefficient, as the attack surface grows and the potential attack configurations increase in number. Semi-unsupervised and unsupervised IDS models do not require attack vectors to be present within training data, but provide low detection accuracy for identifying specific threats (which help identify the source and type of attack), because training either develops a concept of normal operation (semi-unsupervised), or is used to express clusters of membership between data points (unsupervised). Furthermore, if a dataset for classification is particularly noisy, this can have a dramatic effect on performance, especially as there is little (semi-unsupervised) to no (unsupervised) guiding information as to potential attack states. We have opted to design our deep learning IDS using a supervised signature-based detection model. To increase practicality, we develop a single deep learning model trained on a dataset that comprises all three types of attacks considered here. The presence of multiple attack signatures within the model form the basis of being able to determine whether the vehicle is under attack or not, but with the increased intelligence of potentially understanding what type of attack it might be.

IV. Experimental Methodology

A. Testbed: Robotic Vehicle

The testbed we have developed for the experimental evaluation is a 40 cm long remote-controlled 4x4 robotic vehicle, with an on-board computer based on a dual-core Intel Atom D525 CPU and 2 GB of DDR3 RAM, running the Fedora operating system. The vehicle's motors are controlled by an Arduino micro-controller. Network access is via Wi-Fi or Ethernet cable, and remote control is over a TCP socket to the vehicle's control board. In addition to network traffic related to control of the movement of the vehicle, there is also network traffic generated by the onboard camera video streaming and control of its actuation (pan and tilt).

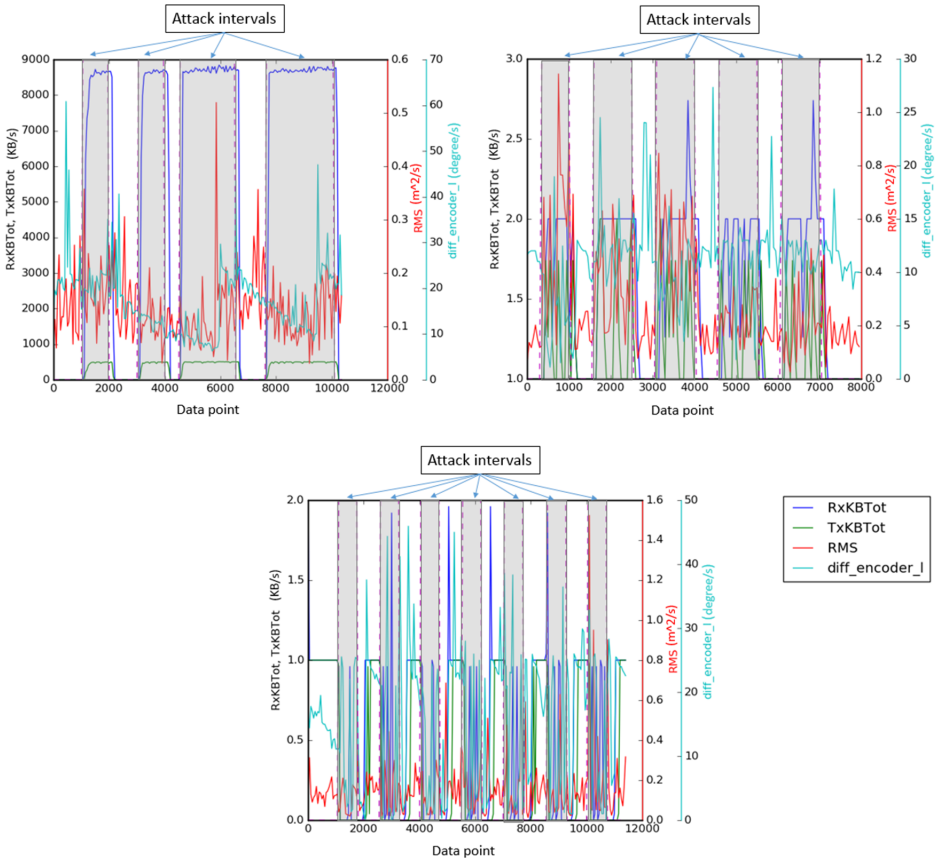
Fig. 5: The robotic vehicle architecture. The input features used for the detection are shown in black background



B. Features

It has been previously shown that for cyber-physical systems, taking into account also the physical manifestation of cyber attacks on the vehicles can improve detection accurady and reduce detection latency [15]. In other words, the adverse impact of a cyber-physical attack can also be seen as an opportunity for improved intrusion detection. Here, we adopt the same approach, which allows observing an attack's impact on computation, communication and physical operations of the vehicle. We focus on types of data that are available and can be extracted on most vehicles without considerable overhead. We have identified eight input features, four related to communication and processing, which we refer to as the cyber input features, and four related to the physical properties of the robot, which we refer to as the physical input features. The attack label is the ground truth for the scenario.

Fig. 6: The impact of the three attacks on an indicative selection of the features monitored, for Denial of Service attack scenario (top left), Command Injection attack (top right), Malware (Net) attack (bottom)



- **Network Incoming:** Received network traffic rates.
- **Network Outgoing:** Transmitted network traffic rates.
- **CPU:** The total CPU utilisation.
- **Disk Data:** The rate of data being written to the disk.
- **Encoder:** Magnetic encoders have been fitted to the rear wheel motors, which provide real-time values of their angular position. The speed is represented by measuring the difference between two consecutive encoder value readings in a fixed period of time. The default value for this period is 33 ms.
- **Accelerometer:** Represents the vibration of the chassis (using accelerometer measurements). An external device has been attached to the chassis to capture these readings.
- **Power:** The overall power consumption of the vehicle as measured by a *WattsUp?* energy meter[1].
- **Current:** The overall current drawn by the vehicle.
- **Attack label:** This is the ground truth label, which states whether there is an attack or not at a particular point in time. This is used to train the model and also to evaluate its performance.

C. Attacks used in training the model

As representative of a wide range of possible families of attacks against a robotic vehicle, we have conducted experiments where the robot is under denial of service (DoS) attack, command injection attack, and malware attack targeting the network interface. The attacks are intermittent. They appear in-between sections of normal operation, where all network traffic and applications running are legitimate, and correspond to the operator's legitimate interaction with the vehicle, including commands and sensor values exchanged. Their precise timings are highlighted in Figure 6 in the form of grey sections (the attack intervals).

- **Denial of Service (DoS) attack**. Here, the aim of the attack is to flood the vehicle's network interface with TCP traffic to disrupt the communication between the legitimate operator and the vehicle. In the particular testbed, an attack rate of 100 Mbits/s was sufficient to overwhelm the communication channel. The resulting intermittent connectivity causes the vehicle to trigger temporarily its fail-safe mechanism, which is simply to stop when communication is lost, and then resume its movement. This causes intermittent physical vibration of very short duration. The top left in Figure 6 shows the effect of the attack on some of the features monitored on the vehicle. Naturally, for a denial of service attack based on volumes of network traffic, it is Network Incoming (in the figure, referred to as *RxKBTot*) that is the feature most obviously affected during an attack, but even physical features (e.g., RMS value) seem to be affected by the

[1] www.wattsupmeters.com

accompanied vibration.

- **Command injection attack**. The robot receives commands from its legitimate operator to move forward, and at the same time receives rogue "stop" or "turn left" commands from an attacker. The conflicting commands cause consistent and frequent physical jittering, as the vehicle attempts to process and act on both commands within very small periods of time and continuously. This effect can be observed in the top right graph of Figure 6, especially in relation to the instantaneous speed value for each wheel, as represented by the encoder value (in the figure, referred to as *diff_encoder_l*), as well as by the very high RMS values, which are the result of the consistent physical jittering.
- **Malware attack**. A piece of malware already installed in the robot's onboard control software utilises the Linux kernel's network scheduler to modify the network traffic control setting and introduced network delay. The result in physical space is that the robot's movement becomes erratic with frequent and relatively consistent stops during the attacks. The bottom graph in Figure 6 illustrates the effect of the attack on some of the features monitored on the vehicle, as captured experimentally.

D. Experimental deep learning results

We have designed the recurrent neural network (RNN) architecture as shown in Figure 3, with the configuration parameters of Table I; here we have used the same parameters for the MLP architecture with the exception of the activation function and sequence group (k) which is not relevant to MLP. As primary metric of the performance of the intrusion detection model, we have used accuracy $ACC = (TP + TN)/(TP + FP + TN + FN)$, where TP corresponds to the true positives (correct detection of attack), FP to the false positives (incorrect detection of attack), TN to the true negatives (correct detection of non-attack), and FN to false negatives (incorrect detection of non-attack).

1) Performance against previously seen types of attacks: Here, we evaluate the performance of the model in real-time as it is exposed to the three types of attacks that the model has already seen. So, the actual data collected are new, but the same types of attacks have been seen previously in training and validation. In terms of the sizes of the dataset collected for each attack, attack and non-attack intervals in each experimental run. The denial of service experiments produced 3,114 data points, of which 2,451 were distributed in four attack intervals and 663 were non-attack data points in intervals between the attacks. The command injection experiments produced 3,432 data points, of which 1,402 were in five attack intervals and 2,030 were non-attack data points in intervals between attacks. The malware (Net) experiments produced 2,390 data points in seven attack intervals, of which 950 were attack and 1,439 were non-attack data points in intervals between attacks.

Figure 7 shows the overall accuracy of the RNN LSTM model against the number of neurons in the hidden layer. We vary the number of neurons from 600 to 1,000 neurons. We see that 800 neurons are sufficient to achieve high average detection accuracy (85.7%) across the three previously seen attack types, and indeed accuracy does not increase as we increase the number of neurons to 1,000. Figure 8 provides a comparative measure for the deep MLP model.

Parameter	**Value**
Interval T	1 s
Single point sample interval τ	0.02 s
Sample size n per interval	50
Grouping k	10*
Number of hidden nodes	600, 800, 1000
Number of epochs	300
Dropout ratio	0.3
Validation ratio	0.3
Loss function	Binary cross entropy
Optimiser	adam
Activation function	Sigmoid*, LReLU**
Metric	Accuracy (ACC)
Decision threshold	0.5

*RNN **MLP

TABLE I: Deep learning training parameters

We also compare the detection accuracy of the deep learning approach against standard statistical machine learning algorithms, which are considered safe generalist choices for classification problems across a range of domains, including intrusion detection. A comprehensive study by Delgado et al. [48], which put to the test 179 different machine learning classifiers across 121 different datasets from different areas of application, showed that random forest was the overall best performing technique, with an average accuracy of 94% and reporting over 90% accuracy across 84% of the datasets, followed closely by Support Vector Machines (SVM) with an average of 92% accuracy. Random forest and SVM are commonly used in intrusion detection [49], and for this reason, in Table II, we compare our deep learning based approach with random forest, two SVM variants (with radial kernel and with linear kernel), as well as with the standard lightweight approaches of using logistic regression and decision trees (C5.0) previously utilised for cyber-physical intrusion detection in [15].

Logistic Regression employs Bernoulli distribution to estimate the probability of a binary response based on one or more independent features and their relationship with the attack label. SVMs employ the concept of a hyperplane in n-1 dimensional space that best separates two classes of data points with the maximum margin. In SVM, data points that support either side of the hyperplane are the "support vectors" and in cases where these data points are not linearly separable, are projected to a higher dimensional space where linear separation is possible. In our case where multiple classes are present (multiple feature variables and a dependent variable), a one versus many binary classification approach is taken. C5.0 functions as a decision tree classifier employing Boolean logic in series of decision rules, inferred by the feature data, to determine which class the data belongs to. Random Forest is

an ensemble tree classifier that trains a number of decision trees with different re-sampled versions of an original dataset, reducing the high variance inherent in a decision single tree and improving the generalisation of model performance by averaging the standard error of all trees across the ensemble in order to produce a final model with low variance.

In our experimental comparison, the deep learning based approach achieves the highest overall detection accuracy rate as an anomaly-based supervised classification model reporting an average classification accuracy of 86.9% compared to SVM with radial kernel at 79.9%. Surprisingly, the SVM radial classifier outperformed the deep MLP overall, and for two out of three of the attacks (DoS, command injection). For individual classification, deep learning with RNN using LSTM outperformed all ML algorithms for detecting command injection with an accuracy of 83.2% compared to random forest at 78.6% accuracy. However, for the deep MLP, detection accuracy for command injection was the second lowest of all algorithms only beating logistic regression by 1%. The RNN LSTM deep learning model effectively equaled decision trees (C5.0), reporting a detection accuracy of 82.2% compared to 82.4% for the network malware attack (wht MLP close behind at 80.7%), but was slightly worse than SVM with a radial kernel for detecting the network denial of service attack, with 95.4% accuracy compared to SVM's 97.4%. By comparison, the deep MLP outperformed the RNN LTSM deep learning classifier for DoS attack detection by 0.8%. Overall, these experimental results give a good indication of utilising the RNN deep learning models' general ability to perform accurate detection across a range of different attacks when compared to more lightweight machine learning techniques, which were much less consistent across the three attacks tested. Furthermore, the capability of the RNN LSTM to learn attack behaviour over time proved superior to the MLP deep learning architecture. However, this consistency achieved by deep learning with a RNN comes at the expense of greater processing requirements and consequently long processing times.

Note that in Figure 7, for the unseen type of attack (bottom curve of the figure), increasing the number to 1,000 neurons proves useful; for the deep MLP this was not the case, adding a second layer improved accuracy slight which also decreased when adding further hidden layers. We detail this part of our experimentation in the next subsection.

2) Evaluating the deep learning IDS on an unseen type of attack: Here, we introduce a fourth attack against the robotic vehicle which takes the form of a malware which generates a random, but significant processing load on the vehicle's CPU. We will be referring to this here as malware (CPU) to differentiate from the malware (Net) included in the attacks used in the training of the model. The purpose of this fourth attack is to serve as an unseen type of attack. The malware (CPU) experiments produced 11,383 data points, of which 6,483 were attack and 4,900 were non-attack data points in intervals between attacks.

ML algorithm	Attack	ACC (%)	Overall ACC(%)
Logistic Regression	Denial of Service	95.5	73.7
	Command Injection	56.9	
	Malware (Net)	68.6	
Decision Tree (C5.0)	Denial of Service	73.2	74.0
	Command Injection	66.5	
	Malware (Net)	**82.4**	
Random Forest	Denial of Service	71.8	77.3
	Command Injection	78.6	
	Malware (Net)	81.6	
SVM (Radial Kernel)	Denial of Service	**97.4**	79.9
	Command Injection	67.7	
	Malware (Net)	74.5	
SVM (Linear Kernel)	Denial of Service	95.2	71.6
	Command Injection	48.0	
	Malware (Net)	71.6	
Deep learning (MLP)	Denial of Service	96.2	78.3
	Command Injection	58.0	
	Malware (Net)	80.7	
Deep learning (RNN)	Denial of Service	95.4	**86.9**
	Command Injection	**83.2**	
	Malware (Net)	82.2	

TABLE II: Comparing the performance of our deep learning based approach against popular machine learning algorithms for intrusion detection

ML algorithm	Attack	ACC (%)
Logistic Regression	Malware (CPU)	59.0
Decision Tree (C5.0)	Malware (CPU)	53.4
Random Forest	Malware (CPU)	58.8
SVM (Radial Kernel)	Malware (CPU)	62.9
SVM (Linear Kernel)	Malware (CPU)	59.0
Deep learning (MLP)	Malware (CPU)	55.0
Deep learning (RNN)	Malware (CPU)	**66.9**

TABLE III: Deep learning Vs. other machine learning algorithms detection accuracy for detecting the "unseen" malware CPU attack

As shown in Table III, here the RNN LSTM deep learning classifier (configured as a 1,000-neuron model) produces the highest detection accuracy for the CPU malware attack, with a 4% increase in detection accuracy over SVM (Radial kernel). Similar to the command injection attack, the deep MLP reported low detection accuracy (second lowest), only 2% higher than decision trees in this case, but almost 12% lower than RNN. In this case the deep MLP is using two hidden layers, with the same number of neurons which reported the most optimal accuracy at 55%, compared to 53.5% with a single hidden layer; adding further hidden layers failed improve detection accuracy. This result indicates that ability for the RNN classifier to learn different attack behaviours over time contributes in a meaningful way to predicting attacks which might share similar behavioural characteristics (observed in cyber and physical features) in the lead up to or execution of an attack.

The relatively lower accuracy rates across all algorithms are not unexpected, as these are all techniques designed for seen types of attacks. So, to further evaluate the usefulness of utilising deep learning in this context too, we compare also with an unsupervised and a semi-supervised technique

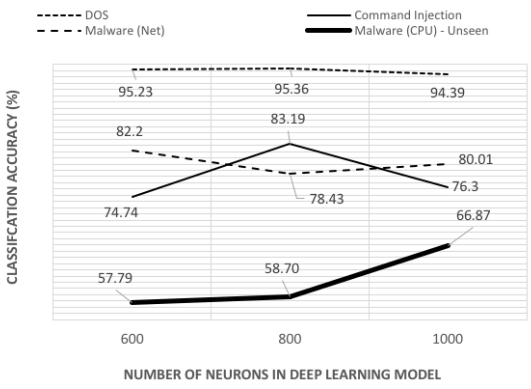


Fig. 7: Detection accuracy using the RNN LSTM deep learning model. The top three are attack types that have been previously seen, while the bottom one is an attack not previously seen.

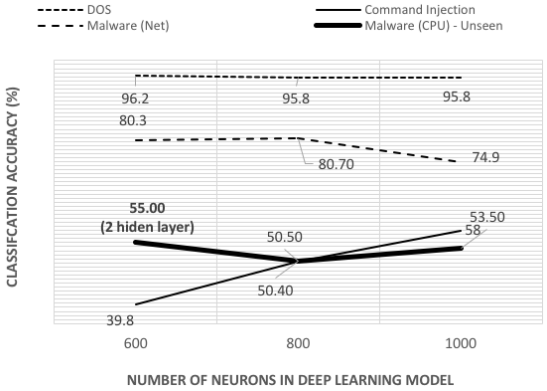


Fig. 8: Detection accuracy using the MLP deep learning model. The top three are attack types that have been previously seen, while the bottom one is an attack not previously seen.

typically employed for unseen types of attack. We have chosen a one-class SVM as a semi-supervised approach which trains a model based on a single-class "concept", which refers to the understanding of a single known state (e.g., normal operation behaviour=TRUE) based on the training data supplied. For prediction, if the model identifies that this state has deviated or the "concept" has drifted, then an anomaly state is classified (e.g., normal operation behaviour=FALSE). For unsupervised learning, we employ k-means clustering to produce two cluster classes.

Again, the deep learning approach outperforms the semi-unsupervised and unsupervised machine learning algorithms we have tested for the unseen type of attack (and naturally also for the seen ones). Moreover, as with the supervised machine learning models tested, our deep learning model outperformed both the one-class SVM and k-means in detecting the malware (CPU) attack. These results are encouraging especially as the unsupervised algorithms selected have been proven to work well within the field of anomaly-based IDS systems for cyber attacks in different contexts [50]–[52]. Of course, it can be argued that different attacks lead to different detection accuracy, but see sufficient evidence that for types of attack that are meaningful in the context of the cyber-physical security of a robotic vehicle, deep learning with LSTM is more dependable than the standard machine learning approaches traditionally used.

However, due to the processing limitations of the vehicle, running deep learning onboard the vehicle can take too long to be practical. Indicatively, when running the model periodically every 1 s based on the last 1 s worth of data collected, the detection latency for the 600, 800 and 1000-neuron architectures is on average (over five runs) 1.163 s, 1.541 s and 1.704 s respectively (Table V). So, a detection result for one detection run may not have been produced before the next periodic run starts.

ML algorithm	Attack	ACC (%)	Overall ACC(%)
K-means	Denial of Service	92.0	57.26
	Command Injection	46.82	
	Malware (Net)	42.14	
	Malware (CPU)	48.2	
One-class SVM	Denial of Service	75.91	57.84
	Command Injection	47.82	
	Malware (Net)	42.14	
	Malware (CPU)	65.5	
Deep learning (RNN)	Denial of Service	**94.39**	**79.39**
	Command Injection	**76.3**	
	Malware (Net)	**80.01**	
	Malware (CPU)	**66.87**	

TABLE IV: Comparing the performance of the RNN 1000 neuron deep learning model against k-means clustering and one-class SVM unsupervised learning

Number of neurons	Detection latency (s)
600	1.163
800	1.541
1000	1.704

TABLE V: Mean detection latency of local deep learning based detection on particular robotic vehicle

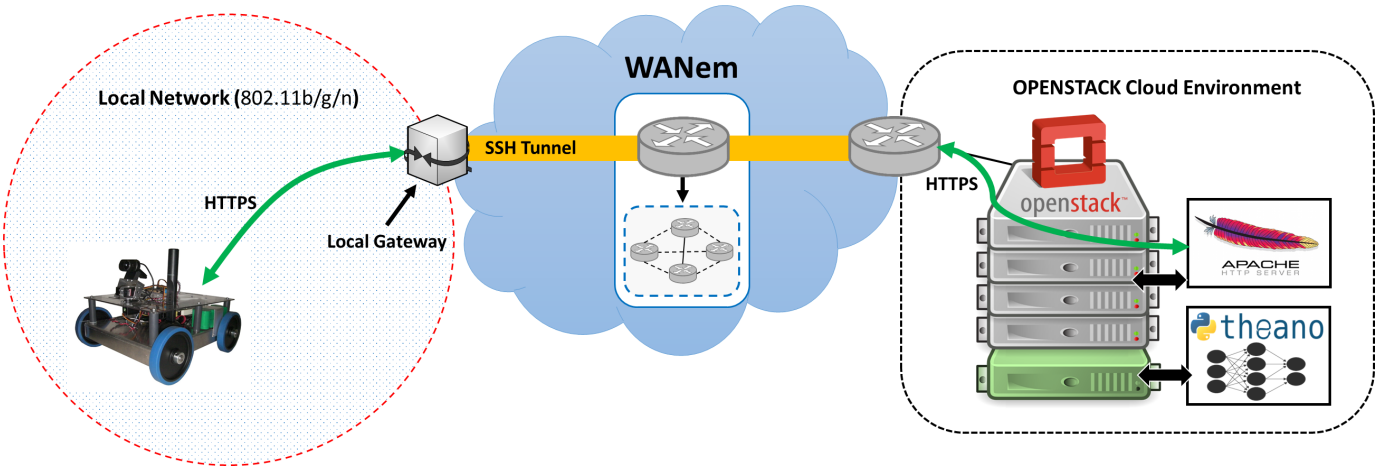
V. Offloading intrusion detection

To reduce the processing time for reaching a detection decision, we turn to the concept of offloading, where a higher-end computing infrastructure carries out the processing. However, in doing so, we introduce network delays and potential for network failures, which place extra volatility on the overall detection latency. In the following sections, we discuss the network configuration of an experimental testbed for evaluating the feasibility of offloading detection to the cloud. We then present a mathematical model to estimate the conditions under which offloading is practical and preferable to local on-board detection. We validate the model by comparing to experimental measurements in our testbed.

A. Testbed: Cloud-based IDS

For our offloading experiments, we have implemented a private cloud using *OpenStack* for cloud provisioning of six "datacentre-class" server nodes; each node containing an 8-core (16 threads) Intel Xeon E5-2640 v3 CPU, 16 GB of

Fig. 9: Experimental testbed including vehicle and offloading infrastructure



RAM and 1 TB of storage. The private cloud was configured using the standard Ubuntu Openstack reference architecture [53], where MAAS (Metal as a Service) was employed to provision one of the six physical servers solely to run the deep learning IDS system. Very similar cloud-based resources are available via existing cloud providers such as EC2 of Amazon Web Services, with options to lease virtual machine instances with dynamic or fixed resources, or even independent physical servers in the providers datacentre supplied as Infrastructure-as-a-Service (IaaS). Here, we have opted to emulate the latter.

B. The networking configuration of offloading

The network testbed consists of three modules: an 802.11n wireless local area network (WLAN), a point to point wide area network (WAN) utilising the *WANem* wide area network emulator and a remote Openstack cloud platform. The WLAN provides the vehicle with mobile connectivity to a local network gateway conducting port forwarding between the vehicle and the deep learning server for offloading, through an SSH tunnel over the WAN. Using the client-side URL transfer library *libcurl* [54] and *PyCURL* (Python Interface to libcurl), the vehicle offloads detection tasks by uploading sensor data samples, at interval period T.

We have designed the offloading process to employ a lightweight mechanism that is both robust to different network conditions and suitably secure, enforcing data confidentiality, integrity and authenticity. For this the HTTPS (HTTP over Transport Layer Security 1.2 (TLS)) protocol was selected to perform network offloading to a web server (via *PyCURL*), using the traditional client-server model to transfer over a authenticated and encrypted communication channel. Certificate-based public key server authentication was employed to guarantee the identity of the cloud test-bed (and the trustworthiness of the detection results source). Aside from data confidentiality and integrity protection supplied by TLS 1.2, HTTP was selected as a robust network transport protocol due to its native reuse of existing persistent connections via keep-alive functionality; ensuring the TLS handshake is performed only during the initial connection. As a result, a HTTP request to offload data to the web server will reuse an existing HTTP connection as long as the detection offload period and transport latency is smaller than the HTTP keep-alive timeout configured. In this configuration latency incurred by the TLS handshake is effectively avoided after the vehicles initial offload (e.g., when the vehicle turns on, or connects to a network), with subsequent cipher-text introducing negligible encryption and decryption latency. Moreover, HTTP provides a lightweight choice for data transport as it employs data transfer pipelining and automatic data compression which helps optimise TCP performance and packet transfer speed, reducing network load and symptomatically decreasing detection latency.

All data communication between the vehicle and remote web server is vehicle initiated, through use of python scripts making calls to the libcurl library. The scripts operate as a set of continual loops. On initiation, a sensor sample generated at interval T is retrieved and transferred via an HTTP POST to the deep learning cloud for every detection period T_d. If the POST is successful, another loop is then spawned, continually polling the server with HTTP GET requests until a detection result is successfully retrieved. On receipt of the detection result, the next sensor sample is then collected when the next detection period is reached and the HTTP data transfer process is repeated. Figure 9 shows a high level overview of the network topology used to remotely access the cloud.

C. Evaluating the practicality of offloading detection

Recently, Canziani et al. conducted a study that compared the computational performance between multiple state-of-the-art neural network architectures in terms of classification accuracy, memory footprint, parameters, operations count, inference time and power consumption [55]. The study showed that for a minor increase in classification accuracy, the computational cost (e.g., processing time) was also significantly increased. Therefore, given the resource constraints inherent in a power-limited vehicle, it would be more efficient to offload this task to an external and likely more powerful system in order to reduce computational processing time and as a result minimise detection latency.

In this manner, realising the benefits of offloading detection across a distributed service to a server, cloud or cloudlet, the transportation of detection data requires network connectivity that is resilient to and practically useful over variable conditions; especially over the public Internet where no reliability or quality of service is guaranteed. For cyber-physical systems relying on fast and reliable attack detection, the problem is exacerbated by the risk of dropping crucial data due to unreliable network connectivity. Therefore, a trade-off between on-board detection and remote offloading is largely determined by the available local resources and the quality of network conditions to the remote detection system.

We consider the task of offloaded periodic cyber-physical intrusion detection, which involves uploading the latest sample of data collected from the vehicle to a remote server in time t_x, processing the data on the remote server to produce a detection result in time t_s, and transmitting the result back to the vehicle in time t_r, followed by possible idle time t_i until the next iteration. So, the detection period T_d is $T_d = t_x + t_s + t_r + t_i$. Detection can be practical (and not cause an infinitely increasing queue of delayed results) only if $t_i \geq 0$, as presented visually in Figure 10.

Assuming that detection is accurate and attack has occurred at a random point in time within the previous interval T_d, then the detection latency t_l is the time between occurrence of attack and beginning of next detection cycle plus the time to upload the data, process them, and return the result. Assuming uniform distribution of the probability of the attack having occurred at a random time within T_d, then the mean corresponding delay is $\frac{T_d}{2}$, and overall the mean detection latency is:

$$\bar{t_l} = \frac{T_d}{2} + \bar{t_x} + \bar{t_s} + \bar{t_r} \tag{1}$$

The time to complete the processing to produce the detection result depends on the algorithm chosen, implementation approach and the processing resources that are available.

Our aim is to evaluate the upper limits for t_s with CPU resources available in a typical public cloud platform that allow periodic offloaded detection to be practical, which translates into t_i not dropping below 0. In ideal communication conditions, where there are no packet losses or network failures, and in accordance with the standard practice in computation offloading modelling [56], [57], the time to upload or receive data over a communication channel can be modelled in relation to the data size uploaded D_x and received D_r by the vehicle and the corresponding transmitting rate R_x and receiving rate R_r as:

$$\bar{t_x}^{(ideal)} = \frac{D_x}{R_x} \tag{2}$$

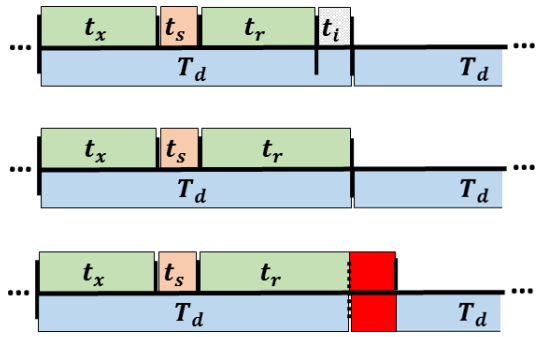
$$\bar{t_r}^{(ideal)} = \frac{D_r}{R_r} \tag{3}$$

Thus, the mean detection latency in ideal communication conditions can be represented as:

$$\bar{t_l}^{(ideal)} = \frac{T_d}{2} + \frac{D_x}{R_x} + \bar{t_s} + \frac{D_r}{R_r} \tag{4}$$

In non-ideal communication conditions, where we consider packet loss with a probability p, we assume that, the delay in establishing that a packet is lost and retransmitting means that each bit lost incurs an increase in communication delay equivalent to the time it would take to transmit l bits, where $l \in \mathbb{R}^+$. Mean detection latency in the presence of packet loss becomes:

Fig. 10: Example of variable offloading detection latency within the constraints of detection period T_d. The top and middle figure correspond to the practical cases, where $t_i > 0$ or $t_i = 0$ respectively, while the bottom figure corredsponds to the impractical case, where $t_i < 0$.



$$\bar{t_l}' = \frac{T_d}{2} + t_s + (1 + lp)(\bar{t_x}^{(ideal)} + \bar{t_r}^{(ideal)}) \tag{5}$$

$$= \frac{T_d}{2} + t_s + (1 + lp)(\frac{D_x}{R_x} + \frac{D_r}{R_r}) \tag{6}$$

Mean detection latency increases further if we also take into account the likelihood of a network failure occurring after random time t_θ since last failure and being repaired after random time t_ξ. We assume that failures occur independently and the number of failures occurring in a period T_d follow a Poisson distribution with constant mean $\frac{1}{\theta}$, where $\theta \in \mathbb{R}^+$ is the mean time between failures (MTBF). We assume that the time to repair after a failure follows a normal distribution with a mean time to repair (MTTR) ξ.

Communication mechanisms used to transmit the data sample or receive the detection result may implement a form of "keep alive" functionality, which keeps a network session alive for up to T_K time after a failure has occurred. If this time elapses, the session needs to be re-established with handshakes, e.g., for SSL. We denote the delay incurred by the handshakes as t_h.

So, the extra delay incurred by one failure can be represented as:

$$\mathbb{1}[t_\xi < T_K]t_\xi + \mathbb{1}[t_\xi \geq T_K](t_\xi + t_h)$$
$$= t_\xi + \mathbb{1}[t_\xi \geq T_K]t_h$$

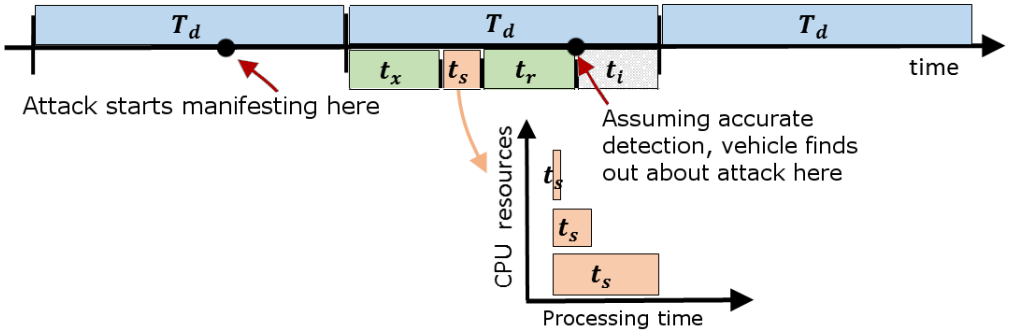
The mean extra delay due to one failure becomes:

$$\xi + (1 - P(t_\xi < T_K))\bar{t_h}$$

In a detection period T_d, due to the Poisson property, the expected number of failures is $\frac{1}{\theta}T_d$. So, the mean detection latency in the presence of both packet losses and network failures is:

$$\bar{t_l} = \bar{t_l}' + \frac{T_d}{\theta}(\xi + (1 - P(t_\xi < T_K))\bar{t_h}) \tag{7}$$

Fig. 11: Network offloading time sequence for offloaded IDS detection with detection period T_d. The practicality of offloading depends largely on the time t_s needed to complete detection on the server, which in turn depends on the server's processing resources



$$\bar{t_l} = \frac{T_d}{2} + \bar{t_s} + (1+lp)(\frac{D_x}{R_x} + \frac{D_r}{R_r}) + \frac{T_d}{\theta}(\xi + (1 - P(t_\xi < T_K))\bar{t_h}) \tag{8}$$

To evaluate the maximum mean time that processing should take to produce the detection result, we take the extreme case of no idle time between detection intervals, hence $t_i = 0$ and consequently $T_d = \bar{t}_{s,max} + \bar{t_x} + \bar{t_r}$. Equivalently:

$$\bar{t}_{s,max} = T_d - (\bar{t_x} + \bar{t_r}) \tag{9}$$

The mean latency introduced after the data sample has been collected on the vehicle is (based on (1)):

$$\bar{t_x} + \bar{t_s} + \bar{t_r} = \bar{t_l} - \frac{T_d}{2} \tag{10}$$

$$\bar{t_x} + \bar{t_r} = (1+lp)(\frac{D_x}{R_x} + \frac{D_r}{R_r}) + \frac{T_d}{\theta}(\xi + (1 - P(t_\xi < T_K))\bar{t_h}) \tag{11}$$

So, (9) becomes:

$$\bar{t}_{s,max} = T_d - (1+lp)(\frac{D_x}{R_x} + \frac{D_r}{R_r}) - \frac{T_d}{\theta}(\xi + (1 - P(t_\xi < T_K))\bar{t_h}) \tag{12}$$

Now, let us consider the general case where the detection period may be different to the data sample collection period T_c. So, $T_c = aT_d$, $a \in (0, 1]$. If $a = 1$, the detection mechanism runs often enough to ensure complete coverage of time, while $a < 1$ means that the mechanism covers a fraction of the time and an attack may be missed if it occurs outside this fraction. Substituting T_d by $\frac{T_c}{a}$, (12) yields:

$$\bar{t}_{s,max} = \frac{T_c}{a} - (1+lp)(\frac{D_x}{R_x} + \frac{D_r}{R_r}) - \frac{T_c}{a\theta}(\xi + (1 - P(t_\xi < T_K))\bar{t_h}) \tag{13}$$

We evaluate our mathematical model experimentally by emulating multiple network scenarios to determine the practicality of cloud-based intrusion detection and its associated performance over different network conditions; for this we have used the wide area network emulator *WANem* [58]. *WANem* enables the design and development of a variety of network scenarios and has been utilised in multiple research studies for the evaluation of defences against cyber attacks against critical infrastructures [59], lightweight security schemes for vehicle tracking [60], and wide area network emulation for testing automated covert channel modelling [61].

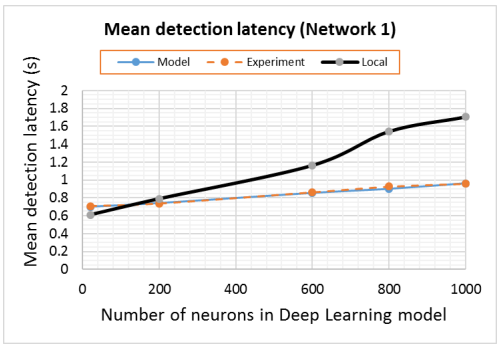
Profile	Network 1	Network 2	Network 3	Network 4
Round-trip time	4 ms	12 ms	54 ms	200 ms
Packet Loss (p)	0	0.001	0.01	0.03
MTBF (θ)	160 s	60 s	20 s	10 s
MTTR (ξ)	4 s	4 s	5 s	5 s
R_x bitrate	174 Kbps	158 Kbps	116 Kbps	29 Kbps
R_r bitrate	82 Kbps	21 Kbps	5.62 Kbps	0.392 Kbps
Keep alive time (T_K)	5 s	5 s	5 s	5 s
Handshake time (t_h)	37 ms	52 ms	102 ms	416 ms

TABLE VI: Network Scenarios used in experiments and by mathematical model

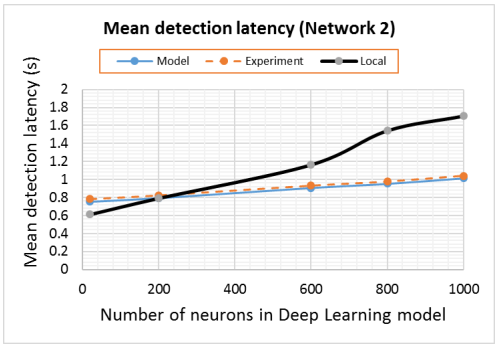
To form a realistic set of cloud-based offloading scenarios that form the basis of our experimental testing, we configure WANem with representative network latency parameters from different network types. For network 1, we use the existing lab test-bed server to present a LAN-based server. Network 2 represents an ideal cloud service, profiled by measuring real cloud services via the *cloudharmony*[1] web service. The network latency results were used to determine a baseline average transmission delay from London (the experiment test-bed location) to *Google Compute Engine* cloud platform for the round-trip time (RTT) of a HTTPS GET request. For network 3, we have utilised the performance statistics from the 2014 OFcom mobile broadband study in the UK [62]; providing an accurate measurement of latency for a typical 4g/3g network service. Network 4 represents a highly unstable network with a high packet loss, frequent connectivity failures and increased latency. Table VI provides each set of configuration parameters defined in the *WANem* configuration for each of the network scenarios in the detection offloading experiment.

Figure 12 shows the comparison between model and experiment for the four network configurations specified in terms of the detection latency. Our experimentation for each case involved five runs, each 300 s of continuous offloaded

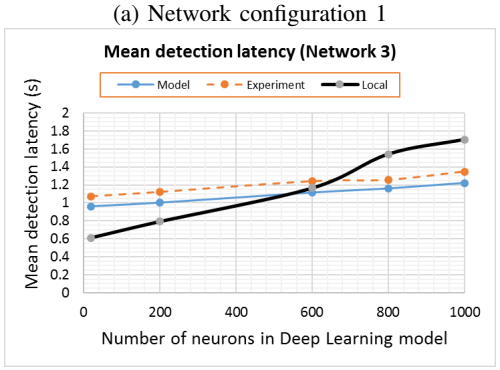
[1] https://cloudharmony.com/speedtest



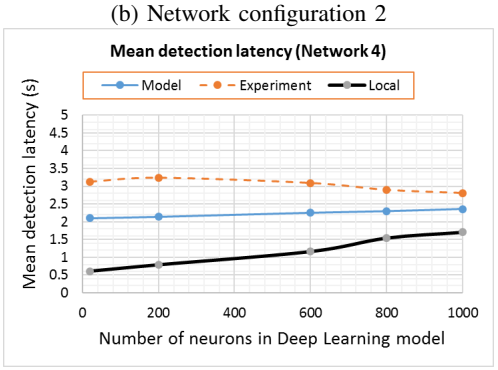
(a) Network configuration 1



(b) Network configuration 2



(c) Network configuration 3



(d) Network configuration 4

Fig. 12: Mean detection latency as measured experimentally and estimated mathematically for the case of network configurations 1-4. The black curve corresponds to the detection latency when the processing occurs on the vehicle itself without offloading via a network.

detection with $T_d = T_c = 1s$. Here, we report the mean detection latency values. For network configurations 1-3, the model's estimation is very close to the actual detection latency values obtained via the experiments. In the case of the very unreliable network (network configuration 4), the model is less accurate (off by 15-33%), mainly because it assumes that a network returns to its healthy state immediately after recovery and handshakes. In practice, some residual delays may occur in highly congested networks. Nevertheless, we have found that the model's accuracy in reasonably reliable networks is excellent and can be used to take offloading decisions (whether detection should run onboard or offloaded).

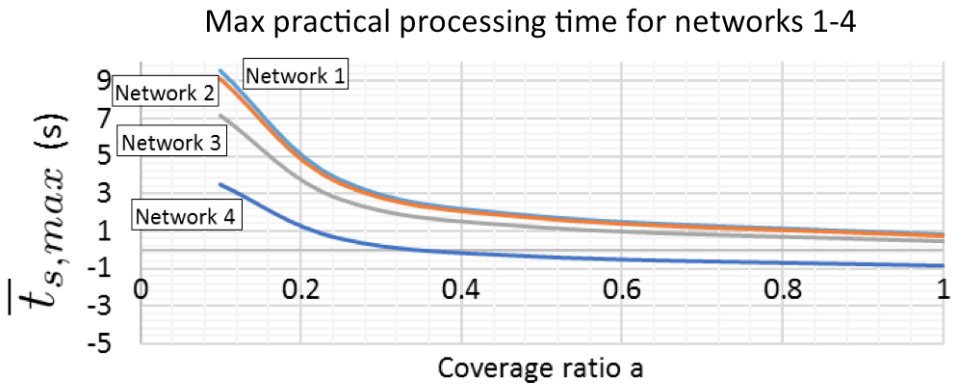
For a more clear view of the evaluation of the mathematical model across different scales, we have extended experimentation to lower numbers of neurons too (20 and 200). Note that the observations below hold only for the specific processing power of the vehicle (and hence processing delays) we worked with. Naturally, for a more powerful vehicle, offloading will be practical only for a larger number of neurons, and for a less powerful one, it will be practical for a smaller number of neurons. It is beyond the scope of this paper to predict processing delay based on processing resources. We consider the processing delays as input to the model. Both model and experimental evaluation agree that, from the perspective of mean detection latency, offloading detection via networks 1 and 2 is preferable to running it onboard, if the deep learning architecture includes 200 neurons and above. This number increases to approximately 600 neurons for network 3. With the same criterion of reducing detection latency, it is never practical to offload detection via network 4 in any of the cases evaluated (between 20 and 1000 neurons).

As we are satisfied with the accuracy of the model in enabling offloading decisions, we can utilise it to estimate configuration parameters that render it practical, in terms of achieving detection latency that is lower than the onboard (local) case, as above, or lower than the T_d. Figure 13 illustrates the latter. Specifically, it shows $t_{s,max}$ as represented in equation 13, for different values for a in the four network configurations.

We observe that out of the four network configurations utilised, only the fourth, which corresponds to the least reliable network, would be impractical for offloading the task of continuous deep learning based intrusion detection. For the cloud infrastructure used in our experiments, the mean time to complete the processing $\bar{t_s}$ was 0.279 s for the 1000-neuron case and lower for the other cases. So, offloading was not only practical, but also preferable for reducing overall delay in networks 1-3.

However, for network 4, $\bar{t}_{s,max}$ would drop below 0, making it impractical for coverage ratio a above 0.4 regardless of the processing power of the remote infrastructure. Naturally, reducing a, increases $t_{s,max}$, but also increases the likelihood that particularly short-duration attacks with no lasting cyber or physical impact may be missed by the detection process altogether if not captured within the time periods covered.

Fig. 13: $\bar{t}_{s,max}$ against different values of a for the four network configurations



VI. Conclusion

We have shown experimentally that utilising RNN-based deep learning enhanced by LSTM can increase considerably intrusion detection accuracy for a robotic vehicle, when comparing against standard machine learning classifiers or MLP-based deep learning, which cannot take into account the temporal elements of a cyber attack. We have also shown that the key disadvantage of a deep learning based approach, which is detection latency due to the increased processing demands, can be addressed through cloud-based computational offloading. For this, we produced a practical implementation and have also presented and validated experimentally a mathematical model for evaluating when offloading is practical from the detection latency perspective.

However, we also need to consider that there is a fundamental difference between running a cyber security task (such as intrusion detection) onboard the vehicle or offloaded remotely, which is the reliance on an external communication network, not only in terms of its availability and performance, but also its security. For vehicles, this is particularly true because, in almost all realistic cases, offloading needs to be carried out via a wireless medium, and consequently is vulnerable to security threats itself. The security of the wireless medium was not within scope here, but needs to be taken into account in real-world deployment of such an approach. For this work, we have utilised HTTPS as a means to provide a satisfactory level of confidentiality and integrity of the process, but have not taken any measures against a physical availability threat, such as communication jamming. A reasonable approach here would be to resort to a lightweight machine learning classifier, such as logistic regression, random forest or SVM, for as long as the vehicle is in a communication-denied environment, whether naturally or as a result of an attack on the wireless medium.

Further considerations are the availability, cost and security of the remote infrastructure used for offloading. Here, we have used a trusted private cloud, but other options could be using another, resource-rich vehicle (for example, when operating within a platoon of driverless vehicles), which may itself have been compromised, or extend to potentially "unfaithful" clouds. For the latter, there is excellent work being produced in the area of secure computation offloading [63], [64], which could be adopted in this context too. As for the cost of offloading, it could be monetary or energy-related, both being interesting directions of further research.

Perhaps the greatest advantage of offloading is the potential of having a common cloud-based infrastructure that can be used by a large number of vehicles of different owners, operational patterns and environments. This would allow collecting data regarding the normal or attack behaviour of a type of vehicle much more widely than in the limited conditions experienced during the training of the IDS of a single vehicle. There is also a strategic strength in this direction of research. Both deep learning and cloud security are areas of considerable activity. By positioning this work where the two meet, the approach of cloud offloading of deep learning based intrusion detection will benefit further in the future by advances in these two fields.

References

[1] G. Loukas, *Cyber-Physical Attacks: A Growing Invisible Threat*. Butterworth-Heinemann (Elsevier), 2015.

[2] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *IEEE Security and Privacy*. IEEE, 2010, pp. 447–462.

[3] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Usenix Security Symposium*. USENIX, 2011.

[4] D. Ward, I. Ibarra, and A. Ruddle, "Threat analysis and risk assessment in automotive cyber security," *International Journal of Passenger Cars*, vol. 6, no. 2, pp. 507–513, 2013.

[5] M. Ring, J. Durrwang, F. Sommer, and R. Kriesten, "Survey on vehicular attacks-building a vulnerability database," in *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2015, pp. 208–212.

[6] A. Javaid, W. Sun, V. Devabhaktuni, and M. Alam, "Cyber security threat analysis and modeling of an unmanned aerial vehicle system," in *IEEE Conference on Technologies for Homeland Security (HST)*. IEEE, 2012, pp. 585–590.

[7] G. McGraw, "Cyber war is inevitable (unless we build security in)," *Journal of Strategic Studies*, vol. 36, no. 1, pp. 109–119, 2013.

[8] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, pp. 617–636, 2014.

[9] J. Petit and S. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 546–556, 2015.

[10] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium*. USENIX, 2015, pp. 881–896.

[11] T. Vuong, A. Filippoupolitis, G. Loukas, and D. Gan, "Physical indicators of cyber attacks against a rescue robot," in *IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2014, pp. 338–343.

[12] A. Greenberg, "Hackers remotely kill a jeep on the highway with me in it," 2015.

[13] A. Bezemskij, R. J. Anthony, G. Loukas, and D. Gan, "Threat evaluation based on automatic sensor signal characterisation and anomaly detection," in *The Twelfth International Conference on Autonomic and Autonomous Systems (ICAS 2016)*. IARIA, 2016.

[14] A. Bezemskij, G. Loukas, R. J. Anthony, and D. Gan, "Behaviour-based anomaly detection of cyber-physical attacks on a robotic vehicle," in *Eighth International Symposium on Cyberspace Safety and Security*. CPS, 2016.

[15] T. Vuong, G. Loukas, and D. Gan, "Performance evaluation of cyber-physical intrusion detection on a robotic vehicle," in *Proceedings of 13th International Conference on Pervasive Intelligence and Computing (PICOM)*. IEEE, 2015.

[16] T. Vuong, G. Loukas, D. Gan, and A. Bezemskij, "Decision tree-based detection of denial of service and command injection attacks on robotic vehicles," in *Proceedings of 7th International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2015.

[17] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *24th ACM Conference on Computer and Communications Security (CCS17)*. ACM, 2016, pp. 164–170.

[18] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell, "Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: a data-driven approach to in-vehicle intrusion detection," in *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*. ACM, 2017, p. 11.

[19] F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone, "Car hacking identification through fuzzy logic algorithms," in *Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–7.

[20] A. Lauf, R. Peters, and W. Robinson, "A distributed intrusion detection system for resource-constrained devices in ad-hoc networks," *Ad Hoc Networks*, vol. 8, no. 3, pp. 253–266, 2010.

[21] M. Strohmeier, V. Lenders, and I. Martinovic, "Intrusion detection for airborne communication using phy-layer information," in *12th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. Springer, 2015.

[22] Z. Birnbaum, A. Dolgikh, V. Skormin, E. O'Brien, and D. Muller, "Unmanned aerial vehicle security using recursive parameter estimation," in *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014, pp. 692–702.

[23] C. Gwak, M. Jo, S. Kwon, H. Park, and S. Son, "Anomaly detection based on recursive least-square filter for robust intelligent transportation systems," in *Proceedings of the 2015 Korea Institute of Communication Sciences Summer Conferences*. KICS, 2015, pp. 438–440.

[24] J. Schumann, P. Moosbrugger, and K. Rozier, "R2u2: Monitoring and diagnosis of security threats for unmanned aerial systems," in *Proceedings of 15th International Conference on Runtime Verification*. Springer, 2015.

[25] A. Bezemskij, G. Loukas, D. Gan, and R. Anthony, "Detecting cyber-physical threats in an autonomous robotic vehicle using bayesian networks," in *Proceedings of IEEE Cyber, Physical and Social Computing (CPSCom)*. IEEE, 2017.

[26] G. Pearson and M. Kolodny, "Uk mod land open systems architecture and coalition interoperability with the us," in *Proc. of SPIE Vol*, vol. 8742, 2013, pp. 87 420C–1.

[27] R. Mitchell and I. Chen, "Specification based intrusion detection for unmanned aircraft systems," in *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications*. ACM, 2012, pp. 31–36.

[28] ——, "Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 5, pp. 593–604, 2014.

[29] S. Martini, D. Di Baccio, F. Romero, A. Jimnez, L. Pallottino, G. Dini, and A. Ollero, "Distributed motion misbehavior detection in teams of heterogeneous aerial robots," *Robotics and Autonomous Systems*, vol. 74, pp. 30–39, 2015.

[30] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[31] K. Alheeti, A. Gruebler, and K. McDonald-Maier, "An intrusion detection system against malicious attacks on the communication network of driverless cars," in *Proceedings of the 12th Consumer Communications and Networking Conference (CCNC)*. IEEE, 2015, pp. 916–921.

[32] K. M. A. Alheeti and K. McDonald-Maier, "An intelligent intrusion detection scheme for self-driving vehicles based on magnetometer sensors," in *Students on Applied Engineering (ICSAE), International Conference for*. IEEE, 2016, pp. 75–78.

[33] K. M. A. Alheeti, R. Al-Zaidi, J. Woods, and K. McDonald-Maier, "An intrusion detection scheme for driverless vehicles based gyroscope sensor profiling," in *Consumer Electronics (ICCE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 448–449.

[34] K. M. A. Alheeti, A. Gruebler, and K. McDonald-Maier, "Intelligent intrusion detection of grey hole and rushing attacks in self-driving vehicular networks," *Computers*, vol. 5, no. 3, p. 16, 2016.

[35] G. Hu, W. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network*, vol. 26, no. 3, pp. 21–28, 2012.

[36] A. Houmansadr, S. Zonouz, and R. Berthier, "A cloud-based intrusion detection and response system for mobile phones," in *IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*. IEEE, 2011, pp. 31–32.

[37] G. Portokalidis, P. Homburg, K. Anagnostakis, and . Bos, H., "Paranoid android: versatile protection for smartphones," in *26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 347–356.

[38] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl4md: A deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN)*. WorldComp, 2016, p. 61.

[39] Z. Dong, K. Kane, and L. Camp, "Detection of rogue certificates from trusted certificate authorities using deep neural networks," *Transactions on Privacy and Security (TOPS)*, vol. 19, no. 2, p. 5, 2016.

[40] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2016, pp. 1–5.

[41] A. Y. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) on 9th*. ICST, 2016, pp. 21–26.

[42] J. W. Kang, M. J.and Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS One*, vol. 11, no. 6, 2016.

[43] T. P. Vuong, "Cyber-physical intrusion detection for robotic vehicles," Ph.D. dissertation, University of Greenwich, UK, 2017.

[44] T. G. Barbounis, J. B. Theocharis, M. C. Alexiadis, and P. S. Dokopoulos, "Long-term wind speed and power forecasting using local recurrent neural network models," *IEEE Transactions on Energy Conversion*, vol. 21, no. 1, pp. 273–284, 2006.

[45] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2015, pp. 1110–1118.

[46] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[47] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[48] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems," *J. Mach. Learn. Res*, vol. 15, no. 1, pp. 3133–3181, 2014.

[49] M. A. M. Hasan, M. Nasser, B. Pal, and S. Ahmad, "Support vector machine and random forest modeling for intrusion detection system (IDS)," *Journal of Intelligent Learning Systems and Applications*, vol. 6, no. 1, p. 45, 2014.

[50] W. J. . M. A. Wang, Y., "Anomaly intrusion detection using one class svm," in *nformation Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*. IEEE, 2004, pp. 358–364.

[51] P. R. D. R. M. . R. F. Giacinto, G., "Intrusion detection in computer networks by a modular ensemble of one-class classifiers," *Information Fusion*, vol. 9, no. 1, pp. 69–82, 2008.

[52] H. S. . L. B. Jianliang, M., "The application on intrusion detection based on k-means cluster algorithm," in *Information Technology and Applications, 2009. IFITA'09. International Forum on*. IEEE, 2009, pp. 150–152.

[53] Canonical, "Ubuntu openstack reference implementation," 2014.

[54] D. Stenberg, "curl," 2016. [Online]. Available: https://curl.haxx.se/

[55] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv preprint arXiv:1605.07678*, 2016.

[56] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[57] G. Loukas, Y. Yoon, G. Sakellari, T. Vuong, and R. Heartfield, "Computation offloading of a vehicle's continuous intrusion detection workload for energy efficiency and performance," *Simulation Modelling Practice and Theory*, 2016.

[58] H. K. Kalitay and M. K. Nambiarz, "Designing wanem: A wide area network emulator tool," in *Third International Conference on Communication Systems and Networks (COMSNETS 2011)*. IEEE, 2011, pp. 1–4.

[59] L. Aniello, D. Luna, L. G. A., G., and R. Baldoni, "A collaborative event processing system for protection of critical infrastructures from cyber attacks." in *International Conference on Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg., 2011, pp. 310–323.

[60] A. Ukil, S. Bandyopadhyay, A. Bhattacharyya, and A. Pal, "Lightweight security scheme for vehicle tracking system using coap," in *Proceedings of the International Workshop on Adaptive Security*. ACM, 2013, p. 3.

[61] F. Rezaei, M. Hempel, and H. Shrestha, P. L. andSharif, "Evaluation and verification of automated covert channel modeling using a real network platform," in *2014 IEEE Military Communications Conference*. IEEE, 2014, pp. 12–17.

[62] OFcom, "Measuring mobile broadband performance in the uk: 4g and 3g network performance," 2014. [Online]. Available: https://www.ofcom.org.uk/__data/assets/pdf_file/0014/32054/mbb-nov14.pdf

[63] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in CryptologyCRYPTO 2010*. Springer, 2010, pp. 465–482.

[64] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. Wong, "Rocking drones with intentional sound noise on gyroscopic sensors," *Transactions on Information Forensics and Security*, vol. 10, no. 1, pp. 69–78, 2015.