Machine Learning and Software Defined Network to secure communications in a swarm of drones

Christophe Guerber, Mickaël Royer, Nicolas Larrieu

Machine Learning and Software Defined Network to secure communications in a swarm of drones

Christophe GUERBER, Mickaël ROYER, Nicolas LARRIEU

ENAC, Université de Toulouse, France

Abstract

As drones become more and more frequent in industry and perhaps tomorrow in everyday life, the variety and sensitivity of their missions will increase. Securing the communication taking place with the drones and especially in the network of a swarm, is of primary importance to allow a safe integration of Unmanned Aerial Vehicles into air traffic.

Drones are subject to a range of attacks, from GPS jamming to application bug exploits. Among these attacks, and irrespective to whether they have already been implemented or not, communication is one of the main contributors, both as a vector and as a target.

In this article, we use previous work on security threats concerning drones to identify two main types of attack in a network of drones: intrusion from the outside and network usage from inside. We take advantage of Software Defined Network (SDN) architecture to build a secure network for a swarm of drones that allows the identification of most outsider attacks, except eavesdropping and address spoofing. In addition, traffic injection using address spoofing is detected using statistics monitoring and corresponding countermeasures are applied using SDN.

Finally, concerning attacks from the inside, we show how a machine learning solution based on Random Forest Classifier can be implemented to detect common network attacks such as denial of service, port scanning and brute force.

Email addresses: christophe.guerber@enac.fr (Christophe GUERBER), mickael.royer@enac.fr (Mickaël ROYER), nicolas.larrieu@enac.fr (Nicolas LARRIEU)

To train this algorithm, a SDN dataset is built from capture files originating from an Intrusion Detection System dataset and specific features easily available from the SDN controller are extracted. Detection performance of these abnormal behaviors are promising, both in terms of true positive and false positive, and in terms of detection delay. Detection of these common attacks will allow tightening of security in such wireless network by denying further access to the network by rogue nodes.

Keywords:
UAV, UAV Ad hoc NETwork, UAANET, AODV, SDN, OpenFlow, swarm of drones, security architecture, Machine Learning, Random Forest Classifier

---

1. Introduction

With an anticipated market of tens of billions of dollars worldwide, Unmanned Aerial Vehicles (UAV) are expected to become more and more widespread. Drones will become actors in an increasing number of missions, meeting a larger variety of objectives. Though the current typical mission usually only considers individual drones, swarms of collaborating machines open new perspectives for future missions. The missions undertaken by both single UAV and swarms will become more and more critical. At the same time, they will have to be integrated into air traffic management. The security requirements of these systems thus have to be addressed. It is even more important in the case of a swarm where the communication means and network are crucial for the success of the mission for collaboration and coordination.

Security is commonly defined by three goals: confidentiality, integrity and availability. In the context of a wireless network where any station in the coverage of the transmitter may receive its signal, security is a challenging task. Given the nature of the drones, which have limited computational resources (memory and computation power), the challenge is even more difficult.

Our approach is to build a twofold architecture capable of providing more insight in the network. First, we will rely on Software Defined Network techniques

(SDN), that allow both a better control and monitoring information about what is happening in the network. Second, we will use these mechanisms to provide data about misbehavior that a Machine Learning algorithm will be able to detect. Countermeasures can then be installed as SDN already provides all the necessary building blocks.

In this article, we will reintroduce the AODV/SDN architecture principles as a reminder of a previous published work. However, our focus will be its usage as a support for innovative security mechanisms to protect the swarm network. Compared to the previous contribution, the current paper demonstrates the immunity of this architecture against several attacks from nodes outside the legitimate network and proposes some detection and mitigation techniques against traffic injection. Finally, it proposes machine learning detection of attacks within the network and contains important results showing the performance of the proposed classification model.

Section 2 describes research fields related to SDN, security and machine learning. Section 3 provides some insight into the architecture that will be used during the rest of the article. Section 4 provides the threat model on which the network threat analysis is based and devides the problem into two separate attack scenarios. It then describes how the previously described architecture addresses one of these scenarios. Section 5 focuses on the second scenario of attacks, namely the situation where one of the nodes (drone or ground station) has been infected and uses the network to discover its environment and propagate. Finally, Section 6 shows the results of the previously detailed detection method and Section 7 concludes this paper.

2. Related work

2.1. Security in UAVs

As security in UAVs is a key enabler for a wider integration into a shared airspace and thus for broader adoption, several studies have addressed security issues in the past.

The author in [1] provides a risk analysis on UAVs compliant with European standards [2]. He studies the targets and vectors of attacks on Unmanned Aerial Systems (UAS) and considers the confidentiality, integrity and availability of the data or subsystems, thus following the Cyber-Security Thread Model defined by [3]. He proposes block diagram models of UAS. The communication means are, however, not identified, "as it encompasses all the modules and any incoming/outgoing command and control signals and data signals pass through it." The likelihood and the impact of each threat is evaluated. The risk is then presented as the product of these two evaluations. The threats considered as critical are eavesdropping, spoofing, command and control message modification, denial of service (DoS), signal integrity and Viruses, Malwares, Trojans and Keyloggers. Non critical threats are jamming, scrambling/distortion, cross layer and multi-protocol attacks, social engineering, data traffic modification and malicious code and subroutine exploits.

The contribution presented in [4] complements the first study by listing all the attacks that have been implemented, either on real UAS or as a simulation. Although the attacks targeted military UAVs, the same attacks will probably also be effective on small commercial drones. Though a significant proportion of the identified attacks concern GPS signal spoofing or jamming, approximately a half do concern the communication stream as target or vector of the attack. The study also lists the vulnerabilities of the Bebop drone for which an exploit has been successfully implemented:

- buffer-overflow Attack (crash) "entry in JSON command requesting to become the controller for the UAV was around 1000 characters";
- DoS (crash) "parallel multiple (around 1000) JSON requests to become the controller for the UAV";
- ARP Cache Poisoning approach (landed) "the controller got disconnected from UAV when ARP replies were sent continuously";
- control and modification of the drone trajectory by using shared data to

fool the anti collision system.

The authors propose a taxonomy of cyber vulnerabilities of drones, starting from a similar study on autonomous vehicle [5]. Then, the attacks listed previously are bound to the taxonomy to show that most attacks target the GPS signal but that there is currently very little research on data communication stream attacks.

Although the target of these attacks are on-board applications, the vector used for their implementation exploits the flaws of the network. Moreover, as there is no limit to the extent of the attack on the network, any vulnerability is widely exposed.

The authors in [6] and [7] present several implementations of attacks on commercial drones, some of which specifically target the network link:

- wifi de-authentication;
- injecting commands to take control of the drone (overwhelming the drone with attacker's commands);
- unauthorized access (telnet as root without password) allowing other attacks to be prepared (uploading virus files, killing processes, starting malwares or backdoors...);
- ARP cache poisoning;
- DoS by exploiting vulnerabilities in the application protocol.

These vulnerabilities are either bound to the network (ARP cache poisoning) or the consequence of poor design and could sometimes be avoided if the UAS had more control over its network.

2.2. SDN and security

As summarized in [8] when it comes to SDN and security, "a twofold research context has been identified: on the one hand, leveraging SDN features to enhance security; while on the other hand one can find the pursual of a secure

SDN system architecture." Though security by SDN is the major focus of the current article, the importance of security for SDN should not be ignored.

Concerning the latter, [8] identifies seven main objectives of attacks, from unauthorized access, to disclosure and modification of information, to service disruption. Orthogonally, attacks may be categorized by the origin or cause, which is one of the SDN layers: application, control, control channel and infrastructure. The present article does not explore these questions in depth. Recommendations will be provided for the most obvious barriers to be installed to strengthen the architecture against some of the above mentioned objectives. All other objectives will be assumed to be unreachable.

As suggested in [9], enhancing network security by SDN is made possible thanks to the following four main features provided by this technology:

- dynamic flow control providing the building blocks for the dynamic access control function and allowing to separate malicious flows from benign ones dynamically;
- network-wide visibility with centralized flow control eases network monitoring, and provides a more holistic view of the network, facilitating detection of and protection against malicious flows;
- network programmability, removing ad hoc network security middle boxes that are difficult to move and replacing them by SDN applications or tunnels through the network, which makes SDN an important complement to Network Functions Virtualization;
- simplified data plane, enabling extension and thus greater suitability for security.

SDN research addresses security in several manners. One approach is linked to Network Functions Virtualization (NFV) for firewalls, network intrusion detection systems (NIDS) or intrusion protection systems (IPS). The power of the tool however, comes at the expense of the number of transmissions required to forward the packet. In the case of small devices such as drones, sending packets

back and forth (from the source to the function and then to the destination) is not compliant with energy consumption requirements.

Another approach relies on deep packet inspection, where some features in the packets transported in the network are analyzed to detect previously identified attack signatures. However, this technique also requires that packets selected for inspection are transmitted to the feature extractor. Once again, this is not suited to our constrained environment: forwarding packets to the controller consumes battery and performing inspection in the drone is limited by CPU and battery resources.

2.3. Machine learning and security

Machine learning techniques have been used for almost every problem in networking as shown in [10] and security is of course no exception.

Although some of the listed techniques exhibit remarkable performance, most of the proposals based on machine learning use either the KDD'99 cup dataset [11] which is known to contain important issues as described in [12], or its enhanced version NSL-KDD [13]. However, some of the very diverse features contained in these datasets are outside the network domain and thus inaccessible to an SDN controller or switch (e.g. number of root accesses, number of file creation operations or number of shell prompts) or are not available at the beginning of the flow (e.g. its duration).

Few proposals are solely based on pure network features. As an example, the authors in [14] propose the use of SDN features for each flow: number of packets, total byte count, byte rate, packet rate, length of first packet and average length of packets. The proposed model targets malware traffic detection. The delay before the attack is identified has not been measured. However, as some of the features are known only at the end of the flow (inter alia flow duration, number of packets, average length), the classifier will most probably not be able to detect malware immediately at the beginning of the flow.

Others use packet header information as in [15], but are more dedicated to intrusion detection systems (IDS): they continually snif packets and seek to

detect the intrusion in the middle of the flow. They do not have requirements with regard to detection latency and may take time to process and alert the network administrator.

3. Mobile architecture for security

As a prerequisite for the following discussion, the network architecture is presented as proposed in a previous published work.

3.1. Network

An architecture to secure the network of a swarm of drones and the communications taking place between the nodes was proposed in [16]. The size of the swarm is not limited but remains small in comparison with usual networks. In addition, drones are mobile nodes in the network with limited lifetime, CPU power and memory capacity, while the ground station is fixed and does not have these limitations.

All the nodes will have a set of network applications exchanging data either between the drones and the ground station (e.g. command and control, telemetry, mission data) or between drones with the intent to coordinate or collaborate.

The possibility of encrypting all the communications between the UAVs has not been considered possible due to the limit in CPU and memory capacity, in addition to the limited battery lifetime [17]. Indeed, ciphering/deciphering would be an important bottleneck in communication, in addition to depleting the battery by an intensive CPU usage, if this option were implemented on board the UAVs. Not ciphering the data will, however, allow the UAVs communications to be eavesdropped.

3.2. The data plane using Software defined network

Access to the network is controlled using Software Defined Network technology, which defines SDN switches that are controlled by an SDN controller. For the sake of simplicity, we will consider the ground controlling station to host

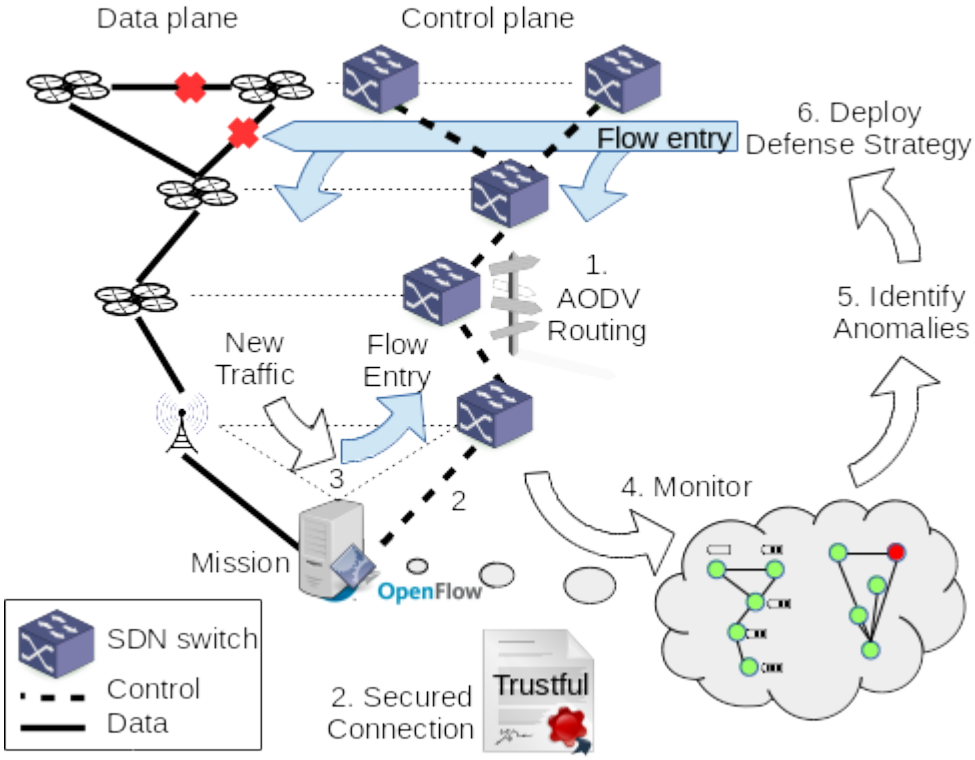

Figure 1: Security use case for SDN

the SDN controller, but the latter may also be hosted on a specialized UAV. The communication between the switches and the controller uses OpenFlow to exchange some SDN events and will be described in more detail hereafter.

As opposed to a more traditional network layout where SDN switches are specific and dedicated devices, we will consider each node (i.e. UAV or GCS) hosts an SDN switch, managing each flow entering or leaving, as shown in Figure 1.

For each new flow, the switch will generate a PacketIn event for the controller which will in turn take a decision on the legitimacy of the flow and send FlowMod events to all the switches along the path to the destination. FlowMod events will install the required forwarding information for the packet to arrive at its destination while performing any necessary frame checking and modifications.

The granularity of the elementary FlowMod events may vary from a very precise (MAC and IP addresses, protocol and ports) to a more general level (destination IP address). Very precise flow entries provide a very small attack surface while increasing the required size of the flow tables to achieve a given routing. Generic flow entries lower the required flow table size for the same routing, but allow more traffic to cross a node, thus increasing the potential attack surface. In addition, more generic flow entries require less updating in case of a topology change.

At the minimum, the destination node requires flow entries to be precise enough to guarantee that the received packets belong to a controller approved flow. Any packet not matching with these flow entries will be ignored without requesting approval.

Forwarding information is critical within the network. No external node should be able to eavesdrop on the control plane, nor modify or inject forwarding data. This is why OpenFlow has to be ciphered and OpenFlow will use TLS [18] or even better DTLS [19] that is not subject to TCP [20] attacks such as TCP reset attack or TCP sequence number attacks [21].

This setup needs to be complemented with the required routing between the nodes and the controller, and the ability to efficiently detect topology changes. Indeed, the network supporting the control plane communication is usually out of band, i.e. different from the one supporting the data plane. In addition, SDN has not been designed to support mobility and topology change events in a wireless environment. Topology discovery is usually based on Link Layer Discovery Protocol which is inefficient here [16]. In the proposed architecture, these two functions are provided inband using AODV [22].

3.3. Control plane routing and mobility

The routing between the nodes and the controller uses a secured version of AODV, a well known protocol in the UAV networking field. This protocol provides neighborhood monitoring and on-request route determination. In fact, each node will always maintain only a single route to the controller. The rout-

ing information consequently forms a spanning tree where the controller is the root. Another consequence is in the case of a change in topology where a node loses neighborhood with the next hop to the controller, any remaining or newly discovered neighbor already has a replacement route and may quickly take the forwarding over. Recovery is thus facilitated.

In addition, AODV Hello packets may provide some signed information to the neighboring nodes in a secured way. In particular, a signed MAC address of the sending node is added so that ARP is not required anymore within the network. This is intended to prevent man in the middle attacks using ARP cache poisoning.

In addition, the appearance (or disappearance) of a node detected by AODV Hello packet reception (resp. timeout expiration) is forwarded in a new dedicated OpenFlow event so that topology changes are forwarded to the controller. The latter is then able to determine which flow entries to modify so that the forwarding of packets within the network is recovered.

Finally, having the control plane sent inband (i.e. on the data plane) requires a set of static flow entries to be installed on the SDN switch.

The public key infrastructure (PKI) required to use both TLS/DTLS and the secured version of AODV is out of the scope of the present discussion.

4. Threat model and first defenses

The architecture described above provides interesting security features by design. In order to present these, the hypothesis taken in the rest of the article is first stated.

4.1. Threat: model and characterization

The assumptions made concerning the architecture, the UAS and its mission are described below:

- the size of the swarm is relatively small with about ten nodes (drones and ground controlling station) and a limited number of applications on board the UAVs;

- a secured PKI infrastructure exists, the setting of the PKI and the exchanges of public keys and certificates is not studied here;
- the mission has a limited duration and the keys used during the mission have the same duration as the mission (i.e. they are changed for each mission);
- the node operating system and all the on board applications in the network have proper authentication mechanisms, configurations and security measures so that no unauthorized access will be considered here. In particular, we considered that connections from unauthenticated devices will be rejected. We will not consider privilege escalation in this article;
- the SDN applications cannot be stopped nor altered and we do not consider any zero day exploits. The controller generates only coherent flows that never overlap or conflict with each other. This may require some additional software that is not described here.

We will consider that the controller does not allow new applications to be installed, and that the installed SDN applications have the highest security standards.

In addition, we will assume the following concerning the intruder that tries to harm the UAS or the mission:

- the intruder has free access to listen to the communication channel, and thus receives all the transmissions of the drones nearby;
- the intruder has free access to send on the communication channel, and thus is able to send any signal that may be received by the drones nearby;
- the intruder has a limited computational power. In particular, its computation capabilities are not powerful enough to break the keys during the mission;
- the intruder does not have the credentials of the ground control station or the drones;

- the intruder does not have physical access to the drone during the mission.

As described in the previous sections, there are vulnerabilities in the communication system that a hacker may exploit for different purposes. From security holes to bad design, several techniques were successfully used to gain some if not total control over the UAV trajectory, or to stop the mission.

If most of these attacks could be taken into account at the application level or during the design of the UAV, the lack of security features at the network level is the first missing barrier in most of these scenarios. Conversely, the network security features described in this article will not be able to prevent any sort of application level attacks, especially for eavesdropping as transmissions remain unciphered. Security analysis of UAV applications and protocols is still one of the first steps in the design of such applications and protocols.

We will consider two main scenarios in the following subsections: either the hacker is an outside node trying to interfere with the drones, or it is a drone that has been corrupted prior or during the mission, by any means giving access to the node (drone or GCS). The latter will be referred to as attacks from the inside.

4.2. Attacks from the outside

As explained in the previous sections, in this SDN based network, new flows will be validated and authorized by the controller before being installed on the SDN switches hosted by every node.

4.2.1. Usual attacks

The first subsystem that may be targeted by a hacker is of course routing and consequently, as routing is enforced by SDN, OpenFlow. The latter is assumed to be operated on top of TLS or DTLS. In addition, we considered the usage of a secured variant of AODV, e.g. SUAP as described in [23].

By construction, each node is able to determine whether a transmission is part of the network or not: only those flows that have a flow entry in the switch will be processed. This means that any attempt to send a frame to a UAV will

be simply ignored by the network if it is not part of the acceptable flows in the network (based on its IP addresses, its transport protocol and, if applicable, its transport ports).

We also added a custom field in the AODV announcement packets that contains the MAC address of the sending node. This field has to be authenticated so MAC spoofing inside AODV field is not possible. By doing this and by deactivating ARP, ARP cache poisoning is made inoperative. Man in the middle attacks based on this technique are thus very unlikely.

A test bed containing two drones and an intruder has been created with the above described setting. Drones are started either with pure AODV or using our AODV/SDN architecture. The intruder can start different attacks using nmap and ettercap tools:

- ARP cache poisoning;
- port scanning;
- ICMP redirect attack;
- smurf attack;
- SYN flood attack.

The success or failure of different attacks on the victim drone are shown in Table 1.

4.2.2. Packet injection

As the intruder is allowed to send forged packets according to our threat model, we also have to consider packet injection attacks. This technique has been successfully used in some attacks to take control of a UAV by overwhelming the drone with the intruder commands in the hope that the drone will execute more unlawful than authorized actions. Of course, in the context of the work presented in this article, MAC and IP address spoofing of a legitimate and pre-existing flow have to be used in conjunction by the attacker.

Table 1: AODV vs. AODV/SDN vulnerability against common attacks

Attack	AODV	AODV/SDN
Eavesdropping	Vulnerable	Vulnerable
ARP cache poisoning	Vulnerable	Not vulnerable[1]
Port scan (with or w/o address spoof)	Vulnerable	Not vulnerable
OS fingerprinting	Vulnerable	Not vulnerable
SYN flood	Vulnerable	Not vulnerable[2]
MAC or IP spoofing	Vulnerable	Vulnerable[3]

[1] This does not prevent an attacker from performing a Man in the middle attack at the application level.

[2] SYN are dropped so the goal of the attack is not reached, but network performance would be affected due to numerous transmissions on the channel.

[3] The SDN switch is not able to detect packet injection that uses both MAC and IP addresses on a pre-existing flow as described in section 4.2.2.

It should be noted that injection of transport packets will not be transparent in the case of TCP streams. Indeed, the sending node will check the sequence number of the ACK packets and react depending on its local counter value. This makes packet injection more visible as the sequence number has to remain congruent with local counter and will lead to send supernumerary TCP ACK if this condition is not met [21].

It should also be remarked that some packet injection may have significant impact on the performance of the communication while remaining very difficult to detect. Sending an RST or a FIN is enough to terminate an open connection. This kind of attack will require an ad hoc mechanism and will not be considered here.

Given V the set of drones in the swarm and E the set of oriented edges representing the current radio visibility (or neighborhood) of drones as reported by AODV, we have:

$$E = \{(x, y) \in V^2\}$$

Let $G(V, E)$ be the oriented graph of neighborhood in the swarm of drones.

In this scenario, the intruder sends forged packets on a pre-existing flow. Let $f \in \mathbb{N}^n$ designate that flow defined by the n different attributes (addresses, protocol, ports). The controller, by accepting this flow, given the current topology G, has built a path P between the source node a and destination node b.

$$\begin{cases} P = \mu[a, b] = (u_0, \dots u_l), \forall i, u_i \in V, \\ u_0 = a, u_l = b, \\ \forall i \in [1..l], (u_{i-1}, u_i) \in E \end{cases}$$

In SDN, each node maintains counters for each flow table, flow entry, port, etc. In particular, when a packet matches a flow entry, the corresponding packet and byte counters are incremented. As a consequence, each node on the path P will count packets and bytes for the flow f.

Let $c_i(t) \in \mathbb{N}$ be the count of received packets or bytes, at node u_i of the path P at time t. As the propagation of data is not instantaneous and downstream nodes cannot receive a packet before it has been sent by the upstream node, we have:

$$\forall i, j \in [0..l], i < j \Rightarrow c_i(t) \geq c_j(t) \tag{1}$$

This invariant requires that the counters of all the nodes in a flow are read at the exact same time t and then sent to the controller. This does not imply that the drones have to have their clock synchronized, but only that they synchronize the readings.

This constraining real-time constraint may be relaxed by taking into account the monotonically increasing nature of bytes and packets counters. We have then:

$$\forall \Delta t > 0 \; c_i(t + \Delta t) \geq c_i(t) \tag{2}$$

Let t_i be the time at which $c_i(t)$ has been measured and t_j, the time at which $c_j(t)$ has been measured. By combining (1) and (2) we obtain:

$$\begin{cases} \forall t_i, t_j \in \mathbb{R}, t_i \geq t_j, \\ \forall i, j \in [0..l], \\ i < j \Rightarrow c_i(t_i) \geq c_j(t_j) \end{cases}$$

By checking if this invariant is true along the path, we can check if traffic injection occurred and locate the place where it occurred. However, there are two factors that limit the applicability of this invariant: they require some form of synchronization (either synchronizing the readings or synchronizing clocks to allow time comparisons) and they assume that no packet losses occur. Unfortunately, losses occur in a mobile network.

Moreover, mobility will reset counters on each path modification, making any further counter value comparison more difficult. That is why we suggest limiting counter checking to the start and end of the flow: flow entries specifically designed to count packets on the source and destination nodes resolve the issue with mobility counter resets.

Counter values can be read in OpenFlow using the Multipart Request/Reply primitive. In order to lower the required throughput, we implemented an automatic sending of Multipart requests as an experimental interaction. It has to be noted that the default format of Multipart request is highly modular, hence not very efficient. As an example, sending a single statistic entry for a single flow in a switch will require up to 242 bytes. A more specific data structure would require only about one hundred bytes for the flow description and a small number of counters: 82 bytes are due to the protocol overhead, each flow will require about 20 bytes.

A validation of this detection method has been successfully implemented in a static drone network and shows that packet injection is detected in at most two times the counters reading period.

The implemented counter measure consisted in changing the source port and destination ports to a randomly chosen unused couple at the source node, and

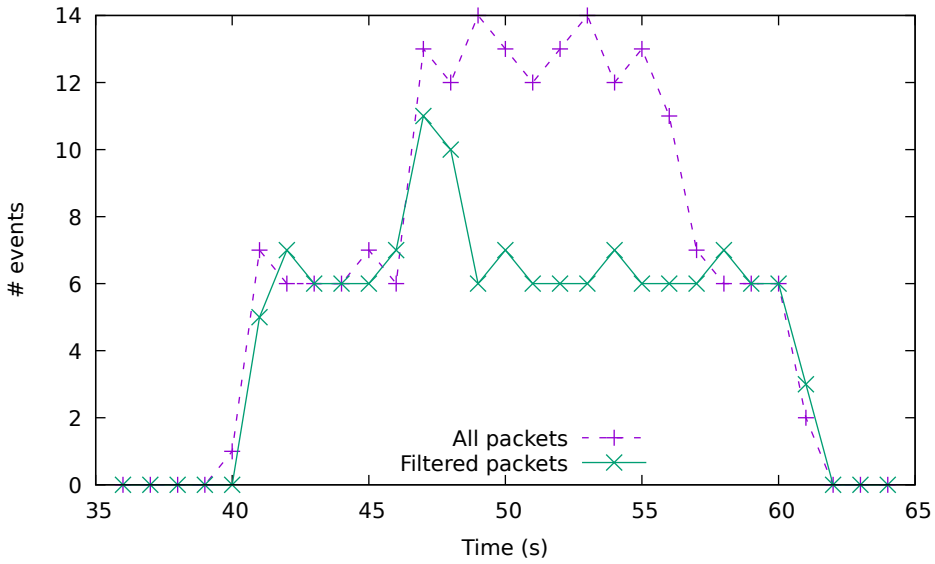

Figure 2: Incoming vs. filtered traffic during traffic injection attack

changing it back to normal at the destination. This counter measure is efficient enough to allow the traffic to be filtered out, as long as the intruder is not able to dynamically adapt the injected traffic to the new port couples. This requires being able to identify a protocol based only on the data and not on the ports. Obfuscating the behavior and what actually happens in the network by continuously changing the port pairs, may make the protocol detection and dynamic adaptation of the injected packets very difficult, but requires more OpenFlow transmissions and may thus impair battery lifetime. As a complement, the user application may implement steganography techniques [24] [25] as a way to check or to convey sensible data, but is outside the scope of this article.

Figure 2 shows the filtering of incoming packets using this invariant checking method and the proposed counter measure. The dashed line shows the number of packets received at the simulated wifi interface while the solid one shows the packets that went through the SDN switch and will be received by the application. Monitoring messages are sent one per second. The scenario is as follows:

- at $t = 40$, some traffic is received figuring a command and control protocol whose rate is about six packets per second;
- at $t = 47$, the intruder starts injection of packets;
- at $t = 49$, the intruder continues to inject packets as shown on the dashed line, however the controller has already filtered the traffic resulting in the difference with the solid line which returns to the usual value;
- at $t = 57$ the intruder has stopped injecting packets and at $t = 62$ the scenario is finished.

It should be noted that the simulated attack is not particularly strong (i.e. one injected packet for one legal one). It is not certain that it would have achieved its goal: taking control of the drone may require more injected packets. Injecting more packets would make the method event more efficient as the source counter and that of the destination would diverge more rapidly.

With a one second monitoring period, it took less than twice this time for the controller to detect and react. However, the automatic sending of counters should be limited to avoid sending too much data. Especially when the node experiences the kind of attacks that are dealt with in the next subsection.

4.3. Attacks from the inside and counter measures

We also considered the case of a drone being corrupted, perhaps by access gained before the mission e.g. when the drone is interconnected with a ground facility, or during the mission e.g. if the hacker managed to corrupt an on board application by traffic injection as shown in the previous subsection. Scenarios leading to this situation range from opening a remote shell (e.g. telnet) allowing the hacker to launch commands and attacks, to a virus, worm or Trojan horse executed on one drone, including arbitrary code execution though buffer overflows.

We did not consider the possibility of having the drone interconnected to the internet through the GCS for example, as it may be considered a more common

network setup where the GCS will act as a firewall and protect the UAVs from the outside.

In the above described cases, the hacker will have the possibility to make any usual attack as the controller authorizes any traffic as long as it starts from a node and arrives at another: trying to extend the control over other nodes or just trying to stop the mission. Starting from the list of attacks described in both the KDD99 dataset that has been used in many other studies, and the IDS2018 dataset, we selected denial of service (e.g. syn flood), unauthorized access from a remote machine (e.g. guessing password), and surveillance and other probing (e.g., port scanning) as the most probable attacks. Unauthorized access to local superuser (root) privileges (e.g. various buffer overflow attacks) is also a possible attack, but its detection requires more insight into the exchanged data and has been proven to be more difficult [10].

In addition to being possible attacks, they will also lead to unusual behavior on the control plane of the SDN network. The clue of these can then be tracked by the controller.

Moreover, once detected, the controller will be able to isolate the identified attacker by denying any new suspicious flow or even by deleting some flows. However, if it seems reasonable to deny new flows on the basis of the current behavior of the drone, identifying already established flows to be deleted will probably be prone to errors.

5. Detecting rogue nodes

Attack detection is usually devoted to network intrusion detection systems that analyze traffic in the network, or to host intrustion detection systems (HIDS) that also monitor the system and its resources.

Though the SDN switch is hosted by the node in the proposed architecture, it does not require access to host information like memory and CPU usage, processes, system calls, files, etc. and does not compare well with host intrusion detection systems.

As stated in section 2, usual signature based NIDS with deep packet inspection techniques will require additionnal transmissions and thus increase the risk of premature battery exhaustion on UAVs. This is why we focused on analyzing events at the controller interface only (also known as the southbound API), possibly complemented with traffic monitoring techniques like the one proposed in sub section 4.2.2.

Identification of an abnormal behaviour can then be addressed in machine learning using two approaches: anomaly detection in unsupervised learning or classification algorithms in supervised learning.

For unsupervised learning, there are effective online real-time techniques like those described in [26] to perform clustering and anomaly detection in a stream of data. However, the method raises two concerns.

First, whatever the method, the availability and the representativeness of a dataset is of primary concern when Machine Learning techniques are at stake. As far as we know, there is no dataset containing network traffic of different typical missions of a swarm of drones that may be used to evaluate a model in a close-to-reality environment.

Second, the unsupervised learning is based on the idea that normal traffic is significantly more common than abnormal traffic. This assumption cannot be guaranteed for all the scenarios and missions, and may be inefficient if the attack arrives very early in the mission at a moment when the model has not yet captured the normal traffic. As an example, continuous camera streams with command and control flows do represent a possible scenario with very steady flows. Unfortunately, steadiness takes any feedback away from the controller: a few PacketIn are received at the beginning, but once the required flow entries are installed, no more events will be received. Only monitoring counters such as that proposed for traffic injection detection could give feedback about the amount of data exchanged. However, monitoring data cannot be received in real-time as opposed to controller available data: requesting counter values from the nodes requires time, while the controller has to take a decision as soon as possible. Moreover, the increase in the amount of monitoring data to be transmitted

cannot be limitless as it will use the battery of the drones.

As long as no network capture is produced on a typical swarm of drones mission, results of an unsupervised learning model will remain questionable with regard to its applicability to real-life situations. Moreover, due to the lack of data about the *quality* of the exchanged information, the model may be difficult to train. As a consequence, we did not consider unsupervised learning any further.

5.1. Dataset

Having no datasets containing the kind of data required to train a new model, we had to create a new dataset from an existing one by building the point of view of the controller from a network capture: receiving PacketIn events for the first packet only and flow deletion events. We did not consider AODV events (e.g. route requests).

In doing so we made a firm assumption: an attack always exhibits a very similar behavior in OpenFlow events, regardless of the network. This assumption implies that a model trained on one dataset would be able to detect a similar attack on a different network, although the applications, the topology and the throughput of the network are different. They are at least as different as a wired network with web applications compared to the ad hoc network of a swarm of drones.

This assumption seemed quite reasonable as most of the targeted attacks (mainly volumetric) imply sending some network transmission "as quickly as possible". However, this may prevent detecting at least a part of the attacks that require lower transmission rates (application DoS) or attacks that are intentionally conducted slower than usual (e.g. brute force attack with only one attempts every second). We thus accepted to not be able to detect any sort of attack and considered that slowing an attack down sufficiently will annihilate its effect, given the limited duration of a drone mission.

5.1.1. Content

On the first packet of a flow for which an SDN switch has no flow entry, the latter sends a PacketIn to the controller. This is the starting point of the validation using the proposed model and thus has to be captured in the dataset. Date and all the details of the received packet are stored and the information of the flow is kept in memory. The dataset must contain all the flows, regardless of legitimacy, in order not to filter any PI event.

The controller will accept the flow and send a FlowMod command in a real SDN network. Subsequent packets will follow the newly created flow entry, and thus will not be sent to the controller. However, the building process uses them to check whether the flow entry is kept, considering an idle timer of 20s as with our test SDN switch.

In the case a second packet from the same stream arrives before the FlowMod is received by the switch, a new PacketIn event will be created for the controller. However, as the flow is already recognized, it will pass the packet to the destination as if it had been captured by the ongoing flow entry, without any validation. These events are not simulated in the dataset.

Mobility events are also filtered by the controller and will not be considered in the validation process: mobility is supposed to be transparent from a validation point of view. Mobility events (i.e. topology changes) do not need to be simulated in the dataset.

Finally, flows that are requested by the SDN switches for the backward direction (e.g. opening the second half connection of TCP) are ignored. Indeed, as the attacker is always the initiator in the kind of attacks that we have selected, the PI event of the response will probably not bring much information to the model. Moreover, it is possible for the controller to install reverse flow entries beforehand in the case of TCP, to lower the SDN overhead and latency. In such cases, no PI event would be generated.

The dataset will have to contain the following data to allow monitoring of the activity in the network, aggregate flows together to compute useful metrics,

etc:

- start and end date of the flow;
- source and destination addresses (MAC, IP, ports);
- all the protocol fields considered of any value, especially TCP flags, etc.

5.1.2. Building

We first considered the datasets commonly used in this field for similar detection techniques as possible sources for our new dataset. Any dataset with network captures and sufficient information concerning the attack scenarios along with benign traffic can be used. However, KDD99 and NSL-KDD do not contain enough information about the traffic to rebuild the OpenFlow events.

We finally selected the Intrusion Detection System dataset 2018 from the University of New Brunswick in collaboration with the Canadian Institute for Cybersecurity [27] as it contains the captured packets from all the hosts in the network in PCAP format with several typical attack scenarios like those that we selected. These data allowed us to determine the OpenFlow events occurring at the controller interface and thus to rebuild a separate dataset with the flow information:

- start and end of the flow, assuming 20s idle time;
- IP header fields: source and destination addresses and protocol;
- TCP, UDP or ICMP headers: ports, flags, reason, etc.;
- length of the first packet.

The dataset was generated on an AWS computing platform simulating a network of five subnets (R&D, Management, Technician, secretary and operation, and IT department) of about hundred hosts each, in addition to a servers subnet. The IT department machines run under Ubuntu operating system, while all the others are Windows machines (8.1 and 10 for the end users and 2012 and 2016

for the servers). Benign behaviors were simulated by a specific agent running on each end user machine, while attacks were started from a separate attacker subnet of 50 machines with the appropriate software, outside the previously described subnets. The attacks conducted in the network are listed below:

- infiltration of the network from inside (portscan);
- HTTP denial of service (DoS and DDoS);
- brute force attacks;
- SQL injection;
- botnet (screenshot and keylogger).

We developed and validated a toolset allowing the creation of a dataset from a list of PCAP files and the storage of the result in a database for later investigation and feature computation.

5.1.3. Validation

We have conducted several validations of the initial dataset, comparing the description of the dataset with what was observed through the identified flows. What were given as the main scenarios for the different capture files were confirmed. However, a number of deviations and some test cases were excluded from the list of potential training data.

Through this validation, we gathered the necessary data to build the attack scenarios that have been used to label the flows. In particular, although the so called "infiltration" scenario where a malicious file that exploits an application vulnerability to launch a script is received by email has not been used per se, it contains several port scanning attacks that were selected in the training dataset. The list of selected scenarios is provided in Table 2.

Capture files showed that a third of the flows come from the DNS application although DNS is never the target of the attack. Then comes HTTPS with about a quarter of the flows, followed by RDP and finally HTTP. Figure 3 show the

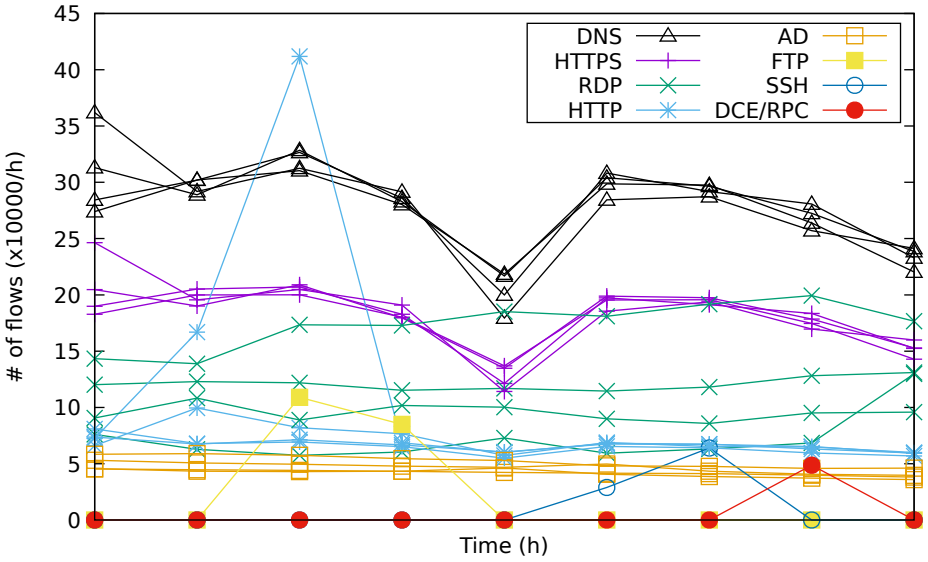

Figure 3: Number of flows per application in the dataset

Table 2: Selected scenarios for training

Date	Attack
14/02/18	Brute force
15/02/18	DoS
20/02/18	DDoS
28/02/18	Scan

evolution of the number of flows per application during the different scenarios. Except for one clearly identifiable HTTP DoS attack, the other attacks do not appear obviously separable in terms of number of induced flows.

5.2. Features

Using the description of the PacketIn events and the flows as described in a previous section, we have computed some statistics or measurements that

could be used as a feature in our model: node degree of the source and of the destination note in the graph of current flows, current average number of flows per second of the source and of the destination. Nevertheless, none were kept as final feature though. We also considered counting the number of events corresponding to some errors, especially ICMP port unreachable. However, the presence of this symptom may depend on the IP stack implementation and on the transport protocol.

In addition, we also incorporated two measurements which are presented in the following subsections, for which we extracted several features with the following process.

We hypothesized that characterizing flows based on the previous ones created by the same source or same destination could be a valuable feature for a model to identify the kind of misbehavior we intend to detect. Thus, for a single mathematical expression, we computed one feature per group having the same value(s) for given protocol field(s), providing a list of PI and their associated t_i times of arrival.

We extended this to every protocol field that could be used to group flows together and defined different grouping criteria based on the fields in the protocol headers of the PI. The fields used as grouping criteria are:

- source and destination IP addresses;
- IP protocol, which will differentiate between UDP and TCP flows;
- source and destination ports for TCP and UDP ports;
- type and code for ICMP

It has to be noted that the number of criteria has an impact on the required memory and thus potentially on the performance as the controller will need more time to reach the previous value for the same group. The higher the number of criteria, the fewer events per second we should have, and the higher the required memory to store the ongoing groups: in a network with N nodes, there will be at most N sources but $\frac{N(N-1)}{2}$ couples (source IP, destination IP).

Garbage collector techniques to delete groups that would be too old may be required here, depending on the number of criteria.

Two specific features were computed that represent the activity of the group and the port span. In the following, the set of t_i represents the time of arrivals of the PI events of the same group, regardless of the criteria used to define this group. The mathematical definitions of these features are described hereafter.

5.2.1. Measuring activity

The aim of this measurement is to give feedback about the activity, as perceived from the controller of an SDN network. As described in a previous section, the controller does not receive all the packets but only the first packet of each flow in the form of a PacketIn event. There are cases where it could receive several packets at the beginning of a flow, but only the first will be sugmitted for a validation.

PacketIns are discreet events that occur at times t_i, while a flow has a duration, packet count, byte count etc. Activity should be understood as the number of created flows, not the amount of data a node generates. It would be possible to count the number of flows a node has (the degree of a node) but this does not take into account the time or speed of flow creation. To obtain a measurement of the speed, we need to count the number of PacketIn events per time unit. Computing the average number of PacketIns per unit time or in a time window requires keeping some history which may require a huge amount of memory if there are many flows. Another approach would be to use the average value or the Inter Arrival Time (IAT) of these events, either using a moving window or with an exponentially weighted moving average (EWMA).

We propose to use another measurement that employs exponentially decreasing properties as for the EWMA, but with the following formula:

Let $E = \{t_i \in \mathbb{R}\}$ be the set of times where the selected PacketIns occurred. The selection criteria may be any criteria. In our application, this consists of a set of characteristics that bind the flows that generated these PacketIn events together (e.g. same source node). At time t, let $F = \{t_i \in E : t_i \leq t\}$ be the

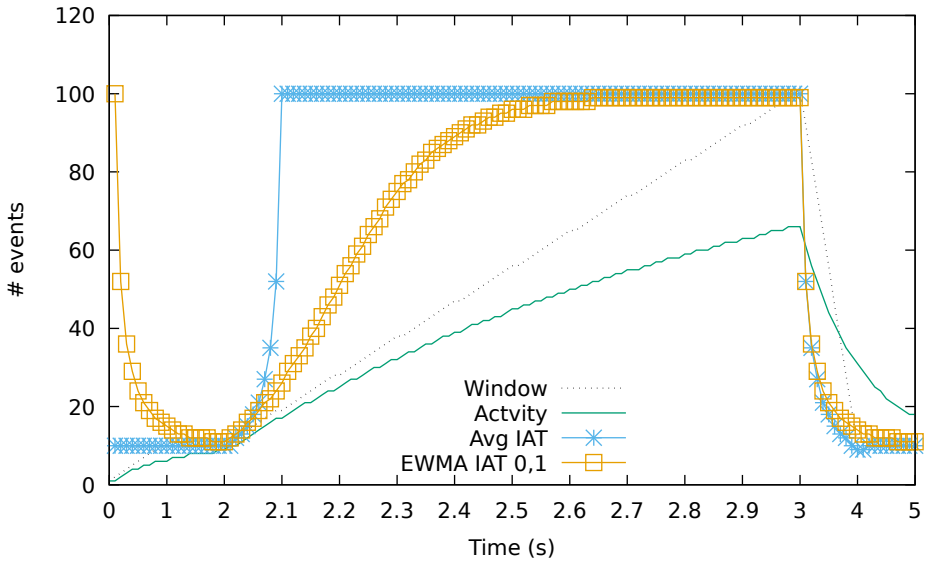

Figure 4: Activity vs. IAT average and EWMA

set of previous events and $n = |F|$ the number of events in F. A can then be written:

$$A(t) = \sum_{i=1}^{n} \alpha^{t-t_i} \tag{3}$$

with $\alpha \in \mathbb{R}$ and $0 < \alpha < 1$.

However, we will examine the activity value only on PacketIn event arrival time (i.e. at each t_i). At time t_n, (3) can be written:

$$A(t_n) = 1 + \sum_{i=1}^{n-1} \alpha^{t_n - t_i} = 1 + \alpha^{t_n - t_{n-1}} \sum_{i=1}^{n-1} \alpha^{t_{n-1} - t_i}$$

And then:

$$S_n = A(t_n) = 1 + \alpha^{t_n - t_{n-1}} A(t_{n-1}) = 1 + \alpha^{\delta t_n} S_{n-1} \tag{4}$$

where $\delta t_n = t_n - t_{n-1}$.

The advantage of this activity definition, compared to a moving average of the inter arrival time (or its multiplicative inverse), is that (4) directly uses the time in the exponent part. As a comparison, exponentially weighted moving

average of the form $S_n = \alpha\delta t + (1-\alpha)S_{n-1}$ does not "forget" about the past based on t.

Figure 4 shows the variation of the different activity measurements proposed so far. The scenario is as follows: the PI arrives each tenth of a second starting at $t = 0$ and for two seconds, then every hundredth of a second for one second, and finally back to every tenth of a second for two seconds. It should be noted that the x axis is not linear in order to ease the variation comparison.

The dotted line shows the actual number of PI events during a one second sliding window. This curve represents the most accurate value of the activity but does not have any form of *memory*. The average IAT reacts very quickly as soon as the frequency of events varies. The EWMA with $\alpha = 0.1$ is closer to the window value but also moves very quickly and presents an unexpected peak at the beginning. This artifact is due to the α value and could be reduced with higher values. In this case, however, the EWMA curve will get closer to the average curve rather than that of the windowed.

On the contrary, the proposed activity measurement, though slower in catching the maximum value, exhibits interesting properties both in terms of similarity with the shape of the window curve and of smoothness, which can be considered as a memory effect of the previous values.

Moreover, if δt is high enough, (4) will neglect the S_{n-1} term while a moving average will not, as $1 - \alpha$ does not depend on δt.

Concerning the determination of the α parameter, instead of the well known half life duration method used for exponentially declining functions, we preferred a more practical approach. Our method consisted in estimating the number of events in the last N seconds (counting the number of events in a fixed size sliding time window). As this method will require keeping the history or at least a part of it and will require too much memory, we will compute the value of the parameter α for which (3) provides the same value as this time windowed method, in a steady inter arrival time situation.

For a given constant flow of PacketIn events at frequency f, we have $S_n = S_{n-1}$ and thus $S_n = 1 + \alpha^{\frac{1}{f}} S_n$. We would also like to have α such that $S_n \approx$

Algorithm 1 onEvent(pi)

```
1: inc ← pi.length/B
2: t ← pi.time
3: for all feat ∈ activity_features do
4:    if ∃groups[feat][pi.get(feat.criteria]) then
5:       group ← groups[feat][pi.get(feat.criteria)]
6:    else
7:       group ← new group() {Initializes s_n to 0}
8:       groups[feat][pi.get(feat.criteria)] ← group
9:    end if
10:   group.s_n ← inc + feat.α^(t−group.last) ∗ group.s_n
11:   group.last ← t
12:   feat.result ← group.s_n
13: end for
```

Table 3: Alpha value for different parameter values

Delta T	2Hz	10Hz	100Hz	1000Hz
1s	0.25	0.348678	0.366032	0.367695
10s	0.9025	0.904382	0.904792	0.904833
100s	0.990025	0.990045	0.990049	0.990050

$\Delta T \times f$. Resolving these equations gives:

$$S_n = \frac{1}{1 - \alpha^{\frac{1}{f}}} \tag{5}$$

$$\alpha(f, \Delta T) = \left(1 - \frac{1}{f \times \Delta T}\right)^f \tag{6}$$

As α is not constant for any frequency f, we will select one value for a reasonable value of f. See Table 3 for different values of the parameter α.

Empirical analysis of the error shows that the error is relatively small: the

Table 4: Maximum activity at different transmission lengths and throughput, with $\alpha = 0.75$

	L=226		L=1334	
B (bps)	Eq. 4	Eq. 7	Eq. 4	Eq. 7
10^4	1.97x10^1	3.57	3.78x10^0	4.04
10^5	1.92x10^2	3.48	3.30x10^1	3.53
10^6	1.92x10^3	3.48	3.26x10^2	3.48
10^7	1.92x10^4	3.48	3.25x10^3	3.48
10^8	1.92x10^5	3.48	3.25x10^4	3.48

error for $\alpha(1000Hz, 10s)$ is higher for lower frequencies and about 5% for $f = 1Hz$ ($M_n(t_n) = 10.5$ instead of 10) and less than 1% $\forall f > 4Hz$.

The definition of the activity as described by (3) is an absolute value whose maximum is not generic as it depends on two factors: the speed and the length of a transmission. Actually, the maximum activity of a node and hence the measurement we provided an algorithm for, is limited by the maximum number of PacketIns that a node could possibly generate, which is directly bound to the speed of the transmission and the length of the data sent in the first packet of the flow and contained in the PacketIn event.

If B is the network throughput and L is the length of each packet, then the maximum frequency f at which the PacketIn events arrive is: $f = \frac{B}{L}$.

Hence, according to (5), with the same throughput but different length of packets, the maximum activity will lead to different values of S_n. Similarly, for the same length of packet, if we modify the physical throughput, S_n is modified.

As we intend to train our model on a dataset captured on one network and use it on another with a throughput that may be slower by several orders of magnitude, we have to take care that the features are not too dependent on the physical characteristics of these networks. We expected the model to detect

high speed activity that needs to be relative to the maximum capacity. As a consequence, the model should be fed with a measurement of the activity relative to the maximum activity rather than with absolute values.

Computing the maximum value obtained depending on the length of the packet and the throughput of the network shows that the ratio is proportional to the throughput and inversely proportional to the length. For the ranges of values of L and B that correspond to our scenario, multiplying S_n by $K = \frac{L}{B}$ results in a maximum value that is almost constant whatever L and B. However, a variation depending on L remains that may become significant for low values of B and α. See Table 4.

Then (4) transforms into:

$$S_n = \frac{L_i}{B} + \alpha^{\delta t} S_{n-1} \tag{7}$$

Algorithm 1 shows how to compute and update S_n for each PacketIn event.

5.2.2. Transport level port span

This feature represents the measurement of the difference between the minimum and the maximum port number used at the transport protocol level. The intent is to capture a misbehavior where a node starts to scan or only creates many connections in a short period of time, like in a brute force attack. Indeed, depending on the attack and on the IP stack software, either the source or the destination or both port numbers will increase for a short period of time during these attacks: it is the aim when one tries to scan all the open ports, it is a consequence of opening several of connections in the case of a brute force attack.

One could compute this by taking the currently opened flows and extract the minimum and the maximum values. However, this algorithm would fail to measure the behavior as the deletion of a flow depends on external parameters: idle timer, whether a flow is closed on TCP FIN or not, whether the flow is actually accepted by the controller, etc. This is why we preferred to use a different definition of the port span.

We will compute maximum and minimum values as being the maximum

(resp. minimum) between the transport port received in the PacketIn and a decreasing (resp. increasing) function depending on the time and the port value in the previous PI.

This function will memorize and artificially maintain the maximum value for some time ΔT and then linearly drop the maximum down to 0 (and conversely, increase the minimum to an absolute maximum value). We used the same function for both the minimum and maximum, except that the slope of the linear part is negative for the maximum and positive for the minimum. This requires storing the times of the maximum or the minimum last update.

Let $\delta t_m = t - T_m$ and $\delta t_M = t - T_M$ be respectively the time difference between the current time and the last time minimum has been reduced (resp. maximum has been increased). Let the function g be defined, for given p, k ($k > 0$ for the minimum and $k < 0$ for the maximum) and ΔT, as:

$$g(p, k, \Delta T, \delta t) = \begin{cases} p & \forall 0 \leq \delta t \leq \Delta T \\ p + k\ (\delta t - \Delta T) & \forall \delta t > \Delta T \end{cases}$$

$$S(t) = min(0, g(M, -k, \Delta T, \delta t_M) - g(m, k, \Delta T, \delta t_m)$$

5.3. Machine Learning algorithm and training set

As explained, the detection of malicious behaviors will be performed on the controller side when receiving a PacketIn, just before the controller accepts a flow by issuing a FlowMod request to install the flow entries on the drones. We made the assumption that the controller is located in the ground station, which will have more processing power than a UAV platform. However, we do not want to exclude the possibility of having the controller located somewhere else. Moreover, the machine learning algorithm should not impose additional requirements on the processing unit of the ground station and stay responsive even on a slow PC.

The list of candidate features include both numeric values (e.g. the activity as defined in the previous section) and symbolic ones (e.g. IP protocol, flags), and the chosen algorithm should handle both types of features. In addition,

Algorithm 2 onTransportPIEvent(pi)

```
1:  t ← pi.time
2:  for all feat ∈ pspan_features do
3:      feat.result ← 0
4:      port ← feat.select_port(pi) {source or destination}
5:      if ∃groups[feat][pi.get(feat.criteria]) then
6:          group ← groups[feat][pi.get(feat.criteria)]
7:          min ← g(pi.min, feat.k, feat.ΔT, t − pi.tmin)
8:          max ← g(pi.max, −feat.k, feat.ΔT, t − pi.tmax)
9:          if port ≤ min then
10:             group.min ← port
11:             group.tmin ← t
12:             min ← port
13:         end if
14:         if port ≥ max then
15:             group.max ← port
16:             group.tmax ← t
17:             max ← port
18:         end if
19:         if max > min then
20:             feat.result ← max − min
21:         end if
22:     else
23:         group ← new group(pi.port, t)
24:         groups[feat][pi.get(feat.criteria)] ← group
25:     end if
26: end for
```

Table 5: Class balance in training datasets

Dataset	Benign	Brute Force	DoS	DDoS	NMAP
Binary	50%	12.5%	12.5%	12.5%	12.5%
Multiclass	20%	20%	20%	20%	20%

as feedback is provided to the operator of the swarm, the machine learning algorithm should construct a model that allows interpretation.

For these reasons, Neural Networks have been set aside. We finally chose the Random Forest Classifier algorithm for its simplicity of usage as it does not require the features to be normalized prior to training.

Given the amount of data extracted from the UNB IDS dataset (several hundreds of thousand PI), we draw the learning curves to determine the maximum number of samples of the training set to avoid overfit. A quantity ranging from 1000 to 10000 samples seems to be enough for the model to converge while avoiding the risk of a high overfit.

As we had different attacks in the dataset, we generated both a dataset for a binary classifier (benign or attack) and another for a multiclass classifier (benign and each attack) for comparison purposes.

In addition, we created balanced and unbalanced datasets. The balance of the dataset is constructed by selecting the same number of samples per class, for benign and each of the attacks concerning the multiclass classifier. However, in the case of a two class classifier, we have the same number of benign and attack samples, the latter containing the same number of samples for each type of attack. Ratios are given in Table 5.

Building the model was then achieved by randomly selecting 10000 samples in the balanced dataset, of which 20 per cent were kept for testing purposes.

Some testing on several datasets with different ratios between the classes confirmed that the model was sensitive to the balance of the dataset, showing more false negative (and conversely fewer false positive classifications) when the

number of benign samples was increased.

As in most security detection model, a good model should have a low false positive ratio. That is why we will not focus the evaluation only on precision, but specifically on recall and F1 score. Moreover, while the precision, recall or F1 scores are the usual performance criteria proposed for machine learning models, the usability of the model is determined by its rapidity of detection.

6. Results

Using the above described dataset and applying the selected machine learning algorithm, we build and evaluate two classifier models with the most important features: one for binary and one for multiclass classification, the latter being able to distinguish between the different types of attack.

6.1. Selected features

The covariance matrix of the features shows high scores for similar groups (i.e. having several common grouping criteria). As an example, activity grouped by IP address couple is very similar to the activity grouped by IP address couple and protocol. Recursive Feature Elimination, which starts with all the features and removes the less significant recursively, proved to be inefficient and prone to errors. Indeed, the model replaces the removed features by similar ones when they existed, usually leading to an insignificant score difference.

We decided to start with only one feature, selecting the one that would have the best F1 score, using the cross validation dataset. For each feature not yet selected, we trained the model with the new feature and noted the score difference with the model without the new feature. We selected the feature that achieved the best improvement in the classification as described in [28].

There was however one exception to this method. In fact, providing IP protocol or TCP flags as features gives the model a way to determine whether the traffic is TCP or not. Scans using TCP or UDP should be very similar though we had only TCP scans in the dataset. The score increase of these

features that are based on TCP specific features (TCP flags) is thus biased. Instead of crafting UDP scans from the TCP ones into the dataset, we put aside those features that may have led the model to be transport layer aware.

It is to be noted however that the transport port span feature is directly bound to the underlying transport protocol and that the model may not be able to generalize on some ICMP based attack scenarios.

With the method described above, we selected the four following features:

- the activity of the destination node application;
- the activity of a pair of nodes, per transport protocol;
- the source port span grouped by destination address and protocol;
- the same for the destination port.

Table 6 lists the parameters of the Random Forest Classifier algorithm used to train our model, as returned by get_params in SciKitLearn v0.22.2.post1.

6.2. Test scenarios

The evaluation of the model was made using data from the initial dataset that contains internet like traffic of a wired network. However, as we intend to use the model on drones, a validation using a dataset with more realistic traffic is needed. This is the reason for building separate datasets for testing: a normal scenario, based on 4 drones and a ground station, with command and control, telemetry for each drone and camera flows for three of them. In addition, they exchange data with each other representing the mission data. Mission data are simulated using iperf, web and ping exchanges. Table 7 summarizes the different scenarios.

Starting from this normal scenario, we started different attacks from one drone to another, using different techniques and different parameters: brute force, port scan, DoS using SYN flood, OS fingerprint or network explore. Some scenarios use the same attack but with different speeds or a different transport protocol. Table 7 summarizes the attack scenarios.

Table 6: SciKitLearn Random Forest Classifier parameters

Parameter	Value
n_estimators	100
min_weight_fraction_leaf	0.0
class_weight	None
min_samples_leaf	1
max_leaf_nodes	None
random_state	None
max_depth	None
bootstrap	True
max_samples	None
ccp_alpha	0.0
min_samples_split	2
max_features	'auto'
criterion	'gini'
warm_start	False
min_impurity_decrease	0.0

We then extracted the data as we made it on the data from the UNB IDS captures, labeled the sample according to the scenario and compared the predictions of our model with the labels.

6.3. Attack detection

Table 8 shows the scores of the model when applied to our validation test scenarios. The most important scores are benign recall (the higher the better) as

Table 7: Test scenarios with parameters

Scenario	Attack	Parameters
no_attack	None	iperf + web + ping
patator	Brute force	default
scan_normal	NMAP	normal speed
scan_polite	NMAP	polite speed
scan_sneaky	NMAP	sneaky speed
synflood	DoS	
OS fingerprint	NMAP	
UDPscan	NMAP	normal speed
explore	NMAP	polite speed

they show the ratio of false positive and attack precision. In each scenario where the model detected an attack, the model identified only one (attacker, victim) pair and correctly identified each of them. Finally, the last column shows how much time it takes for the model to identify the attack.

Although the performance of the multiclass classifier shown in Table 9 is close to that of the binary one, it usually failed in providing useful additional information. Indeed, if it identifies the fact that an attack is underway, it sometimes fails in determining accurately the type of attack the network is undergoing. As a consequence, no specific countermeasure can be applied. This may not be surprising as the model uses the same number of features as in the binary class but needs to separate more classes. In addition, brute force and SYN flood imply the same type of behaviour: one attacker continuously opening new flows to one victim on the same port.

The binary classifier with only two classes (benign or attack), achieves good performances in general: recall of benign traffic is close to 1.0 and misclassifi-

Table 8: Binary RFC model scores with detection speed

		Benign		Attack		
Scenario	Acc	prec.	recall	prec.	recall	Δt
no_attack	1.0	1.00	1.00	-	-	N/A
finger	0.88	0.70	1.00	1.00	0.84	<1ms
polite scan	0.94	0.81	1.00	1.00	0.92	1ms
synflood	0.99	0.88	1.00	1.00	1.00	10ms
normal scan	0.90	0.74	1.00	1.00	0.87	14ms
patator	0.84	0.51	0.96	0.99	0.83	27ms
UDP scan	0.98	0.97	1.00	1.00	0.98	1.2s
sneaky scan	0.84	0.84	1.00	0.00	0.00	-
explore	0.18	0.18	1.00	0.00	0.00	-

cations are linked to the fact that benign traffic took place during the attack. The model seems to classify the nodes as either benign or offender, rather than classifying the flows themselves. This is compliant with the countermeasures we discussed previously: an infected node should not be trusted and must be taken out of the mission. It is important to note that the technique consists in denying any new flow, but that already established traffic may still be allowed, in the hope of sending a return to base command and obtain telemetry data.

There are three principal exceptions to the good performances of our model. UDP scan is detected but its detection takes more time. When the victim's ports are slowly scanned, the model fails to identify the attack. However, slowing down the scan will leave less time to explore and detect open ports. The explore scenario is an attempt to ping different addresses in the network to find other victims. This attack is not present in the dataset.

As the training dataset and the verification dataset are based on different

Table 9: Multiclass RFC model scores with detection speed

		Benign		Attack		
Scenario	Acc	prec.	recall	prec.	recall	Δt
no_attack	1.0	1.0	1.0	-	-	N/A
finger	0.78	0.55	1.0	1.0	0.70	2ms
polite scan	0.95	0.82	1.0	1.0	0.93	3s
synflood	0.84	0.93	1.0	1.0	0.84	23ms[1]
normal scan	0.8	0.58	1.0	1.0	0.72	<1ms
patator	0.15	0.68	0.96	0.0	0.0	<1ms [2]
UDP scan	0.99	0.98	1.0	1.0	0.99	<1ms
sneaky scan	0.84	0.84	1.0	0.0	0.0	-
explore	0.18	0.18	1.0	0.0	0.0	-

[1] This value corresponds to the time between the start of the attack and the first correct classification. The model did detect an attack after 19ms but misclassified the type of attack.

[2] The model did detect an abnormal behavior but misclassified the attack.

network topologies, it tends to exclude obvious data leakage that would explain the good performances of the model.

In terms of performance, the mean execution time for a single prediction varies between 4 and 45 micro seconds on a Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz using SciKitLearn v0.22.2.post1.

7. Conclusion and future work

In this article, we showed how the SDN based mobile network architecture presented in a previous article can be used and complemented to secure the network of a swarm of collaborating drones. First, the network design provides the capability to differentiate between outside and inside traffic. The SDN flow entries can thus be designed in such a way that transmissions from the outside will be ignored by the nodes in the majority of cases (i.e. except for traffic injection with address spoofing). We then demonstrated two techniques to protect the network from outsider attacks using traffic injection and common insider attacks.

Outsider attack detection is based on counter comparison with dedicated flow entry to count packets at the edge of the network. The counter measure demonstrated in this article is based on TCP header modification and proves to be efficient, as long as the attacker is not able to adapt to the new conditions. It is however dependent on the monitoring frequency, which has to be balanced with the increased amount of monitoring data transmissions. This counter measure can be extended to apply long term network obfuscation, by regularly changing TCP header fields (like TCP ports).

Insider attack detection is based on a signature based detection that differs from other proposals by using only OpenFlow events as input for abnormal behavior identification. We built a dataset starting from an existing set of network traffic capture files. We proposed two functions along with an event selection process for computing the features of the new dataset. Recursive feature addition was applied and four features were selected. Supervised machine learning is then applied, using a Random Forest Classifier model. We showed that the detection has a good F1 score (high recall while precision may be lower) with a small detection delay. The hackers may still throttle the attack to remain undetected, but will then most probably not achieve their goal (e.g. DoS, port scan, password detection), especially as the mission is supposed to last for a limited period.

We intend to implement these techniques on our controller along with some counter measures to test the efficiency of the solution. We then plan to apply the whole system to a real drone.

It would also be worthwhile checking different values of the parameters, and comparing different machine learning algorithms, in terms of accuracy and F1 score and in terms of CPU usage and detection time. The influence of the size of the network could also be evaluated, both in terms of controller CPU usage and its effect on the detection performances.

Other forms of attacks could also be tested to evaluate whether the model is able to generalize. If the model is unable to generalize, the training set will have to be extended with corresponding samples in order to evaluate if the model is able to correctly classify new attacks.

Finally, unsupervised learning could be evaluated with the same features, as soon as a representative dataset for swarm of drones mission becomes available.

References

[1] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, M. Alam, Cyber security threat analysis and modeling of an unmanned aerial vehicle system, in: 2012 IEEE Conference on Technologies for Homeland Security (HST), 2012, pp. 585–590. doi:10.1109/THS.2012.6459914.

[2] Etsi tr 101 771 v1.1.1; tiphon release 4; service independent requirements definition; threat analysis (2001).

[3] E. A. Oladimeji, S. Supakkul, L. Chung, Security threat modeling and analysis: A goal-oriented approach, in: Proc. of the 10th IASTED International Conference on Software Engineering and Applications (SEA 2006), 2006, pp. 13–15. doi:10.1.1.103.2997.

[4] C. G. L. Krishna, R. R. Murphy, A review on cybersecurity vulnerabilities for unmanned aerial vehicles, in: 2017 IEEE International Sympo-

sium on Safety, Security and Rescue Robotics (SSRR), 2017, pp. 194–199. doi:10.1109/SSRR.2017.8088163.

[5] V. L. L. Thing, J. Wu, Autonomous vehicle security: A taxonomy of attacks and defences, in: 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016, pp. 164–170. doi:10.1109/iThings-GreenCom-CPSCom-SmartData.2016.52.

[6] F. Fayollas, A. Vacher, Drone predator, one drone to rule them all, Master's thesis, TLS-SEC, accessed on October 2020 (2019).
URL `https://git.tigre-bleu.net/antoine.vacher/predator-drone`

[7] K. Ledieu, F. Pennel, A. Pernot, Attaques sur un ar drone 2.0 parrot, Master's thesis, TLS-SEC (2019).

[8] J. C. Correa Chica, J. C. Imbachi, J. F. Botero Vega, Security in SDN: A comprehensive survey, Journal of Network and Computer Applications 159 (2020) 102595. doi:https://doi.org/10.1016/j.jnca.2020.102595.
URL `http://www.sciencedirect.com/science/article/pii/S1084804520300692`

[9] S. Shin, L. Xu, S. Hong, G. Gu, Enhancing network security through software defined networking (SDN), in: 2016 25th International Conference on Computer Communication and Networks (ICCCN), 2016, pp. 1–9. doi:10.1109/ICCCN.2016.7568520.

[10] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, O. M. Caicedo, A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, Journal of Internet Services and Applications 9 (16). doi:10.1186/s13174-018-0087-2.

[11] U. of California, Knowledge discovery and datamining cup 1999

data, https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, accessed on July 2020 (1999).

[12] M. Tavallaee, E. Bagheri, W. Lu, A. Ghorbani, A detailed analysis of the kdd cup 99 data set, IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA 2. doi:10.1109/CISDA.2009.5356528.

[13] NSL-KDD dataset, https://www.unb.ca/cic/datasets/nsl.html, accessed on July 2020 (2009).

[14] L. Boero, M. Marchese, S. Zappatore, Support vector machine meets software defined networking in ids domain, in: 29th International Teletraffic Congress, Genoa, Italy, 2017, pp. 25 – 30.

[15] P. Sangkatsanee, N. Wattanapongsakorn, C. Charnsripinyo, Practical real-time intrusion detection using machine learning approaches, Computer Communications 34 (18) (2011) 2227 – 2235. doi:https://doi.org/10.1016/j.comcom.2011.07.001.
URL http://www.sciencedirect.com/science/article/pii/S014036641100209X

[16] C. Guerber, N. Larrieu, M. Royer, Software defined network architecture, in: ICUAS 2019, 2019.

[17] A. V. Uskov, Information security of ipsec-based mobile vpn: Authentication and encryption algorithms performance, in: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 1042–1048.

[18] Rescorla, E., The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446 (2018).

[19] Rescorla, E. and Modadugu, N., Datagram Transport Layer Security Version 1.2, RFC 6347 (2012).

[20] J. Postel, Transmission control protocol, RFC 793 (1981). doi:10.17487/RFC793.

[21] L. Joncheray, A simple active attack against TCP, in: Proceedings of the 5th Conference on USENIX UNIX Security Symposium - Volume 5, SSYM'95, USENIX Association, USA, 1995, p. 2.

[22] C. Perkins, E. Belding-Royer, S. Das, Ad hoc On-Demand Distance Vector (AODV) Routing, RFC 3561 (Jul. 2003). doi:10.17487/RFC3561.

[23] J. Maxa, M. S. Ben Mahmoud, N. Larrieu, Extended verification of secure uaanet routing protocol, in: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), 2016, pp. 1–16.

[24] K. Szczypiorski, Steganography in TCP/IP networks. state of the art and a proposal of a new system-hiccups, Warsaw University of Technology, Poland Institute of Telecommunications, Warsaw, Poland.

[25] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr, K. Szczypiorski, Information hiding in communication networks: fundamentals, mechanisms, applications, and countermeasures, John Wiley & Sons, 2016.

[26] M. Carnein, D. Assenmacher, H. Trautmann, An empirical comparison of stream clustering algorithms, in: Proceedings of the Computing Frontiers Conference, 2017, p. 361–366. doi:10.1145/3075564.3078887.

[27] U. of New Brunswick, Cse-cic-ids2018 on aws, `https://www.unb.ca/cic/datasets/ids-2018.html`, accessed on July 2020 (2018).

[28] T. Hamed, R. Dara, S. C. Kremer, Network intrusion detection system based on recursive feature addition and bigram technique, Computers & Security 73 (2018) 137 – 155. doi:https://doi.org/10.1016/j.cose.2017.10.011.
URL `http://www.sciencedirect.com/science/article/pii/S0167404817302274`