DisPEED: Distributing Packet flow analyses in a swarm of heterogeneous EmbEddeD platforms

Louis Morge-Rollet*‡, Camélia Slimani *‡, Laurent Lemarchand†‡, Frédéric Le Roy*‡, David Espes†‡, Jalil Boukhobza*‡
*ENSTA, Institut Polytechnique de Paris,
† Université de Bretagne Occidentale,
‡ Lab-STICC, CNRS, UMR 6285, F-29200 Brest. France
Email: {frederic.le_roy, jalil.boukhobza}@ensta.fr,
{laurent.lemarchand, david.espes}@univ-brest.fr

***Abstract*—Security is a major challenge in swarm of drones. Network intrusion detection systems (IDS) are deployed to analyze and detect suspicious packet flows. Traditionally, they are implemented independently on each drone. However, due to heterogeneity and resource limitations of drones, IDS algorithms can fall short in satisfying Quality of Service (QoS) metrics, such as latency and accuracy. We argue that a drone can make profit from the swarm by delegating part of the analysis of their packet flows to neighbor drones that have more processing power to enforce security. In this paper, we propose two solving methods to distribute the packet flows to analyze among drones in a way to ensure that it is processed with a minimum communication overhead to limit the attack surface, while ensuring QoS metrics imposed by the drone mission. First, we propose a formulation of the distribution problem using both an Integer Linear Programming (ILP) and a Maximum-Flow Minimum-Cost (MFMC). Furthermore, we propose two specific solving methods for the distribution problem: (1) a Greedy Heuristic (GH), a non-exact solving method, but with small time overhead, and (2) an Adapted Edmonds-Karp (AEK) algorithm, an exact method, but with a higher time overhead. GH proved to be a very fast solution (up to more than 2000x faster than ILP with Branch and Bound), while AEK solution proved to find the exact solution even when the problem is very difficult.**

***Index Terms*—IDS, swarm drones, distribution solving methods.**

I. Introduction

Drones are increasingly used for missions, such as surveillance or inspection. Swarms of drones are used to address the capacity limitations of single drones (e.g. autonomy and payload) [1]. According to the United Nations Institute for Disarmament Research (UNIDIR), swarm robotics (including drone swarms) is a large group of robots defined by the following five properties [2]: Mass (from two to thousands of units), diversity (heterogeneous drones with different configurations [3]), collaborative behavior, communication (e.g. using Mobile Ad-Hoc Networks (MANET) [4]), autonomy and decentralization (to react without human intervention).

Swarm of drones are still in their early development stages as several challenges remain open. One of the most salient one is security [2]. In particular, MANET networks used in distributed drone swarms are vulnerable to network intrusion threats. Intrusion Detection Systems (IDS) are a promising solution, they are systems that identify malicious actions by analyzing the network traffic. In the case of drone swarms, IDS operates as a standalone system, where each drone independently analyzes its local traffic packet flows [5].

Swarms of drones are heterogeneous: (1) in terms of platform, that is, they consist of different embedded processing units (some with CPUs, others with GPUs or FPGAs), and (2) in terms of state, as some may have a lesser battery energy or more processes running, etc. We argue that in cases where a single drone cannot comply with QoS metrics, it is necessary to be able to offload part of the IDS analysis to one or more (powerful) neighbors to timely identify the threat. Several state-of-the-art studies investigated collaborative IDS [5]–[7], but mainly to share the results of the analysis. We argue that it is indispensable to be able to share also the packet flows to guarantee on-time detection on heterogeneous embedded platforms such as drone swarms.

This paper investigates IDS distribution by proposing a method for analyzing the packet traffic of the swarm to detect intrusions. The method takes into account the drone states (available energy, available memory), mission states (criticality level, traffic load), and the swarm topology, while minimizing the communication cost related to the distribution to limit the attack surface. In particular, our contribution relies on a distribution phase that makes it possible to offload part of the total traffic to analyze for each drone. To do so, we proposed a formulation of the distribution problem using Integer Linear Programming (ILP), that can be solved by Branch and Bound method. However, solving an ILP problem is NP-hard, and thus can take too long on an embedded platform. Thus, we devised an equivalent formulation using Maximum-Flow Minimum-Cost (MFMC) problem, that can be solved with polynomial algorithms such as Edmonds-Karp algorithm. Furthermore, we proposed two specific solving methods for the distribution problem. The first one is a non-exact Greedy Heuristic (GH) method, with small time overhead. The second solving method is an adaptation of the Edmonds-Karp algorithm to our distribution problem: Adapted Edmonds-Karp (AEK). Contrarily to the GH, it is an exact method, but produces a higher time overhead.

We performed experiments on: (1) simple missions, (2) scalability (swarm size) and (3) influence of solving difficulty. We compared our proposed methods (GH and AEK) with an ILP solved with a Branch and Bound method (B&B) and an

978-3-9826741-0-0/DATE25/© 2025 EDAA

MFMC solved with Edmonds-Karp method (EK). AEK got the best performance on each experiment, as it reaches optimal solutions in more than 95% of the cases, while improving the solving time by 6.63x compared to EK. For simple to mildly complex missions, GH got very high accelerations (more than 2000x faster than ILP with B&B), while reaching the optimal solutions in over 83% of cases. AEK finds the exact solution for most experiments.

The remainder of this paper is structured as follows. Section II describes the encompassing project and motivates this work. In Section III, we describe our proposal DisPEED. Section IV evaluates the distribution solving methods. Section V summarizes related work int he area. Finally, Section VI concludes this paper and identifies some future work perspectives.

II. Background and Motivation

A. Background

This study is part of a project that aims to investigate the deployment of IDS on a drone swarm by exploiting their hardware heterogeneity. The project is structured as shown in Fig. 1. The first step consists of **(1) characterizing several IDS models [8], [9]** based on different machine learning (ML) algorithms (e.g. random forests, DNN) on each processing element available on the drones (e.g. CPU, GPU, FPGA) in terms of QoS and resource metrics (see Fig. 1 for metrics description). The characterization is conducted offline and the IDS implementations are then embedded in the drone to be run adaptively. **(2) The distribution module** is implemented in each drone, captures the traffic and the maximum number of packet flows that each drone can handle, and runs a strategy to decide which drone analyzes which part of the packet flows to guarantee QoS metrics (Fig. 1(b)). **(3) IDS map module** then selects the best IDS implementation for each drone (which IDS model on which processing element) to run according to its state, the characteristics of the IDS models and the mission state. Once the IDS implementation is selected, the model is deployed on its associated processing element. **This paper focuses on the "distribution module".**

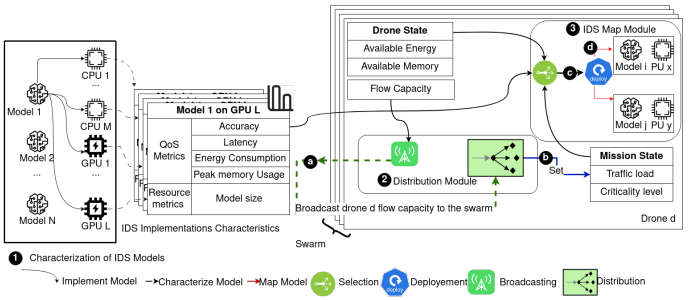

Fig. 1: Overview of the Project

The characterization phase showed that a drone embeds a set of IDS implementations with important disparities in terms of accuracy (guaranteed security level), energy consumption, inference time, and memory footprint. Table I shows part of the characterization results. This disparity can be leveraged to accommodate different drone and mission QoS requirements.

Platform	Model	Throughput (flow/s)	Energy (µJ)	Memory Peak (MB)	Accuracy (%)
Raspberry Pi 4	NoFS-RF	23675.64	74.875	0.033	81.59
	ES-RF	35714.29	133	0.011	77.28
	ES-DNN1	76263.11	125.375	0.066	70.34
Nvidia Xavier GPU	NoFS-RF	201005.03	2462.5	1.500	81.59
Nvidia Xavier CPU	NoFS-RF	27932.96	775	0.090	81.59
	ES-RF	784313.73	542.125	0.019	77.28
	ES-DNN1	233918.13	537.5	0.015	70.34
	AE-DNN1	42553.19	1097.875	0.018	74.9
	AE-DNN2	4631.50	3037.5	0.021	79.03
Pynq-Z2 FPGA	ES-DNN1	176678.45	10.7875	0.455	65.73

TABLE I: Characteristics of the IDS implementations embedded on drones - the Model Indicates the ML model on which relies the IDS (RF for random forest and DNN for Dense Neural Network), NoFS, AE and ES are different methods to select features)

B. Motivation

As one can observe in Table I extracted from our past work in [9], according to the platforms available on the drone (CPU, GPU, FPGA), the maximum IDS throughput varies substantially. For example, a drone equipped with a GPU can analyze around 10x more flows than a Raspberry's CPU. In case of a traffic load that exceeds 80,000 flow/s (with a WiFi connection for instance), the drone equipped with a Raspberry Pi-like CPU cannot analyze the whole traffic on time, leaving it vulnerable to serious security threats. We argue that a relevant solution to this challenge is to offload (or share) part of its traffic to (with) more powerful neighbors equipped with more processing resources, such as GPUs or FPGAs (such as NVIDIA Xavier or Pynq-Z2 in the Table that can handle more than 150,000 flows each). The same applies in case of reduced energy budget or a specific accuracy target. One can take advantage of drones heterogeneity to fit within the mission QoS requirement through a relevant flow distribution.

In a previous work, we proposed IDS-DEEP [10], a strategy to deploy the best IDS implementation depending on the context, as a standalone system on a single drone. To do so, we proposed two different phases: (1) an offline phase that selects the IDS implementations to embed depending on static constraints (available embedded platforms, storage space dedicated to IDS implementations, ...), and (2) an online phase that selects the best IDS implementation to deploy depending on dynamic constraints (security level required, battery-level, ...). For a drone swarm configuration, one can directly apply IDS-DEEP locally on each drone to select the best IDS implementation to execute. However, in some circumstances, some drones cannot process all packet flows, because no embedded IDS implementation matches the dynamic constraints.
To overcome this issue and generalize IDS-DEEP, we propose DisPEED, a strategy able to distribute the packet flows processing among the drones of the swarm. To the best of our knowledge, this is the first paper to propose a strategy to distribute the traffic in a swarm of drones to comply with QoS metrics.

III. DisPEED

A. Overview

The objective of DisPEED is to ensure that the whole traffic flow within the swarm is analyzed while respecting (1) each of

the drones capabilities in terms of available energy and memory resources ; (2) mission constraints in terms of minimum level of security to guarantee (according to the criticality level). **The distribution module** is structured around two steps :

- **Drone Capacity self-assessment:** each of the drones embeds a set of IDS implementations with heterogeneous levels of security, energy consumption and available memory. According to the drone state (available energy and memory), and the mission state (criticality level), the IDS implementations that do not satisfy the constraints are excluded. Then, the flow capacity c_i of drone i is determined as the throughput of the fastest remaining implementation.

$$c_i = 1/min_{j \in J}\{L_j\} \tag{1}$$

  L_j is the time for implementation j to process one traffic flow, and J is the set of implementations that satisfy the drone and mission constraints. The drone broadcasts its flow capacity to the other drones of the swarm (see Fig. 1(a)).
- **Flow distribution:** once all the drones have shared their flow capacities, each one of them has a holistic view of other's capacities, their actual loads, and the existing communication routes. Then, the objective is to distribute the traffic flow in a way to process the whole traffic while minimizing flow offloaded between drones, in order to reduce the attack surface. To do so, we propose two distribution solving methods (described in the next section) that hit different tradeoffs in terms of performance and time overhead.

As explained in the Motivation, after the execution of **the distribution module**, each drone will select the best embedded IDS implementation to process the packet flows, by executing the **the IDS map module** (the online phase from IDS-DEEP [10]). Indeed, the packet flows to process by a particular drone will give the latency constraint for the online phase.

All drones independently and consistently calculate the distribution of packet flows: (1) to reduce the attack surface; (2) to make the distribution deterministic, thus guaranteeing that all nodes converge to the same solution. Additionally, low time overhead for distribution solving methods is mandatory to have as little impact as possible on the swarm drones mission.

B. Packet Flow Distribution Formulation

1) Input data: We consider a swarm $D = \{1, 2, ..., n\}$ of n drones. Each drone i is related to the following data:

- w_i is the workload addressed to drone i for analysis.
- c_i is the analysis capacity of the IDS embedded on drone i. Notice that if $w_i > c_i$, drone i cannot process completely its workload and thus must offload a part of it to another drone.
- $N_i \subset D = \{j | i$ can send data to $j\}$ is the set of neighbor drones of drone i to which i can offload its workload.

Of course, the problem is unfeasible if

$$\sum_{i \in D} w_i > \sum_{i \in D} c_i \tag{2}$$

2) Integer Linear Programming (ILP) formulation: The goal is to minimize the communication cost of the balancing of the workload over the drones. The decision variables are as follows:

- f_{ij} is the flow transmitted from drone $i \in D$ to drone $j \in N_i$
- p_i is flow processed by i
- r_i is the flow received by i from other drones in the swarm.
- e_i is the flow emitted by i, implying a communication cost.

The objective function for minimizing communication cost with a fixed cost for all neighbor-to-neighbor communication (eq. (3)) is defined as:

$$\min \sum_{i \in D} e_i \tag{3}$$

It can be modified to take into account latency or IDS processing costs as desired. The constraints are the following:

$$\forall j \in D \qquad r_j = \sum_{\{i | j \in N_i\}} f_{ij} \tag{4}$$

$$\forall i \in D \qquad e_i = \sum_{j \in N_i} f_{ij} \tag{5}$$

$$\forall i \in D \qquad e_i + p_i = r_i + w_i \tag{6}$$

$$\forall i \in D \qquad e_i \leq w_i \tag{7}$$

$$\forall i \in D \qquad p_i \leq c_i \tag{8}$$

$$\forall i \in D, j \in N_i \qquad f_{ij} \geq 0 \tag{9}$$

The decision variables are p_i and f_{ij}, denoting respectively locally processed and transferred to neighbors flows.

Eq. (4) and (5) set the values for received and emitted flows with the neighborhood of drone i. Eq. (6) ensures that flows addressed to drone i (either local workload or received flow from its neighbors) is processed, either locally or transferred to the neighborhood. Eq. (7) states that a drone cannot transfer more than its own workload, i.e. a workload is to be processed locally or by the direct neighbors of a drone: a flow cannot be sent from i to j and then from j to k. Eq. (8) verifies that drone i processing capacity is not exceeded. The last constraint (9) leads all decision variables to be positive.

The ILP formulation of the distribution can be solved by classic methods such as branch and bound or cutting plane [11].

3) Maximum-Flow Minimum-Cost (MFMC) formulation: We argue that ILP formulation of the distribution problem can be expressed as a Maximum-Flow Minimum-Cost problem [12], consisting in finding the maximal flow that can be sent from a source node to a sink node in a graph labeled with capacities on its edges. The flow on each edge cannot exceed its capacity and incoming and outgoing flows must be balanced on all intermediate nodes. If more than one flow is optimal (with the max flow value), one with the minimum cost is selected. Cost is computed according to a second label representing the prize per unit of flow associated to each edge of the graph. Ford-Fulkerson [13] or its variant Edmonds-Karp [12] solve it in polynomial time, e.g. $O(|V|.|E|^2)$ for a flow graph $G = (V, E)$ with Edmonds-Karp. As with MILP solver, optimal solution is guaranteed. Furthermore, MFMC execution time is polynomial while MILP approach can lead to exponential execution times.

G is built as follows. For edges, the following notation is used: $< i, j, \alpha, \beta >$ stands for an edge with tail node i, head node j, capacity α and cost β.

- node $v_0 \in V$ is the source
- node $v_{2n+1} \in V$ is the sink

- $\forall i \in D$, nodes $v_i, v_{n+i} \in V$ are respectively workload hosting and processing nodes associated to drone i
- $\forall i \in D, < 0, i, w_i, 0 > \in E$ links the source to the hosting node of drone i
- $\forall i \in D, < i, 2n+1, c_i, 0 > \in E$ links the processing node of drone i to the sink
- $\forall i \in D, < i, n+i, \infty, 0 > \in E$ links hosting and the processing nodes of drone i
- $\forall i \in D, j \in N_i, < i, n+j, \infty, 1 > \in E$ links the hosting node of drone i and the processing nodes in its neighborhood.

Fig. 2 shows an example for a set of 4 drones. If eq. (2) holds, the max flow corresponds to the (minimal cut) stage of edges connected to the sink (node v_9 on the figure), else it is limited by the cut of edges from source v_0 to hosting nodes (v_1 to v_4 in the example). The cut can also correspond to the fact that no way exists for transferring from an overloaded node to an unsaturated one. Thus, maximizing the flow will lead to processing a maximum amount of the flows. The processed load can correspond to the whole traffic of the swarm in case the overall capacity is higher than the overall load, and there exist network routes that allow to proceed to the offload. As the cost of edges between hosting nodes and processing nodes in neighborhood is 1 whereas the cost of edges between hosting and processing node on the same node is 0, the Edmonds-Karp will naturally favor local processing, thus minimizing communications. The cost of the balancing depends on the fact that edges of value 1 are used or not on the intermediate stage: these edges correspond to transfers among drones, and thus to communication costs.

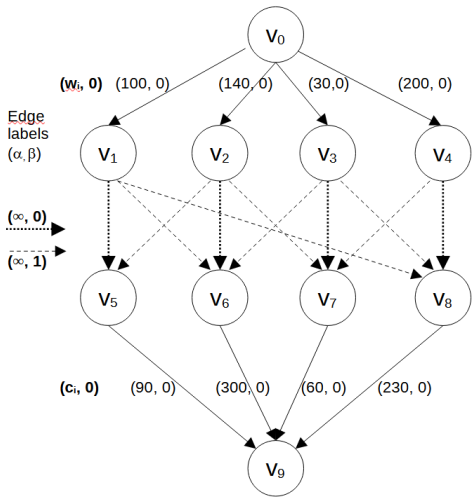

Fig. 2: Flow graph for $n = 4$ drones, with $W = [100, 140, 30, 200]$, $C = [90, 300, 60, 230]$, $N_1 = \{2, 4\}, N_2 = \{1, 3\}, N_3 = \{2, 4\}, N_4 = \{3\}$

The Edmonds-Karp algorithm computes the maximum flow of minimal cost by determining iteratively *augmenting paths* of minimal length in the *residual network* G^Δ of the graph G. Such a shortest path from the source to the sink node is computed using, for example, Dijkstra algorithm. G^Δ is updated according to this path (for details, refer to [12]). The algorithm ends when source and sink are disconnected in G^Δ.

C. Flow Distribution solving methods

To solve the distribution problem, we propose two solving methods based on the previously introduced formulations.

1) Greedy heuristic: The first proposed solving method is a nonexact greedy heuristic, which is described in algorithm 1. This method is divided into two steps: (1) local allocation and (2) iterative global allocation. Local allocation step assigns the maximum of flows processing in local, according to the drone capacity c_i. The iterative global allocation step assigns the remaining flows to process iteratively to the swarm. Particularly, this method uses two internal variables for each drone:

- ac_i: the available capacity of drone i. This variable allows to track the number of flows the drone could still process, during the algorithm execution.
- rf_i: the number of remaining flows to be processed. This variable allows tracking the number of drone flows remaining to be processed, during the algorithm execution.

Data: $\{c_i\}_{i \in D}$: the drone capacities, $\{w_i\}_{i \in D}$: the drone workloads and $\{N_i\}_{i \in D}$: the drones neighbors
Result: Packet flows distribution
Step 1: Local allocation
for $i \in D$ **do**
  $min(c_i, w_i)$ are assigned locally
  $ac_i = max(c_i - w_i, w_i - c_i)$
  $rf_i = max(0, c_i - w_i)$
end
Step 2: Iterative global allocation
for $i \in D$ **do**
  for $j \neq i$ **do**
    if $(j \in N_i)\ and\ (ac_j > 0)$ **then**
      $min(ac_j, rf_j)$ flows from drone i are assigned to drone j
      if $rf_i > ac_j$ **then**
        $rf_i = rf_i - ac_j$
        $ac_j = 0$
      end
      else
        $rf_i = 0$
        $ac_j = ac_j - rp_i$
      end
    end
  end
end

Algorithm 1: Greedy heuristic strategy

This is not an exact method, i.e. it cannot always find a solution to a problem that has one that can be found by exact methods. However, it is faster than solving the exact formulations ILP (using B&B) and MFMC (using Edmonds-Karp). Furthermore, its spatial and temporal complexity are $O(|D|^2)$.

2) Adapted Edmonds-Karp (AEK): The second proposed solving method is an adaptation of Edmonds-Karp algorithm, for solving the distribution problem based on the MFMC formulation but using more a priori knowledge about the problem. In fact, in the Edmond Karp algorithm, G^Δ is initialized to G. Since local processing does not require communication, we know that the first shortest paths (of cost 0) correspond to $v_o \rightarrow v_i \rightarrow v_{n+i} \rightarrow v_{2n+1}$, i.e. source $\rightarrow$ hosting node $\rightarrow$ processing node $\rightarrow$ sink for each drone $i \in D$. We compute directly the flow on those paths and update G^Δ accordingly, saving n executions of Dijkstra algorithm. It can be noted that the initialization step of the AEK algorithm based on a priori

knowledge about the distribution problem also corresponds to the local allocation step of the greedy heuristic method.

IV. Evaluation

A. Experimental methodology

All experiments were run on a Raspberry Pi 4B (RPi4B) as it is frequently used to prototype drones. We evaluated different swarm configurations for each experiment described below. For each configuration, the solving methods have been executed on 100 instances. Note that all the methods have been run on the same instances to mitigate any configuration-related bias.

For all the experiments, the evaluated metrics are **the solving time** and **success rate**. Particularly, the success rate corresponds to the ratio between the number of instances for which the tested method finds the optimal solution (all the packet flows are distributed) and the overall number of instances. By optimal solution, we mean a solution that succeeds to distribute all the packet flows to analyze.

Our methods (GH and AEK) were evaluated against EK and ILP with Branch and Bound (B&B). The EK and AEK algorithms were manually implemented in C language, the GH solution was manually implemented in C++ and B&B solver was based on the C++ library: https://github.com/gymitoso/ILP/tree/master. This tool was chosen for the sake of implementation simplicity on an embedded platform (RPI4B).

B. Experiments Description

1) Simple use-cases experiment (proof of concept): The first experiment consists in evaluating all the solving methods on simple use-cases. These missions are described in figure 3: a swarm composed of 10 drones and a sub-swarm composed of the first 6 drones. It was inspired by a decentralized swarm topology illustrated in [14]. Drone capacities were determined by real-world results from our extensive characterization step [9]. Indeed, we considered three types of drones: (1) a single drone equipped with CPU, GPU and FPGA (drone 4), (2) three drones equipped with CPU and FPGA (drones 3, 7 and 9), and (3) seven drones equipped solely with a CPU (drones 2, 5, 6, 8 and 10). The capacity is determined by the throughput achievable by their fastest IDS implementation considering the table I. Finally, we assigned for each drone 90000 flows to be processed, except for the drone equipped with CPU, GPU and FPGA that needs to process 200000 flows. In these configurations, we can observe that drones equipped with a CPU cannot process all their flows locally and need to distribute their remaining flows.

For each configuration, we used the same 100 test case files for compliance with the methodology.

2) Scalability experiment (swarm size): The second experiment consists in evaluating the scalability of EK, GH and AEK (for reasons explained in Section IV-C1) on two configurations: a swarm composed of 10 drones and a swarm of 100 drones. For each configuration, we generated 100 test cases for methodology compliance. To be coherent with the previous experiment, we keep the same platform proportion and capacities: 10% of drones are equipped with CPU, GPU and FPGA

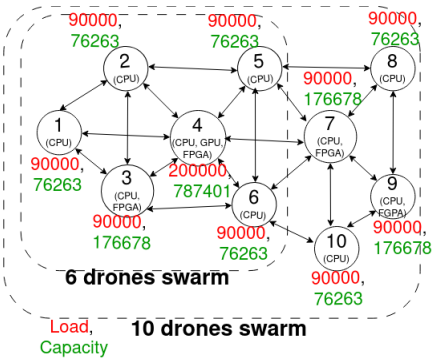

Fig. 3: Simple missions for swarm packet flows distribution

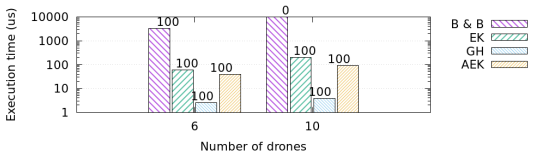

Fig. 4: Simple Missions Experiment Results The labels above the bar charts represent the success rate of the methods

($w_i = 200000, c_i = 787401$), 30% of drones are equipped with CPU and FPGA ($w_i = 90000, c_i = 176678$) and 70% of drones equipped with only CPU ($w_i = 90000, c_i = 76263$). Furthermore, the probability of a connection between two drones is fixed to 50%. If a drone is not connected to any drone, we randomly select a drone with neighbors to connect with it. As for the previous experiment, for each configuration, we generated 100 test cases for each configuration. The distribution of GPU, FPGA, and CPU over nodes depends on the number of node connections. The more the node has connections, the higher its computing resources (it is assigned a GPU and/or FPGA). All nodes are assigned a CPU.

3) Influence of solving difficulty (load/capacity ratio): The third experiment consists in evaluating EK, GH and AEK (as for experiment 2) for different solving difficulties on a 10 drones swarm. The solving difficulty is defined as the ratio between the sum of all the flows to be processed and the sum of the capacities for each drone. The evaluated configurations for this experiment are the following: 0.5, 0.6, 0.7, 0.75, 0.8, 0.9 and 1.0 (worst case). Thus, each drone has the same traffic load that corresponds to the solving difficulty (sd) multiplied by the mean of the drone capacities of the swarm ($c = sd \times (\sum_{i \in D} c_i)/|D|$). However, to be coherent with the previous experiments, we kept the same platform ratio: 10% of drones are equipped with CPU, GPU an FPGA ($w_i = c, c_i = 787401$), 30% of drones are equipped with CPU and FPGA ($w_i = c, c_i = 176678$) and 70% of drones equipped with only CPU ($w_i = c, c_i = 76263$). Connection generation follows the same process as in Experiment 2.

C. Results and Discussion

1) Simple missions experiment (proof of concept): Fig. 4 shows the solving time and success rate for the 6 drones and 10 drones test-scenarios, for the two reference solving methods

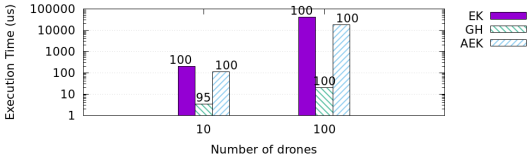

Fig. 5: Scalability Experiment Results - The labels above the bar charts represent the success rate of the methods

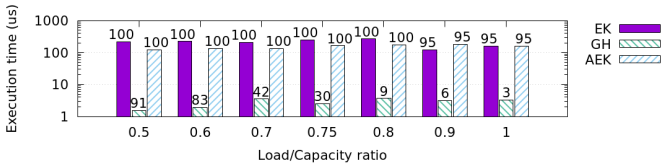

Fig. 6: Solving Difficulty Experiment Results - The labels above the bar charts represent the success rate of the methods

(B&B and EK) and our two proposals (GH and AEK). We observe that for the 6 drones scenario, the EK solving method is around 54x faster than B&B. As compared to EK, our proposals GH and AEK are, respectively, 24.51x and 1.5x faster. In terms of success rate, the four solving methods succeeded to reach the optimal solution for all test instances. For the 10 drones mission, we observe that the B&B method surpasses EK by 48x for the solving time without reaching a solution (0% success rate), probably due to some limitations of the used ILP solver. As compared to EK, AEK and GH are respectively 2.21x and 52.91x faster while reaching the optimal solution. In light of this experiment, we conclude that the two proposals, especially GH, are very competing compared to traditional solving methods for simple missions where the solving difficulty is around 0.54, which is considered low.

It can be noted that we determined experimentally that the threshold from which B&B cannot converge is 7 drones, probably due to the chosen implementation. For this reason, B&B is no longer considered in the remaining experiments.

2) Scalability experiment (swarm size): Fig. 5 shows the solving time and success rates for 10 and 100 drones test-scenarios, to demonstrate the scalability of the methods. For the 10 drones case, we observe that GH is respectively 33.09x and around 60x faster than AEK and EK. Regarding the success rate, we observe that GH fails to find the optimal solution in 5% of the cases, while both EK and AEK succeeded in 100% of the cases. For the 100 drones configuration, the accelerations performed by GH as compared to EK and AEK are substantial (respectively 2060x and 881x faster), while succeeding in 100% of the cases. This experiment shows that for low solving difficulty, GH is the best option when the problem size increases.

3) Influence of solving difficulty (load/capacity ratio): Fig. 6 shows the solving time and success rate for different solving difficulties on a 10 drones swarm, for the reference solving method (EK) and our two proposals (GH and AEK). We can observe that for solving difficulties between 0.5 and 0.8, GH and AEK succeeded to solve all the test cases, while the percentage of success of GH decreases when the solving difficulty increases. Furthermore, we can see that all the execution times of the solving methods increase with solving difficulties. For the range of 0.9 and 1.0, the success rate of exact methods (EK and AEK) is about 95%, because some distribution problems are not solvable at all, while success rate of GH is between 6% and 3%. Regarding execution times, it is lower as compared to solving difficulty of 0.8, probably because of the 95% of success rate. This experiment shows the low effectiveness of GH to solve high-difficulty problems, but also illustrates a link between solving difficulty and execution time for medium solving difficulty between 0.5 and 0.8.

V. Related Work

Existing studies considered IDS systems for drones swarms [15]–[18]. [16] proposed a centralized IDS applying multifractal analysis on packet flow and used a RNN model for detection. In [15], the authors proposed a model to estimate the real malicious traffic from packet flow using a state observer. [17] proposes a CNN based-IDS that analyses encrypted Wi-Fi traffic from 3 different drones models. Finally, [18] developed an extensive distributed IDS for drones swarm. Those studies are orthogonal to ours, as the models designed could be integrated to our platforms.

Other studies have considered the case of Collaborative IDS (CIDS) or Distributed IDS (DIDS), relying on two approaches: (1) propagating data about locally detected threats, whether in (a) a centralized way, where each drone propagates the detected threats to a central node which re-analyzes the suspected communication and propagates the information [5], or in (b) a decentralized way, where a node shares the detected threats with the others [6], [19]. Another decentralized approach consists in collaboratively establishing the reputation of a traffic source [7]; (2) Federated Learning-based approaches, where nodes collectively train a holistic IDS model based on the collected data on each node [20], [21].To the best of our knowledge, no previous work addressed the issue of distributing packet flows to analyze on a drone swarm.

VI. Conclusion and Future Work

This paper proposes an approach to distribute the IDS analysis of the packet flows in drone swarm to be able to process all the packets in a timely manner and by considering drone properties in terms of computing capacity, battery level and heterogeneity. To do so, we designed two problem formulations: (1) an ILP formulation and (2) a Max-Flow Min-Cost formulation. Furthermore, we presented two specific solving methods that hit different trade-off between performance and solving time: (1) a Greedy Heuristic (GH), a non-exact solving method, with low time overhead and (2) an Adapted Edmonds-Karp (AEK) algorithm that finds an exact solution at the expense of a higher time overhead. Our experimental results show that both methods make it possible to successfully cover different scenarios in terms of swarm size and solving difficulty problem. For future work, we plan to implement a hybrid approach between GH and AEK, to leverage on the advantages of both solving methods. Also, the solution will be integrated in the overall project with the mapping part for extensive evaluation.

REFERENCES

[1] M. Campion, P. Ranganathan, and S. Faruque, "Uav swarm communication and control architectures: a review," *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, 2019.

[2] UNIDIR, "Swarm robotics," 2020.

[3] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace sciences*, vol. 91, pp. 99–131, 2017.

[4] Q. Cui, P. Liu, J. Wang, and J. Yu, "Brief analysis of drone swarms communication," in *2017 IEEE International Conference on Unmanned Systems (ICUS)*, pp. 463–466, IEEE, 2017.

[5] J. Arshad, M. A. Azad, R. Amad, K. Salah, M. Alazab, and R. Iqbal, "A review of performance, energy and privacy of intrusion detection systems for iot," *Electronics*, vol. 9, no. 4, p. 629, 2020.

[6] K. Zaidi, M. B. Milojevic, V. Rakocevic, A. Nallanathan, and M. Rajarajan, "Host-based intrusion detection for vanets: A statistical approach to rogue node detection," *IEEE transactions on vehicular technology*, vol. 65, no. 8, pp. 6703–6714, 2015.

[7] K. M. A. Alheeti, A. Gruebler, and K. McDonald-Maier, "Using discriminant analysis to detect intrusions in external communication for self-driving vehicles," *Digital Communications and Networks*, vol. 3, no. 3, pp. 180–187, 2017.

[8] C. Slimani, L. Morge-Rollet, L. Lemarchand, F. L. Roy, D. Espes, and J. Boukhobza, "Characterizing intrusion detection systems on heterogeneous embedded platforms," in *26th Euromicro Conference Series on Digital System Design (DSD*, 2023.

[9] C. Slimani, L. Morge-Rollet, L. Lemarchand, D. Espes, F. L. Roy, and J. Boukhobza, "A study on characterizing energy, latency and security for intrusion detection systems on heterogeneous embedded platforms," *Future Gener. Comput. Syst.*, vol. 162, p. 107473, 2024.

[10] L. Morge-Rollet, C. Slimani, L. Lemarchand, F. Le Roy, D. Espes, and J. Boukhobza, "Ids-deep: a strategy for selecting the best ids for drones with heterogeneous embedded platforms," in *2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 138–147, IEEE, 2024.

[11] L. A. Wolsey, *Integer programming*. John Wiley & Sons, 2020.

[12] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[13] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[14] U.S. Government Accountability Office, "Drone swarm technologies," 2023.

[15] J.-P. Condomines, R. Zhang, and N. Larrieu, "Network intrusion detection system for uav ad-hoc communication: From methodology design to real test validation," *Ad Hoc Networks*, vol. 90, 2019.

[16] R. Zhang, J.-P. Condomines, and E. Lochin, "A multifractal analysis and machine learning based intrusion detection system with an application in a uas/radar system," *Drones*, 2022.

[17] Q. Abu Al-haija and A. Al Badawi, "High-performance intrusion detection system for networked uavs via deep learning," *Neural Computing and Applications*, vol. 34, pp. 10885 – 10900, 2022.

[18] F. Tlili, S. Ayed, and L. Chaari Fourati, "Exhaustive distributed intrusion detection system for uavs attacks detection and security enforcement (e-dids)," *Computers & Security*, 2024.

[19] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 656–666, IEEE, 2017.

[20] J. Shu, L. Zhou, W. Zhang, X. Du, and M. Guizani, "Collaborative intrusion detection for vanets: A deep learning-based distributed sdn approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4519–4530, 2020.

[21] J. Cui, H. Sun, H. Zhong, J. Zhang, L. Wei, I. Bolodurina, and D. He, "Collaborative intrusion detection system for sdvn: A fairness federated deep learning approach," *IEEE Transactions on Parallel and Distributed Systems*, 2023.