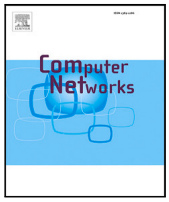
Computer Networks 250 (2024) 110599

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



P4 FANET In-band Telemetry (FINT) for AI-assisted wireless link failure forecasting and recovery

Layal Ismail [a,*], Domenico Uomo [a], Andrea Sgambelluri [a], Faris Alhamed [a], Francesco Paolucci [b]

[a] Scuola Superiore Sant'Anna, Via G. Moruzzi 1, Pisa, 56124, Italy
[b] CNIT, Via G. Moruzzi 1, Pisa, 56124, Italy

ARTICLE INFO

Keywords:
SD-FANET
AI
P4 Language

ABSTRACT

This paper introduces a novel framework for enhancing quality of service predictability in Flying ad hoc Networks (FANET) by leveraging P4 data-plane programmability. The proposed solution, P4 FANET In-Band Telemetry (FINT), is specifically designed to tailor the limited resources in wireless networks and is extended to collect not only the standard INT metadata but also novel essential Unmanned Aerial Vehicles (UAV) real-time metrics, including Received Signal Strength Indicator (RSSI), geolocation information and CPU load. These parameters are then fed into an artificial intelligence (AI) system, enabling proactive prediction of FANET link failures. By integrating P4 FINT and AI, our framework aims to improve the availability and overall performance of UAV-based networks through advanced link failure forecasting.

1. Introduction

1.1. Motivation and paper contribution

In an era characterized by rapid automation advancements, offloading various jobs to Artificial Intelligence (AI) is becoming the standard practice, and many AI experts believe that in the next few decades AI will match or even outperform humans in many areas [1].

Among the different networking scenarios, Flying ad hoc Networks (FANET) represent the most challenging framework, since node mobility and flying state and conditions introduce relevant physical topology dynamicity issues, involving variable link states, link/node disconnection events and flying node platform limitations (e.g., battery). With technological advances in networking technologies in terms of programmability, higher transmission speeds, and lower latencies, it is now possible to collect comprehensive telemetry data on the state of devices in a given network [2].

For such networks, AI-assisted recovery procedures require standard in-band network telemetry (INT) [3] augmented with key mobility and flying performance metadata (for example, the actual quality of the transmission of a wireless link, or the current position of the flying node).

In this paper, we propose a new framework for improving the robustness of connectivity in a FANET based on the collection of an extended set of P4 In-Band Telemetry [3] data. By feeding these telemetry data to an AI system as input features, the AI will be able to make real-time decisions about implementing changes to the state of the network devices. These changes would come as a response to resolve unwanted situations such as link congestion, packet loss, and high latencies.

For this reason, with our work described in this paper, we aim to take advantage of data plane programmability with the P4 language [4] for the collection of telemetry data in FANET. We can summarize our contributions in the following points:

1. Introducing data plane programmability using P4-enabled nodes in a FANET for enhanced networking operation. In our implementation we achieved this by introducing a P4 programmable software switch that handles the forwarding of the packets and the collection of extended telemetry data based on flow entries that keep being updated through an SDN controller as the topology of the network changes.
2. Implementing an extended FANET In-Band Network Telemetry (FINT) to collect state information on FANET nodes, their actual properties and additional metadata. The extended telemetry includes non-standard P4 metadata such as the load of the CPU of the UAV, its coordinates in the 3-dimensional space and the RSSI of the radio links. This extended telemetry is used for the prediction of link failures in the network.
3. Implementing an AI algorithm that uses FANET In-Band extended telemetry data to predict disconnections between the nodes and prevent link failure using link recovery in advance, minimizing traffic disruption.

* Corresponding author.
E-mail address: layal.ismail@santannapisa.it (L. Ismail).

https://doi.org/10.1016/j.comnet.2024.110599
Received 6 February 2024; Received in revised form 26 April 2024; Accepted 9 June 2024
Available online 12 June 2024

In Section 2, we briefly explain the scientific background and technologies used in our work, while the rest of this paper comprises the following topics:

1. The related works in the literature that includes FANET and data plane programmability (Section 3);
2. The architectural framework of P4 FANET, including the network and the node design (Section 4).
3. The proposal of extended in-band telemetry for FANET, i.e. FINT (Section 5);
4. The proposal of AI-assisted FANET link failure forecasting and recovery exploiting the FINT reports (Section 6);
5. The experimental testbed and extensive results carried out to evaluate and validate the FINT-based FANET link failure forecast (Section 7);
6. The conclusions of the work (Section 8).

2. Background

2.1. Mobile ad hoc networks

Wireless networks can generally be classified into two broad categories: 1. Infrastructure-based wireless in which nodes communicate through an Access Point (AP). 2. Infrastructure-less (Ad Hoc) wireless networks. Mobile Ad Hoc Networks (MANETs) are a type of wireless network that connect two or more wireless mobile nodes without relying on a central infrastructure to establish the connections between different nodes in the network. In this type of wireless network, each wireless node performs the functions of both a host (generating its own traffic) and a router (forwarding traffic generated by other nodes). The topology in MANETs is constantly changing as the wireless nodes in such a network are allowed to move freely in any direction, and the nodes can dynamically join or leave the network. Thus, the nodes frequently break and establish wireless connections to other nodes on an on-demand basis [5].

2.2. Unmanned aerial vehicles

In recent years the world has seen rapid development in the field of Unmanned Aerial Vehicles (UAVs), commonly known as Drones. These Drones are flying aircraft that do not carry a human operator and can operate either autonomously or via remote control through a Base Station (BS). UAVs can be built in different sizes and shapes and be adapted to carrying different types of payloads. Also, they can be equipped with a variety of sensors for multiple purposes such as Global Positioning System (GPS) and proximity sensors which can be used for navigation, cameras for navigation and data collection, and many other types of sensors adapted to the application. This flexibility in the design makes drones suitable for many applications, including parcel delivery, exploration, search and rescue, disaster recovery, and many others. Most UAVs are designed for high maneuverability, so they can be deployed quickly and easily, and rely on batteries as their power source [6,7].

2.3. Flying ad hoc networks

Many applications employ a group of UAVs to increase coverage area or carrying capacity, thus requiring communication and coordination between UAVs, leading the UAVs to form what is known as Flying Ad Hoc Networks (FANETs). A FANET can be described as a wireless ad hoc network that connects two or more UAVs. Essentially, a FANET is a particular case of MANET, where the mobile nodes are UAVs [8]. A FANET provides many advantages over the use of a single UAV system. Some of these advantages can be summarized here:

1. Reduced costs when using multiple small UAVs than a single large one [9].
2. Increased coverage range [10].
3. Redundancy: if one of the UAVs fails its role can be covered by other nodes in the network. [11].
4. Task speed up: having a group of UAVs working towards achieving a goal would take less time than using a single UAV. [12]

However, the flying nature of a FANET introduces various challenges in coordinating and sustaining the network. These challenges can be summarized in the following points:

1. Routing traffic between nodes: the topology is frequently changing because nodes have a high degree of movement freedom in a 3-dimensional space at high speeds that can be more than 400 km/h [11].
2. Power consumption: batteries are the main power source in small UAVs, their low energy density permits a small UAV to operate for about 20 to 30 min. Moreover, the relatively slow charging speeds sometimes call for a manual replacement of the batteries [13–15].
3. The need for accurate location information at high speed of movement [16,17].
4. Latency and bandwidth issues [18,19].
5. Managing and organizing the network [20].

2.4. Programmable data plane with P4

Programmable Data Planes (PDPs) allow a network administrator to tune the processing logic of a network device to their own specific needs even after the devices have been deployed in the network. This feature separates the packet processing functions from the underlying hardware and makes it possible to implement new customized and non-standard networking protocols, designed to achieve the goals of the network administrators. This programmability is supported by the introduction of programmable data plane languages. Among those, the Programming Protocol-independent Packet Processors Language (P4Lang) [4,21] is a domain-specific language designed to describe the processing logic of Software Defined Networking (SDN)- enabled devices and to specify how data plane devices process packets.

2.5. In-band network telemetry

In-band Network Telemetry (INT) is a framework designed to allow the collection and reporting of the state of the network by the data plane, without requiring the intervention of the control plane [3]. In INT, it is possible to embed telemetry data (also known as metadata) into the packet by each network node along the packet path. This process is generally enforced in the following steps:

1. The first node in the telemetry path (known as the ingress node) inserts two or more telemetry headers. The first header conveys information about the telemetry protocol and format, such as the length of the telemetry data. In addition, it can carry instructions for the downstream nodes. Finally, the ingress node adds one or more metadata headers, according to the instructions from the control plane.
2. Following nodes in the INT path (known as transit nodes) also include their metadata following control plane instructions and possibly can make changes to the INT header.
3. The last node in the INT path (e.g., the sink node) inserts its metadata like a transit node, it extracts all the added INT headers from the packet and then sends them to a telemetry collector in a new packet (called INT Report). This way, the original packet is restored to be delivered to the destination transparently.

3. Related work

Several works have been proposed in the literature to identify and address the challenges faced in deploying FANETs (Table 1). For

Table 1
Overview and features of key related works.

Work	Description	Features
LEPR [22]	Link stability metrics for preemptive estimation of link failure.	The estimation is local to the node, no overall view of the network, no INT.
DPTR [23]	Tree-based grouping takes into account multiple ground stations.	No Link estimation, priority-based grouping, no INT.
SOCS [24]	Clustering-based, self-organizing, nodes exchange information with neighbours.	Requires electing cluster heads, no dynamic forming of links, no INT
Fekher et al. [25]	Uses fuzzy logic for localization and energy-efficient routing, uses node clustering.	Operates in no GPS environment, uses signal to estimate location, no INT.
Scano et al. [26]	Uses INT in UE in 5G to predict latency and link failures.	Studies the concepts of using INT for link state estimation in a cellular environment instead of FANET.

example, the work by Frew and Brown [27] identifies operational and networking requirements to be met by the nodes in a FANET for specific applications. As an example, one scenario reported in [27] is tracking the boundaries of a toxic plume by drones; for many practical reasons, it may not be possible or desirable to equip every drone with the same set of sensors. In this case, drones with a similar set of sensors need to identify and locate each other for efficient cooperation and routing of communications. In [22] the authors proposed Link-stability Estimation-based Preemptive Routing (LEPR) with the goal of adapting traffic routing to the dynamically changing network topology, enhancing packet delivery ratio and minimizing delay and overhead. In [23] the authors proposed a tree-based routing scheme partitioning the FANET into the aerial and the ground segments. Their thorough analysis showed comparable results between ground and flying ad hoc networks in terms of packet delivery ratio and end-to-end delay and showed good connectivity in networks composed of up to 30 ground nodes and up to 10 UAVs. On the other hand, the authors in [28] propose Delay and Link Stability Aware (DLSA) routing, which is based on DPTR [23] and LEPR [22] and combines the upsides of both schemes to achieve an improvement in the end to end delay and the packet delivery ratio.

Authors in [24] propose a Self-Organized Clustering Scheme (SOCS) to address routing and management issues in FANET. The SOCS makes use of the Glowworm Swarm Optimization [29] in which every node exchanges information with its neighbours about distance and connectivity to the ground control station, then based on this information cluster headers are elected and clusters of UAVs are formed. When tested against similar clustering algorithms, SOCS offered a substantial improvement in terms of cluster building time, energy consumption, and cluster lifetime. However, the scheme in SOCS requires that nodes adjust their position to stay close to cluster heads which may not always be desirable, and the tests were simulated in MATLAB which may not reflect real-world scenarios. A dynamic UAV positioning method based on Particle Swarm Optimization is proposed in [30], intending to maximize the value of sensor data collected from multiple UAVs. The scenario assumes that numerous types of sensors are deployed in a wide area and that multiple UAVs are used to collect the data from the sensors in real-time. The protocol works by assigning a value to each sensor, the value goes higher in time the longer the sensor data has not been read and is set to the minimum value after a sensor has been read. The authors in [30] used a MATLAB simulation to prove the worthiness of their method. However, a real-world evaluation is still desired to provide reliable test results. Another mechanism for clustering nodes and routing data is proposed in [31]. The proposed mechanism is based on the localization of nodes using weighted centroid localization [25] that uses Fuzzy Logic inference based on RSSI values between different nodes to compute their locations. The calculated locations of the nodes are used to select a cluster head, which is then used to optimize routing, to reduce energy consumption and hence increase network lifetime. Simulations are run in MATLAB which has proven some improvements over similar algorithms. However, we still believe that real-world validation is necessary to take into account unforeseen events. Another work that takes link state into account is presented in [32] in which the authors study and simulate multiple link state routing protocols, available in the literature. The authors aim to enhance the performance of these protocols by optimizing their parameters such as coverage, directional gain, number of nodes, and many other relevant parameters, as opposed to proposing new protocols. Their results show that it is possible to gain significant improvements in terms of delay and packet delivery ratio by fine tuning the routing parameters. However, this approach may not be very practical as many of the network parameters may not be known beforehand. Concerning data plane programmability, in [26] INT operation has been extended with geo-location tracking of user equipment in edge networking with the aim to steer the application traffic from the wireless end-user equipment to the latency closest edge node, also considering the wireless link in the latency computation budget.

4. P4 FANET

4.1. Research scenario

In this work, the considered scenario focuses on a mesh SDN FANET responsible for the reliable reception, processing and delivery of traffic generated by a number of distributed aerial sensors (e.g., IoT devices, and other flying objects such as rescue helicopters), unable to communicate directly with a ground station using a point-to-point connectivity. Possible causes of this limitation can be the limited radio link budget or the instability of the line-of-sight conditions. In this case, the FANET acts as a wireless network extender within a given geographical domain, allowing multiple aerial traffic sources to be received and forwarded by the FANET towards the closest ground station.

The elements of the FANET are placed and moved to guarantee a suitable network coverage for the selected sources and may move in order to optimize the reach due to geographical and weather conditions. The dynamicity of the network, in terms of topology, depends on two factors. First, the FANET elements (P4-UAV) may move and /or may change (e.g., due to maintenance operation). Second, the traffic sources may move independently concerning the FANET, thus they may disconnect and reconnect to the same or to a different P4-UAV. This means that a given traffic flow may be subject to dynamic QoS performance and requires careful monitoring that depends strictly on the FANET conditions. Moreover, topology changes, occurring at the FANET, may induce flow disruptions potentially affecting the traffic QoS, especially in the case of link disconnection and re-connection, whereas SDN control has to update the topology and the network routing of all the disrupted flows.

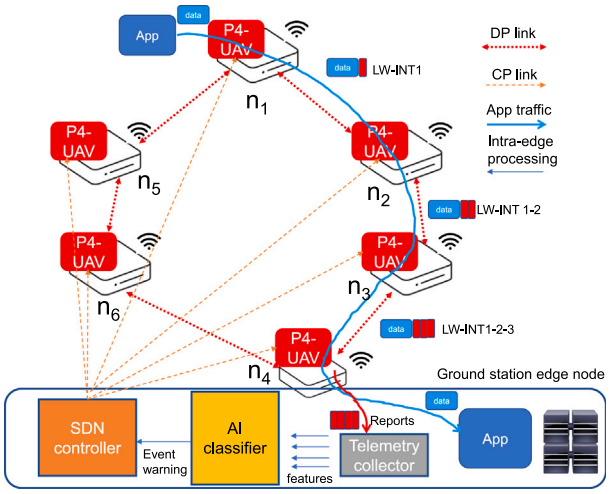


Fig. 1. SD-FANET architecture: aerial network and ground station.

4.2. Network architecture

The general architecture of the P4-FANET, with particular reference to the aerial SDN network solution, is detailed in Fig. 1. The three functional pillars of the solution are clearly distinguishable in the figure: a set of P4-UAVs with wireless interfaces (n_x nodes in the figure), the SDN controller and the AI engine. The latter two are located on the edge node at the ground station and are responsible for the SDN network configuration and detection or forecast of critical events. For simplicity, the SDN controller and AI engine are placed on the same edge node at the ground station. The n_x aerial nodes are interconnected through data plane (DP) wireless links and are instructed to forward application traffic to the ground station through such links. The radio links are created using standard wireless technology in the ISM band, with the creation of transparent L2 links. Two solutions were considered: the Wireless Distribution System (WDS)-based solution and the overlay solution (adopting VXLAN tunnelling for Ethernet over IP encapsulation). In this paper, the VXLAN-based solution has been considered. In particular, when a client connects to an AP, it obtains an IP address and thus the IP reachability with the AP. Then, both sides instantiate a VXLAN. The AP node sets its IP address as local and the client IP address as remote, while the opposite occurs on the client side. Once ready, the VXLAN is assigned to the P4 switch instances. With this method, we enable L2 connectivity among two switches using IP connectivity. The proposed approach can be applied to various radio technological solutions, with the only requirement of having Ethernet L2 access to the packets passing over the interface. Furthermore, an LLDP-based solution was also studied and implemented to update the state of the links, discovering and deactivating the adjacency between two devices. This functionality, traditionally managed centrally by the SDN controller, is distributed to the switches, in order to better handle the dynamism of the wireless network, which is difficult to track by the controller.

The SDN controller maintains the topology and the routing policies of traffic flows. The logical connection between the controller and each single P4-UAV element occurs through the control plane (CP) link. In this work, since focusing on data plane traffic, we assume the CP links are always active and not affected by failures or disconnections. In a real scenario, this assumption is hardly verified. For this case indeed, it is crucial to take additional countermeasures. For instance, it is possible to leverage the programmability of the P4 devices to provide them with a backup path. In that way, whenever a link failure is detected and the controller is unable to overwrite the routing table, the switch is still able to properly forward the packets.

The controller is designed to manage and configure a series of data forwarding and telemetry management policies allowing the nodes to be able to operate and, possibly, modify forwarding rules at runtime, even in the case of temporary disconnection from the control plane. The main functions of the controller, i.e. topology management, forwarding and in-band telemetry management, are implemented as general multi-option policies that can be configured on the nodes both in the deployment/bootstrap phase and in the operational phase.

The AI engine is a set of software applications operating on the resources of the edge node at the ground station. The AI software (i.e., the classifier) receives the updated values of a predetermined set of features as input and computes a series of outputs representing indicators of critical events predicted with a given advance time interval. The AI output is then encoded and interfaced with the SDN controller, responsible for the mapping between the event warning and the most suitable network reconfiguration. The processed features are related to telemetry data that describe the state of the P4-UAV switches and the interconnections between switches. Based on such information, the AI model is trained to recognize any critical issues in the FANET, consequently triggering a reaction by the controller in order to maximize the aerial network lifetime operation (i.e. minimizing the traffic outage time due to failures and recovery procedures), adapting to its dynamic nature. Specifically in this work, the AI model detects wireless connections between pairs of switches that could disconnect within a few seconds, thus alerting the controller of the need to modify in advance the routing routes accordingly, following a make-before-break policy.

The figure also shows the mapping of the use case on the network. The aerial sensor is placed in source node n_1 which has both the P4S and the traffic generator module active. The traffic flow, exiting the generator, is intercepted by the local P4S, configured to manage forwarding and to insert the INT (INT source node) header which includes the metadata of the local node. The traffic is routed to air transit nodes (nodes n_2 and n_3 in the figure), which are also configured to manage forwarding and to insert their telemetry data (INT transit node) into the data packet. Finally, a wireless node co-located with the ground station (node n_4 in the figure) is configured as a telemetry terminal (INT sink node). In n_4 the data traffic is purified of INT headers and is delivered to the application instance to be processed. In parallel, the INT data of each packet is inserted into an INT Report packet containing all the metadata accumulated during the wireless path. INT Report packets are processed by a Telemetry Collector which saves all telemetry data in a local in-memory structure. The data is used to be mapped onto the input features of the AI Classifier. The output generates discrete events on the interface towards the controller thus notifying the event. The SDN controller is programmed to react by implementing topology adjustment and routing reconfiguration policies, anticipating any failure event to minimize the impact on the transmission and reception of application traffic on the data plane network. For example, still considering Fig. 1, degradation on the n_1-n_2 link or on the n_2-n_3 link, due to the relative distancing of the drones or to congestion events at n_2, or even to packet loss events due to an excessive level of CPU load at node n_2, may induce the AI engine to predict a future link disconnection or low quality of service, so as to anticipate the network reconfiguration operations by the controller and avoid significant packet loss events due to link disconnection.

4.3. P4-UAV architecture

The P4-UAV FANET node encompasses drone functionalities (e.g., mobility) along with mesh connectivity and programmable forwarding through an internal P4 switch (P4S). This component represents the fundamental element of the aerial topology capable of implementing the following functionalities:

- Exposing control plane connectivity to the SDN controller.

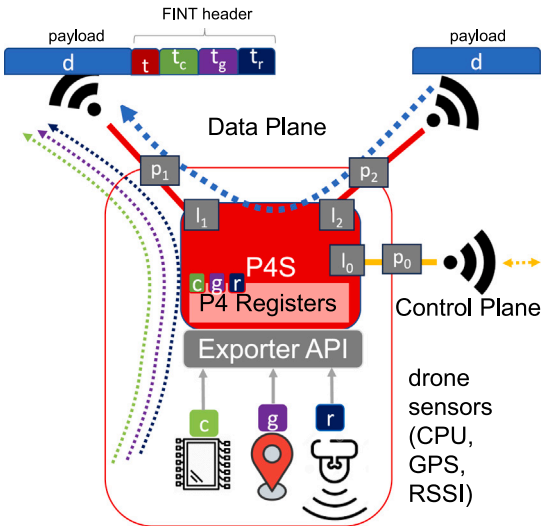


Fig. 2. P4-UAV internal architecture.

- Managing geolocalization and radio channel information through GPS receiver and radio modules.
- Ensuring forwarding of sensor traffic flows through the P4-UAV network via P4-UAV interfaces.
- Implementing the FINT solution by adding additional drone metadata at the packet level of the data flow.

The P4-UAV is implemented as a white box. It is described through a list of external wireless interfaces that connect the P4-UAV to other FANET peers or to the ground station (e.g., another P4-UAV, the SDN controller, the telemetry edge node). Fig. 2 shows the architecture of the node. The central P4S is connected through internal logical interfaces l_x to all the radio interfaces of the UAV p_x using an overlay mechanism based on VXLAN. The different sensors of the drone platform, for example, the CPU load level c, the geolocation tracking g and the radio link level r, provide their data to the Exporter API. The Exporter manages the incoming sensor data and writes their updated values inside specific P4 registers, resorting to the P4 switch command line interface. The packets of the flows carrying payload d are received by the P4S through l interfaces and are then processed. If matched, they are augmented with local FINT header, including standard INT metadata t (e.g., timestamps, hop latency) and specific FINT metadata t_c, t_g and t_r read from the P4S local registers.

5. FANET in-band network telemetry

Network monitoring is a vital mechanism for UAV Networks, especially in dynamic environments where UAVs operate and network conditions change rapidly. Therefore the nodes need to be monitored periodically to ensure their functioning, detect relevant problems and apply network optimization.

Today, different types of network monitoring exist, including traffic probing and network equipment polling. Both of these approaches yield additional traffic that needs to travel over the network and are prone to performance issues.

In–Band Network Telemetry (INT) is an innovative approach to network monitoring wherein monitoring information is directly appended to the data packets by each node along the path, instead of being sent in dedicated packets and without requiring intervention by the control plane. The network state is obtained at the precise moment the real user traffic passes through, reflecting the instantaneous network performance and the exact treatment that an application packet undergoes. INT provides real-time insights, facilitating proactive issue detection

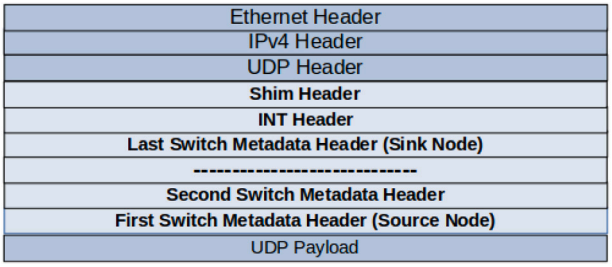


Fig. 3. Standard INT over UDP format.

and enabling the identification and addressing of potential problems before they escalate.

In this work, we have developed and implemented a dedicated FANET In-band Telemetry (FINT), leveraging the standard design of INT over UDP outlined in [3]. This standard structure, shown in Fig. 3, consists of:

1. Shim header: A 4-byte header that indicates the presence of telemetry data. It contains information about the type of INT header following it and the total length of INT header and metadata stack encapsulated between it and UDP payload. The structure of this header is demonstrated in Fig. 4.
2. INT header: An 8-byte header that follows the shim header and contains information about the stack of metadata that follows it, such as the length of metadata to be inserted at each node and the remaining number of nodes allowed to add their metadata to the packet. The fields of this header are also clarified in Fig. 4.
3. Metadata stack: Following the INT header, it consists of a stack of metadata headers. Each header is inserted by an INT node along the monitored path and contains standard metadata, such as ingress timestamp, egress timestamp, and hop latency.

Our contribution lies in extending the standard in-band telemetry structure not only to monitor standard metadata but also to customize the monitoring operation by tracking new key parameters, which is crucial for optimizing UAV network performance. These new parameters are the Received Signal Strength Indicator (RSSI) of wireless links, the geolocation information (GPS longitude, latitude and altitude) and the CPU load. The designed FINT can be activated for specific data flows and its structure is shown in Fig. 3.

In our implementation, every node has the flexibility to serve as a source node, transit node or destination node. Furthermore, a single node can perform multiple roles at the same time: it can act as both a source and transit node or as both a destination and transit node for different flows.

INT source node is the first node in the packet's path of the INT domain. This node inserts a 12-byte structure (i.e., standard shim and INT header) after the UDP header. Subsequently, the source node acts as a transit node, appending 40 bytes of its metadata information into a metadata header known as 'Switch Metadata Header'. The information in this header includes both the standard metadata and the new information specifically tailored for UAV network, as illustrated in Fig. 4. The standard information encompasses: 1. Switch ID. 2. Ingress timestamp. 3. Egress timestamp. 4. Hop latency (the difference between egress and ingress timestamp). 5. Flow ID [33]. The new information includes: 1. CPU load. 2. Longitude. 3. Latitude. 4. Altitude. 5. The RSSI of the incoming wireless interface.

At the next INT transit node, 40 bytes of information are appended to a metadata header called 'Second Switch Metadata Header'. All these metadata headers have the same structure as the Switch Metadata Header illustrated in Fig. 4 . This process is repeated for each subsequent INT transit node along the monitored path. Eventually, at the sink node, the received packet is duplicated using the P4 clone

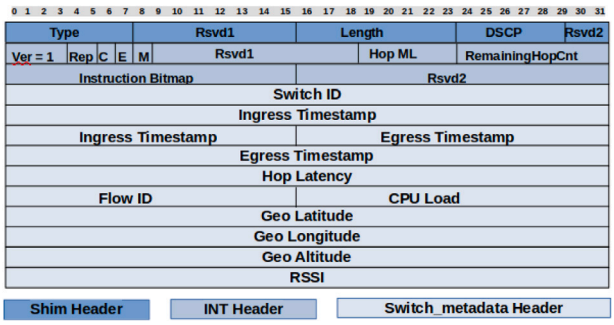


Fig. 4. FINT headers structure.

function. The payload is removed from the cloned packet and the sink node sends it, along with its own metadata information, as a final report to the collector. in parallel, the sink node removes all the FINT information from the original packet and forwards it to the receiver. Fig. 5 demonstrates the P4 pipeline architecture.

5.1. FINT metadata exporter

Regarding the FINT metadata exporter at the P4-UAV level, an API, called Metadata Exporter, is utilized to perform the following tasks:

1. It retrieves the RSSI value from the wireless module and stores it in a specified P4 register array (power_antenna register). The size of this register is denoted by 'n,' where 'n' equals the number of interfaces.
2. It obtains the P4-UAV CPU load value, representing the system load average for the last minute, and then it writes this value in the dedicated P4 register (cpu_register).
3. It gathers geo-location information from the Global Positioning System (GPS) device. The collected data, including longitude, latitude, and altitude, is then written into three distinct P4 registers (longitude_register, latitude_register, and altitude_register).
4. It updates the values in the aforementioned registers periodically.

5.2. P4 pipeline implementation

The considered P4 implementation is based on the V1 architectural model specified in the reference P4 software switch, the Behavioural Model version 2 (BMv2) [34]. The model includes several stages:

1. The parser: It is represented by a finite state machine, where the states are responsible for extracting data from the individual header struct into runtime variables. Transitions within the state machine occur conditionally, depending on the value of a parsed header field.
2. Ingress and egress control blocks: each block consists of several match-action tables that match the packet based on different header fields and determine the appropriate actions to take when a match is detected. It also defines a control function that decides the order in which the tables should be executed. The ingress stage is utilized to perform forwarding, while the egress pipeline is utilized to perform further operations after determining the output port for forwarding.
3. The deparser: Its role involves converting the modified headers back into a serialized format within the packet, preparing it for transmission to the subsequent switch in the network.

The internal structure of the parser, illustrated in Fig. 5, begins by parsing the Ethernet header followed by the IPv4 header. State transitions are determined by the examination of a specific header field, indicating the presence of the subsequent protocol. The parser subsequently processes UDP, Shim, and INT headers. Ultimately, it extracts the Sw_m header from the packet, characterized by a variable length with a predefined maximum size. This header represents a stack of metadata headers, including information appended by every transit node. In order to parse the appropriate variable length of this header at each node, the parsing process has been achieved based on the length field in the Shim header. This field encompasses the total length of Shim header, INT header and the stack of metadata headers (Sw_m header) in 4-byte words. By subtracting the length of the Shim header (1 word) and the length of the INT header (2 words) from the value of this field and converting the result from words to bits, the correct length for parsing this header is determined.

After parsing, packets proceed through the ingress pipeline, depicted in Fig. 5, where three flow tables are executed: Forwarding, Activate Source and Activate sink. The Forwarding table determines the output port based on Layer 2 (L2) MAC addresses, essentially implementing switch behaviour. The Activate Source table initiates the FINT process by invoking the associated action. This action involves inserting both Shim and INT headers into FINT packets received at a designated port with a specific UDP port. Additionally, it updates the length of IP and UDP headers to accommodate the new headers. Meanwhile, the Activate sink table is responsible for duplicating FINT packets by executing the relevant action for packets destined for a particular interface with a specific UDP port.

In the egress pipeline, the control function directly triggers the addition of metadata through the 'adding metadata' action. This action retrieves FINT metadata from various registers (CPU load, Antenna power register, Geo-latitude, Geo-longitude, and Geo-altitude registers), and writes them into the corresponding header fields. It also decreases the number of hops by 1 and assigns this modified value to the remaining hop field in the INT header, which determines the number of traversed nodes. Finally, it updates the lengths of the IP, UDP, and Shim headers to accommodate the new metadata information added by the node. The 'Table Report Forwarding' is tasked with altering the destination of the duplicated FINT packet (report) by modifying the MAC address of this packet.

6. FANET link state forecast

FINT data analytics from the network are used to perform the forecast of the state of the FANET connections used by the application traffic. The main idea is to deploy an AI model capable of predicting a wireless link disconnection event and sending a warning to the SDN-Controller. The controller, aware of the overall topology, performs recovery decisions about that link in advance with respect to actual disconnection. The ground node responsible for the telemetry data consumption is composed of three elements:

1. Data collector: it parses incoming packets to extrapolate telemetry data and generates feature reports to the AI model;
2. AI model: it reads reports extrapolated by the collector, performs classification, and sends a warning whenever a disconnection has been predicted;
3. SDN-Controller: it runs re-routing algorithms whenever a warning has been received, trying to avoid the critical connections, and sending updated flow rules to the involved P4S of the FANET.

6.1. Data collector

The data collector is the component responsible for parsing the packets, retrieving telemetry data, elaborating them and eventually sending them to the AI-model for prediction. The FINT metrics reports are sent to an edge node that reads and interprets them to extract flow and device-related metadata. Those pieces of information anyway are

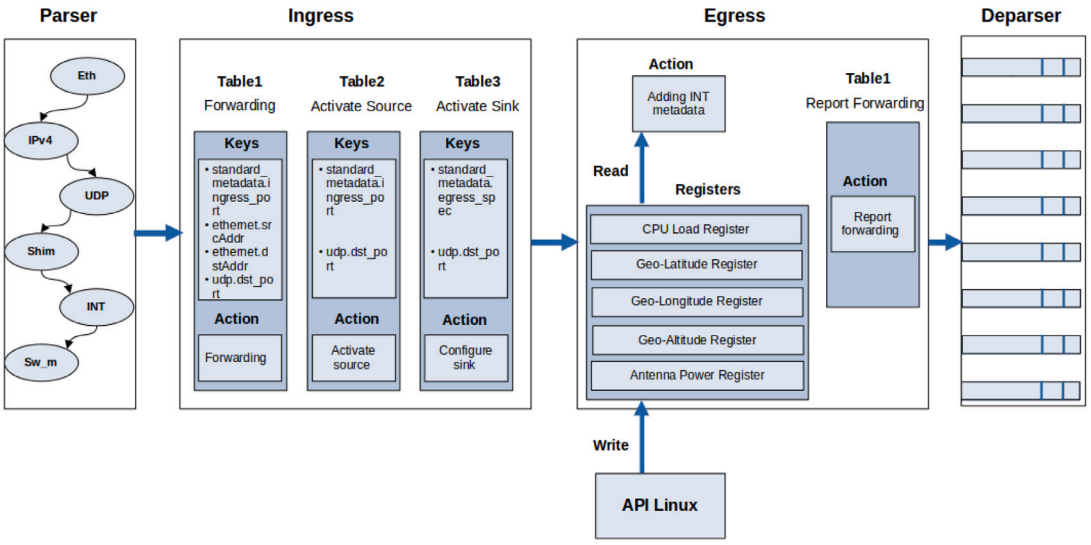


Fig. 5. P4 pipeline architecture.

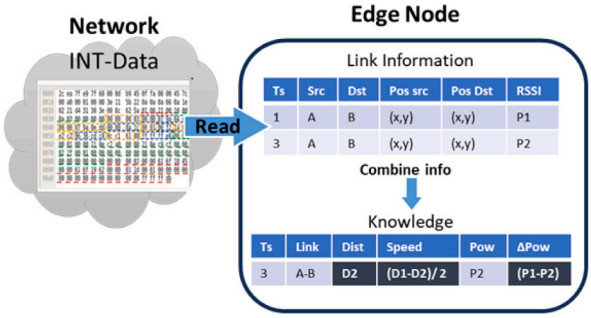


Fig. 6. Processing of telemetry data.

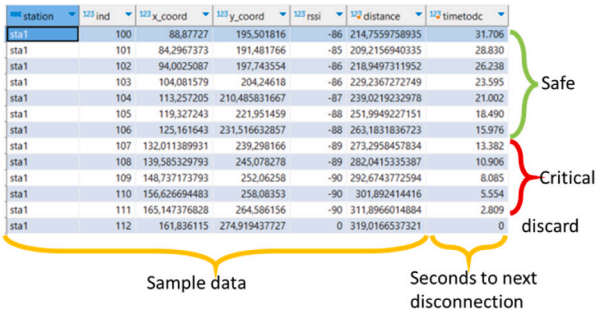


Fig. 7. Dataset: example of sample labelling for a link disconnection pattern due to distancing.

not enough to derive anything about the link status. Thus, the collector correlates information from different reports to generate more interesting metrics, such as the node distance or the relative speed among them and thus generates finer knowledge data about the network status. (Fig. 6). The obtained final report about the link status is sent to a message queue, retrieved and consumed by the AI model.

6.2. AI-model

The deploying of the AI model has been designed through the following steps:

1. Analysis of which kind of data is possible to extrapolate from the network and creation of a dataset for the training;
2. based on the previous stage, defining whether the learning process will be for a classification or regression problem;
3. Model selection;
4. Final test of the model.

6.2.1. Dataset creation

In the first step, the telemetry data used to train and validate the AI model have been generated through MininetWiFi, an open-source emulator of wireless networks that includes wireless channel modelling estimation [35]. In fact, the extended "WiFi"version of Mininet provides the modules *mobility* and *telemetry*. The module *mobility* can be used to emulate the movement of one or more nodes of the network, while the module *telemetry* can be used to collect telemetry data, like position, RSSI, channel errors etc. Therefore, we used this emulator to recreate a network with several moving nodes, following a pseudo-random path, while collecting telemetry data.

We assume the presence of only one access point and several mobile stations that move at different speeds. This is sufficient to collect info about several links with different characteristics. The collected data are stored in an SQL database to provide long-term retention and reuse of the data for several tests. Moreover, the SQL language allows for a rapid combination of link data as shown in Fig. 6. The final dataset used for the training and validation consists of 5485 rows, while the test has been performed on 438 rows obtained in a different execution of Mininet.

6.2.2. Problem definition

After gathering the data related to the connection, it is necessary to select the metric on which to perform the prediction. We decided, for each link, to calculate the time (e.g., the seconds) to the disconnection, and thus defining a threshold to separate a critical status from a safe state. In Fig. 7 an example is shown of how each sample has been labelled.

Note that the threshold has to be set according to several parameters of the network, such as the expected speed of the nodes, the sample rate and the controller efficiency. For this case of study, the bottleneck is the sample rate. MininetWifi indeed allows to sample one data point each 2.8 s roughly. Thus, to ensure not to miss a critical sample, the threshold must be greater than 3 s. Moreover, to consider possible packets lost, or misclassification, it can be useful to set a higher threshold so that more than one report can be classified as critical. For these reasons, we set the threshold at 15 s.

	RFRecall	RFPrecision	SVMRecall	SVMPrecision
1	0.7711	0.5981	0.8193	0.6071
2	0.7742	0.5970	0.7935	0.6029
3	0.7763	0.6378	0.8092	0.6181
4	0.8065	0.5924	0.8065	0.6068
5	0.7810	0.5632	0.7956	0.5798
6	0.7692	0.5911	0.8333	0.6311
7	0.7746	0.6395	0.8028	0.6404
8	0.7910	0.5699	0.8507	0.6096
9	0.6547	0.5353	0.6978	0.5301
10	0.8058	0.5517	0.8417	0.5680

Fig. 8. Precision and recall of the SVM and RF models.

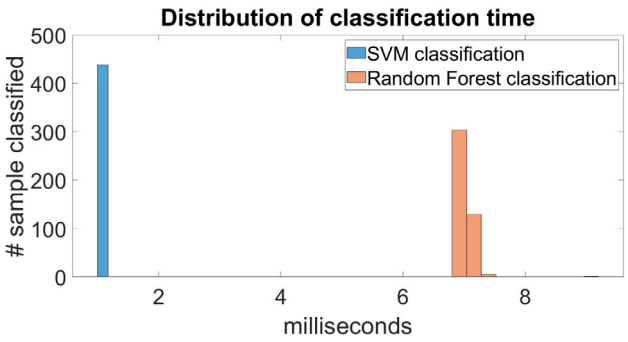


Fig. 9. Benchmark of the SVM and RF models.

	precision	recall	f1-score	support
CRITICAL	0.70	0.78	0.73	185
SAFE	0.82	0.75	0.79	253
accuracy			0.76	438
macro avg	0.76	0.76	0.76	438
weighted avg	0.77	0.76	0.76	438

Confusion Matrix:

[[144 41]
[63 190]]

Fig. 10. Confusion Matrix and results of SVM on the test set.

In a different scenario, with a higher sample rating and faster nodes, it is recommended to set a lower threshold.

Once the label has been assigned, before training, it is necessary to perform the features selection. This step has the double goal of eliminating the features that are useless for learning (e.g., the timestamp has no correlation with the link status) and eliminating the features with low variance, and thus poor information content.

In this first training, the features we include are the following:

- coordinates: we suppose that the nodes can only move on the x and y axis
- RSSI: the received signal strength indicator is a negative number in the range [-30; -90]
- the distance between the two nodes at the edge of the link
- the relative speed between the two nodes. It can be either a negative number if the nodes are getting closer, or a positive value if the nodes are moving away

All the features are numerical values, and the target is the link status, which can be either *Safe* or *Critical*.

6.2.3. Model selection

Due to the nature of the network, the AI model must be topology-independent. For this reason, the learning is performed at link level rather than topology. Indeed, the topology can change its shape due to the moving nature of the nodes and the limited autonomy of the UAVs.

Given the characteristics of the data collected for the forecast, we decided not to rely on Deep Neural Network. For this particular case, indeed, more effective and simple ML models are available. In our previous work [36], we showed that two models are suitable for this scenario. Those two models are *Random Forest* and *Support Vector Machine*. Here, we perform again the training, validation and testing steps on a wider dataset, and provide a comprehensive set of results including performance and accuracy that identify the best model and the best hyperparameters for this case of study.

To evaluate precision and recall, we tune the two models finding the best hyperparameters for each of them, and then we use a K-fold cross-validation, with $K = 10$ to compare those metrics on each fold. In Fig. 8 the results of the classification are shown on the validation set. In this stage, the two models show similar classification performances, with SVM having a slightly higher recall value. Nevertheless, those differences are not enough for a T-test to reject the null hypothesis. This means that the differences are not statistically significant and we cannot really state which model is outperforming.

However, when we compare the classification time, we achieve far better results for SVM, as shown in Fig. 9. The benchmark shows that SVM requires 1 ms to perform a classification, against the 7 ms of Random Forest.

6.2.4. Model test

Once the model has been selected, we perform again the training on the training set and validation set and test it again against a novel test set, a set of fresh data unseen by the model. In this way, overfitting issues are avoided, moreover, an in-depth analysis of the results can be performed. In Fig. 10 the results of the test set are reported. As expected, the precision and recall are in line with the results in the validation stage, meaning that there is no overfitting. It is interesting to perform an in-depth investigation of the errors. Particularly, we focus on the false safes that are, in our opinion, the critical part of the network. As mentioned before, the considered approach introduces an information loss about "how critical" a sample is. Therefore, it is interesting to evaluate the false safes distribution based on their time proximity to the disconnection. This distribution is shown in Fig. 11.

The results show that most of the errors occur when the disconnection happens in more than 12 s. The closer a critical sample is to the disconnection, the higher the accuracy. Eventually, when the next disconnection is within 4 s, the recall is 100%.

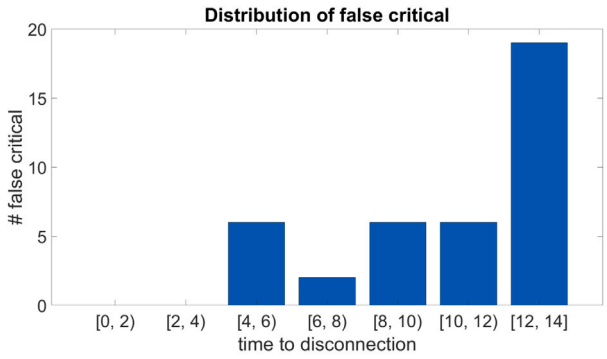
6.3. SDN-controller

The SDN-Controller has been designed to execute the following task:

1. Computation of the shortest path among two hosts;
2. Managing the warning received from the AI-model;
3. Configuring the P4-switches accordingly with the results of the previous two points.

The actions of the controller are triggered through Rest-API. Therefore, whenever it is necessary to instantiate an application flow among two nodes of the network, an API request is submitted to the controller, which computes the shortest path, using the Dijkstra algorithm, and eventually pushes the forwarding actions in the flow tables of all the involved switches. Whenever the AI-model finds a critical link, a warning API message is sent to the controller. As a response, the controller sets a higher link cost on the critical link and recomputes the shortest path for all the couple of endpoint nodes.

Moreover, since the AI may return a false critical, the controller stores all the links classified as critical in a hash table, associating them with the relative timestamp of the last warning received. Then,



(a) Histogram of the results on the test set

sec to next disc	# samples	false safe
$[0, 2)$	20	0
$[2, 4)$	20	0
$[4, 6)$	39	6
$[6, 8)$	19	2
$[8, 10)$	37	6
$[10, 12)$	17	6
$[12, 14)$	31	19

(b) Table of the results on the test set

Fig. 11. Distribution of the false safes based on the proximity to the next disconnection.

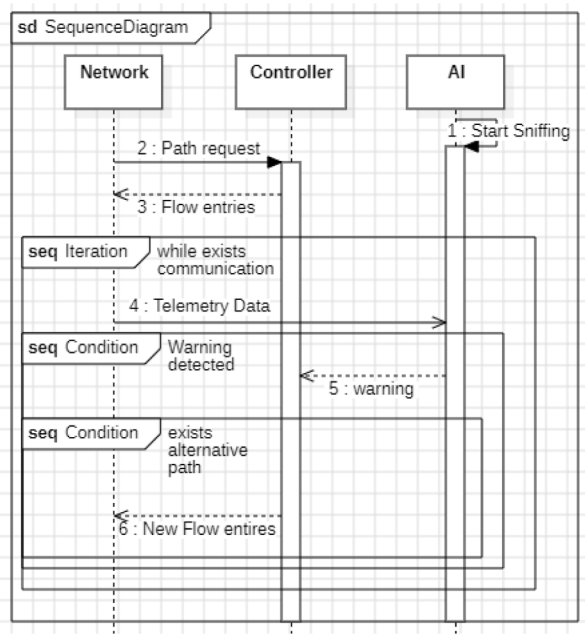


Fig. 12. Sequence diagram of the forecasting at the edge node.

periodically, the controller will delete the link from this table if no more warnings or failures are received in the last seconds, assuming thus that the link is safe again.

6.4. Implementation: components interfaces

The communication between the AI-model and the collector has been realized with RabbitMQ [37], a managed message queue written in Erlang [38]. This solution allows the decoupling of the two components, taking into account the different performances of a sniffer and an AI-model.

In order to prevent an overload of the AI, the collector performs a filter on the reports, sending no more than one report per link per 100 ms. Thus, the model must process no more than 10 reports per active link per second. On the other hand, the AI process interacts with the controller through Rest-API.

The edge node's components need to communicate with the P4-switches as well. Particularly, three types of messages need to be exchanged:

- *Telemetry Data Reports* from switches to data collector;
- *Path Computation requests*, triggered whenever two hosts need to communicate, or when a topology change occurs;
- *Flow entries*, sent by the controller to the switches.

In order to obtain telemetry data, the collector listens to a predefined port (e.g., in the tests we used the UDP port 12345). The sniffer is based on the *libpcap* library. Path computation requests are triggered by the P4S switches, calling a Rest-API whenever a new neighbour is detected, or an old one becomes unreachable. The control plane communication between the SDN-controller and the P4S nodes resorts to the P4Runtime protocol using the P4Runtime-shell library [39], written in Python. Control operations include the bootstrap phase and the runtime phase. In the bootstrap phase, the controller instantiates a shell through each switch and pushes the P4 pipelines. In the runtime phase, whenever a new or recovered path is computed, flow entries are derived and sent by the controller to the switches to activate the related match-action rule.

To summarize the forecasting workflow, we show in Fig. 12 a sequence diagram of the use case. Note that the network (i.e., FANET) is hereby considered as a single entity.

The diagram shown in Fig. 12 does not consider the case of link creation and link deletion, since these scenarios do not involve forecasting. The diagram steps are the following:

1. The AI collector starts sniffing Report packets using *libpcap* library;
2. A host of the FANET is required to communicate with another host and thus requests the controller to calculate a path;
3. Once the controller finds a path, it has to configure the switches of the network, providing them with flow entries;
4. While packets are flowing among the two hosts of the FANET, FINT will be generated and attached to those packets, and thus the AI will process those data;
5. AI will use telemetry data to perform predictions about the links traversed by the traffic. Whenever a critical link is found, AI will send a warning to the controller using the exposed Rest-API;
6. Eventually, the controller checks whether a different path is feasible and, if this is the case, it pushes the updated flow entries into the involved switches.

7. System performance and experimental results

This section summarizes the experimental tests carried out to evaluate the performance of the proposed solution.

7.1. Testbed description

The proposed FINT scenario and architecture has been deployed and evaluated utilizing the experimental setup depicted in Fig. 13.

The four red devices, labelled S1, S2, S3 and S4 respectively, are the P4-enabled switches that will be onboarded on the UAVs. In this setup, the WiFi links among the devices are made using coaxial cables, avoiding interference with the rest of the laboratory equipment. The architecture of the devices is similar to the one described in [40]. All

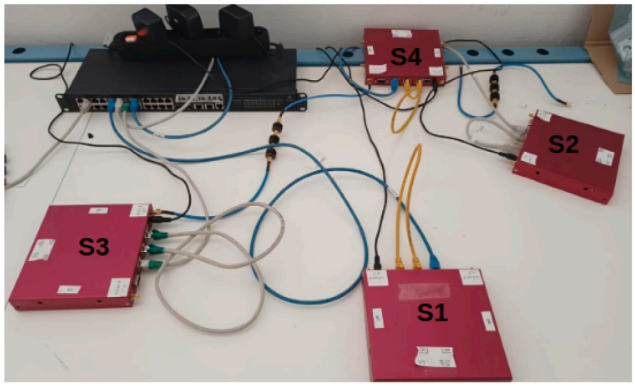


Fig. 13. FINT Testbed: P4S aerial segment topology.

Fig. 14. P4-UAV drone integration with the APU platform hosting the P4S.

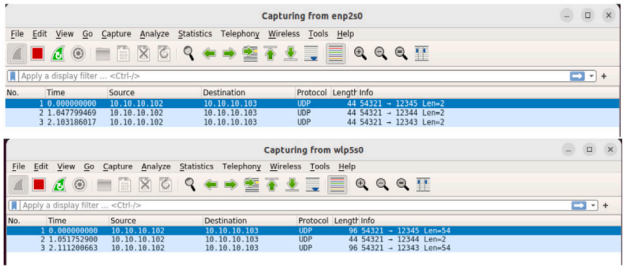


Fig. 15. Wireshark capture of FINT implemented on specific flows.

those switches are then connected to a management switch (Netgear) that is responsible for the forwarding of the packet to the ground station, where the telemetry will be parsed and elaborated.

Each device is built of a versatile x86 board called APU1 produced by PC Engines. This platform ensures compatibility with a diverse range of software and applications. The devices are equipped with an AMD G series T40E CPU. The CPU is a Bobcat dual-core with a clock frequency of 1 GHz and is compatible with both 64-bit and 32-bit applications. Each core has a 32 K data cache, a 32K instruction cache, and a 512K L2 cache. The inclusion of 2 GB DDR3-1066 RAM in each device ensures efficient management of active applications and processes and enables APU1 unit to effectively handle workloads ranging from light to moderate. In addition, the devices natively host three gigabit Ethernet interfaces based on the Realtek RTL8111E chipset and a DB9 serial port which allows the management of the device even in the absence of a video card or disabled networking. Moreover, every APU1 is equipped with three mPCIe slots, providing flexibility for further expansion. Among these slots, one is designated for the integration of a 64 GB m-SATA SSD drive, the other two have been adopted for the WiFI radio modules, allowing adaptability for running in a FANET node. The APU1 board's dimensions measure 152.4 x 152.4 x 25 mm, rendering it compact and well-suited for environments with spatial constraints. The APU1 board has been outfitted with Linux operating system (Ubuntu), and augmented with two dual-band miniPCIe Qualcomm DAXA-F1 QCA9980 radio modules. Each module offers a triadic chain configuration for harnessing multi-input multi-output (MIMO) technique and supports 802.11 a/b/g/n/ac standards. The wireless channels have been established using the standard 802.11a at the 5 GHz frequency band. The WiFi interfaces have been enabled and configured on each node in a dedicated configuration. The core of the node's architecture lies in the use of the dockerized BMv2 instance [34] to effectively onboard the P4S and ensure an isolated environment for running services. This methodology yields numerous benefits, including a simplified deployment process, scalability, the capacity to update the software independently of the underlying operating system, and the guarantee of enduring software stability.

The picture of Fig. 14 shows the possible P4-UAV deployment integration between a real UAV and the APU platform hosting the P4S. In a minimal configuration, only a power supply connection is needed between the two platforms. In this way, the drone native OS and the APU OS are kept independent. Moreover, additional metadata exposed by the drone OS (e.g., battery level) may be easily integrated into FINT by resorting to an additional Ethernet interface connecting the drone case and the APU.

The implementation uses a unique and efficient approach, based on the Python P4Runtime-shell library, to send the flow entries to the different P4Ss. The library allows the controller to open a shell to the P4 devices, and use this shell to write or edit the routing table. On the other side, the topology events, like a link disruption or creation can be sent using REST API deployed through the Flask library [41]. The same mechanism is used by the classifier to send warnings whenever a dangerous link is detected.

The picture in Fig. 13 illustrates the section of the aerial network with the four APU1 nodes. The nodes are connected to each other with wireless data plane interfaces adopting coaxial cables and attenuators or in free space, depending on the tests to be performed. The ground node control plane and data plane interfaces are wired over Ethernet and are connected in star centre mode to an Ethernet switch, attached to the subnet that includes the edge node and the SDN-P4 controller.

7.2. P4S performance

7.2.1. Functional validation

The functional validation has been implemented in the testbed demonstrated in Fig. 13. An API in Linux has been launched on each node to collect information about the antenna power of wireless interfaces, CPU load, and geo-location information. It gathers geo-location information from a Global Positioning System (GPS) device using a Python script that utilizes the gpsd library. This library facilitates interaction with GPS devices through the gpsd service. Then, the API writes this information inside the appropriate register and updates these values every 1 s. Three flows of traffic with different destination UDP ports (i.e., 12345, 12344, 12343) have been generated in P4S4. The three flows are injected into the P4S container for processing. Here the P4S4 acts as a source node, selectively inserting shim and INT headers only into the flows destined for UDP ports 12345 and 12343, based on predefined flow entries, shown in Fig. 15.

The switch adds its metadata to these two flows and forwards the three flows to P4S2. P4S2, in turn, appends its metadata to the incoming flows and forwards the three flows to P4S1, the sink node.

The figure shows that the length of the FINT Report header related to each node is 40 bytes. Therefore The total length of the extra FINT

Table 2
The latency on the P4-UAV path with and without FINT.

Rate (p/s)	RTT Without INT			RTT With INT		
	2 switches	3 switches	4 switches	2 switches	3 switches	4 switches
1	4.896	6.953	14.243	5.627	8.980	13.276
10	5.164	6.461	13.045	5.712	8.980	12.991
100	4.272	5.687	10.895	5.103	8.980	11.217
1000	4.039	6.595	9.470	5.830	8.980	10.037

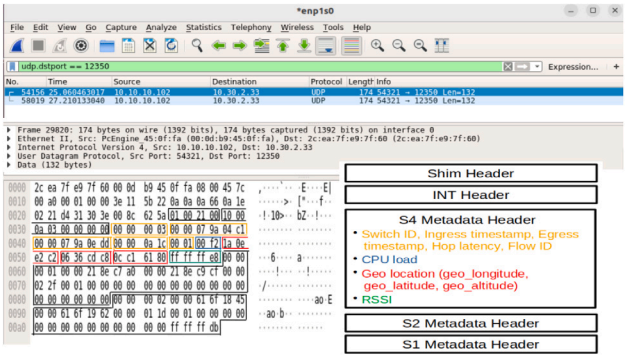


Fig. 16. Wireshark capture of FINT-tagged traffic.

Table 3
The hop latency of P4S node exported as FINT metadata.

Rate	Num. nodes traversed	Hop latency (μs)			
		Node1	Node2	Node3	Node4
1 p/s	2	587	587	–	–
	3	341	340	527	–
	4	314	347	347	496
10 p/s	2	334	578	–	–
	3	335	369	552	–
	4	356	411	384	465
100 p/s	2	324	464	–	–
	3	332	399	508	–
	4	435	374	403	424
1000 p/s	2	535	589	–	–
	3	670	401	451	–
	4	854	375	360	364

header is $12 + 40N$ bytes, where N is the number of nodes traversed. In the implementation, the maximum FANET path length is assumed equal to 6, with a maximum FINT size of 252 bytes in the worst case (at the sink node).

At this stage, P4S1 duplicates each FINT packet, removes the payload from the cloned packets and sends them as a final report to the collector after adding its metadata. The two report packets related to the two flows are captured and illustrated in Fig. 16.

The functional validation confirms the effectiveness of the proposed FINT protocol extension and the ability to activate FINT on specific traffic flows while guaranteeing the transparent forwarding of additional flows. Moreover, it quantifies the amount of FINT overhead required by the header extensions.

7.2.2. End-to-end latency

The purpose of this set of experiments is to check the ability of the P4S container to process FINT information with a limited additional latency compared to the latency introduced with standard forwarding, with variable topology of P4S nodes with different path depths and variable traffic. Moreover, we investigate the ability of the P4S container to manage and process FINT information for any switch role (source, transit and sink). The impact of data plane programmability in terms of accumulated latency was evaluated. The ICMP traffic has been used for the generation of packets at variable rates. The latency is evaluated as a function of the traffic bitrate and path length, ranging from a minimum of 2 to a maximum of 4 P4-UAVs. In these experiments, the Round Trip Time (RTT) latency measured by Ping includes all the P4-UAV system components, including the radio interfaces and the wireless links.

The tests were carried out on the topology of Fig. 17. The paths P4S4-P4S2, P4S4-P4S2-P4S1 and P4S4-P4S2-P4S1-P4S3 were considered in the different measurements. The S4 node always acts as a traffic generator and receiver. Ping packets are generated externally to the P4S container and are routed inside the P4S container which processes the packet. In the set of measurements without INT, each P4S was configured only in static forwarding mode, allowing pings to be routed to the next hop in a bidirectional fashion. In the set of measurements with FINT, the P4S has been populated with flow rules to correctly manage the FINT. Then, a first FINT stack was configured up to the sink node. The sink node eliminates all FINT headers and forwards the ping request packet to the external radio interface of the last node. The interface responds with a ping reply in the backward direction. In this case, the local node acts as the source node for a new FINT stack backwards to the source. The experienced RTT results are shown in Table 2 when varying the rate of the packet per second (p/s) and the number of the switches.

The table shows that the additional contribution due to FINT processing is always less than 0.5 ms per node in a single direction. The non-linear latency values concerning data plane bitrate are attributed to the behaviour of the BMv2 container which is well known in the literature [42]. Specifically, the switch demonstrates superior performance within an aggregate bitrate range of 150 to 800 Mbit/s, exhibiting increased latency at lower bitrate. However, it is clearly noticed that the latency is always less than 5 ms per node, with or without FINT.

7.2.3. Hop latency

The intra-node latency values measured by the P4S and entered as FINT metadata are reported as a function of the bitrate and the number of the nodes traversed, as shown in Table 3.

Results show that, as expected, the processing of the source and sink nodes is more intense and requires a longer execution time. In general, the sink node seems to require the greatest contribution of latency (variable from 400 to 500 μs) while the transit nodes introduce the least contribution of delay (from 300 to 400 us). The source node has a latency that strongly depends on the number of nodes traversed, especially at high bit rates. The contribution of the P4S in terms of latency is always less than 0.9 ms in the case of a source node, always less than 0.4 ms in the case of a transit node, and always less than 0.6s in the case of a sink node.

7.2.4. CPU load and RAM resources

The performance of the P4S container has been evaluated with regard to its capacity to handle packet information using FINT without imposing a significant additional CPU load compared to the standard load introduced by the P4 switch with regular forwarding. This evaluation involves different FANET topologies, featuring different path depths and fluctuating traffic conditions. Additionally, we evaluate the ability of the P4S container to manage and process FINT information for any switch role (source, transit and sink) with a RAM requirement

Table 4
CPU load performance of P4S container with and without FINT.

Rate (p/s)	#Node	% CPU INT – NO INT			
		Node1	Node2	Node3	Node4
1	2	0.75–0.93	0.61–0.69	–	–
	3	0.76–0.86	0.60–0.69	0.59–0.71	–
	4	0.78–0.91	0.62–0.71	0.77–0.68	0.27–0.35
10	2	3.95–4.42	2.68–3.12	–	–
	3	4.1–4.61	2.72–2.83	2.63–3.04	–
	4	2.35–4.62	2.85–2.85	2.83–2.86	1.12–1.61
100	2	25.1–28.74	24.09–28.2	–	–
	3	27.35–30.72	24.28–25.66	24.17–28.66	–
	4	27.49–29.19	23.58–23.90	23.2–23.60	8.74–12.17
1000	2	50.9–55.88	45.43–52.43	–	–
	3	31.74–35.33	27.39–28.88	28.92–32.42	–
	4	28.78–33.22	24.57–27.35	24.54–26.56	9.31–16.77

Table 5
RAM performance of P4S container.

	P4S Off	P4S On	P4S 1 p/s	P4S 10 p/s	P4S 100 p/s	P4S 1000 p/s
Rate (p/s)	260 428	278 768	285 613	286 056	288 308	288 514

compatible with the resources available at the individual APU platform level. In this case, the evaluations are conducted by comparing the performance in terms of CPU and RAM of the P4S solution with simple static forwarding against forwarding accompanied by complete FINT processing.

Table 4 shows the obtained results. From the performance analysis, it becomes evident that FINT processing requires an additional contribution of CPU load compared to the performance required by the entire P4S in a simple forwarding configuration. The additional load is approximately 2% at maximum for low rates and 7% at maximum for high rates. The most significant contributions are observed in the sink nodes, while the smallest contributions are observed in the transit nodes. Importantly, the P4S occupies at most a maximum of 50% of the CPU in any configuration, ensuring a substantial operational margin for all other essential processes of the P4-UAV.

Table 5 depicts the RAM requirements of the P4S container at various data rates, demonstrating the total RAM utilization on the P4-UAV (APU1) platform. In the absence of the P4S container, the P4-UAV system consumes around 260MB of RAM. The P4S container powered up with P4 code compiled without flow rules enabled requires approximately 18MB of additional memory. Activating the essential flow rules for forwarding and FINT functionality incurs an additional RAM demand of about 10MB. This incremental RAM requirement ranges from 7MB to 10MB, depending on the fluctuation in traffic, varying from 1 packet per second (1 p/s) to 1000 packets per second (1000 p/s). The results show that the system does not require excessive amounts of RAM in any operating configuration. Considering that the total amount of RAM the APU1 platforms offer is 2 GB, the additional percentage of RAM needed in the system for the activation and complete functioning of the P4S stands at approximately 1.5%.

7.3. Edge node evaluation results

In order to evaluate the edge node we connected all the pieces we described in this paper and simulated a link degradation. The server hosting the controller, the AI classifier and the collector has the following properties:

- 16 core Inter Xeon Gold 2.20 GHz, with hyperthreading (2 thread per core);
- 77 Mebibyte (MiB) L3 Cache;
- 8 MiB L2 cache;
- 512 Kibibyte (KiB) L1 cache;
- 64 Gibibyte (GiB) RAM (4x16 GB DDR4) with 0.3 ns access delay;

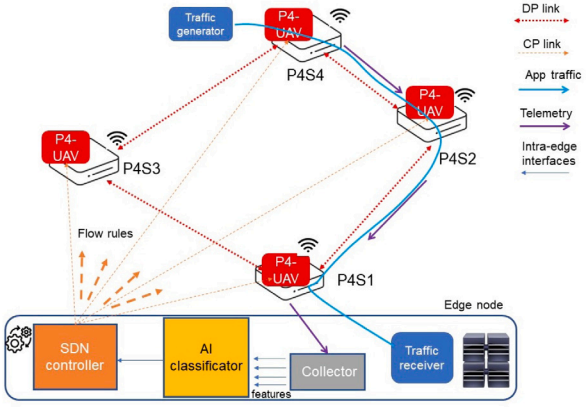


Fig. 17. Forecasting: Initial state.

- 2 SDD of 479 GB for storage;
- OS Ubuntu LTS 20.04 with kernel Linux 5.15.0-87-generic.

We carried out both functional and non-functional tests. The first ones aim to prove the capability of the edge node to handle a dynamic topology. The second class of test are needed in order to estimate the workload that the server can handle. It should be noted that this second type of test depends both on the implementation of the algorithms and on hardware resources.

7.3.1. Link failure forecasting

This test scenario is divided into two phases: in the first phase, shown in Fig. 17, the switch P4S2 is progressively getting close to P4S4. In this case, thus, the two switches will establish a connection and the traffic that has P4S4 as the source and P4S1 as the destination, will use this link.

In the second phase (Fig. 18), the switch starts to move away. The AI, after a while, will classify the link P4S2-P4S4 as no longer safe to use. The controller then, will receive the warning and will search for another possible path. In this case, it is possible to use P4S3 instead of P4S2. The controller will then recompute the path and will send new flow entries in the routing tables. the traffic will then use the new path, avoiding the dangerous link. In the test, we will show the ability of our solution to take preventive countermeasures to avoid packet loss in case of topology changes.

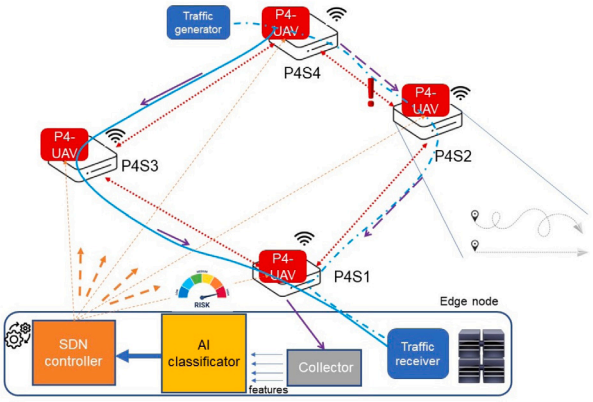


Fig. 18. Forecasting: Link degradation.

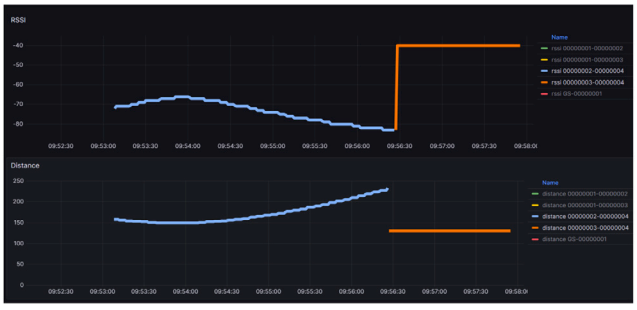
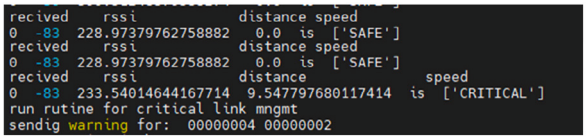


Fig. 19. Graph of the evolution of RSSI and distance of the used link.



(a) AI-model find a potentially critical link.

(b) Controller receives a path computation request and, later, a warning for a used link.

Fig. 20. Workflow of the edge node.

Name/ID	Tx Port Name	Rx Port Names	Aggregated Rx Port Count	Tx Count (Frames)	Rx Count (Frames)	Dropped Count (Frames)
sw_4 - sw_1/131072	Port //1/4	Port //1/3	*1*	*130.703*	*128.576*	*1.075*

Fig. 21. Packet loss without prediction.

Name/ID	Tx Port Name	Rx Port Names	Aggregated Rx Port Count	Tx Count (Frames)	Rx Count (Frames)	Dropped Count (Frames)
sw_4 - sw_1/131072	Port //1/4	Port //1/3	*1*	*81.763*	*81.763*	*0*

Fig. 22. Packet loss when the AI predict the failure.

In the experimental testbed, wireless links are connected using coaxial wires and attenuators and the switches do not move. Therefore, to emulate mobility, we created a script in the Metadata Exporter that periodically updates P4 registers with values of geolocation and RSSI. During the tests, we emulated a node (P4S2) moving far from the topology, thus inducing a progressive degradation of the quality of the signal. The AI will notice the degradation and will predict that the quality of the signal will not improve. Thus, it will send a warning to the controller, which will check for an alternative path. The observed behaviour in the experiment is shown in Fig. 19.

The distance between nodes P4S4 (ID: 00000004) and P4S2 (ID: 00000002) is increasing, with the related RSSI degradation. We can notice this behaviour by observing the blue line in Fig. 19, that represents the link 00000004-00000002. At a given time, the link is not used anymore and FINT telemetry data will not be collected. The old connectivity is replaced by the link P4S4-P4S3 (ID:00000004-00000003), represented in Fig. 19 by the orange lines.

In Fig. 20 it is shown how the controller and AI work together to prevent a link failure and avoid packet loss in the network.

7.3.2. Handover performance

In order to validate the deployed solution, we check how the unavailability time (i.e., the outage time of the flow traffic) changes whenever a predictable failure happens. For this purpose, after setting up the testbed, we have emulated a progressive degradation of the link, as in Section 6.4. The test has been repeated twice: with and without the AI node. A Spirent N4U [43] traffic generator has been connected to two end-points of the testbed topology, emulating a host source and a host destination. The Spirent is configured to send 1000 packets per second, thus the packet loss is equal to the number of milliseconds the network is unavailable (i.e., the outage time). In Fig. 21, the packet loss measured by the Spirent is shown, when the AI prediction is deactivated. In this case, recovery accounts for 1 s of unavailability.

In Fig. 22, the same results are shown when the AI is activated and anticipates the failure link. In this case, the controller recomputes the path and pushes the flow entries into the forwarding table of the switches and no packet loss occurs. This means that the recovery based on the forecast is able to drastically reduce the outage time of more than three orders of magnitude (i.e., from 1s to under 1 ms).

Without AI, several contributions account for the recovery time, such as fault detection, the recalculation of a new path and, eventually, the reconfiguration of the switches along the new path. On the other hand, using AI prediction, the system circumvents a true recovery from the fault since it is actually anticipated. Therefore, the detection time since recomputation and reconfiguration are irrelevant since re-optimization takes place while the link is still working.

If we denote as t the working time of link P4S4 - P4S2, the availability in the classical scenario is $(t)/(t+1s) < 1$. To improve the overall dependability, we are not acting on the resilience of the link, but we are working on the recovery time. For this specific case thus, bringing this time to 0, we reach an availability of 100%.

7.3.3. Edge node capacity test

We conclude our evaluation of the edge node performance by testing its capacity. This is useful in order to understand how many packets the edge node can handle and thus correctly size the telemetry of the network.

We used the Spirent traffic generator that creates UDP packets and sends them to the first switch of the network (P4S4). The flow entries are configured so that when a UDP packet with the given source and destination comes into switch P4S4, it will be sent to the host connected to switch P4S1. Here, a copy of the packet is mirrored to the edge node that will parse it and perform the forecast. We tested first the telemetry collector alone, and then the combination of the collector and AI model.

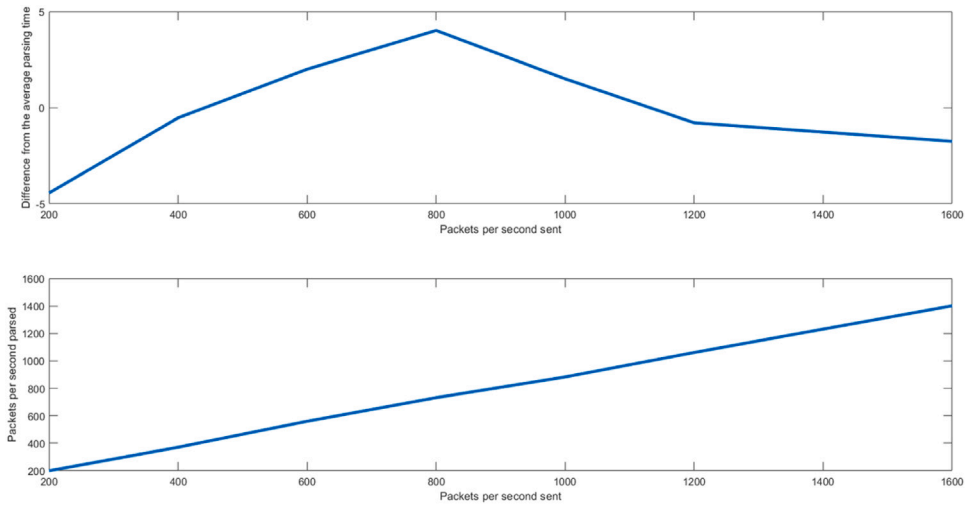


Fig. 23. Capacity test results.

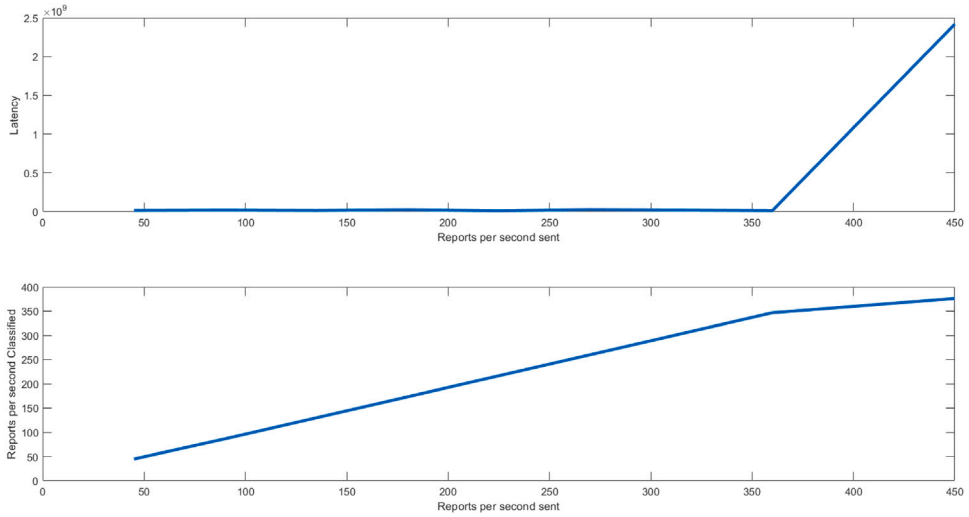


Fig. 24. Capacity of the AI model.

The reason behind this choice lies in the different speeds expected by the two components. Indeed, the forecasting is expected to be slower than the parsing.

In the first test, the traffic generator emulates a growing number of users, from 1 to 8, generating UDP packets. Each user generates packets 2000 packets at the rate of 200 packets per second. Thus at the beginning, the edge node receives 200 packets per second and finishes by receiving 1600 packets per second. The purpose of this test is to check if we can saturate the capacity available of the collector before reaching the maximum bandwidth established. To do this, we check the latency and the throughput of the collector as the input grows. The results are shown in Fig. 23.

In Fig. 23 the first graph shows the latency variation compared to the average value. As we can see, the latency never diverges, meaning that the packets are never queued and the collector manages the amount of packets generated. This is also confirmed by the throughput plot. Indeed we never observe a knee on the curve. This basically means that the collector can keep up with this amount of requests, and potentially can handle even additional incoming rates.

In the second step of the capacity test, we also included the classification time. In this case, we expect a significant overhead, therefore we decided to reduce the number of packets sent in the network. We plan to handle this different speed by implementing a filter on the collector (e.g., the collector sends no more than one packet each 0.100 ms per link). With this test, we aim exactly to figure out how this filter should perform. In this case, we do not consider the number of packets sent, but the number of the reports classified. Indeed, a single packet can include more reports, based on how many switches are traversed.

Fig. 24 shows the results of the test. In this case, we managed to saturate the capacity of the server. Indeed we can notice how, when the throughput of input data becomes too high, the throughput of the classification stops growing and the latency starts increasing, due to queuing time. The maximum capacity of the AI model is nearly 360 reports per second. After this value, the model cannot keep up with the request rate anymore, resulting in packets queued and thus a growth in latency. Due to this test, for a server with the aforementioned capability, we believe that the classifier should take in input no more than 340 packets per second while running, preserving some spear capacity for possible spikes in the number of active links. A heavier workload could be dangerous since in case of traffic spikes the server would not have the capacity to handle the event. At the same time, using a lighter workload would be a waste of resources since the server would be under-exploited.

Lastly, in Fig. 25 we show the results of the resource utilization during the test. In Fig. 25 we immediately find out the different stages of the test. Indeed we can notice a growth of used resources as time increases. Even though the CPU utilization is growing, it never exceeded

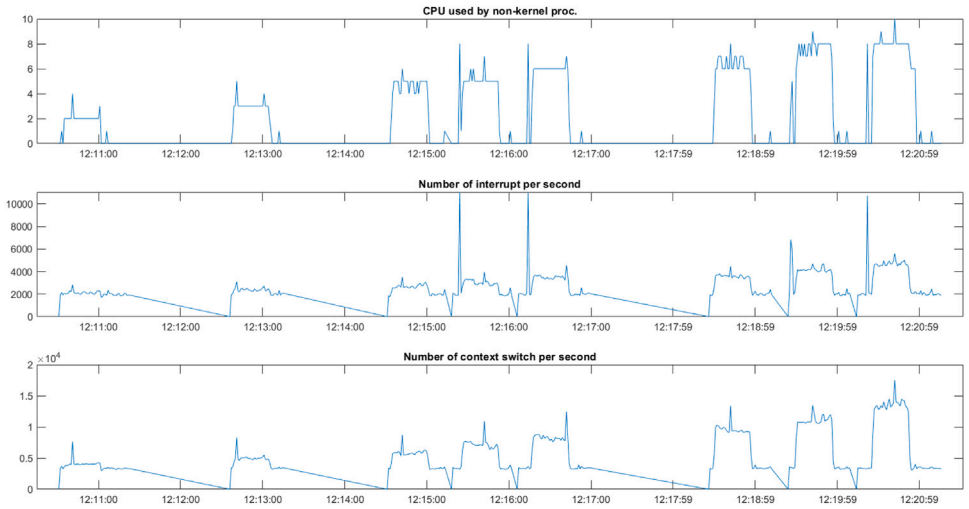


Fig. 25. Resource metrics of the server during the test.

10%. What we notice growing up is the number of context switches, that seem to be strictly correlated with the number of interrupts. Therefore we can conclude that, in our case, the capacity is full because the server spends more time performing context switches than time performing the actual classification. At the same time, we have to consider that we used a general-purpose OS, simultaneously hosting the controller and the AI model, along with additional background processes. Thus, the capacity can surely grow up using a Real-Time OS on dedicated servers.

8. Conclusions

This paper presented a novel architecture of SDN-enabled FANET nodes enriched with a programmable P4 switch capable of managing augmented FANET in-band telemetry of key aerial network node vital parameters, including real-time geolocalization, radio link levels and internal resource occupancy. The proposed FINT was exploited to perform data analytics on the FANET state at the ground edge node. Wireless link state prediction using AI allowed the system to be robust against mobile topology changes, enabling anticipated recovery thus minimizing traffic outages occurring in the FANET. Experimental evaluations showed the ability of the P4 switch to effectively enable FINT with limited resource burden, while allowing the AI forecasting mechanism to reduce the link failure outage time by three orders of magnitudes, guaranteeing zero packet loss at the millisecond time scale.

This work focuses on the P4 capabilities and machine learning model accuracy for failure prediction. However, in a similar scenario, it is also crucial to consider the power consumption of the component of the network. UAVs indeed have a limited capacity in terms of power supply and therefore it is fundamental to optimize energy resources. In our next works, we will consider extending the current framework in order to address this challenging topic, integrating FINT with metadata about energy consumption, allowing the AI to consider power metrics and taking the most suitable decisions to allow the network to work with a sustainable and reduced energy footprint.

CRediT authorship contribution statement

Layal Ismail: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Data curation. **Domenico Uomo:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Investigation, Formal analysis, Data curation. **Andrea Sgambelluri:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Faris Alhamed:** Writing – original draft. **Francesco Paolucci:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Layal Ismail reports financial support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Sant'Anna School of Advanced Studies. Domenico Uomo reports financial support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Sant'Anna School of Advanced Studies. Andrea Sgambelluri reports financial support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Sant'Anna School of Advanced Studies. Faris Alhamed reports financial support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Sant'Anna School of Advanced Studies Francesco Paolucci reports financial support, article publishing charges, equipment, drugs, or supplies, and travel were provided by CNIT. Layal Ismail reports article publishing charges, equipment, drugs, or supplies, and travel were provided by CNIT. Domenico Uomo reports article publishing charges, equipment, drugs, or supplies, and travel were provided by CNIT. Andrea Sgambelluri reports article publishing charges, equipment, drugs, or supplies, and travel were provided by CNIT. Faris Alhamed reports article publishing charges, equipment, drugs, or supplies, and travel were provided by CNIT. Francesco Paolucci reports article publishing charges, equipment, drugs, or supplies, and travel were provided by Sant'Anna School of Advanced Studies. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgements

This work has been partially funded by the European Commission Horizon Europe SNS JU DESIRE6G project, under grant agreement No. 101139285, and by the European Union's Horizon RIA research

and innovation programme under grant agreement No. 101092908 (SmartEdge).

Moreover, the work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of the NextGenerationEU partnership on "Telecommunications of the Future" (PE00000001, program "RESTART").

References

[1] K. Grace, J. Salvatier, A. Dafoe, B. Zhang, O. Evans, When will AI exceed human performance? Evidence from AI experts, 2017, CoRR abs/1705.08807. arXiv:1705.08807.

[2] W.L. da Costa Cordeiro, J.A. Marques, L.P. Gaspary, Data plane programmability beyond OpenFlow: Opportunities and challenges for network and service operations and management, J. Netw. Syst. Manage. 25 (4) (2017) 784–818, http://dx.doi.org/10.1007/s10922-017-9423-2.

[3] In-band network telemetry, 2023, https://p4.org/p4-spec/docs/INT_v2_1.pdf. (Accessed: August, 2023).

[4] P4.org, 2023, https://p4.org. (Accessed: August, 2023).

[5] J. Loo, J.L. Mauri, J.H. Ortiz, Mobile Ad Hoc Networks: Current Status and Future Trends, first ed., CRC Press, Inc., USA, 2011.

[6] K.P. Valavanis, G.J. Vachtsevanos, Handbook of Unmanned Aerial Vehicles, vol. 1, Springer, 2015.

[7] S.A.H. Mohsan, M.A. Khan, F. Noor, I. Ullah, M.H. Alsharif, Towards the unmanned aerial vehicles (UAVs): A comprehensive review, Drones 6 (6) (2022) http://dx.doi.org/10.3390/drones6060147.

[8] T. Kim, S. Lee, K.H. Kim, Y.-I. Jo, FANET routing protocol analysis for multi-UAV-based reconnaissance mobility models, Drones 7 (3) (2023) http://dx.doi.org/10.3390/drones7030161.

[9] H. Chao, Y. Cao, Y. Chen, Autopilots for small fixed-wing unmanned air vehicles: A survey, in: 2007 International Conference on Mechatronics and Automation, 2007, pp. 3144–3149, http://dx.doi.org/10.1109/ICMA.2007.4304064.

[10] B.S. Morse, C.H. Engh, M.A. Goodrich, UAV video coverage quality maps and prioritized indexing for wilderness search and rescue, in: 2010 5th ACM/IEEE International Conference on Human-Robot Interaction, HRI, 2010, pp. 227–234, http://dx.doi.org/10.1109/HRI.2010.5453190.

[11] K. Kashyap, A. Agrawal, FANET: Survey on design challenges, application scenario and communication protocols, IJRAR (2018).

[12] E. Yanmaz, C. Costanzo, C. Bettstetter, W. Elmenreich, A discrete stochastic process for coverage analysis of autonomous UAV networks, in: 2010 IEEE Globecom Workshops, 2010, pp. 1777–1782, http://dx.doi.org/10.1109/GLOCOMW.2010.5700247.

[13] C.H. Choi, H.J. Jang, S.G. Lim, H.C. Lim, S.H. Cho, I. Gaponov, Automatic wireless drone charging station creating essential environment for continuous drone operation, in: 2016 International Conference on Control, Automation and Information Sciences, ICCAIS, 2016, pp. 132–136, http://dx.doi.org/10.1109/ICCAIS.2016.7822448.

[14] A. Purohit, F. Mokaya, P. Zhang, Collaborative indoor sensing with the sensorfly aerial sensor network, in: Proceedings of the 11th International Conference on Information Processing in Sensor Networks, IPSN '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 145–146, http://dx.doi.org/10.1145/2185677.2185720.

[15] C.H. Choi, H.J. Jang, S.G. Lim, H.C. Lim, S.H. Cho, I. Gaponov, Automatic wireless drone charging station creating essential environment for continuous drone operation, in: 2016 International Conference on Control, Automation and Information Sciences, ICCAIS, 2016, pp. 132–136, http://dx.doi.org/10.1109/ICCAIS.2016.7822448.

[16] G. Mao, S. Drake, B.D.O. Anderson, Design of an extended Kalman filter for UAV localization, in: 2007 Information, Decision and Control, 2007, pp. 224–229, http://dx.doi.org/10.1109/IDC.2007.374554.

[17] D. Jung, P. Tsiotras, Inertial attitude and position reference system development for a small UAV, in: AIAA Infotech@Aerospace 2007 Conference and Exhibit, ARC, 2007, http://dx.doi.org/10.2514/6.2007-2763.

[18] M. Quaritsch, K. Kruggl, D. Wischounig-Strucl, S. Bhattacharya, M. Shah, B. Rinner, Networked UAVs as aerial sensor network for disaster management applications, e & i Elektrotech. Inf.tech. 127 (3) (2010) 56–63, http://dx.doi.org/10.1007/s00502-010-0717-2.

[19] E.W. Frew, T.X. Brown, Networking issues for small unmanned aircraft systems, J. Intell. Robot. Syst. 54 (1) (2009) 21–37.

[20] J. Baillieul, P.J. Antsaklis, Control and communication challenges in networked real-time systems, Proc. IEEE 95 (1) (2007) 9–28, http://dx.doi.org/10.1109/JPROC.2006.887290.

[21] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming protocol-independent packet processors, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95, http://dx.doi.org/10.1145/2656877.2656890.

[22] X. Li, J. Yan, LEPR: Link stability estimation-based preemptive routing protocol for flying ad hoc networks, in: 2017 IEEE Symposium on Computers and Communications, ISCC, 2017, pp. 1079–1084, http://dx.doi.org/10.1109/ISCC.2017.8024669.

[23] V. Sharma, R. Kumar, N. Kumar, DPTR: Distributed priority tree-based routing protocol for FANETs, Comput. Commun. 122 (2018) 129–151, http://dx.doi.org/10.1016/j.comcom.2018.03.002.

[24] A. Khan, F. Aftab, Z. Zhang, Self-organization based clustering scheme for FANETs using glowworm swarm optimization, Phys. Commun. 36 (2019) 100769.

[25] J. Blumenthal, R. Grossmann, F. Golatowski, D. Timmermann, Weighted centroid localization in zigbee-based sensor networks, in: 2007 IEEE International Symposium on Intelligent Signal Processing, 2007, pp. 1–6, http://dx.doi.org/10.1109/WISP.2007.4447528.

[26] D. Scano, F. Paolucci, K. Kondepu, A. Sgambelluri, L. Valcarenghi, F. Cugini, Extending P4 in-band telemetry to user equipment for latency- and localization-aware autonomous networking with AI forecasting, J. Opt. Commun. Netw. 13 (9) (2021) D103–D114, http://dx.doi.org/10.1364/JOCN.425891.

[27] E.W. Frew, T.X. Brown, Networking issues for small unmanned aircraft systems, J. Intell. Robot. Syst. 54 (1) (2009) 21–37.

[28] A. Hussain, T. Hussain, F. Faisal, I. Ali, I. Khalil, S. Nazir, H.U. Khan, DLSA: Delay and link stability aware routing protocol for flying ad-hoc networks (FANETs), Wirel. Pers. Commun. 121 (4) (2021) 2609–2634, http://dx.doi.org/10.1007/s11277-021-08839-9.

[29] J. Wang, Y. Cao, B. Li, S. Lee, J.-U. Kim, A glowworm swarm optimization based clustering algorithm with mobile sink support for wireless sensor networks, J. Internet Technol. 16 (5) (2015) 825–832, http://dx.doi.org/10.6138/JIT.2015.16.5.20140918.

[30] H.J. Na, S.-J. Yoo, PSO-based dynamic UAV positioning algorithm for sensing information acquisition in wireless sensor networks, IEEE Access 7 (2019) 77499–77513, http://dx.doi.org/10.1109/ACCESS.2019.2922203.

[31] F. Khelifi, A. Bradai, K. Singh, M. Atri, Localization and energy-efficient data routing for unmanned aerial vehicles: Fuzzy-logic-based approach, IEEE Commun. Mag. 56 (4) (2018) 129–133, http://dx.doi.org/10.1109/MCOM.2018.1700453.

[32] E.A. Tuli, M. Golam, D.-S. Kim, J.-M. Lee, Performance enhancement of optimized link state routing protocol by parameter configuration for UANET, Drones 6 (1) (2022) http://dx.doi.org/10.3390/drones6010022.

[33] D. Scano, A. Giorgetti, F. Paolucci, A. Sgambelluri, J. Chammanara, J. Rothman, M. Al-Bado, E. Marx, S. Ahearne, F. Cugini, Enabling P4 network telemetry in edge micro data centers with kubernetes orchestration, IEEE Access 11 (2023) 22637–22653, http://dx.doi.org/10.1109/ACCESS.2023.3249105.

[34] Behavioral model version 2 (BMV2), https://github.com/p4lang/behavioral-model.

[35] MininetWiFi, 2024, https://mininet-wifi.github.io/. (Accessed: January, 2024).

[36] D. Uomo, A. Sgambelluri, P. Castoldi, E. De Paoli, F. Paolucci, F. Cugini, Failure prediction in software defined flying ad-hoc network, in: Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, 2023, pp. 355–357.

[37] RabbitMQ, 2024, https://www.rabbitmq.com/. (Accessed: January, 2024).

[38] Erlang, 2024, https://www.erlang.org/. (Accessed: January, 2024).

[39] P4Runtime-shell, 2024, https://github.com/p4lang/p4runtime-shell/. (Accessed: January, 2024).

[40] P. Castoldi, A. Sgambelluri, L. Ismail, F. Paolucci, F. Cugini, D. Bowden, Network programmability for smart factory mobile robotics: the SmartEdge project approach, in: 2023 23rd International Conference on Transparent Optical Networks, ICTON, 2023, pp. 1–5, http://dx.doi.org/10.1109/ICTON59386.2023.10207553.

[41] Flask, 2024, https://flask.palletsprojects.com/en/3.0.x/. (Accessed: January, 2024).

[42] F. Paolucci, F. Cugini, P. Castoldi, T. Osiński, Enhancing 5G SDN/NFV edge with P4 data plane programmability, IEEE Netw. 35 (3) (2021) 154–160, http://dx.doi.org/10.1109/MNET.021.1900599.

[43] Spirent, 2024, https://www.spirent.com/assets/u/spirent_n4u_chassis_datasheet. (Accessed: January, 2024).

Layal Ismail received her B.S. in information and communication technology engineering from University of Tartous (2013), Tartous, Syria. She received her M.S in industrial automation from University of Tartous (2020) , discussing a research thesis on wireless data acquisition system for control purposes. In 2023 she got a Research Training Grant at CNIT, Pisa, Italy. Currently, she is Ph.D. student at Scuola Superiore Sant'Anna . Her research interests are software defined networking and wireless networks.

Domenico Uomo received his M.Sc. in computer science and engineering from the University of the Studies of Napoli Federico II (2020) with full marks, discussing a research thesis on Real-Time virtualization for railways signalling as a service where he has deepened the studies of safety–critical and real-time architectures. The same year he started working as a data engineer on highly sensitive data. In 2022 he got a scholarship with Scuola Superiore Sant'Anna di Pisa. His research topics involved safety assessment of critical networks and Artificial Intelligence.

Andrea Sgambelluri is an Assistant Professor at the Scuola Superiore Sant'Anna of Pisa, Italy, since 2019. He published around one hundred papers (source Google Scholar, May 2021) in International Journals and Conference Proceedings. In March 2015 he won the grand prize at 2015 OFC Corning Outstanding Student Paper Competition with the paper "First Demonstration of SDN-based Segment Routing in Multi-layer Networks". His main research interests are in the field of control plane techniques for both packet and optical networks, including Software Defined Networking (SDN) protocol extensions, network reliability, industrial Ethernet, switching, segment routing application, YANG/NETCONF solutions for the dynamic management, telemetry, (re)programming and monitoring of optical devices.

Faris Alhamed received his B.S in Telecommunication and Electronics Engineering in 2017 from Tishreen University, Latakia, Syria. and in 2021 he also received a joint Master's Degree form Aston University, Birmingham, UK, and Scuola Superiore Sant'Anna, Pisa, Italy, in Photonic Integrated Circuits, Sensors and NETworks - PIXNET. He is a currently a Ph.D. student at Scuola Superiore Sant'Anna expected to graduate by the end of 2024. His research focuses on data-programmability and its applications in switches and SmartNICs.

Francesco Paolucci (M) received the Laurea degree in telecommunications engineering from the University of Pisa in 2002, and the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, in 2009. In 2008 he was granted a research Merit Scholarship at the Istitut National de le Recherche Scientifique (INRS), Montreal, Quebec, Canada. Currently, he is Head of Research at CNIT, Pisa Italy. His main research interests are in the field of network control plane, orchestration for edge/cloud platforms, traffic engineering, network disaggregation, advanced network telemetry, SDN/P4 data plane programmability. He has been involved in many European research projects on next generation control networking (BRAINE, DESIRE6G, SMARTEDGE, CLEVER, SEASON). He is co-author of 3 IETF Internet Drafts, more than 200 publications in international journals, conference proceedings and book chapters, and filed 4 international patents. He is Associate Editor of the IEEE/OSA Journal of Optical Communications and Networking (JOCN).