SmartEdge

Semantic Low-code Programming Tools for Edge Intelligence

This project is supported by the European Union's Horizon RIA research and innovation programme under grant agreement No. 101092908

Deliverable D4.1

Design of Dynamic and Secure Swarm Networking

Editor N. Zilberman (UOXF)

Contributors H. Chen, D. Ding, C. Zheng (UOXF), R. Abu Bakar, F. Cugini, L. Ismaeel, M. Mtiri, F. Paolucci, D. Uomo (CNIT), J. J. Vegas Olmos, F. Alhamed (NVIDIA) P. Cudré-Mauroux, A. Lerner (FRIB), D.M. Nguyen, D. Le Phuoc, A. Le Tuan (TUB)

Version 1.0

Date 31st January 2024

Distribution PUBLIC (PU)

DISCLAIMER

REVISION HISTORY

Revision	*Date*	*Responsible*	*Comment*
0.1	*November 5, 2023*	*N. Zilberman (UOXF)*	*ToC*
0.2	*January 10, 2024*	*N. Zilberman (UOXF)*	*Partner contributions*
1.0	*January 31, 2024*	*N. Zilberman (UOXF)*	*Final version*

LIST OF AUTHORS

Partner	*Name Surname*
UOXF	*Noa Zilberman, Damu Ding, Changgang Zheng, Hongyi Chen*
CNIT	*Filippo Cugini, Mohamed Mtiri, Francesco Paolucci, Layal Ismaeel, Domenico Uomo, Rana Abu Bakar*
NVIDIA	*Juan Josè Vegas Olmos, Faris Alhamed*
TUB	*Duc Manh Nguyen, Danh Le Phuoc,* Anh Le Tuan
FRIB	*Alberto Lerner, Philippe Cudré-Mauroux*

GLOSSARY

Acronym	***Description***
ACC	accuracy
ACK	Acknowledgment
ACL	Access Control List
AP	Access Point
AUC	Area Under the ROC Curve
BMV2	Behavioral model version 2
BT	British Telecom
CDF	Cumulative distribution function
CNN	Convolutional Neural Networks
CP	Control Plane
CPU	Central processing unit
DDoS	distributed denial-of-service
DDS	DDoS Detection System
DHCP	Dynamic Host Configuration Protocol
DL	Deep Learning
DP	Differential Privacy
DPDK	Data Plane Development Kit
DPI	Deep packet inspection
DPU	Data Processing Unit
DT	Decision Tree
FL	Federated Learning
FNR	false negative rate
FTG	Flow Traffic Graph
GNN	Graph neural network
GRU	Gated Recurrent Unit
ICMP	Internet Control Message Protocol
ID	Identifier
IDS	Intrusion detection and prevention system
ILP	integer linear programming
INT	In-Band Telemetry
IoT	Internet of Things
ITS	Intelligent Transportation System
ISP	Internet service provider
KM	k-means
MAC	Media Access Control
ML	Machine Learning
M/A	match-action
NB	Naïve Bayes
NIC	Network Interface Card
NN	Neural Network
PDP	programmable data plane
P4	Programming Protocol-independent Packet Processors
QoS	Quality of Service
RF	Random Forest
RSU	Road-side unit

RTT	Round Trip Time
SQL	Structured query language
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UC	Use Case
TPR	true positive rate
UDP	User Datagram Protocol
URLLC	ultra-reliable low-latency communications
UUID	Universal Unique Identifier
VLAN	Virtual local area network
VNID	VXLAN Network Identifier
VXLAN	Virtual Extensible LAN
WP	Work Package
XGB	XGBoost

EXECUTIVE SUMMARY

Deliverable 4.1 is the first iteration of the Dynamic Swarm Networking technical Work Package. The Work Package combines three tasks: (1) Automatic discovery and dynamic network swarm formation (2) Embedded network security and isolation and (3) Hardware accelerated in-network operations.

These three intertwined tasks lead the way to enabling the networking layer of the SmartEdge project and for providing high performance and low latency operation by providing innovative network-programmability based solutions.

By considering the requirements set in SmartEdge's global architecture, and building upon the SmartEdge use cases, an initial architecture of the Work Package is proposed. The architecture covers how SmartEdge swarms are formed and managed, what are the network security considerations and how to implement active security techniques within the network layer, and how programmable network devices can be used to accelerate SmartEdge operation.

SmartEdge aims to demonstrate how adaptable, but resilient, swarms can operate autonomously at the edge offering innovative practical solutions in the daily life of people in specific set of representative Use Cases.

In the following sections we identify building blocks in the architecture of the Work Package, and describe the planned design of each component, as well as present preliminary results.

By enabling these building blocks, we aim to provide valuable insights into the development of a dynamic, scalable, secure and high-performance solution for decentralized edge intelligence in swarms, ultimately contributing to the project's objectives. The solution provided by the Work Package will enable SmartEdge's use cases and support their operation. These will be validated by the SmartEdge Integration work package as part of the Use Cases evaluation environment.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 OVERVIEW

The Dynamic and Secure Swarm Networking Work Package, also known as Work Package 4 (WP4), covers aspects of the SmartEdge project that are responsible for the communication between the SmartEdge system components and for development of swarm intelligence within the network.

The Work Package covers three networking related aspects: automatic discovery and dynamic network swarm formation, embedded network security and isolation and hardware accelerated in-network operations. The three parts, translated into three development tasks, are intertwined and closely integrated, with elements developed under one task being used or accelerated under another task.

To position the Work Package, it is located above the physical layer of the network, and below the software application layer (or middle layer). The Work Package uses existing physical layer communications protocols (e.g., 4G/5G, wireless protocols) as-is, while building upon emerging network programmability to introduce innovation from the network layer (layer 3 in the OSI model) and above. By offloading operations previously done on servers or in the cloud, it aims to increase performance, reduce latency and introduce novel functionality.

1.2 OBJECTIVES

WP4 has three objectives:

1. Automatic Discovery and Dynamic Network Swarm formation in near real time – providing the functionality required to create a swarm of SmartEdge devices, including adding, maintaining and removing nodes. This functionality will happen during operation time based on communication between devices, and is not based on a pre-set list of devices.
2. Embedded network security and isolation – providing network security for nodes that are part of the swarm, including protection against common network attacks (e.g., DDoS) and detection and mitigation of anomalies and emerging attacks. Note this does not cover end-to-end security (e.g., application level). The tasks also isolate swarm communication from nodes outside a swarm.
3. Hardware-accelerated in-network operations for context-aware networking – providing acceleration to SmartEdge functionality by offloading it to run within the network. This functionality will range from network security to acceleration of stream processing, while building upon emerging network-hardware technologies such as DPUs.

The scope of WP4 is illustrated in Figure 1.1. As the figure shows, the functionality provided by the Work Package is distributed across the SmartEdge network, providing intelligence within the network for swarm formation, network security and context-aware acceleration.

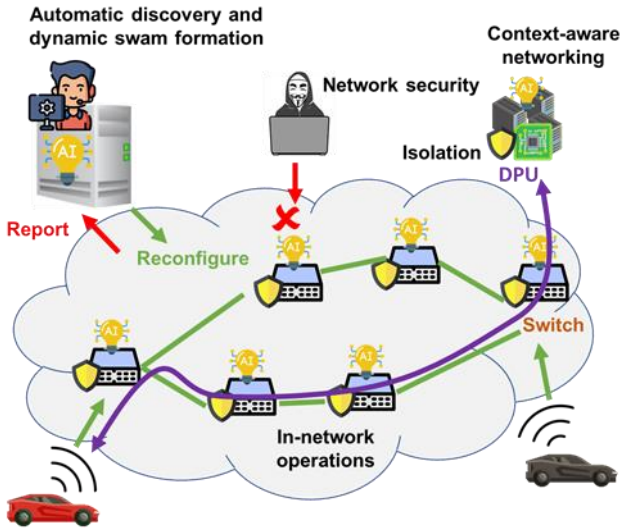


Figure 1.1 High Level View of Dynamic and Secure Swarm Networking

While Use Case 2 (UC-2) is depicted in the figure, the Work Package will support a range of SmartEdge Use Cases, with a focus on nodes dynamism, rapid response time and high performance.

1.3 REQUIREMENTS

The requirements for WP4 are set by the over-all architecture of the SmartEdge project as defined by WP2. The detailed list of requirements, their related Use Cases and the priority set by the Use Case owners is provided below.

1.3.1 Swarm Management

ID	Partner Short Name	Related Use Case(s)	Priority
SW-003	A swarm coordinator is a swarm smart-node that must support the advanced functionality necessary to form and maintain a swarm. For example, the ability to onboard or offboard nodes to and from the swarm, or in UC5 the provide medical staff with such node that support coordination, holistic management, and monitoring of a group of patients represented by their swarm nodes.		
	DELL, IMC	UC-3, UC-5	High
SW-004	A manifest swarm must consist of one or more swarm smart-nodes and at least one of those smart-nodes must be a swarm coordinator. Typically, the swarm will consist of two or more swarm smart-nodes, but during swarm formation or reformation there may only be one seed swarm smart-node that will enlist other nodes into the swarm, to perform the desired behaviour.		
	DELL, IMC	UC-3	High
SW-007	The swarm coordinator(s) must maintain the state of the swarm. For example, which nodes are part of the swarm.		
	DELL	UC-2, UC-3	High
SW-008	Swarm smart-nodes must reach a consensus on any action that affects the swarm. For example, the swarm smart-nodes must agree on the offboarding of a new node from the swarm. If a smart-node (A) requests to leave the swarm, but another smart-node (B) is relying on it to perform some operation, then smart-		

	node (B) must have the opportunity to request smart-node (A) to remain in the swarm. There may be exceptions to this rule, e.g., if all swarm coordinators agree to eject a node (C) from the swarm, then node (C) cannot block this action.		
	DELL	UC-3	Medium
SW-011	A swarm smart-node (A) must have the right to leave a swarm if it chooses, but another swarm swarm-node (B), can request it to remain if it needs the assistance of smart-node (A). Smart-node (A) must then decide if it stays or leaves the swarm.		
	DELL, IMC	UC-3, UC-5	Medium
SW-012	A swarm node may belong to more than one swarm at the same time. For example, when a node is providing a sensor stream that could be of use to multiple swarms. However, there must be no crossover of information between the swarms, as this could lead to a possible attack vector, or at the very least a leakage of swarm information. A shared node should never be the target of actuator commands, as this could lead to conflicting actions between swarms.		
	DELL, IMC	UC-3, UC-4, UC5	High
SW-013	A swarm coordinator should only belong to one swarm at a time. There may be exceptions where it is desirable, or even required, for a swarm smart-node to orchestrate more than one swarm. For example, when two swarms must collaborate to achieve a common objective, but this is likely to be the exception rather than the rule.		
	DELL, IMC	UC-3, UC-5	Medium
SW-014	A protocol must exist to onboard a smart-node into a swarm.		
	DELL, IMC	UC-3, UC-5	High
SW-015	A protocol must exist for a smart-node to request to join a swarm.		
	DELL, IMC	UC-3, UC-5	High
SW-016	A protocol should exist for a swarm to broadcast a request for a smart-node with a specified set of capabilities to join the swarm. Based on the responses received, the swarm will select a candidate and onboard the smart-node. This feature is particularly important for when a swarm is missing a critical smart-node to enable an operation to be performed. It may be useful for the responding smart-nodes to offer a time limited option to join the swarm. When the option expires there is no obligation for them to join the swarm.		
	DELL, IMC	UC-3, UC-5	Medium
SW-017	There may be cases where a device (A) in a swarm may object to a device (B) joining the swarm. For example, if device (A) has prior knowledge of the untrustworthiness of device (B). In this case device (B) is not allowed to join the swarm. There may be some limits or time constraints on device (A) objecting to device (B).		
	DELL	UC-3	Low
SW-019	A swarm must handle units disappearing without proper protocol due to network failure or similar external factors.		
	Aalto, IMC	UC-2, UC-3, UC-5	High
SW-020	Standard protocol for accepting new smart-nodes to swarm has to be defined for SmartEdge capable vehicles.		
	Aalto, IMC	UC-2	High

SW-022	The swarm smart-nodes and the swarm must have a universally unique identifier (e.g., UUID), encoded in the form of a URI. In many use cases when assigning a URI to a smart-node it is not possible to have a single controlling agency to issue the URI or provide an Internet (or even intranet) resolvable URL. Whilst UUIDs have no implicit meaning, but simply strings of hexadecimal characters, they do (for all intents and purposes) guarantee that the resource is uniquely identified.		
	DELL	UC-3	High
SW-027	A cloud or brown field system can be part of a swarm and could be regarded as a type of swarm node with its own specific capabilities. This may be necessary if the swarm needs to communicate with legacy systems or use their resources. For example, to transfer data for long-term storage or deep data analysis.		
	DELL	UC-3, UC-4	High
SW-028	To keep latency low the Cloud should not take on the role of swarm coordinator or orchestrator. The Cloud should implement the SmartEdge stack but may be restricted to only certain swarm protocols. There may be challenges if the Cloud has too much control over the swarm.		
	DELL	UC-3	High

1.3.2 Swarm Communication

ID	Partner Short Name	Related Use Case(s)	Priority
SC-002	SmartEdge must provide a procedure so that a Swarm can communicate events about its state to swarm participants that registered for certain swarm events.		
	FhG, BOSCH, IMC, DELL	UC-1, UC-3, UC-5	High
SC-003	It must be possible to register for application specific events that can be published by SmartEdge applications. The events must support application specific data if the application needs to communicate payload data.		
	FhG, IMC, DELL	UC-1, UC-3, UC-5	High
SC-006	Events should contain data about the creation source, time, and date.		
	FhG, BOSCH, IMC, DELL	UC-3, UC-5	High
SC-008	Internal swarm communications must be limited to only current nodes in the swarm, e.g., information about the swarm must not leak out of the swarm or be sent to nodes that have been offboarded from the swarm.		
	DELL, IMC	UC-3, UC-5	High
SC-012	A network node join message must be implemented for a swarm to request an external smart-node to join a swarm, or an external smart-node requesting to join a swarm. This would typically follow a network node characteristics message, where the swarm has identified a suitable smart-node and is requesting it to join the swarm. Alternatively, it could be used by a smart-node that has discovered a swarm and wishes to join it.		
	DELL, IMC	UC-3	High
SC-013	A network device leave message must be implemented for a swarm device to request to leave a swarm it is currently part of. This message must be broadcast to all other members of the swarm.		
	DELL	UC-3	High

SC-014	A network node remain message must be implemented for a swarm smart-node to request a leaving smart-node to remain in the swarm as it needs its services.		
	DELL, IMC	UC-3	Medium
SC-024	Smart-nodes in a swarm must regularly exchange heartbeat messages to other smart-nodes in the swarm. Rather than each smart-node communicating to each other smart-node in the swarm, smart-nodes may syndicate recent heartbeat messages they have received. In this way other smart-nodes, and in particular the swarm coordinator(s), can assess the cohesion of the swarm without having to constantly communicate with other members of the swarm.		
	DELL	UC-3	High
SC-025	When a swarm coordinator sends a heartbeat message, the message should contain a hash of its swarm state. Smart-nodes can cache the hashes from the swarm coordinator(s), to identify if the swarm state changes. If swarm coordinator (A) detects that swarm coordinator (B)'s state has changed, it will immediately initiate a swarm update to reconcile the two states.		
	DELL	UC-3	Medium
SC-026	Smart-node heartbeat messages should never be propagated outside of the swarm. Where a node is shared between several swarms or a Cloud is part of a swarm, they should not provide a bridge for heartbeat message.		
	DELL	UC-3	Medium

1.3.3 Hardware and Protocols

ID	**Partner Short Name**	**Related Use Case(s)**	**Priority**
HP - 002	Vehicles communicating via C-ITS/G5 devices and without SmartEdge SW stack should be able to connect to the Swarm as "dummy" devices.		
	Aalto	UC-2	High
HP-013	The SmartEdge components system must be able to run on ARM-based hardware and Ubuntu/Linux OS.		
	Aalto, DELL	UC-2, UC-3	High
HP-015	The SmartEdge components system must be able to run on x86-based hardware and Ubuntu/Linux OS.		
	DELL	UC-3	High

1.3.4 IDM and Policy & Security

ID	**Partner Short Name**	**Related Use Case(s)**	**Priority**
IDM-001	The visibility of events must be restricted to application, user, "swarm", or public level roles according to the system groups controlled by the swarm admin.		
	FhG, IMC, DELL	UC-2, UC-3, UC-5	High
IDM-002	A swarm can be created within a private secure environment where all nodes are belonging to the same owner. Nodes of other owners cannot join the swarm.		
	FhG, IMC, DELL	UC-1, UC-2, UC-3, UC-4, UC-5	High
IDM-004	Swarms must have the means to detect, monitor and mitigate cyber-attacks. This involves monitoring suspicious behaviour and providing the means to isolate compromised devices and block hostile network traffic.		

	ERCIM, IMC, DELL	UC-1, UC-2, UC-3, UC-4, UC-5	High
IDM-005	Swarms must report suspected cyber-attacks and system failures.		
	ERCIM, IMC, DELL	UC-1, UC-2, UC-3, UC-4, UC-5	High
IDM-006	Swarms should support the means to securely transfer critical participant roles to ensure robust operation of the swarm as a whole, for instance, when a device fails, or is compromised in a cyber-attack, or when the device is scheduled for hardware replacement or a lengthy software update.		
	ERCIM, IMC, DELL	UC-1, UC-2, UC-3, UC-4, UC-5	Medium (with WP5)
IDM-013	Encryption of data in transit using hardware acceleration. This is particularly important when data is being transported across a publicly accessible fabric, such as the Internet.		
	ERCIM	UC-2, UC-3	High
IDM-019	Communications emanating from the swarm should appear to come from the swarm and not a specific device, i.e., the internal structure of the swarm should not be ascertainable from its external communications.		
	IMC	UC-2, UC-5	High
IDM-022	The swarm, and particularly the swarm coordinator(s), should constantly monitor the behaviour of swarm nodes to look for aberrant behaviour, either due to malfunction or malicious intent. For example, a smart-node (A) may incorrectly syndicate to other smart-nodes in the swarm that it has received heartbeat messages from another smart-node (B), when it has not. Where practical, the swarm should try to corroborate swarm information from other smart-nodes, and isolate smart-nodes that are shown to be untrustworthy.		
	DELL	UC-2, UC-3	Medium

1.4 KPI

The Key Performance Indicators (KPI) associated with the project are the following:

K3.1: time required to perform automatic discovery and dynamic network swarm formation, reduced by 60% thanks to HW-accelerated in-network operations compared to regular processor-based execution.

K3.2: Increase performance by enabling in-network processing to reduce end-to-end latency (by 50% or more, assuming propagation delay is not the dominant component) and reduce variability.

K3.3: Provide isolation to data streams and a distributed security barrier for smart anomaly detection (80% or more of events detected and mitigated within the network, with 97% accuracy / F1 score >90%, detection provided in 1µs by ML algorithms (e.g., decision tree) optimized to run on Deep Processing Unit (DPU), >100x faster wrt. CPU baseline.

K3.4: Enable distributed operation of a large number of heterogeneous IoT devices and smart systems (1K devices or more handled by the DPU only, in-line with expected number of devices/vehicles to be considered in the SMARTEDGE scenarios) to achieve higher resilience.

1.5 STRUCTURE OF THE DOCUMENT

This document is organized as follows: Section 2 provides a high-level architecture of the solution developed in WP4. Section 3 introduces the solution for automatic discover and dynamic swarm formation. Section 4 focuses on the design for embedded network security and isolation. Section 5 discusses hardware accelerated in-network operations. Section 6 summarizes the document. Section 7 provides the list of references used in this document.

2 HIGH LEVEL ARCHITECTURE

WP4 is divided into three tasks, each line aligned with one of the objectives. The tasks are:
T4.1. Automatic Discovery and Dynamic Network Swarm formation in near real time
T4.2. Embedded network security and isolation
T4.3. Hardware-accelerated in-network operations for context-aware networking
The three tasks and their relations are illustrated in Figure 2.1.

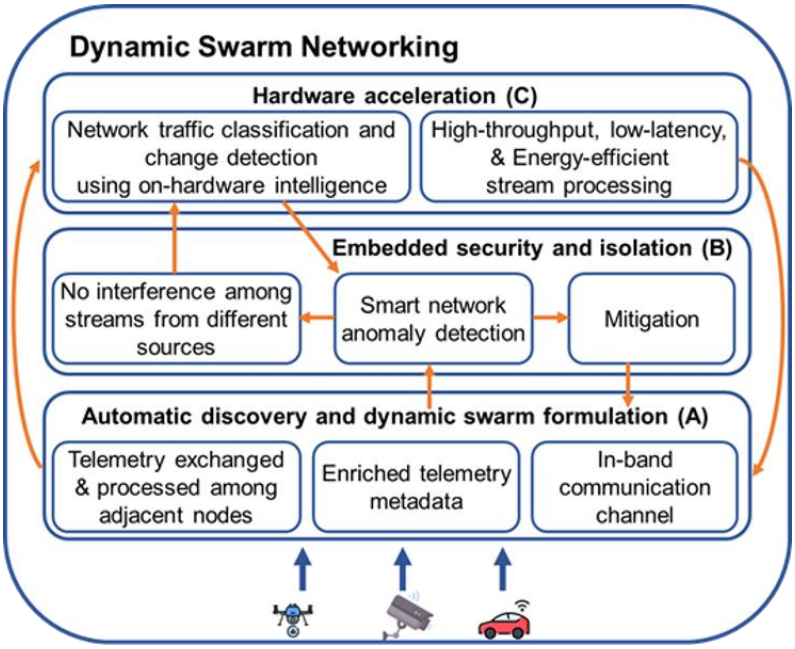


Figure 2.1: High Level Architecture of WP4

Task 4.1 (T4.1, marked A in the figure) is responsible for the formation of the swarm. It detects new nodes and decides if and how to add them to the swarm. To this end, the task uses an in-band communication channel and offers innovation in intelligent network telemetry and its processing.

Task 4.2 (T4.2, marked B in the figure) is responsible for embedded network security and isolation. The task innovates in moving network security intelligence into network devices, primarily using in-network machine learning (ML) for anomaly detection. The task is responsible to mitigate detected threats and to isolate traffic streams from different sources, so that nodes outside the swarm cannot interfere with its operation.

Task 4.3 (T4.3, marked C in the figure) is responsible to accelerate SmartEdge functionality using hardware-based acceleration, with a focus on the acceleration of stream processing and network security applications.

The three tasks are intertwined, with information exchanged between the different parts of the Work Package. T4.1 shares both swarm membership and status information with other tasks, allowing isolation of communication within the swarm by T4.2. Task 4.1 also shares telemetry information, used as features in the ML based anomaly detection of T4.2. T4.2, in turn, reports anomalous nodes behavior, used to isolate malfunctioning nodes and exclude malicious ones from the swarm by T4.1. T4.3 enhances the functionality of both tasks by providing hardware acceleration, improving performance toward KPI goals, while stream processing acceleration supports the overall functionality of SmartEdge and intersects with WP5.

2.1 PROGRAMMABLE NETWORK DEVICES AND P4

Software-defined networking (SDN) decouples the operational planes of network devices, primarily separating the data plane from the control plane. While SDN supports network programmability it assumed fixed or reconfigurable data plane functionality. The introduction of programmability into the data planes, in devices such as Intel Tofino and NVIDIA Spectrum 2, has enabled changing the functionality of the data plane by users. The programmability enables network applications to be executed in an in-network manner, meaning that the program is implemented and executed within the data plane.

Programming Protocol-Independent Packet Processors (P4) is a domain-specific language used by programmable network devices. It allows to customize packet parsing and forwarding while being protocol independent. Combined with runtime API it allows flexible interaction between the programmable data plane and control plane, thereby enabling coordination between the control logic and packet processing logic.

The SmartEdge project builds upon the P4 language and programmable network devices to enable innovation under WP4. The WP takes programmable network devices a step further than their original design, enabling intelligence, including machine learning, within the devices, providing new and innovative functionality.

3 AUTOMATIC DISCOVERY AND DYNAMIC NETWORK SWARM FORMATION

This section reports on the first design and preliminary implementation choices of task T4.1, i.e., the innovative SmartEdge solution providing automatic discovery and dynamic network swarm formation. In particular, this section complements D2.2 by focusing on the SmartEdge Network Layer highlighted in Figure 3.1. In order to provide a more concrete and detailed description of the adopted solution, Use Case 3 on smart factory mobile robotic is used as a leading example.

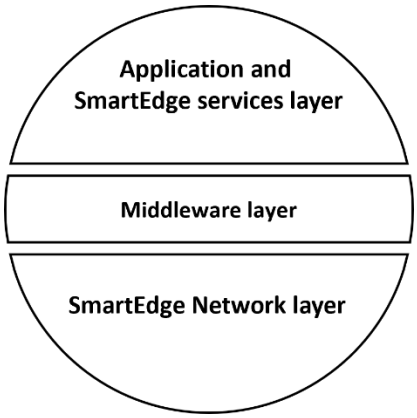


Figure 3.1: SmartEdge smart-node layers

3.1 P4 DATA PLANE PROGRAMMABILITY FOR SWARM FORMATION

Figure 3.2 depicts the strategic direction of automatic swarm formation, which involves providing data plane programming capability using P4 Language to all components in the factory, including mobile robots, racks, cameras, and IoT edge gateways. This allows for efficient processing of necessary communications between these components. For example, if cameras detect an unexpected object in their area and require urgent action, a P4-programmed protocol message can be sent to trigger appropriate responses. Similarly, if mobile robot 1 intends to load rack 1 with a specific trajectory, a message must be sent to other robots in the area to prevent collisions. However, mobile robot 1 may not have knowledge of which robots are present in the area. Hence, data plane programmability plays a crucial role in detecting the elements within the area by facilitating communication of relevant messages. Subsequently, the swarm intelligence system can analyze the received messages and inform the algorithm that controls the motion planning of the robots. The algorithm takes into consideration the position of the detected object and adjusts the motion plans of nearby robots to avoid collisions. This could involve slight alterations in the motion plans of one robot, slowing down of another, or even temporary halting of a robot for a brief period. The integration of swarm intelligence using P4 ensures that the components in the factory work cohesively to minimize the risk of collisions and facilitate effective communication among different elements. This can significantly enhance the productivity and efficiency of the factory by facilitating seamless data sharing and collaboration among devices. Ultimately, this collaborative approach can substantially reduce the occurrences of downtime, incidents, and other issues that may arise in industrial settings.

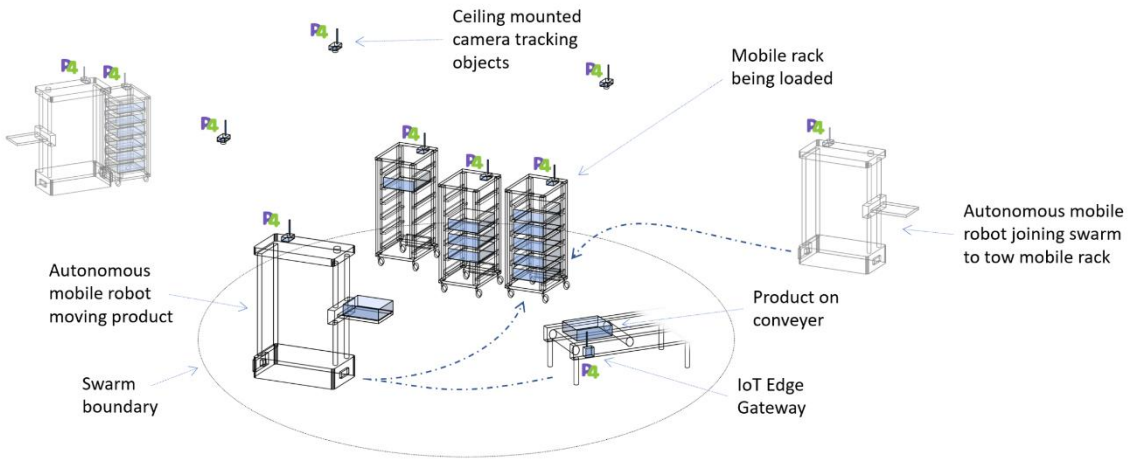


Figure 3.2: Smart factory utilizing P4 network layer swarm formation and management.

3.1.1 Swarm discovery and inclusion

In order to discover other devices, including their capabilities, and enlist them into the swarm intelligence system, the P4-based system may be programmed as follows:

1. Defining a custom discovery protocol using P4 that allows mobile robots, racks, cameras, and IoT edge gateways to exchange discovery messages periodically with each other to build a network topology. This protocol should define the format and semantics of the messages, including fields for device capabilities (such as whether the device is a robot, a camera, or an IoT edge gateway, and what sensors, actuators, or processing capabilities it has) and its location.
2. Programming the programmable switches using P4 to listen for discovery messages on the network and to extract the relevant metadata about device capabilities and locations. The switches should also update their forwarding tables based on the location information of the devices, in order to facilitate communication within the swarm intelligence system.
3. Using the extracted capability and location metadata to build a logical map of the factory floor that can be used by the swarm intelligence system to coordinate the activities of mobile robots, racks, cameras, and IoT edge gateways.

Implementing the protocol in P4 could involve defining the message headers and fields, as well as the processing logic in the P4 switch. For example, a P4 switch could use a combination of match-action tables and stateful processing to route discovery messages to their destination devices.

3.1.2 Controlled swarm exit

To allow mobile robots or other devices to exit the swarm intelligence system in a controlled manner the following steps are identified:

1. Using the previous discovery protocol to signal their intention to exit. This message will include fields for the identity of the device, any pending tasks, and other necessary contextual information.
2. Programming the wireless programmable switches in the network with P4 to listen for the custom control messages for exiting devices. The switches should remove the exiting devices from their forwarding tables to prevent any further communication between them and the other devices in the swarm intelligence system.

By using P4 to create this custom control message format, devices in the swarm intelligence system can communicate with each other effectively and efficiently. This helps ensure that all devices are aware of each other's intentions and can adapt to changes in the system in real-time. By including specific information, such as pending tasks, the system can ensure that any unfinished work is reassigned to other devices, preventing work duplication or lost work.

3.1.3 Unexpected swarm exit and recovery

To enable the swarm intelligence system to automatically recover if a swarm device is unexpectedly lost, the following steps can be taken:

1. Define a mechanism to detect when a device is lost, such as by using a telemetry system with IoT edge gateways and cameras that can report on the status of the devices within the network. This mechanism should identify when a device is no longer responding or is otherwise experiencing issues.
2. Program the programmable switches using P4 to update their forwarding tables and topology information to exclude the lost device's information automatically.
3. The switches should reroute traffic to other available devices in the system to ensure efficient communication.
4. Configure the swarm intelligence system to initiate automated routines to replace the lost device with a new device so that it can integrate into the swarm intelligence system as a backup.

By defining and implementing suitable mechanisms to detect lost devices and automated error remediation routines using the P4 language architecture, devices within the network can continue to work even in the presence of device loss. These implementations enable network-wide fault tolerance and resiliency, which can lead to a more reliable and efficient swarm intelligence system.

3.1.4 Cohesiveness and isolation

A P4-based swarm telemetry system (either in-band or out-of-band) is able to check the presence and status of the mobile robots or other devices within the swarm intelligence system, allowing it to maintain its cohesiveness. The monitoring of the swarm intelligence system continuously for signs of deviations from optimal performance to detect disconnections from mobile robots or other devices.

Key metadata metrics are in this case:

a. The geolocation information of the device, that may be used to forecast its trajectory and allowing anticipating future swarm operation
b. The device state, for example in terms of battery level or computational resources.

Swarm isolation, also covered by T4.2, can be achieved using a firewall or whitelist programmed in P4 that can filter out swarm-specific communication. Specific header fields identify swarm-specific communication (e.g., swarm ID, device ID, and message type).

Each outgoing message from a swarm device must contain these header fields. Additional match-action tables look for these header fields and take action based on the information they contain. A firewall function can be programmed to drop any messages that contain swarm-specific header fields if they are detected outside of the swarm, by relying on geolocation metadata. This approach can help protect the swarm from outside attacks and preserve the integrity of data exchanged within the swarm.

3.2 ARCHITECTURE OF SMARTEDGE NETWORK LAYER

Figure 3.3 depicts the key functional components of each swarm node related to the innovative SmartEdge swarm networking technology: the P4 switch and the Smart-node Network Control Plane (CP) Layer.

The P4 switch forwards all communication messages. It leverages tunneling technology, such as VXLAN, to guarantee isolation. In addition, it inserts/extracts in-band telemetry to provide/retrieve detailed and timely information about the swarm node state. Furthermore, it implements optional reliable communication functionalities, such as 1+1 protection.

The Smart-node Network Control Plane (CP) Layer manages the swarm formation from the networking perspective. It then enforces forwarding rules to the P4 switch.

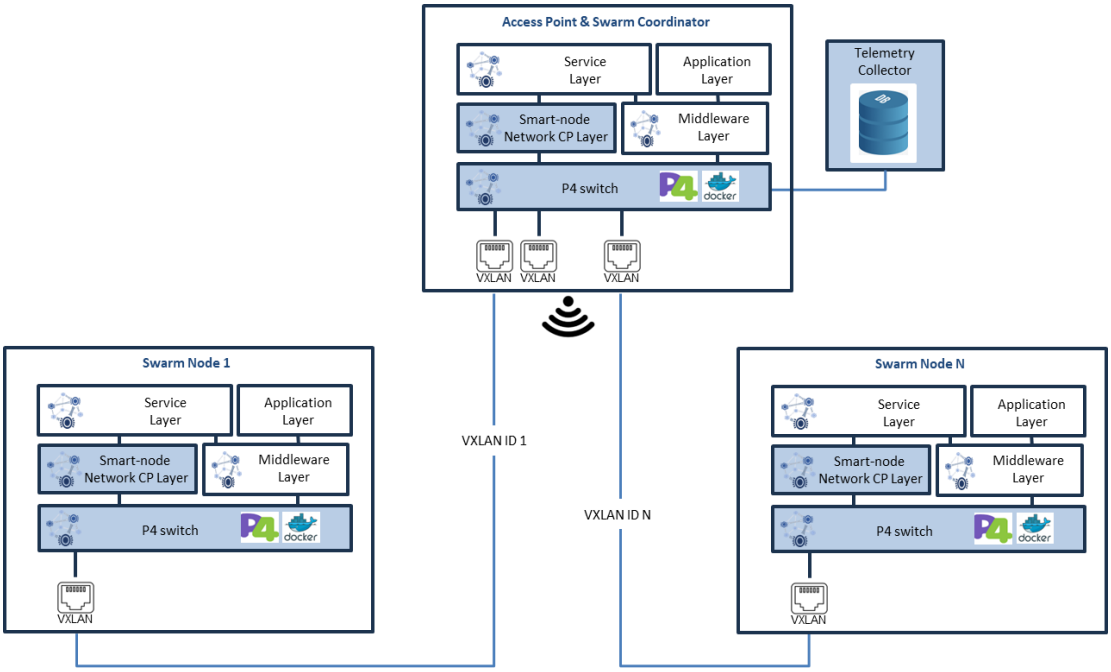


Figure 3.3: SmartEdge Reference scenario

3.3 INNOVATIVE TELEMETRY SYSTEM

3.3.1 Architecture of the telemetry system

SmartEdge designs an innovative in-band telemetry-driven technology for dynamic network swarm formation. In-Band Telemetry (INT), originally designed for data center operations, is here activated from all swarm nodes. The goal is to monitor the experienced performance also including the wireless segment (either wi-fi or 5G). This way, continuous and accurate end-to-end network awareness can be obtained, e.g., in terms of e.g. end-to-end latency or operational status.

Indeed, compared to traditional INT, additional metadata are enforced, such as localization and system/application parameters with low age of Information, thus providing system/application-level information that will trigger dynamic in-network functions. For example, INT augmented

with CPU load and localization information have the potential to trigger dynamic rerouting and offloading towards more suitable nearby edges just operating at network level only. Telemetry will also feed local control loops and AI algorithms for predictive networking against QoS degradations and dynamic operations for dynamic network swarm formation.

The SmartEdge packet header including INT has been designed with the following structure:

- Ethernet header (14bytes)
- IPv4 header (20bytes)
- UDP header (8bytes)
- Shim header (4bytes)
- INT header (8bytes)
- INT data (36 bytes)

The INT data field has been designed with a combination of traditional metadata and innovative ones (in Italic):

- Switch ID: 32bit
- Ingress Timestamp: 48bit
- Egress Timestamp: 48bit
- Hop Latency: 32bit
- Flow Id: 16bit
- *CPU: 16bit*
- *Latitude: 32bit*
- *Longitude: 32bit*
- *Received Signal Strength Indicator (RSSI): 32bit*

It is important to highlight that at this stage of the project no final decisions have been made and specific fields can still be added (e.g., status of the battery of mobile robots) or removed (RSSI, if not reliable in indoor complex environments) also considering specific use case needs or according to the evolution of the whole SmartEdge solution.

3.3.2 Preliminary implementation of the SmartEdge In-Band Telemetry solution

A preliminary implementation of the proposed INT solution is in progress. INT is inserted by each swarm node. At the access point, INT is extracted and forwarded to a telemetry collector. INT can be either removed by the access point or updated with its metadata. In the latter case, INT is removed by the destination swarm node which gets visibility of the entire end-to-end communication.

Figure 3.4 shows the Wireshark capture of a packet exchanged among swarm nodes and including a first implementation of the proposed SmartEdge INT solution

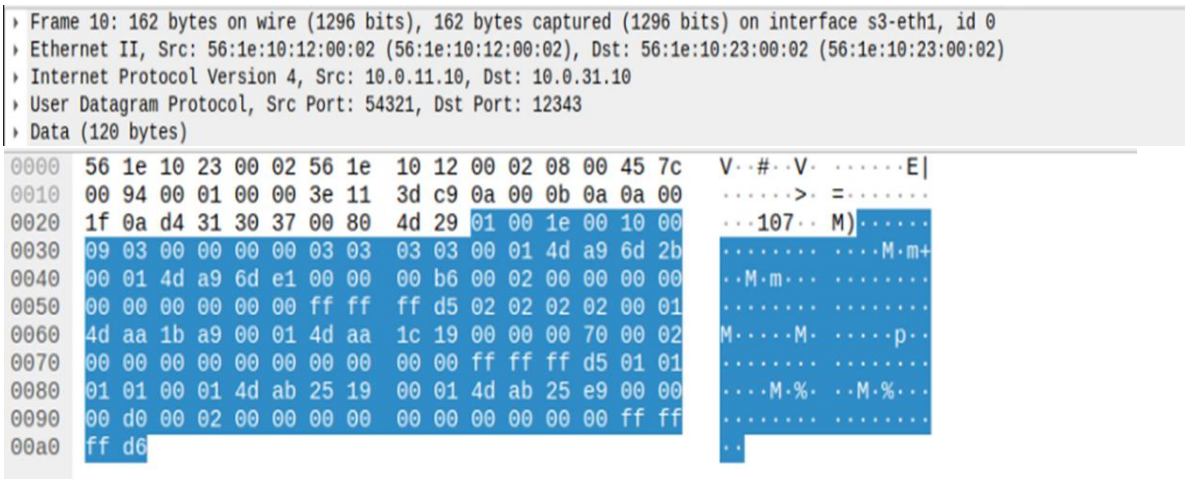


Figure 3.4: Wireshark capture of a packet exchanged among swarm nodes and including a preliminary implementation of the proposed SmartEdge telemetry system

3.3.3 Design and preliminary implementation of the SmartEdge Telemetry Collector

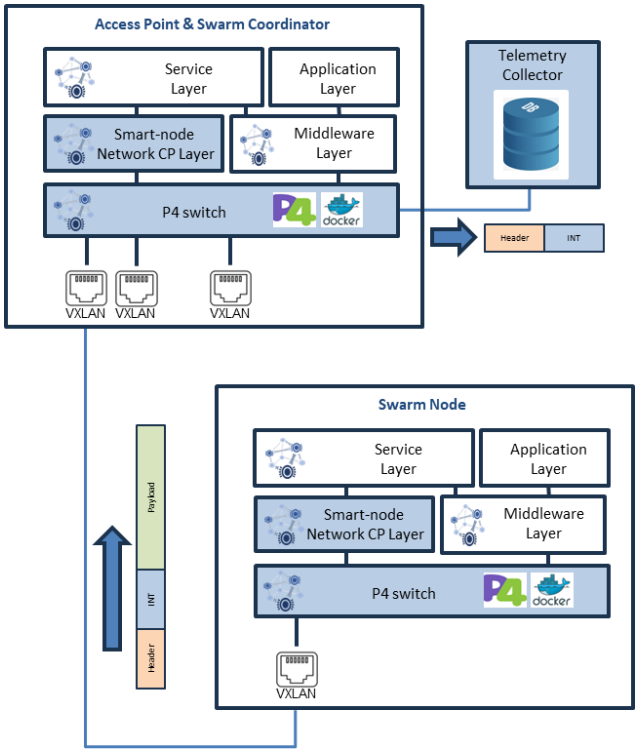


Figure 3.5: Exchange of Telemetry data towards the telemetry collector

The telemetry collector receives INT data from the Access Point. In particular, the P4 switch implemented at the access point once it receives packets data including INT, it removes payload information and forwards INT extra-header to the telemetry collector.

The collector stores the last data about a link in a hash table and links it with the timestamp of the last data received. The timestamp is useful for two reasons: (i) it allows to perform a filtering on the arrival rate of the packets; (ii) enables the removal of obsolete data.

A preliminary implementation of the telemetry collector is in progress. It is written in python3.8, using the library libpcap. It exploits a parallel thread that periodically checks on hash table for

expired data. The implementation enables the setting of the port on which the sniffer will listen the incoming traffic. Furthermore, it can be adapted to forward received information on a SQL database, NoSQL database or on an in-memory data structure for better performance. Specific focus is devoted to the innovative SmaryEdge INT metadata, such as usage of resources (e.g., CPU), location and RSSI.

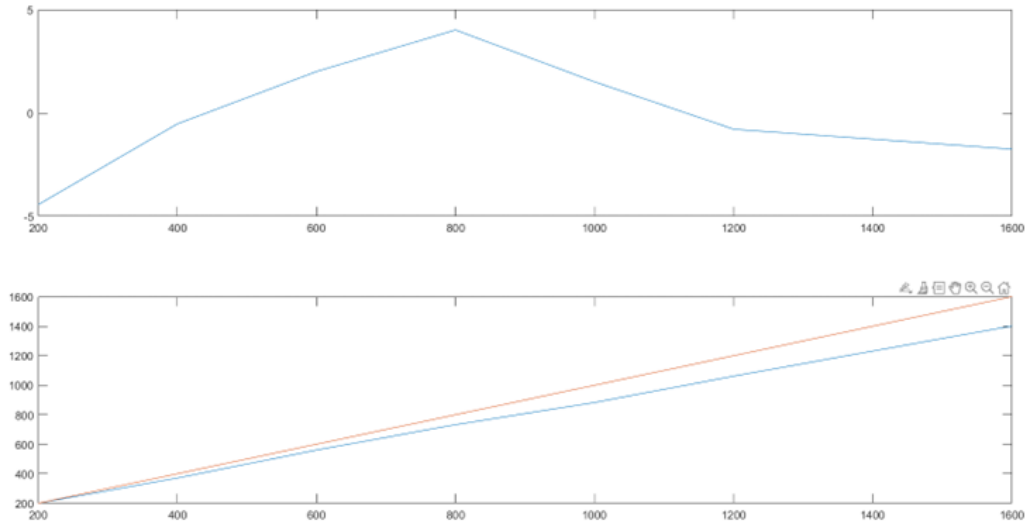


Figure 3.6: preliminary performance assessment of the Telemetry Collector

A preliminary performance test has been performed on the Telemetry Collector to check its capacity and, thus, discover the traffic load that it can handle.

Those tests have been carried out using a traffic generator that periodically increments the number of threads sending UDP packets. It starts with one thread sending 200 packets per second, and finishes with 8 threads, sending 200 packets per second each. Each packets contains 3 INT headers that need to be read.

The aim of this test is to check whether the Telemetry Collector is able to keep up with this traffic load, or, at a certain point, it will experience a drop in performance.

For this purpose, we estimate the output throughput of the collector and the variation of the latency expressed as the difference between the generated packet time and the processing time of the packets.

Every time a group of N threads finishes, before instantiating the group of N+1 threads, is calculate average latency and average throughput.

In Figure 3.6 it is possible to see the test results as the incoming traffic increases. The first figure does not show the absolute value of the latency since our aim is not to check the elapsed time (that could be affected by a lack of synchronization between client and server) but to check whether the performance gets worse as the traffic load increases. Thus we show the difference between the average latency time experimented by a threads group and the global average latency time.

As shown, the latency never diverges and remains in a range of [-5,+5] ms from the average time.

In the second graph, we plot the throughput of the collector (blue) and the ideal throughput of the threads. Even in this case, the throughput of the collector keeps increasing linearly as the throughput of the threads increases, and we can't see a knee on the curve of the capacity.

This test thus shows that this collector, given this hardware, can handle at least 1600 packets per second without saturation of capacity.

3.4 SWARM DISCOVERY

The SmartEdge swarm networking solution exploits Operating System features at the access point (AP) to detect wi-fi clients and to provide IP configuration via DHCP. It is assumed the AP has static IP and dedicated subnet.

The AP runs the SmartEdge P4 switch. Initially, no flow rules are configured. That is, no forwarding among swarm-nodes is allowed. Once swarm nodes become part of the swarm, traffic forwarding is enabled among swarm nodes. Furthermore, as anticipated, the AP forwards INT data to the Telemetry Collector.

The Smart-node Network Control Plane layer manages the swarm networking packets for swarm discovery, joining, heartbeat, and leave.

The joining operation has been designed as follows:

- The AP listens to swarm nodes as wireless (e.g., wi-fi) clients
- When a new node appears, the AP verifies it is authorized to join
- If authorized, the AP provides IP address through DHPC and stores info about all connected nodes (e.g., MAC, IP)
- The AP associates related VXLAN connectivity to the P4 switch virtual ports
- The AP receives and accepts/rejects JOIN requests (Swarm node ID, skills) and stores info about active swarm nodes (info shared with the middleware or embedded in P4)
- The AP enforces flow rules to the P4 switch to enable inter-swarm-node communication (any to any within the swarm)
- The Swarm-node Network Control Plane layer at the AP informs the Middleware layer about the new swarm member, providing node info such as SwarmID, IP and MAC addresses, skills, etc
- The AP distributes the list of active swarm nodes to all nodes in the swarm
    - upon changes (e.g., disconnection)
    - periodically (heartbeat)

From the swarm node perspective, the joining operation has been designed as follows:

- The swarm-node receives IP configuration via DHCP
- The swarm-node runs the SmartEdge P4 switch configured with a pre-established flow rule which indicates the Access Point as default gateway
- The P4 switch at the swarm node adds/removes INT metadata

The exit operation (controlled or uncontrolled) has been designed as follows:

- The AP removes the node from the list of active nodes
- The AP removes related VXLAN and flow entry

3.5 P4 SWITCH

A P4 switch is embedded in each swarm node to forward all communication messages. Swarm networking messages are passed to the Smart-node Network control plane layer. Once the swarm is operational, middleware/application packets are directly forwarded to the middleware layer.

The switch can be implemented as a software switch (e.g., as a docker container) or in hardware (i.e., ASIC). The former software solution is the one initially considered in SmartEdge for the implementation and validation of the proposed swarm networking solution. In particular, the selected switch platform for SmartEdge version 1 is BMV2 which provides full programmability and flexibility even if forwarding performance does not enable extremely high throughput.

A P4 program preliminarily supporting the aforementioned telemetry system as well as the proposed swarm discovery has been designed and preliminarily implemented, mainly to understand if the performance limitations affecting BMV2 platforms would determine critical constraints in the validation of swarm networking concepts.

The following preliminary performance measurements have been experimentally obtained (See Tables 3.1-3.4):

- Hop Latency
  - The contribution of the P4 switch in terms of hop latency is less than 0.6ms per node.
  - P4 switch introduces limited latency for processing INT compared to forwarding without INT.
- CPU Load
  - The processing of INT incurs a modest additional CPU load compared to the overall performance required by the entire P4 switch in a simple forwarding configuration.
  - This additional load is approximately 2% at maximum for low rates and 7% at maximum for high rates.
- Used Memory
  - The system does not require excessive amounts of RAM in any operating configuration.
  - In the absence of the P4 switch container, the P4 system consumes around 260MB of RAM.
  - The P4 switch container powered up with P4 code compiled without flow rules enabled, requires approximately additional 18M.
  - Activating the essential flow rules for forwarding and INT functionality incurs an additional RAM demand of about 10MB.

Data rate (packets/sec)	Hop Latency Node1 (µs)	Hop Latency Node2 (µs)
1	587	587
10	334	578
100	324	464
1000	535	589

Table 3.1: Latency contributed by P4 switch as a function of data rate (preliminary results)

Rate(p/s)	RTT without INT	RTT with INT
1	4.896	5.627
10	5.164	5.712
100	4.272	5.103
1000	4.039	5.830

Table 3.2: Effect of INT on round trip time (preliminary results)

	Without INT		Without INT	
Rate(p/s)	CPU node 1 (%)	CPU node 2 (%)	CPU node 1 (%)	CPU node 1 (%)
1	0.75	0.61	0.93	0.69
10	3.95	2.68	4.42	3.12
100	25.1	24.09	28.74	28.2
1000	50.9	45.43	55.88	52.43

Table 3.3: Effect of INT on CPU load when using P4 Switch (preliminary results)

	P4s off	P4s on	P4S 1p/s	P4S 10p/s	P4S 100p/s	P4S 1000p/s
Used RAM (KB)	260428	278768	285613	286056	288308	288514

Table 3.4: Amount of RAM consumed by P4 Switch under different packet rates (preliminary results)

4 EMBEDDED NETWORK SECURITY AND ISOLATION

Task T4.2 is responsible to provide network security and traffic isolation to SmartEdge nodes. The Task interacts with T4.1, which creates and manages the swarm, and with T4.3, which provides acceleration to functionality developed under this task.

Embedded network security in SmartEdge builds upon two contributions aspects:

- A novel P4 based layer of security that focuses on swarm requires and attends to threat vectors relevant to SmartEdge use cases.
- An innovative intelligent threat detection and mitigation solution that attends to emerging threats and anomalies.

The innovative solutions offered by SmartEdge are complemented by using industry-standard security mechanisms, such as the use of traffic encryption and authentication protocols. These are not developed by T4.2, and are only integrated in WP6.

4.1 PROGRAMMABLE ACCESS CONTROL FOR SWARM INTELLIGENCE

SmartEdge will use the P4 language to define a novel adjustable Access Control List (ACL) within the P4 switch to enable swarm intelligence functionalities. Each node will have a unique identifier (e.g., UUID) carried within the packet header, and only messages between nodes of the same swarm can be exchanged. Any packets sent from low-reputation nodes or of unauthorized protocol are dropped. Only the coordinator is capable of communicating with nodes outside the swarm, such as nodes asking to join or coordinators of other swarms. In this manner, SmartEdge will isolate network traffic streams belonging to different swarms.

To elaborate, the functionality supported for programmable network control will include the following:

- A P4 switch will drop any traffic sent from malicious or low reputation nodes. Such nodes will be identified through a blocklist controlled by the swarm coordinator, and distributed to all P4 switches in the swarm. A blocked node can be identified by its UUID or an identifier relevant to the use case, such as an IP address.
- Only approved protocols will be allowed within the swarms. Any traffic that is not of an allowed set of protocols will be dropped.
- VXLAN will be used to provide traffic isolation between swarms, with each swarm using a different VNID.
- In addition to VXLAN, for each incoming message the Swarm ID of both source and destination nodes will be checked, as well as the role of the nodes. A packet will be forwarded only if both source and destination are part of the same swarm, OR if one of the nodes is the coordinator (allowed to communicate with nodes outside the swarm).
- Nodes that asked to join the swarm but have not been onboarded yet by the coordinator will be allowed to communicate only with the coordinator, and only under constrained conditions (e.g., rate limiting to avoid DoS attack on the coordinator).

The access control functionality described above is programmable and can be easily extended by modifying the P4 switch program or by installing new rules during runtime.

4.2 ML-DRIVEN IN-NETWORK DEFENSE

Network threats can rise unexpectedly, and given SmartEdge's end-to-end low latency millisecond scale requirements, threats can spread faster than cloud-based security analytics can respond. This is particularly pressing in time-critical applications such set by Use Case 2 and Use Case 3.

Threats are also evolving stealthily with changing patterns. Attackers exploit can botnets or alter attack patterns to evade traditional security measures like firewalls, causing disruption of critical services or impact on network infrastructure. Thereby, swift defenses against evolving attacks are imperative to counteract potential widespread damage. Edge devices such as IoT gateways close to swarm nodes play a vital role in defending against emerging threats before they spread.

SmartEdge uses programmable data planes to offload inference to network devices, called in-network ML. It provides a novel approach to **rapidly detect attacks and enforce actions** in high-speed switches with an extension to other network devices like IoT gateways. In-network ML inference, a type of in-network computing, is the offloading of ML inference tasks from servers/cloud to the data plane of network devices [Zhe22, Zhe23-1, Zhe24]. Unlike classical practices of running ML training and inference solely on servers (potentially with GPUs) or in the cloud, this approach offloads inference processes to pipeline logics in programmable data plane within network devices (programmed in P4 language [Bos14]). Thus, ML inference runs concurrently with packet switching. This empowers a direct ML-based identification of malicious traffic and action enforcement within network devices, avoiding Round-Trip Times (RTT) for cloud-based analysis. Moreover, reconfigurable pipelines in programmable data planes also enable **flexible defense** in network devices. Traffic features can be flexibly parsed and in-network ML inference model can be reconfigured to flexibly respond to emerging threats, unlike classical solutions where ML models rely on predefined features from either monitoring protocols [Pas21] or proprietary toolkits

SmartEdge's innovation addresses several challenges in integrating in-network ML efficiently and flexibly to enable fast and continuous attack detection and mitigation within IoT gateway. First, the developed framework supports resource-constrained network devices. Second, there it enables continuous collection and analysis of traffic features to learn emerging threats. Third, it operates continuously to minimize maintenance costs and ensure uninterrupted service delivery. As reconfiguring in-network ML functions requires data plane program recompilation, potentially disrupting normal traffic in gateway during update, SmartEdge develops an innovative solution that protects from traffic interruption.

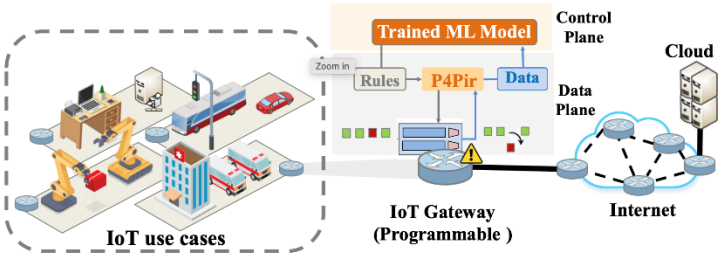


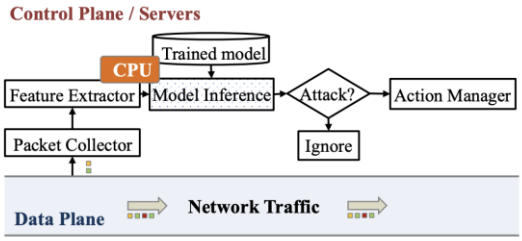
Figure 4.1: Deploying P4Pir within an IoT gateway in SmartEdhe scenario

The solution used by SmartEdge, dubbed P4Pir, will operate within an IoT gateway with programmable data plane on the network's edge. It establishes a seamless workflow for collecting traffic data, updating an in-network ML model, and employing data plane table rules

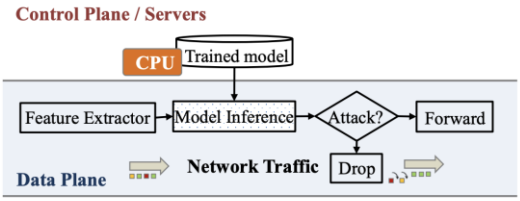
to detect/mitigate threats within the gateway. Specifically, the solution incorporates the following three features: *a) Swift in-network ML-based mitigation.* We integrate in-network ML inference into an IoT gateway, supporting fast threat defense as incoming traffic passes through the pipeline. *b) Continuous learning with proactive logging and labeling.* We introduce proactive traffic data logging and automated labeling to avoid manual intervention, enabling continuous retraining of the in-network inference model and learning on emerging attacks. *c) Seamless updates of in-network inference model.* We propose shadow table updates scheme to allow seamless in-network model updates to identify emerging attacks, avoiding function recompilation or forwarding disruptions in gateway.

4.3 Runtime Model Update For In-network Defence

In-network ML deployment can reduce inference latency by offloading the inference process from the controller to the data plane. Figure 4.2 (b) depicts how the innovative inference model can be offloaded to data plane together with traffic forwarding logic. By training a model offline and mapping the trained inference model to the data plane, malicious traffic can be labeled and mitigated directly based on in-network inference results. Despite accurate detection and swift mitigation, such an approach requires P4 program recompilation instead of runtime updates to achieve continuous learning, interrupting packet forwarding.



(a) Traditional ML deployment: inference on control plane/server.



(b) In-network ML deployment: inference on data plane.

Figure 4.2: (a) Traditional [Esk20] vs. (b) In-network ML deployment used in SmartEdge for traffic analysis and attack detection.

Continuous learning of ML models is necessary to address potential data drift, change of feature distribution or emerging malicious activities [Hin21]. SmartEdge integrates in-network ML inference in IoT gateway and incorporates techniques to support timely ML-based attack mitigation, as well as runtime updates for continuous and seamless model maintenance.

4.3.1 Framework

SmartEdge ML-based in-network defense framework engages the control plane for runtime model updates without interrupting the existing processing on the gateway target.

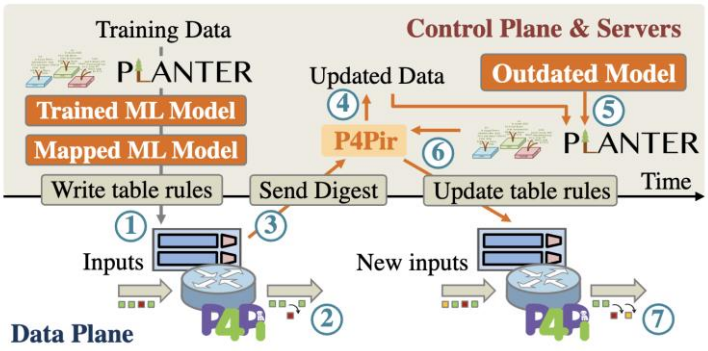


Figure 4.3: Block diagram of P4Pir workflow.

Figure 4.3 depicts P4Pir workflow. Step 1 shows a typical workflow of in-network ML-based detection [Zhe22], where a trained model is mapped to Match/Action table rules and P4 code to analyze the arriving traffic. On top of the existing in-network inference solution, as shown in step 2, SmartEdge identifies suspicious traffic from incoming traffic. While benign traffic is being forwarded, suspicious traffic will be dropped immediately and logged to the control plane by encapsulating the extracted features in a digest (as in step 3). Based on these digests, it retrains the current model to learn from the new traffic pattern (as in steps 4 & 5). A set of new rules will be generated to map the parameters of the new model. Updated rules are then inserted into the data plane and outdated rules are removed (as in step 6). With this updated setup, it is possible to learn from newly arriving traffic and mitigate abnormal traffic continuously (as shown in step 7). SmartEdge achieves continuous learning beyond existing frameworks by actively collecting traffic features and updating the model, shown as steps 2-7, to detect and mitigate emerging threats. Detailed components are illustrated as follows.

4.3.2 Implementation

The implementation of the solution is composed of three main components:

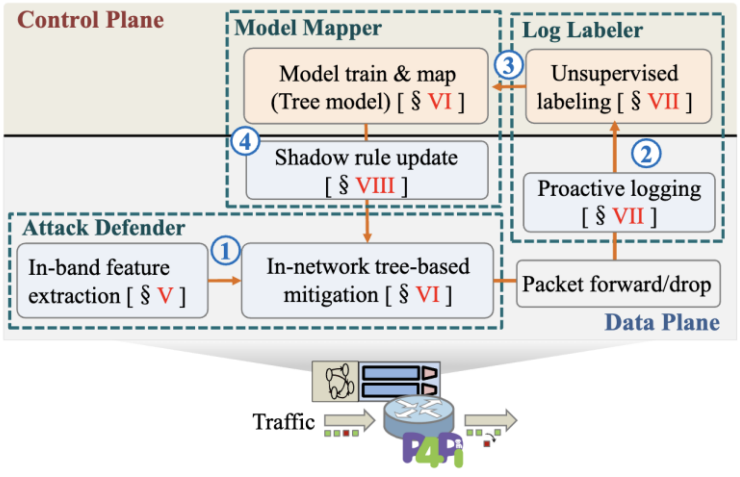


Figure 4.4: Components used in P4Pir workflow.

Attack defender performs ML-based attack detection and fast mitigation within the data plane. This defense mechanism involves two function blocks: in-band feature extraction and in-network tree-based attack mitigation. First, relevant traffic features are extracted as inputs to an in-network model. Next, an in-network inference tree model, which is mapped to data plane, identifies malicious traffic based on these features (Figure 4.4 step 1). As a result, benign traffic

is allowed to pass through, while detected malicious traffic is swiftly dropped, effectively mitigating the potential threat.

Log labeler automates a proactive logging process from the data plane to the control plane. It has two function blocks: proactive logging and unsupervised labeling. During proactive logging, extracted features are promptly logged to the control plane (Figure 4.4 step 2). At the control plane, these logs are labeled using the unsupervised algorithm. This approach allows for continuous learning of incoming traffic patterns.

Model mapper serves the purpose of training and mapping the in-network model (Figure 4.4 step 3). Its main function is to generate and seamlessly update Match-Action table entries, allowing for the runtime configuration of in-network inference. It involves two function blocks: model train & map, and shadow rule updates. The “Model Train & Map” block serves two purposes. Firstly, it retrains the model based on logs, allowing updates to adapt to new traffic patterns. Secondly, new table rules are generated to map the retrained model's parameters. Next, the “Shadow rule update” block inserts these new rules into the data plane using a shadow update method (as shown in Figure 4.4 step 4). This seamless configuration of in-network ML inference ensures uninterrupted traffic forwarding in the data plane.

With this framework design, our solution continuously learns from newly incoming traffic and quickly mitigates abnormal traffic.

4.3.3 Preliminary Results

The first prototype was developed on P4Pi [Zan22, Lak21], using Raspberry Pi 4 Model B with 8GB of RAM, and running P4Pi release v0.0.3 using bmv2 with v1model architecture. We also prototyped it on the Dell EMC Edge Gateway 5200 hardware device [Del23] used in UC-3, where the functionality is encapsulated within Docker and executed on the Dell gateway device. The code is implemented mainly in P4 and Python to provide data plane and control plane functionality, correspondingly. ML Model training and mapping for in-network inference function is based on scikit-learn [Ped11] and Planter [Zhe22].

For performance evaluation, it was connected to another Raspberry Pi and a desktop with an Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz and 64 GB RAM as client and server. Public IoT datasets CIC-IDS2017 [Sha18] and EDGE-IIOTSET [Fer22] are used for model training and evaluation. The captured traces in the dataset are replayed using tcpreplay. In these datasets, attacks are launched in different time slots using a week-worth of data. In order to examine the effectiveness of continuous learning, we assume an initial state in which the gateway learns only one attack on the first day. Then, another type of attack is replayed to simulate emerging attacks on the next day. After deployment, we compare its accuracy with a baseline's accuracy obtained using a static model. The static model is initialized by learning a single attack on the first day and maintained unchanged for incoming traffic replayed from other days. There are several metrics to evaluate detection performance. In SmartEdge, we use accuracy (ACC), F1 score, and true positive rate/false negative rate (TPR/FNR) as the metrics defined as below to evaluate detection performance.

Table 4.1 and Table 4.2 list accuracy and F1 score results of encode-based Decision Tree (DT) and Random Forest (RF) [Zhe22-2] with and without P4Pir's model updates. Static DT/RF models without P4Pir's updates are initialized as a baseline and trained with a single attack (“Base” columns in Table 4.1 and Table 4.2). To keep the results comparable, 5-tuple features {source IP, destination IP, source port, destination port, protocol} are used to train static DT/RF model

in both datasets. To select model parameters, we use grid search methods and use the parameters that give high accuracy. That is, in CIC-IDS2017, DT model is trained with a maximum depth of 5, and RF model is trained with 5 trees and maximum depth of 5. In EDGE-IIOTSET, DT model is trained with a maximum depth of 6, and RF model is trained with 6 trees and maximum depth of 6.

		SCAN	SCAN→DOS		SCAN→BOT*	
		Init	Base	P4Pir	Base	P4Pir
DT	ACC	0.987	0.604	0.932	0.900	0.923
	F1	0.984	0.568	0.868	0.776	0.820
RF	ACC	0.989	0.731	0.942	0.987	0.989
	F1	0.985	0.027	0.869	0.964	0.987

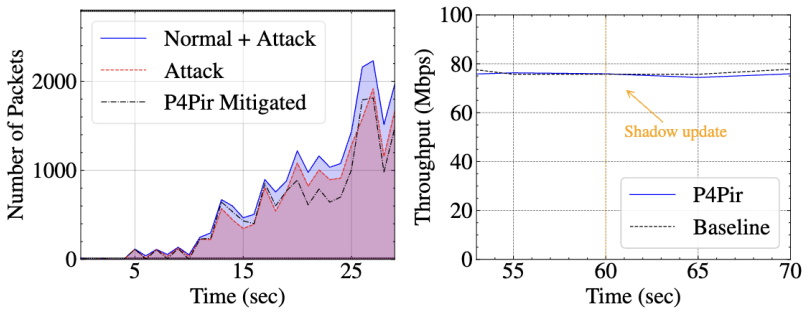
Table 4.1: Detection accuracy on dataset CICIDS 2017.

		SYN	SYN→SCAN		SYN→UDP		SYN→HTTP†	
		Init	Base	P4Pir	Base	P4Pir	Base	P4Pir
DT	ACC	0.910	0.156	0.945	0.435	0.903	0.921	0.941
	F1	0.953	0.270	0.972	0.606	0.949	0.924	0.970
RF	ACC	0.999	0.674	0.999	0.888	0.903	0.791	0.902
	F1	0.999	0.788	0.999	0.934	0.944	0.876	0.943

Table 4.2: Detection accuracy on dataset EDGE-IIOTSET.

The results show the DT/RF mapped to the data plane for in-network inference can achieve the same level of accuracy as the benchmark performance given by both datasets, reaching more than 90% accuracy and F1 score as listed in initial states. Different attack patterns may result in different levels of accuracy decrement for static models (Baseline), and we can efficiently mitigate it through model updates. Accuracy decrement can be seen for baselines in “SCAN→DOS” column in Table 4.1 and “SYN→SCAN” column in Table 4.2. It may be due to the changing attack attributes and feature distributions that static models have not learned. The novel unsupervised labeling mechanism learns changing feature distributions in logs and retrains DT/RF with the logs to detect attacks with increased accuracy. It increases DT's accuracy by more than 50% and RF's accuracy by more than 30% (“SCAN→DOS” column in Table 4.1 and “SYN→SCAN” column in Table 4.2).

To assess the effectiveness of the in-network design in fast attack mitigation, Figure 4.5 (a) presents a snapshot of captured traffic. In this capture, Friday afternoon record with normal and DDoS traffic in CIC-IDS2017 dataset is replayed. Total amount of traffic (including normal and DDoS attacks) is marked in blue, and attack traffic is marked in red. DT is deployed based on five features and the black dash line depicts the number of packets that are mitigated (dropped) by the solution. As shown, it learns the new attack in sub-millisecond and quickly starts dropping attack traffic, as shown by the overlap of red and black dash lines in Figure 4.5 (a).



(a) Mitigation performance. (b) Shadow update impact.

Figure 4.5: (a) A snapshot of mitigation performance of SmatrEdge DT on DDoS attack from CIC-IDS2017 dataset. (b) Shadow update impact of SmatrEdge DT on throughput. Baseline - forwarding setup without any model update.

For the update efficiency, a shadow update scheme is introduced. As the update involves changes to table rules within the data plane, the efficiency of shadow updates is evaluated. A qualitative comparison of throughput is presented in Figure 4.5 (b). At the 60-second mark (indicated by the yellow arrow), the shadow rule update does not affect throughput, as indicated by the similarity between the blue solid curve (shadow update) and the black dashed curve (baseline without our deployment).

This work has been peer-reviewed and published in IEEE Internet of Things Journal [Zan23-1].

4.4 Federated In-network Defense

To provide accurate analysis capability by ML-driven algorithms, a large amount of telemetry data is sent to server/cloud for ML training and inference. Such transmissions increase the load on the communications infrastructure and may lead to long RTT, increasing the response time to mitigate attacks. Although in-network ML-based attack detection offers immediacy and flexibility, scaling such solutions to multiple devices in a distributed system poses challenges, particularly in the case of multiple gateway access at the IoT edge [Zhe23-3], as in SmartEdge.

To tackle this issue, Federated Learning (FL) algorithms can enable ML-based model training and inference with the data maintained locally. Data volume and communication overheads are reduced by keeping the input data locally, while sharing model parameters with the server to maintain data privacy. To do so, SmartEdge needs to efficiently coordinate the tree-based FL design with the in-network ML-based attack detection in distributed IoT scenario.

In SmartEdge, we present a distributed in-network attack detection solution (dubbed FLIP4) based on federated tree-based models. It does model inference in-band to quickly detect and mitigate attacks. It utilizes FL to coordinate an efficient distributed deployment of in-network attack detection, reducing communication overheads and preserving data privacy. Instead of sharing all collected data, the solution trains a global model by sharing local model weights without leaking any source data.

To enable federated in-network defense, SmartEdge offers the following innovation: 1) enabling lightweight learning and update process to adapt to distributed IoT setup. 2) Securing the model-sharing process with low overhead to prevent malicious interception 3) integrating the local information into a global model and driving local update for in-network inference process.

4.4.1 Design

In SmartEdge, we consider a network scenario with multiple switches acting as IoT gateways that are deployed between the IoT end devices and a remote server in the cloud. Each IoT end device is assumed to be at risk of network attacks like Scanning, Man-in-the-Middle (MitM), Distributed Denial of Service (DDoS), etc. An attacker might exploit one of the devices to launch protocol attacks or use several of them as a botnet.

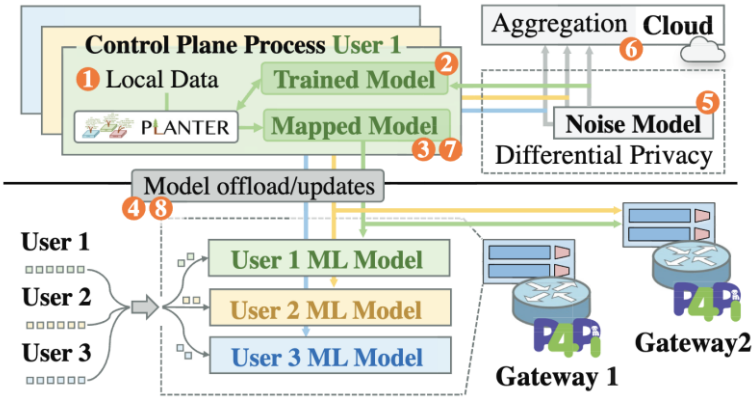


Figure 4.6: System design of Federated In-Network Defense.

The proposed design is a privacy preserving in-network analysis framework based on FL and in-network ML. Under this framework, traffic is being processed inside the IoT gateway with low latency and high throughput, traffic data remain locally, and uploaded models are added with noise to prevent privacy leakage. The complete process consists of the following steps. A gateway (a switch) is first initialized with a local model trained using a local traffic dataset (Figure 4.6 step 1 and 2). The trained model is offloaded and mapped to the data plane pipeline for in-network inference by running a P4 code and inserting mapped table entries reflecting the model structure and parameters (Figure 4.6 step 3 and 4). Incoming traffic from each user device passes through the programmable data plane on IoT gateway for header parsing and feature extraction. The extracted information goes through the M/A pipeline with inference logic for labeling. If a packet is labeled as benign, it will be forwarded, otherwise, it would be dropped. To maintain a global model for the server over the local models on multiple gateways, the parameters of the trained model on each IoT gateway are sent to the server with the protection of Differential Privacy (as Figure 4.6 step 5). When the server receives the updated information from all gateways, the aggregator starts to average the parameters to generate a global model, and the parameters of this global model are sent back to each IoT gateway for local model updates (as Figure 4.6 step 6). The mapper on each gateway coordinates the control plane and data plane to map the updated model's parameters to table rules and insert the new rules to the data plane pipeline at runtime (as Figure 4.6 step 7 and 8). The training process is further split into model training and communication process. In this system workflow, the model training includes local training on the IoT gateway, and global training on the server. Communication refers to the interaction between the control plane and data plane within the IoT gateway, and to the parameter exchange between the IoT gateway and the server.

4.4.2 Implementation

The implemented prototype combines interaction between the data plane, control plane, and server to achieve FL-based in-network attack detection. It combines three types of components: trainer, mapper, and aggregator.

- Trainer: The role of the trainer is to train the local model on every single device. It runs on the IoT gateway control plane and preprocesses local data. It next initializes the local model and conducts the model training.
- Mapper: The completeness of model training will trigger the mapper to map the model inference process to the data plane. It maps the trained model to P4 code and a set of M/A table rules. These rules are inserted by the control plane into the data plane pipeline for in-network inference. The data plane of each IoT gateway runs the generated P4 code, and M/A rules are written to the data plane's tables for in-network inference at runtime.
- Aggregator: When the trainer completes model training based on the local data, the parameters of the local model are sent to the server, where the aggregator computes a federate averaging over all local models to gain the global model. It then sends the averaged parameters back to the IoT gateway for local updates.

Figure 4.6 illustrates a scenario where multiple users (end devices) access through the same gateway in a multiple access gateway scenario. This includes a more dedicated scenario when a gateway serves the data access from a single user.

4.4.3 Preliminary Results

Experimental Setup. The data plane code is implemented in P4 language using bmv2 with v1model architecture and prototyping on Raspberry Pi using P4Pi-v.0.0. Python code provides controller and server functionality. The prototype extends the design in FL algorithm [Mad22] and *Planter* [Zan23-1, Zhe22] for ML training and inference functions. To evaluate performance on multiple nodes, Mininet is used for network emulation. Baseline results are taken from offline model learning using NN in *PyTorch* and XGB in *sklearn*.

Network Setup. As the solution is deployed for a distributed network scenario, a SOHO (Small Office/Home Office) network scenario is taken as a sample scenario where a limited number of gateways are connected. A network topology is built in Mininet to simulate a connection at the network edge with multiple switches acting as gateways and these gateways are linked with an edge switch connecting to the server. The parameters of the topology vary between the experiments.

Dataset. Public dataset CICIDS 2017 [Sha18] is used as source data, which includes both benign and malicious network traffic. In this work, we use Scanning, DDoS, and Botnet attacks as emerging attacks from IoT end devices assuming that remote server/cloud has to learn these attacks.

Evaluation Metrics. Detection Accuracy: Several metrics can be used to evaluate detection performance. In this paper, we use Accuracy (ACC), True Positive Rate (TPR)/ False Positive Rate (FPR), Area Under the ROC Curve (AUC) as the metrics to evaluate detection performance.

		Gateway1	Gateway2	Gateway3	Global	Offline
NN (Baseline)	IID	1.0	1.0	1.0	1.0	1.0
	Non-IID	0.9982	0.8430	0.5125	0.9649	0.9999
XGB (FLIP4)	IID	1.0	1.0	1.0	1.0	1.0
	Non-IID	0.9994	0.8124	0.5836	0.9681	0.9999
DP-XGB (FLIP4)	IID	0.9182	0.8991	0.9467	0.9417	1.0
	Non-IID	1.0	0.9064	0.6367	0.9274	0.9999

Table 4.3: Detection accuracy with/without DP on CICIDS 2017 dataset.

Table 4.3 provides a summary of detection accuracy of the prototype using CICIDS 2017 dataset. Taking the XGB model as an example, Table 4.3 presents the detection performance of the model implemented in an in-network manner with/without DP (DP-XGB/XGB) enabled on IID/non-IID dataset, where IID data is obtained by shuffling the dataset. To clearly list the performance results of local models at gateways and the global model in server, a network topology with 3 access gateways is setup on edge. Learning results on source data at the server are also listed as offline results. The results show that: a) Based on local information and local training, the global model is able to integrate the local models and present accurate attack detection without directly gathering the local data at the server. Due to the privacy trade-off, the global accuracy is slightly lower than offline results that are directly trained from source data, but is higher than NN model's global accuracy. b) The solution provides higher detection accuracy for IID data than for non-IID data. Considering that the global model is an average of the model at each gateway (switch), the distribution of local data may affect the model averaging performance. c) Introducing DP into the FL model can lead to a minor accuracy degradation (e.g., 4% decrement in accuracy for non-IID data). Such degradation is caused by the introduction of noise in DP method, which is a trade-off between accuracy and privacy.

This work has been peer-reviewed and published at IFIP Networking Sec4IoT [Zan23-2].

4.5 Hierarchical Flow-to-Traffic Graph Neural Network for DDoS Attack Detection

Distributed Denial of Service (DDoS) is one of the most common cyber-attack which takes advantage of the capacity limits of a network resource, orchestrating multiple requests from several devices that congest the access to the resource for ordinary usage. There are three main types of DDoS attacks:

- Volumetric attacks: A botnet attack floods the network with a massive volume of legitimate traffic that saturates the bandwidth.
- Protocol attacks: The resources of the servers or the intermediate communication equipment, such as firewalls and load balancers, are consumed by targeting Layer 3 and Layer 4 protocol communications.
- Application attacks: The application layer (Layer 7) is attacked by legitimate requests that target vulnerabilities to consume specific resources, such as database queries or file reads.

An example of a Volumetric Attack is the ICMP Floods that overwhelms a target device with ICMP echo-requests, forcing it to respond with an equal number of echo-replies. An example of a Protocol Attack is SYN Flood, which exploits the TCP handshake mechanism by opening several connections with SYN packets destined for the server. It will send SYN/ACK packets until the client responds with an ACK, that the attacker will never send, or a connection timeout, uselessly consuming bandwidth. To build a flexible DDoS Detection System (DDS) able to face the variety of attacks, it is necessary to observe both the aggregated traffic between servers and hosts and the specific traffic matching the communication flow (e.g., protocol, ports) between two endpoints. The DDS must reach a trade-off between the delay due to the amount of incoming traffic under analysis (to provide a sufficient overview to observe possible malicious patterns) and the reactivity to enforce proper mitigation and security rules promptly before significant damages. Moreover, this control mechanism should not overload the computation capabilities of the network.

DDSs based on traditional Machine Learning (ML) and Deep Learning (DL) techniques exploit meaningful either flow-level or traffic-level features that are statistical aggregations of the information related to the exchanged packets in a flow or in a time window, respectively. Those approaches show good performance when trained on popular Intrusion Detection System (IDS) datasets but lack adoption in real-world scenarios due to their inability to generalize and be flexible to different networks and traffic profiles. This problem can be addressed by replacing the statistical aggregation with the investigation of the structure of flows, given by sequences of exchanged packets between two endpoints, and aggregated traffics, intended as the set of flows established among endpoints in a certain time window. This augmented topological knowledge provides the possibility to detect common structural patterns that appear in specific types of DDoS attacks. It is necessary to analyze both intra- and inter-entity flow level interactions to completely capture all the possible attack patterns.

In SmartEdge, we propose a novel graph structure Flow-to-Traffic Graph (FTG) that incorporates both flow-level and aggregated traffic-level structures in a two-level hierarchical representation, and a GNN model (FTG-Net) able to process those fine-to-coarse graphs and classify the flows as legitimate or malicious. Our approach can represent and embed the structure of a flow between a host and a server, and combine this representation with the structure of the whole traffic, fulfilling the requirement of including each type of DDoS attack in the possibly recognized patterns. This solution is only based on the traffic topology and does not require stateful features, that can be expensive to compute in real-time scenarios and often overfit the characteristics of a specific training dataset, without the ability to generalize. When FTG-Net is deployed in real-world environments, significant stateful features, depending on the network requirements and capabilities, can be added to the graph nodes to enrich the representation and improve the performance.

4.5.1 Graph Structure for DDoS Attack Detection

GNNs are a type of deep learning model specifically designed to process and analyze data represented as graphs. GNNs are a natural extension of traditional neural networks, but they are designed to handle inherently non-Euclidean data such as social networks, communication networks, and biological networks. Unlike traditional neural networks, which can only operate on data that can be represented as vectors or matrices, GNNs can directly process the graph structure itself. This allows them to capture the graph's intricate connections and relationships between entities (nodes).

Network traffic can be represented hierarchically, with flow entities ranging from fine-level packets to coarse-level communications between endpoints. This section introduces a hierarchical graph-based structure called FTG (Flow-Traffic Graph) for detecting DDoS attacks. The FTG captures information at flow and traffic levels, making it a comprehensive tool for detecting such attacks.

4.5.1.1 Flow Graph

The Flow Graph is a crucial component in analyzing network traffic patterns. It is created for each time slot and captures how packets interact between endpoints within that time slot. To construct the Flow Graph, each packet is converted into a node, and the upstream and downstream packets are distinguished by positive and negative packet lengths, respectively. Edges are added between consecutive packets within the same mini-group, which are packets sent sequentially by one of the endpoints. This approach allows us to gain insights into packet behavior and identify potential network performance issues.

Figure 4.7: Example of a Flow Graph. Upstream packets have positive packet length features, whereas downstream packets have negative packet length features. The mini-groups sequence is sorted from left to right.

4.5.1.2 Traffic graph

A Traffic Graph is generated for each time slot to analyze the interactions between network endpoints. The nodes in this graph represent pairs of endpoints that have communicated with each other during that time slot. The features of each node are based on the output vector of the Flow GNN, which processes the corresponding Flow Graph. Edges are added between nodes in the Traffic Graph when they share the source or destination IP address to connect nodes in the Traffic Graph.

4.5.2 FTG-Net Hierarchical Model Architecture

The FTG-Net architecture is designed to handle the multi-level graph representation of FTG. It comprises three steps. In the first step, traffic data is converted into FTG structures. In the second step, the Flow GNN processes the Flow-level Graphs and generates an embedded representation vector for each graph. These vectors are employed as node features in the Traffic-level Graphs, which are processed in the third step by the Traffic GNN. Finally, the Traffic GNN outputs the final predictions for each flow, which represents the communication between two endpoints in a time slot. The architecture of FTG-Net, shown in Figure 4.8.

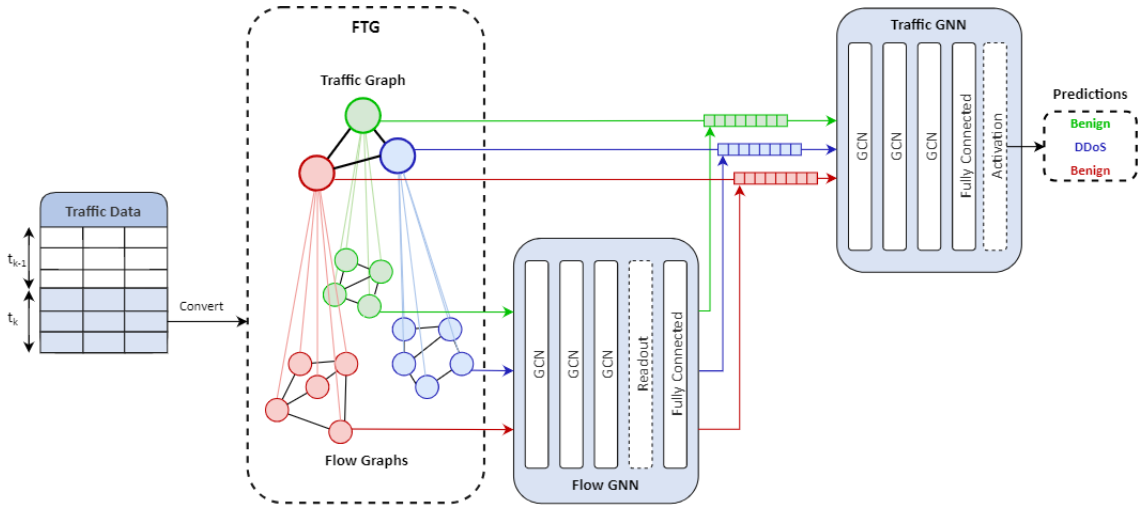


Figure 4.8: Hierarchical model architecture of the FTG-Net solution

The FTG-Net model comprises two GNNs: a Flow GNN and a Traffic GNN. The Flow GNN analyzes the Flow Graphs to extract features at the flow-level, while the Traffic GNN analyzes the Traffic Graph to extract features at the traffic-level. The Traffic GNN's output is then used to determine whether the traffic in the corresponding time slot is malicious or not.

The FTG-Net model is a powerful tool for detecting DDoS attacks. It captures both flow-level and traffic-level information, which means it can identify targeted attacks on specific endpoints and attacks that affect the overall network traffic. Additionally, the model incorporates flow-level patterns, allowing it to learn and recognize patterns indicative of DDoS attacks. Another benefit of the FTG-Net model is its efficiency in both training and deployment. Its relatively simple structure makes it suitable for real-world applications. When evaluated on a dataset of real-world DDoS traffic, the FTG-Net model demonstrated impressive results. It achieved a remarkable detection accuracy of 92.1%, surpassing state-of-the-art DDoS attack detection methods. The FTG structure provides a promising approach for DDoS attack detection. The hierarchical graph-based representation captures flow-level and traffic-level information, and the two-level GNN architecture allows the model to learn flow-level and traffic-level patterns. The experimental results demonstrate the effectiveness of the FTG-Net model for DDoS attack detection.

4.5.3 Results

To evaluate how well FTG-Net performs, we used a method called 5-fold cross-validation. This method randomly divides the dataset into five parts, or "chunks." We trained the model for each chunk using the other four chunks as the training set and the current chunk as the test set. We used a time slot size of 5 seconds and an 8-dimensional vector as the output of the Flow GNN. We used an 8-dimensional vector as the feature vector for the Traffic Graph nodes. We also used 64 hidden channels in the output to the GNN Layers. Initially, the nodes of the Flow Graphs are made up of only one element, corresponding to the packet length.

Method	Accuracy	F1-score
LUCID	0.9967	0.9966
GLD-Net	0.9940	0.9920
GraphDDoS	0.9959	0.9959
FTG-Net	0.9914	0.9913

Table 4.4: Comparison of Results

The results achieved by combining the best model for each cross-validation phase are shown in Table 4.4. These results are comparable with the state-of-the-art results. Our model has an advantage over other models in that it only considers the network structure and does not collect multiple stateful features. However, the time slot size must be configured carefully to use our approach. A larger time slot provides a broader view of the traffic topology, which contains more information to detect potential DDoS attacks. But it is inversely proportional to the system's responsiveness and may not be enough to detect attacks before damage.

Additionally, with large Traffic Graphs, the inference time increases. On the other hand, a smaller time slot may need to carry more information to classify traffic robustly. To evaluate the performance of our approach, we measured the average inference time and weighted F1-Score of four different time slot sizes. We used a random split that included 70% of the data in the training set and the remaining 30% in the test set. The results are shown in Figure 4.9, confirming the importance of carefully selecting the time slot size.

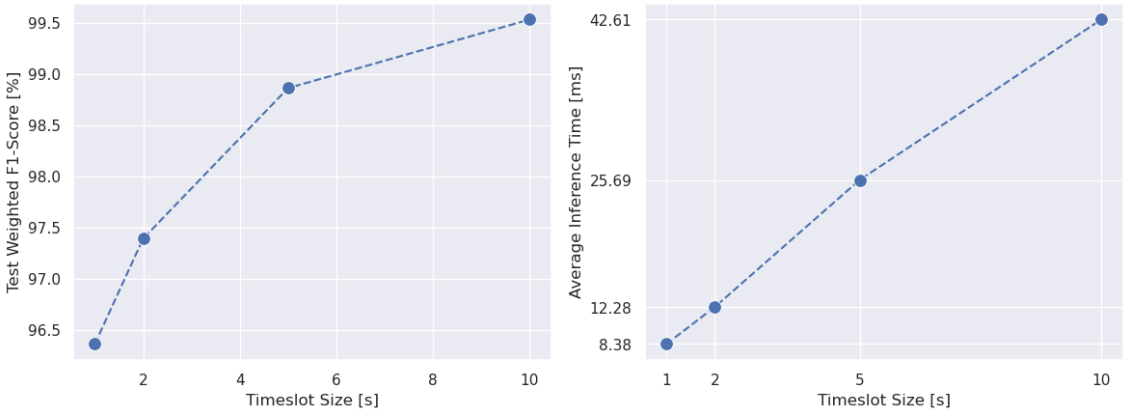


Figure 4.9: Hierarchical model architecture of the FTG-Net solution

This work has been peer-reviewed and published at HPSR Conference [Bar23].

5 HARDWARE-ACCELERATED IN-NETWORK OPERATIONS

5.1 DISTRIBUTED IN-NETWORK OPERATIONS

5.1.1 Introduction and motivation

Scaling in-network services is hard, as programmable network devices are intended for high-efficiency packet processing, with limited resources (e.g., memory, operations, and stages) compared with CPUs. One approach is to optimize algorithms' design for a single-device deployment, yet resource constraints remain a limitation [Zhe22]. An alternative solution is moving to distributed in-network computing, jointly utilizing resources of programmable network devices. Such an implementation is especially important in SmartEdge, where a swarm of devices is being used, and complex intelligent operations need to be supported across the swarm.

Distributed in-network computing raises multiple implementation challenges, especially where resource-heavy applications are considered, such as large machine learning (ML) models. Figure 5.1 illustrates the challenges: (1) Decomposing the single program into multiple segments. (2) Distributing the program's segments across multiple devices without affecting the correctness of its functionality. (3) Satisfying the program's and network's set of constraints, such as latency and resource constraints. (4) Providing the program's functionality for any set of paths within the network without routing rules changes. This last challenge is possibly the hardest, as in a network packet may travel from any node to any node, and operators may use different routing optimization methods.

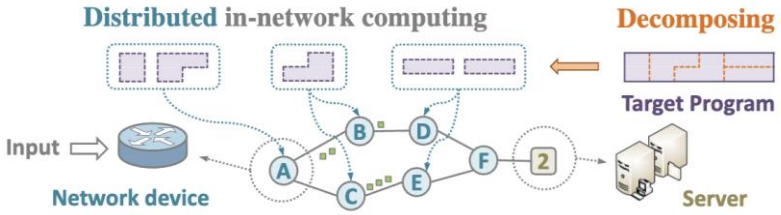


Figure 5.1: Distributed in-network computing paradigm.

In a software-defined network, a controller has a centralized view of the network, including device information, and potential routes through the network [Zhe23-1]. Building upon this information, a controller can be designed to provide a joint resource provisioning plan for distributed in-network computing. It can utilize unused network resources, splitting a single in-network computing service across several devices.

In SmartEdge, we present a Distributed In-Network Computing (DINC) solution. It efficiently plans and implements P4-based service partitions on multiple network devices. The solution's planner supports any-to-any routing and is able to distribute and deploy program segments across network devices while providing full, correct functionality. To ensure co-existence with normal network functionality, it co-designs a code slicer and generator, extracting and generating P4 program slices in accordance with the planner's strategy.

5.1.2 Distributed In-Network Computing Workflow

An overview of the distribution framework is shown in Figure 5.2. The framework is given an in-network computing program with both data plane and control plane code components (shown in ❶), and a network topology (shown in ❷). The P4 slicer extracts the program resource requirements (shown in ❻), dependencies (shown in ❸), and metadata information. The network controller provides the routing table, with all paths identified either by the controller or the framework (shown in ❹) and the resources available on each network device (shown in ❺). The planner uses the outputs from steps ❸ to ❻ to craft an integer linear programming (ILP) problem and outputs a deployment strategy in step ❼. This guides the P4 generator for data plane and control plane codes generation in step ❽.

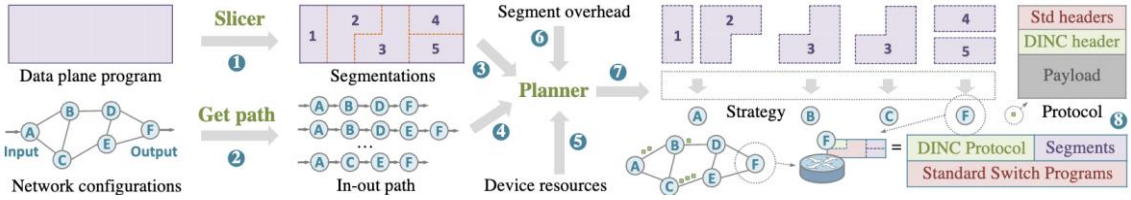


Figure 5.2: DINC workflow overview.

5.1.3 Framework Implementation

The deployment of distributed in-network computing is challenging for two reasons. The first is the complexity of the data plane program. It is hard to correctly extract information from a given data plane program according to user expectations. The second is the complexity of data plane program generation. Reconstruction of a valid data plane program under a certain architecture, coexisting with its use case, and with an embedded DINC header is difficult, especially when the slice is complex.

Our framework is designed to tackle these two challenges. It contains a data plane program Slicer, a strategy Planner, a code Generator, and a Tester, allowing the generation of sliced data plane programs and their deployment on target hardware nodes. The detailed workflow of the framework shown in Figure 5.3 is as follows: ❶ User input: The target data plane program and slicer markers are required as user inputs. ❷ Network configuration: The network topology and its related resources are required and stored in configuration files, which can generate directly from the network controller or emulate by using the framework. ❸ Slicer: The network slicer slices the target program based on the markers, builds the dependency between each segment, and digs the resource information of each segment. ❹ Planner: The planner is used to generate deployment strategies for the target program. Based on the input network topology, segment dependency, remaining resources for each network device, and the resource overhead of each segment from previous steps, the planner applies the ILP to formulate the plan. ❺ P4 generator: The code generator generates data plane codes for all programmable network devices. To generate the codes, the generator jointly combines the selected architecture and use case based on the configurations, and combines the program segments of the input program from the deployment strategy. ❻ Synthetic test: Before loading the generated code to the real hardware, a synthetic test is implemented by the framework. The framework will activate a test environment with the same topology as the input network and load the generated model for each node. The test can then be easily done within the environment while the content varies by use cases. P4 debugging tools can be used in the field to trace and correct bugs in each generated segment [Sto18, Zho19]. ❼ Segment loader: When the test is finished, the generated data plane programs with sliced segments are to be sent to the controller for deployment.

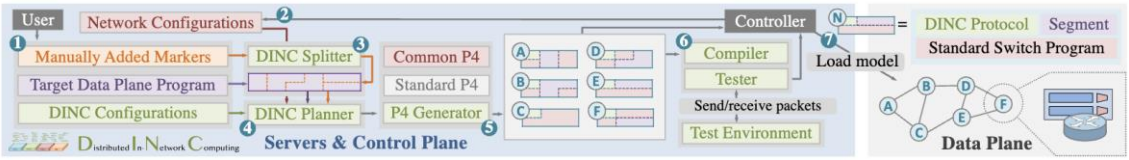


Figure 5.3: Distributed In-Network Computing framework components and workflow steps.

Splitter (Slicer). The slicer is used to segment the input data plane program and extract information from it. It uses manually added markers to slice the program and extract the information. This solution is lightweight and flexible.

Planner. In SmartEdge we address the absence of distributed planning for in-network computing across multiple paths, without influencing the routing rules of the original network. To tackle this, the problem is formulated within the planner as an ILP problem. In this problem, we expect our deployment strategy to optimize the following three major objectives in distributed in-network computing: minimize the resource consumption, minimize the computation delay and minimize duplicate deployed segments, as well as constraints of dependency, integrity, and resource availability. These objectives can be extended, encompassing additional and customized requirements, such as limitations on hops and latency for specific services, constraints on throughput, and variations in resource constraints using heterogeneous devices. The specific constraint conditions applied can be adjusted based on the specific deployment scenario. Searching for the optimum solution of an ILP is NP-hard. SmartEdge tackles this problem using the branch-and-cut method that combines the cutting plane method and the brand-and-bound method.

P4 Code Generator. The P4 generator is designed to generate the data plane program that can be directly applied to each node, composed based on a controller plus multiple architectures and uses case building blocks. Before generating any node, the controller will invoke the selected architecture and use case block based on the network controller or the configurations. Besides writing the skeleton of the data plane program, the selected architecture folder will call the respective use case function for writing use case codes at each position of the program. The deployment strategy will also be applied in the architecture block to drive the regeneration of the sliced segment at the right position. Functions that can auto-generate P4 codes dealing with metadata embedding and extraction as well as bitmap checks are applied and can be called by any nodes or segments during the generation process.

Modular Framework Design. The new architecture and target design appear frequently, and a design that allows iteration is required. The framework applies a modular design, where a centralized controller is used to provoke corresponding modules according to the input configurations. When a module is selected, all the functions under this module will be loaded to the controller. It is currently designed to support modular topology generators, solvers, architectures, targets, slicers, use case generators, and testers. With this modular framework design, the solution can be adapted to the required case easier and with strong customized capacity.

Implementation. The framework is implemented in Python 3.10. The ILP solver is based on *milp*, a mixed-integer linear programming solver in *SciPy v1.10.0*. The topology is stored using *NetworkX 3.0*. The framework is open and available on the DINC's GitHub repository [Zhe23-4]. It supports a range of predefined modules, including topologies, solver, slicer, architectures, targets, and tester. Under topologies, it currently supports Fat-Tree, Folded-Clos, and BT ISP.

The P4 slicers currently support manual configuration inputs, where inputs come from manual marking or are auto-generated by a data plane automation framework like Planter. It supports two architectures: v1model [Bar21] and TNA [Int21] and coexists with simple forwarding, RARE [Lou22], and Intel's switch.p4.

5.1.4 Preliminary Results

We evaluate the solution using two common network scenarios — a Folded-Clos based data center network topology (shown in Figure 5.4 (a) with 3 core, 6 aggregation level, and 24 edge switches, connecting around ten thousand servers), and a large wide area network topology - British Telecom (BT) (shown in Figure 5.4 (b) with more than a thousand nodes: 8 inner core, 12 outer core, 63 metro, and 925 tier 1 switches). The data center topology is relevant to the small-scale deployments in SmartEdge, while the wide-area network is more representative of UC-2 while also evaluating scalability. Using the two topologies, Table 5.1 shows a subset of potential ingress/egress switch setups. A total of five setups are presented and are combinations of two dimensions: communication direction and deployment view.

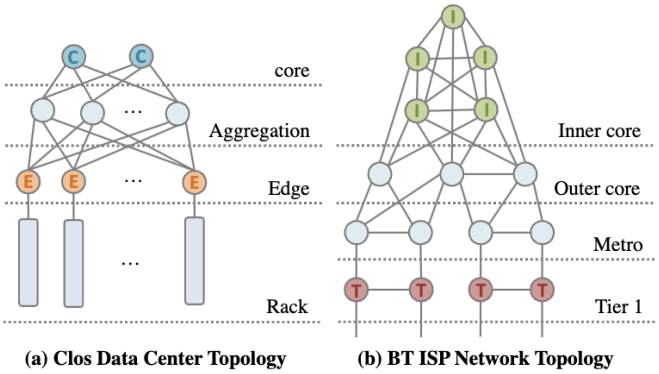


Figure 5.4: Network topologies used for evaluation.

Setup	Setup 1		Setup 2		Setup 3		Setup 4		Setup 5	
Topo.	Clos	BT	Clos	BT	Clos	BT	Clos	BT	Clos	BT
Ingress	All E	All T	All C	All I	One E	One T	All C	All I	One E	One T
Egress	All C	All I	All E	All T	All C	All I	One E	One T	One E	One T

Table 5.1: Network ingress & egress scenarios using the topologies shown in Figure 5.4.

Workloads. We use several popular in-network computing programs from different domains: in-network telemetry (INT) using PINT [Bas20] (especially useful to accelerate SmartEdge's innovative telemetry solution), load balancing using Pegasus [Li20] (relevant for load-balancing accesses to coordinators), and in-network ML inference based on Planter [Zhe22] with 5 ML models (applied to ML-based in-network defense). These applications represent advanced in-network computing services, yet they require significant resources and need optimization when co-deployed with other data plane forwarding functions.

Metrics: The efficiency of distributed in-network computing is evaluated for the following: *1. Number of used nodes:* The number of nodes required to deploy an in-network computing program. This number depends on the total number of available nodes. *2. Hops per path:* The number of hops required to complete a given in-network computing program. Hop number is directly proportional to the latency, and hops are used to represent latency, and to ensure the evaluation is unbiased in terms of network setups and equipment selection. Cumulative distribution function (CDF) of hops provides further insights into the distribution of required

hops per path. *3. Duplication of segments:* in topologies with multiple in-out paths, duplicate segments may exist on different paths. The amount of duplication is relative to the total number of nodes and shows the efficiency of the deployment strategy. *4. Execution time:* Job completion time of the ILP solver is critical to its application. The ILP solver's execution time depends on multiple factors, ranging from the topology to the program, and the solver will be generally scalable if the time increases in a linear manner.

Environment: Both large-scale simulation and small-scale hardware tests are conducted. The simulation is based on Mininet and BMv2, and hardware tests run on Tofino using APS-Networks BF6064X-T (64x100G) and two NetBerg Aurora 710 (32x100G) switches. Tofino compiler is further used for feasibility testing.

Program	Stand Alone			Segment	Setup	Folded-Clos Topology					BT ISP Topology				
	Stage	Alone	Coexist			Path	Dis.	Nodes	Hops	Dup.	Path	Dis.	Nodes	Hops	Dup.
INT-PINT	7	✓	✗	5	4	18	✓	10/33	3/3	9	36	✓	3/1008	1.42/3.92	8
LB-Pegasus	8	✓	✗	4	1	432	✓	30/33	2/3	56	26512	✓	400/1008	3.11/3.81	796
ML-NB	8	✓	✗	2	4	18	✓	9/33	2/3	7	36	✓	6/1008	2.92/3.92	4
ML-SVM	9	✓	✗	3	5	6	✓	8/33	3/3	5	12	✓	7/1008	3/5	4
ML-DT	2	✓	✓	2	3	18	✓	1/33	1/3	0	36	✓	1/1008	1/3.92	0
ML-XGB	6	✓	✗	4	2	432	✓	9/33	2/3	11	26512	✓	410/1008	2.76/3.81	422
ML-RF	6	✓	✗	3	1	432	✓	30/33	2/3	33	26512	✓	400/1008	3.11/3.81	405

Table 5.2: Sample programs Supported by the framework. Segmentation (Seg.) Details can be found in [Zhe23-4]. Setups refers to Table 5.1. Duplication (Dup.) - number of duplicated segments. ✓/✗ Deployment feasibility. Distribution (Dis.) - distribution feasibility. Nodes - nodes used/total. Hops - average used hops/path length.

Simulation: Our solution is evaluated both on the Folded-Clos data center network and BT topology, using the five setups and seven workloads. We measure the resource consumption and number of hops on both single and distributed deployments for all workloads, as summarized in Table 5.2.

The distributed in-network computing framework is capable of processing distributed deployment problems at data center level with about 10,000 servers or at ISP level with around 1000 switches. As Table 5.2 shows, all workloads, including those that cannot coexist with RARE switch functionality (✗ in Coexist column) are feasible in a distributed deployment, coexisting with network functionalities (Dis. column). As some programs, e.g., ML-DT, coexist with other switch functions, the solution selects the best node along the path to optimally utilize resources and minimize latency. Such resource optimization advantages can be found in all distributed deployed programs.

In the comparison between standalone and distributed deployment, as illustrated in Table 5.2 (in the same row), consider the example of ML-RF/XGB. Under Setup 2, with a Folded-Clos data center configuration featuring 3 core, 6 aggregation, and 24 edge switches (equivalent to 1000 servers), the solver successfully resolves 4 segments. In this setup, more than 400 distinct data paths require service deployment. Our novel implemented algorithm deploys segments on only 9 out of 33 devices, achieving service completion with an average of 2 (out of 3) hops. Additionally, there are only 9 secondary segments duplicated, compared to deploying on each individual data path (which would require thousands of repeated segments) or alternatively routing all flows requiring service through a single path or a subset of paths. This significantly enhances deployment efficiency and reduces resource consumption.

In the comparison between distributed deployment under different path setups (among rows in Table 5.3), workloads in BT topology using all-to-all scenarios (setups 1 & 2) require using less

than 40% of the nodes, and each path requires on average less than one duplicated segment. For one-to-one and one-to-all scenarios (setups 3-5), not all the devices are needed for complete task execution. In the data center setting, the number of all-to-all deployments requires one-third of nodes when data is arriving from the core (e.g., incoming to the data center). Tasks for outward flows may require all nodes if a latency optimization is applied (comparing ML-RF and ML-XGB). This shows that our solver can reach optimal planning with efficient node utilization and limited duplication under different topologies and traffic path conditions for both use cases.

Hardware test: We evaluate our prototype on a small-scale hardware setup using the distributed RF-ML program on Point-to-Point and Fat-Tree topologies constructed using three Tofino switches, covering the majority of scenarios in Table 5.2c& Table 5.3. RF-ML is chosen as Planter [Zhe22] provides a functionality validation test. The result shows the model functions correctly with the same accuracy as a standalone deployment.

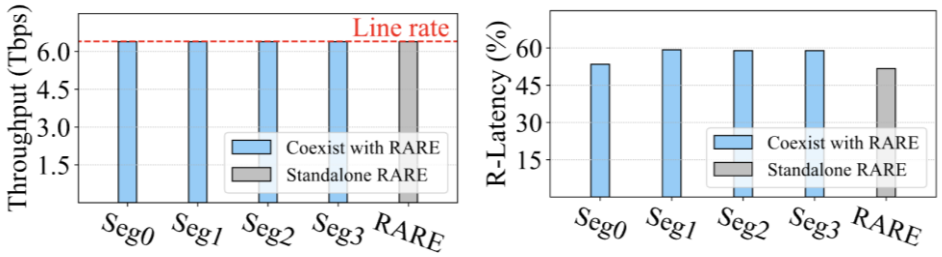


Figure 5.5: Throughput and Latency on Hardware.

For the sample distributed RF segments, As shown in Figure 5.5, all segments are able to coexist with the RARE router program and are able to achieve a full line rate of 6.4Tbps bit rate. For the latency of each segment, the coexistence of the in-network computing application segment will increase the 10% of the relative clock cycle. However, even coexists with RARE, all deployed segments have a relatively low latency, which is lower than 60% of the reference design (switch.p4). The full line rate in the result proved that the added data-sharing bitmap and metadata passing mechanism will not limit the throughput of the system. The relative latency shows that the coexistence of network functions by using the solution will not noticeably increase the consumption of the clock cycle.

This work was peer reviewed and published in the Proceedings of the ACM on Networking [Zhe23-3].

5.2 Hardware-accelerated DDoS Detection using Convolutional Neural Networks (CNNs) on DPU

5.2.1 Introduction and motivation

SmartEdge supports advanced intelligent network security solutions, and under T4.3 our goal is to accelerate these solutions. Traditional software-based IDSs and IPSs face challenges in processing and analyzing network traffic in real-time due to the increase in network speeds and volumes. They also have scalability issues, high false positive rates, and limited effectiveness in detecting modern network security threats. To address these challenges, data plane programmability with P4-based feature extraction and in-network machine learning pipelines have been introduced to partially overcome scalability concerns at the switch level.

In SmartEdge, we propose an innovative hardware-accelerated framework designed to promptly detect and mitigate DDoS attacks by enabling real-time traffic analysis. The framework is built around a data processing unit (DPU) and uses the high-speed packet processing capabilities of the Data Plane Development Kit (DPDK) libraries. We have integrated RegEx for deep packet inspection (DPI) to analyze packet-level content. The framework also includes deep learning methodologies for real-time intrusion detection and Suricata for further fortification against potential attacks.

Our objective is to provide a comprehensive and adaptable solution that not only deals with the current limitations of traditional IDSs and IPSs but also aligns with the unpredictable nature of contemporary cyber threats. The goal is to improve the resilience of networks against various types of attacks, reinforcing the fundamental framework of cybersecurity in a constantly evolving technological era.

5.2.2 Background

Traditional intrusion detection systems rely on signature-based techniques to identify known attack patterns, but these systems often cannot detect new or sophisticated attacks. Deep learning has emerged as a promising approach to network intrusion detection, as it can learn to identify attack patterns based on large amounts of network traffic data. However, training and deploying deep learning models for network intrusion detection can be computationally intensive and require significant hardware resources.

Convolutional Neural Networks (CNNs) are a type of deep learning technique that has been increasingly used in the field of artificial intelligence, leading to remarkable advancements. Although the use of CNNs in cybersecurity is still a matter of ongoing research, they have already demonstrated notable success in certain applications such as malware detection, network traffic analysis, and intrusion detection in industrial control systems. These achievements, coupled with the advantages of CNNs in terms of reducing the need for feature engineering and providing high detection accuracy, motivate us to incorporate CNNs in our work.

Although larger CNN architectures have shown impressive detection rates, their size can be problematic when operating in limited resource environments. Moreover, IoT devices' deployment has increased the likelihood of DDoS attacks via vulnerable IoT devices. As a result, it is imperative to place defense mechanisms appropriately. For instance, high-powered appliances can absorb volumetric DDoS attacks and are located either locally or in the cloud. With the emergence of edge computing to enhance service delivery, it has become necessary to protect against attacks closer to the edge and on resource-constrained devices. Even without resource constraints, it is critical to minimize resource usage for optimal system output.

The BlueField-2 smart network interface card (SmartNIC) from NVIDIA provides a hardware acceleration platform for network traffic processing, using DPDK libraries to offload network traffic processing to the NIC. This can significantly improve performance and reduce the computational burden on the edge CPU.

5.2.3 Proposed Methodology Hardware-Accelerated Framework

Our proposed framework utilizes the NVIDIA BlueField-2 DPU to detect malicious DDoS attacks in real-time. The BlueField-2 is a SmartNIC developed by NVIDIA, which offers a hardware acceleration platform for processing network traffic. By offloading network traffic processing to the network interface card using DPDK libraries, the computational burden on the edge CPU can be significantly reduced, improving performance.

In SmartEdge, we introduce a hybrid framework for live traffic analysis and intrusion detection on the DPU. Our framework uses hardware acceleration to perform deep packet inspection (DPI) and deep learning-based intrusion detection while enabling fast and efficient blocking of malicious traffic. Figure 5.6 illustrates how DDoS detection tasks are offloaded to the DPU. Initially, incoming traffic is received by the DPU, which performs RegEx operations to identify malicious traffic. If the traffic is benign, it is forwarded to the edge CPU for further processing. However, if the traffic is malicious, the DPU immediately drops it.

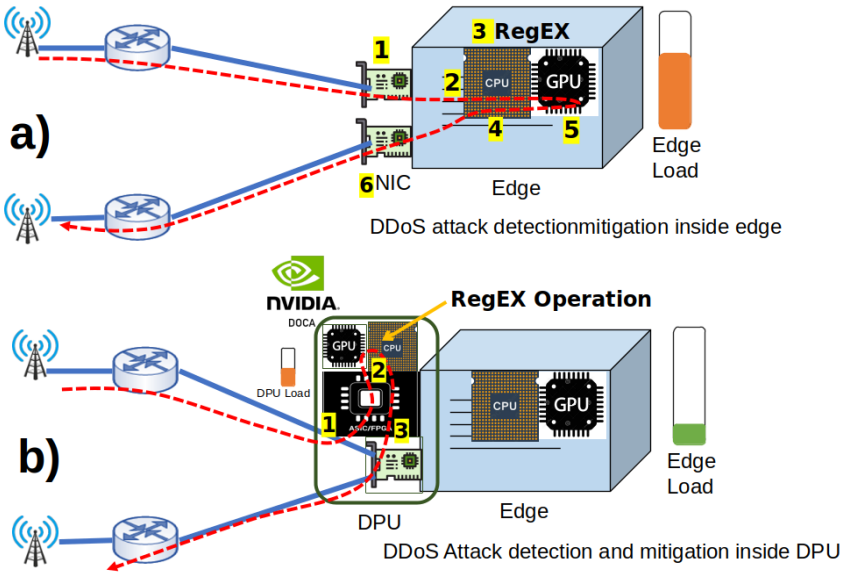


Figure 5.6: (a) Without offload: DDoS attack detection and mitigation inside edge and (b) With offload: DDoS attack detection and mitigation inside DPU

Our system employs two advanced DDoS detection algorithms based on deep learning, namely a fine-tuned CNN and a GRU. These algorithms begin by extracting packet attributes, followed by grouping them into flow. After this, the flows are truncated or padded to a fixed length and normalized. The fine-tuned CNN algorithm predicts whether each flow is normal or an attack. On the other hand, the GRU algorithm extracts features using the fine-tuned CNN and then feeds them to the GRU model for classification. Our system can detect DDoS attacks in real-time with high accuracy.

5.2.4 Implementation

Our proposed framework for detecting and blocking DDoS attacks was tested on a real-world network dataset collected from our testbed. The dataset included normal traffic as well as a variety of DDoS attacks such as volumetric, application-layer, and protocol-based attacks. Our experiment results showed that our framework is effective in accurately detecting and blocking a wide range of network attacks while maintaining high throughput and low CPU utilization.

Our DDoS detection and mitigation testbed consisted of two Dell servers connected via a SONiC-based white box network switch. Our testbed is described in Figure 5.7. Each server was equipped with an NVIDIA BlueField-2 DPU, with one server running the DDoS detection system and the other running the DDoS mitigation system. The detection system used a CNN model to classify traffic as normal or anomalous, while the mitigation system used an IPS to block malicious traffic. The DPUs were utilized to offload traffic processing and REGEX operations, which improved performance. The testbed was connected to a local data center using VLAN 2,

and the DPU admin server was also connected to the testbed using VLAN 2. We simulated a DDoS attack on the testbed to evaluate the performance of our proposed framework.

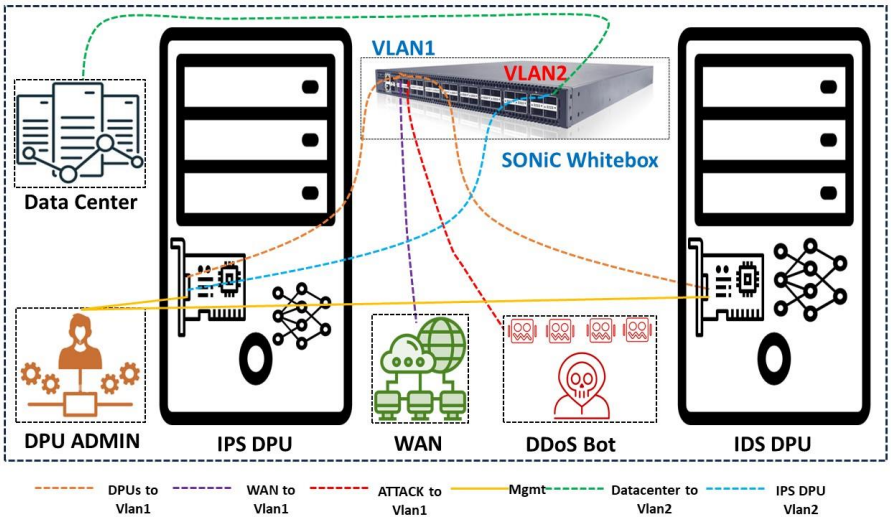


Figure 5.7: Hardware-accelerated DDoS detection and mitigation using DPU Testbed

5.2.5 Results

This section presents a thorough analysis of the performance of our proposed security system's detection and mitigation modules. We used a combined approach of GRU and CNN and a directed mitigation technique to accomplish this objective. To evaluate the effectiveness of our proposed method, we conducted tests on two existing detection methods: DIDDOS's GRU and LUCID's CNN. These methods were selected based on their compatibility with similar application environments and previous research.

The effectiveness of the anomaly detection approaches was evaluated using traditional classification metrics such as accuracy, precision, recall, and F1 score. Accuracy measures the percentage of correctly identified flow records, while precision estimates the percentage of accurately identified abnormal IP flows among all samples identified as unusual. Recall evaluates the proportion of correctly identified anomalous flows, and the F1 score represents the harmonic mean between true-positive rate and precision.

Our performance analysis focuses on two different test scenarios that use the public CICDDOS2019 dataset and a custom dataset. In the first experiment, we simulated IP flows from the CICDDOS2019 dataset comprising 87 IP flow features. We selected 78 features by excluding specific dimensions to avoid data bias. The dataset includes 12 types of DDoS attacks for training and six for testing.

The results of evaluating GRU and CNN deep learning models for detecting DDoS attacks are presented in Figure 5.8. The accuracy, recall, and F1-score were used as evaluation metrics. The GRU model achieved an accuracy of 99.45%, recall of 99.43%, F1-score of 99.66%, false positive rate (FPR) of 99.57%, and false negative rate (FNR) of 0.46%. The CNN model achieved an accuracy of 99.41%, recall of 99.55%, F1-score of 99.64%, FPR of 98.78%, and FNR of 0.44%.

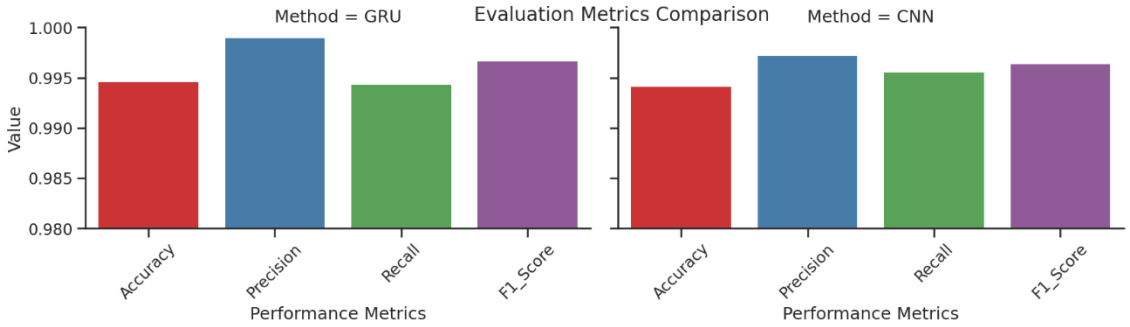


Figure 5.8: Evaluation results of the CNN and the GRU models

We conducted tests on various methods for detecting flooding attacks, and the results were highly promising, with a success rate of almost 99.45%. While the Convolutional Neural Network (CNN) method did not outperform the Gated Recurrent Unit (GRU) method, it is a better choice visually. The GRU method fared slightly better among the methods we tested, achieving a balanced outcome across all four metrics we measured. Furthermore, we conducted a separate analysis to evaluate the effectiveness of the methods in classifying normal and attack flows, which provided us with insightful results.

In Figure 5.9 (a), we can observe that CNN successfully accurately classified 234933 instances of normal traffic as normal, with a True Negatives rate of 99.43%. Similarly, 239530 cases of malicious traffic were correctly identified as malicious, with a True Positives rate of 99.56%. However, there were some instances where CNN made incorrect classifications. It misclassified 3622 instances of normal traffic as malicious, giving a False Positives rate of 0.57%. Moreover, it misclassified 33 instances of malicious traffic as normal, resulting in a False Negatives rate of 0.44%.

The confusion matrix in Figure 5.9 (b) reveals that the GRU Model accurately identified 235640 normal traffic instances as normal (99.57% True Negatives) and 239534 malicious traffic instances as malicious (99.56% True Positives). However, the model incorrectly classified 2915 instances of normal traffic as malicious (0.43% False Positives) and 29 instances of malicious traffic as normal (0.44% False Negatives). This detailed information about the performance of our CNN and GRU models is valuable for further analysis and improvement.

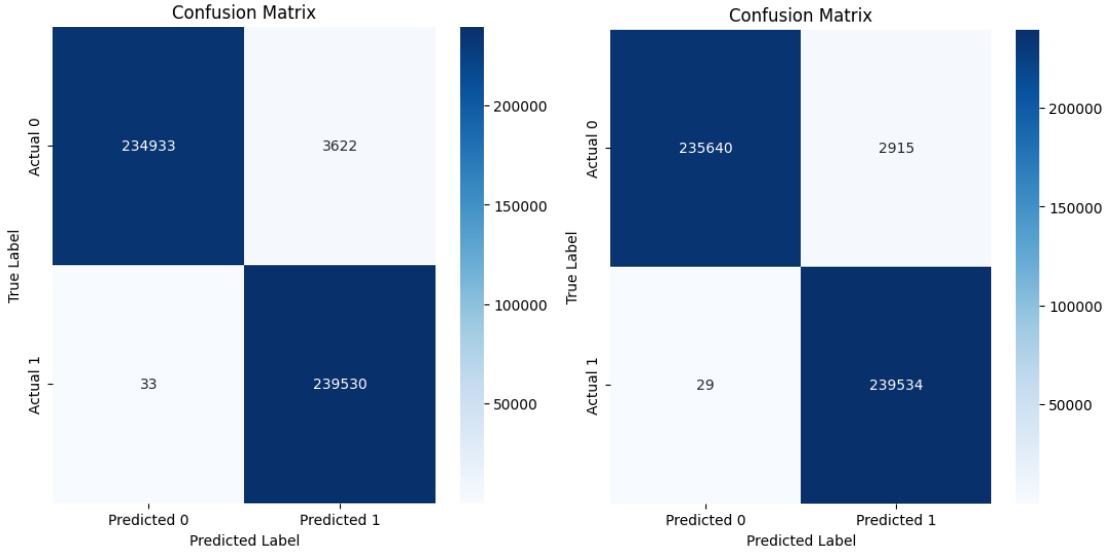


Figure 5.9: (a) Confusion matrix of CNN Model (b) Confusion matrix the GRU Model

5.3 INTELLIGENT ROAD SIDE UNIT

5.3.1 Background

As a pivotal component within the framework of intelligent transportation systems (ITS) such as in UC-2, the roadside unit (RSU) is a multi-functional device strategically positioned alongside roadways. RSU facilitates vehicle-to-everything (V2X) communication, enabling vehicles, infrastructures, and the cloud to exchange crucial information with each other. Nevertheless, its primary function is not limited to being a wireless communication device that provides connectivity support to passing vehicles; rather, it can also serve as an intelligent computation, memory, and networking hub.

Such intelligent RSU offers more task-offloading options for complex ITS scenarios. Compared to computing on the vehicle side, intelligent RSUs naturally facilitate vehicle-to-vehicle and vehicle-to-infrastructure cooperations, cutting down the redundant sensors and repetitive computation across different vehicles; compared to computing on the traditional edge cloud servers, intelligent RSUs decrease latency and network traffic, shortening the data flow pathway.

One of the foremost advantages of utilizing the RSUs for computational tasks is the reduction of round-trip latency, as these tasks no longer need to access the cloud. In time-critical scenarios such as collision avoidance and intersection management, round-trip latency should be minimized to at most a few tens of milliseconds and even smaller for the computation time at RSU. The demand for low-latency computation at the RSU underscores the need for acceleration to fully achieve the potential of intelligent RSUs.

5.3.2 SmartEdge Intelligent RSU

Within the context of SmartEdge Use Case 2, this scenario is centered on the smart junction scenario, encompassing multiple intersections. In this scenario, stationary radars, cameras, and controllable traffic lights are deployed on the roadside, all connected to the RSU. Three categories of vehicles are involved: ordinary vehicles, dummy vehicles, and smart vehicles. Dummy vehicles can send information to the RSU, but do not respond to RSU's messages. Smart vehicles are equipped with cameras, LiDARs, GPS, and wireless communication, enabling bidirectional data exchange with the RSU. This dynamic setting forms the background for our development of an intelligent RSU.

SmartEdge will accelerate sensor fusion and collaborative perception at the RSU to enable the provision of time-critical services for ITS. It will initially focus on accelerating the integration of multi-modal and multi-agent data at the RSU. This integration involves combining raw or processed radar data from the roadside, raw or processed LiDAR data from smart vehicles, and GPS information from smart and dummy vehicles to create a comprehensive data repository. Smart vehicles can query this dynamic repository to enhance their situational awareness and the RSU itself can utilize it as well. The integration of camera data is considered a stretched goal for future consideration, and the primary focus lies in accelerating this data fusion process.

SmartEdge will leverage sensor fusion and collaborative perception results to develop time-critical and resource-intensive services such as collision avoidance. This will be achieved, for example, by controlling traffic lights and other infrastructures, as well as communicating with swarm's nodes. The aim is to harness the capabilities of intelligent RSUs to improve transportation safety and efficiency.

5.3.3 LiDAR Processing Acceleration

The use of LiDAR in the intersection RSUs creates a unique opportunity. These devices naturally combine visual and positional information that would otherwise need to be captured via a fusion of camera and radar sensors. As an added benefit, the data generated by LiDARs, called "point cloud", provides features enough to distinguish between different vehicle types while not allowing privacy-sensitive features---faces or license plates---to be distinguished.

The point cloud, however, needs to be processed to make vehicle recognition possible. Moreover, this processing must occur within the RSU to avoid lengthy transmission times to prevent taking prompt actions, as Use Case 2 requires. In this task, we intend to provide infrastructure to process the point cloud data in an edge device. We are designing a system to efficiently extract features from the LiDAR stream (e.g., bounding boxes of moving objects along with their velocity and reflexivity). The features can feed an object recognition machine learning model that determines the vehicle type of each moving object [Lin18]. Our emphasis is to deliver a language to express how to compute statistical, geometrical, and temporal features out of point cloud data and its corresponding runtime. Focusing on a language rather than pre-selected and hard-coded features allows the flexibility to cover many traffic model scenarios beyond Use Case 2 in the future.

In practice, the expressions over this language (i.e., the features specifications) are executed by a runtime we will also provide. This runtime must be adapted to the computational restrictions of the RSU. Unfortunately, The CPU capacity is that unit is not adequate to deliver the results on time, given the amount of processing necessary to extract useful information from the network stream produced by the LiDAR. Therefore, instead of generating a software binary from the features specifications, we will provide a runtime implemented on an FPGA within the RSU unit that is capable of evaluating the latter. Because the number of features to extract is low, the FPGA area necessary for this runtime should be adequate for the power and physical dimensions of the RSU unit. We are currently experimenting with a low-power Xilinx FPGA unit that is compatible with the restrictions.

6 CONCLUSIONS

This document introduces the initial architecture for the Dynamic and Secure Swarm Networking Work Package (WP4). The document combines both planned parts of the architecture, as well as components that were also prototyped.

As shown, the three components of the Work Package: automatic discovery and dynamic network swarm formation in near real time, embedded network security and isolation and hardware-accelerated in-network operations for context-aware networking are intertwined. Each task provides innovative components towards the project's overall goals, while also contributing novel solutions to the wider community.

The preliminary evaluation of the prototypes shows progress made toward the project's KPI, and in particular:

- Ultra-low latency, including <1µS on a switch, <1ms in container, and ~2ms on a Raspberry Pi running an ML-driven in-network defense solution.
  These are significant steps toward KPI K3.2.
- Varying between datasets, ML-driven in-network defense solution achieves accuracy of >99% and F1>94% for known attacks and 90%-99% accuracy with F1>94% for new attacks.
  These are significant steps towards KPI K3.3.
- Demonstrated distributed in-network computing across 1000 nodes, in-line with expected number of devices/vehicles to be considered in the SMARTEDGE scenarios. This is a significant step towards KPI K3.4.

As WP4 progresses, the architecture will keep evolving as the SmartEdge research questions are investigated in greater depth. It is in the nature of research and innovation projects that new techniques will be investigated and trialed, and better options are chosen. It is therefore important that the architecture provides a solid basis for developing and implementing the innovative features of SmartEdge, without being too proscriptive so as to restrict WP4 growth.

7 REFERENCES

[Bar21] Barefoot Networks, "Version 1.0 Switch Architecture Model. Website," https://github.com/p4lang/p4c/blob/main/ p4include/v1model.p4. 2021.

[Bar23] L. Barsellotti, L. De Marinis, F. Cugini, F. Paolucci, "FTG-Net: Hierarchical Flow-To-Traffic Graph Neural Network for DDoS Attack Detection", HPSR Conference 2023.

[Bas20] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic In-band Network Telemetry," In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 662–680. 2020.

[Blö21] M. Blöcher, L. Wang, P. Eugster, and M. Schmidt, "Switches for HIRE: Resource Scheduling for Data Center In-Network Computing. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems," 268–285. 2021.

[Bos14] P. Bosshart et al., "P4: Programming protocol-independent packet processors," SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, p. 87–95, jul 2014.

[Bus19] C. Busse-Grawitz, R. Meier, A. Dietmuller, T. B ¨ uhler, and L. Vanbever, ¨ "pForest: In-network inference with random forests," arXiv preprint arXiv:1909.05680, 2019.

[Dan20] H. Tu Dang, P. Bressana, H. Wang, K. Suh Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, "P4xos: Consensus as a Network Service," IEEE/ACM Transactions on Networking 28, 4 (2020), 1726–1738. 2020.

[Del23] DELL Technologies, "Dell EMC Edge Gateway 5200 software user's guide," 2023. [Online]. Available: https://dl.dell.com/content/ manual77941191-dell-emc-edge-gateway-5200-software-user-s-guide.

[Esk20] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, "Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices," IEEE Internet of Things Journal, vol. 7, no. 8, pp. 6882–6897, 2020.

[Fer22] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications: Centralized and federated learning." IEEE Dataport, 2022.

[Gur21] V. Gurevich and A. Fingerhut, "P4-16 Programming for Intel Tofino Using Intel P4 Studio," In P4 Workshop, 2021.

[Hin21] H. Hindy, C. Tachtatzis, R. Atkinson, E. Bayne, and X. Bellekens, "Developing a siamese network for intrusion detection systems," in Proceedings of the 1st Workshop on Machine Learning and Systems, ser. EuroMLSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 120–126.

[Hog22] M. Hogan, S. Landau-Feibish, M. Tahmasbi Arashloo, J. Rexford, and D. Walker, "Modular Switch Programming Under Resource Constraints. In 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)," USENIX Association, Renton, WA, 193–207. https://www.usenix.org/conference/nsdi22/presentation/hogan. 2022.

[Int21] Intel, "P4_16 Intel® Tofino™ Native Architecture – Public Version," https://github.com/barefootnetworks/OpenTofino/blob/master/PUBLIC_Tofino-Native-Arch-Document.pdf. 2021.

[Jia20] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, “Flowguard: An intelligent edge defense mechanism against iot ddos attacks,” IEEE Internet of Things Journal, vol. 7, no. 10, pp. 9552–9562, 2020.

[Jin17] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, “NetCache: Balancing Key-Value Stores with Fast In-Network Caching,” In Proceedings of the 26th Symposium on Operating Systems Principles. 121–136, 2017.

[Kol20] R. Kolcun et al., “The case for retraining of ML models for iot device identification at the edge,” arXiv preprint arXiv:2011.08605, 2020.

[Kon15] J. Konecnˇ y, B. McMahan, and D. Ramage, “Federated optimization: ` Distributed optimization beyond the datacenter,” arXiv preprint, 2015.

[Lao21] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, “ATP: In-network Aggregation for Multi-tenant Learning,” In 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). 741–761. 2021.

[Lak21] S. Laki, R. Stoyanov, D. Kis, R. Soule, P. V ´ or¨ os, and N. Zilberman, ¨ “P4Pi: P4 on Raspberry Pi for networking education,” SIGCOMM Comput. Commun. Rev., vol. 51, no. 3, p. 17–21, jul 2021.

[Li20] J. Li, J. Nelson, E. Michael, X. Jin, and D. R. K. Ports, “Pegasus: Tolerating Skewed Workloads in Distributed Storage with in-Network Coherence Directories,” In Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI’20). USENIX Association, USA, Article 22, 20 pages. 2020.

[Lin18] Z. Lin, M. Hashimoto, K. Takigawa, and K. Takahashi, "Vehicle and Pedestrian Recognition Using Multilayer Lidar based on Support Vector Machine," 2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), Stuttgart, Germany, 2018.

[Lou22] F. Loui, C. Mate, A. Gall, and M. Wisslé, “Project Overview: Router for Academia Research & Education (RARE) ,” 2022.

[Mad22] S. Maddock, G. Cormode, T. Wang, C. Maple, and S. Jha, “Federated boosted decision trees with differential privacy,” in CCS ’22, 2022.

[Pal16] K. Palani, E. Holt, and S. Smith, “Invisible and forgotten: Zero-day blooms in the iot,” in 2016 IEEE International Conference on Pervasive Computing and Communication Workshops, 2016, pp. 1–6.

[Par18] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, “A survey on low latency towards 5g: Ran, core network and caching solutions,” IEEE Communications Surveys & Tutorials, vol. 20, no. 4, pp. 3098– 3130, 2018.

[Pas21] A. Pashamokhtari, N. Okui, Y. Miyake, M. Nakahara, and H. H. Gharakheili, “Inferring connected IoT devices from ipfix records in residential ISP networks,” in 2021 IEEE 46th Conference on Local Computer Networks (LCN), 2021, pp. 57–64.

[Ped11] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[Sha18] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP, 2018, pp. 108–116.

[Sir22] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, “Re-architecting traffic analysis with neural network interface cards,” in NSDI 22, 2022, pp. 513–533.

[Sto18] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu, “Debugging P4 Programs with Vera,” In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 518–532. 2018.

[Ten17] C.-C. Teng, J.-W. Gong, Y.-S. Wang, C.-P. Chuang, and M.-C. Chen, “Firmware over the air for home cybersecurity in the internet of things,” in 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), 2017, pp. 123–128.

[Zan22] M. Zang, C. Zheng, R. Stoyanov, L. Dittmann, and N. Zilberman, “P4pir: in-network analysis for smart iot gateways,” in Proceedings of the SIGCOMM’22 Poster and Demo Sessions, 2022, pp. 46–48.

[Zan23-1] M. Zang, C. Zheng, L. Dittmann, and N. Zilberman, “Towards Continuous Threat Defense: In-Network Traffic Analysis for IoT Gateways,” IEEE Internet of Things Journal, 2023.

[Zan23-2] M. Zang, C. Zheng, T. Koziak, N. Zilberman, and L. Dittmann, "Federated Learning-Based In-Network Traffic Analysis on IoT Edge", IFIP Networking 2023 Sec4IoT Workshop, June 2023.

[Zhe22] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, “Automating In-Network Machine Learning,” arXiv preprint arXiv:2205.08824, 2022.

[Zhe23-1] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, “In-Network Machine Learning Using Programmable Network Devices: A Survey,” IEEE Communications Surveys & Tutorials, pages 1–1, 2023.

[Zhe23-2] C. Zheng, B. Rienecker, and N. Zilberman, “QCMP: Load Balancing via In- Network Reinforcement Learning,” In Proceedings of the 2nd ACM SIGCOMM Workshop on Future of Internet Routing & Addressing, pages 35–40, 2023.

[Zhe23-3] C. Zheng, H. Tang, M. Zang, X. Hong, A. Feng, L. Tassiulas, and N. Zilberman. DINC: Toward Distributed In-Network Computing. Proceedings of the ACM on Networking, 1(CoNEXT3):1–25, 2023.

[Zhe23-4] C. Zheng, H. Tang, M. Zang, X. Hong, A. Feng, L. Tassiulas, and N. Zilberman, “DINC - GitHub Repository,” https://github.com/In-Network-Machine-Learning/DINC. 2023.

[Zhe24] C. Zheng, Z. Xiong, T. T Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, “IIsy: Hybrid In-Network Classification Using Programmable Switches,” IEEE Transactions on Networking, 2024.

[Zho19] Y. Zhou, J. Bi, C. Zhang, B. Liu, Z. Li, Y. Wang, and M. Yu, “P4DB: On-the-fly Debugging for Programmable Data Planes,” IEEE/ACM Transactions on Networking 27, 4 (2019), 1714–1727. 2019.