UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

Master Degree in TELECOMMUNICATION ENGINEERING: SMART SENSING, COMPUTING AND NETWORKING

Improving the Quality of Federated Learning using Programmable Networks

Candidate:

Ahmad MAHMOD

Matricola: **2**34454

Supervisors:

Prof. Pasquale PACE **Prof. Antonio IERA**

ACADEMIC YEAR 2022 /2023

Declaration of Authorship

I, Ahmad MAHMOD, declare that this thesis titled, "Improving the Quality of Federated Learning using Programmable Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

AHMAD MAHMOD

Abstract

An emerging modality, increasingly utilized by edge devices for training machine learning models in a distributed and collaborative manner, is Federated Learning (FL). FL offers a unique blend of improved learning quality and the imperative need for data privacy. Despite the numerous advantages of this emerging paradigm, there exists a critical factor that could significantly impact its efficiency in future 5G and 6G application scenarios. This factor relates to potential delays arising from the limited communication resources available for connecting clients to servers, which could excessively slow down the process and diminish its effectiveness, especially in light of the new real-time applications that characterize 5G and 6G scenarios. To address this challenge, the thesis introduces a novel approach to client selection. Unlike the various communication streamlining methods proposed thus far, this approach begins with a networking research perspective and leverages the capabilities of Software-Defined Networking (SDN). It focuses on the dynamic selection and continuous updating of clients participating in the Federated Learning (FL) process. This innovative approach ensures that the distributed learning process remains highly effective and efficient, resulting in an overall reduction in the FL process time under varying network traffic loads. This effectiveness is demonstrated through a performance evaluation campaign conducted using an implemented testbed platform.

Acknowledgements

I am deeply grateful to Prof. **Pasquale Pace** and Prof. **Antonio Iera** for their unwavering support and invaluable guidance throughout the development of this thesis. Their extensive experience in the field, coupled with their dedication to nurturing my academic growth, has been instrumental in shaping the trajectory of my research. I want to take this opportunity to extend my deepest gratitude:

To the pillars of my life, **my father and mother**, who not only provided unwavering support but also instilled in me the values of perseverance and determination. Their sacrifices and encouragement were instrumental in reaching this milestone in my academic journey.. . .

To my siblings, **Wafaa, Samar, Sawsan**, and **Ali**, you are the backbone of my life and the source of support.. . .

To my dearest friend, **Hanan**, your presence during both the challenging and joyous moments of this journey has been a source of strength and comfort.. . .

To my closest companions, **Hani, Abd Alrahman, Bashar,** and **Adeeb**, thank you for your unwavering support and camaraderie throughout this academic endeavor. Our shared experiences have enriched my life in ways I cannot express.. . .

To the friends of childhood **Ali Bassam**, **M. Kartish**, **Khaldoun**, **Basel**.. . .

To my roommate and fellow student in Italy, **Issam**, I am grateful for the shared experiences of living and studying abroad. Your friendship has made this journey more enjoyable and memorable.. . .

To the sweatheart and best colleague of my master's class **Mahmoud Ezzat**. . .

To the friends I had the privilege of meeting during my studies in Italy - **Ghinwa, Jana, Yara, Mohamamd** and **Abd Almonaem**. Your friendship added a unique dimension to my time abroad.. . .

To the classmates **Graziano, Alessandro, Andrea, Giussepe, Darwin**, and **Francesco**.. . .

To all **professors and teachers** that contributed to my knowledge during my MSc study at this programs in The University of Calabria and during my previous Bachelor's Degree in Tishreen University.

Contents

List of Figures

List of Tables

List of Abbreviations

SDN	Software Defined Network
Open vSwitch	OVS
Open Flow	OF
OpenDaylight	ODL
Virtual Machine	VM
QoS	Quality of Service
Restful Configuration	RESTCONF
Network Configuration	NETCONF
Internet of Things	IoT
Machine Learning	ML
Delay/Computational-Resources Selection Strategy	DCSS
Delay Selection Strategy	DSS

Chapter 1

Introduction

Federated Learning (FL), first introduced by Google [1], has garnered increased attention in the Distributed Machine Learning literature across various industries. The fundamental idea behind FL is to keep data from different sources, referred to as 'FL Clients,' locally without transmitting it to a central party for training specific Machine Learning Models. FL enables these data sources to train the model solely on their data and then send only the model's weights to the central aggregator server, known as the 'FL Server' (Client-Server mode) or to other clients (Peer-to-Peer mode), ensuring data privacy. The weights from different FL Clients are aggregated using a specific mechanism to update the global model, which is subsequently used in the next training round for all clients until the training process is completed.

FL is considered a significant player in the future of 6G Networks. FL inherently relies on the resources of the network that connects FL Clients and the FL Server. Consequently, network resource scarcity may pose challenges for FL, particularly in time-sensitive applications like e-Health and Monitoring, where delay and data loss can significantly degrade performance. Even in applications less sensitive to loss and delay, such as IoT applications, managing network resources can enhance network robustness and efficiency.

Existing literature has extensively investigated communication issues in FL, primarily from a learning perspective, with a focus on streamlining between FL Clients and FL Server, such as overhead reduction and efficient client selection. On the other hand, some literature has emphasized selecting clients based on computational resources and memory, crucial for expediting the training process, often overlooking the importance of rapidly transferring trained model weights through available network resources.

This thesis work addresses one of the most recent topics in the literature, "Network for AI," which complements the traditional approach of "AI for Network." The work proposes SDN-assisted FL, where the network becomes programmable and easier to manage with the goal of reducing training time while maintaining performance levels. The proposed framework introduces a new entity called the 'FL Orchestrator,' responsible for mediating interactions between the FL Server and the SDN Controller and selecting the best clients based on the defined scenario.

The proposed framework comprises three main phases:

1. **Dynamic Routing**: This phase selects the most free routes, avoiding congested routes between FL Clients and the FL Server, which may become overloaded due to other network applications or FL data during different FL training rounds.

2. **Delay-based Selection Strategy (DSS)**: The work suggests a Selection Mechanism that chooses clients with the least delay to participate in each training

round, assuming that clients are homogeneous in terms of computational resources and memory. This means they can complete training equally, but they may not be able to deliver their results to the FL Server in the same amount of time due to congestion.

3. **Delay/Computational Resources Selection Strategy(DCSS)**: When clients become heterogeneous (typically the case in real scenarios), their ability to train and deliver data to the FL Server varies. This phase proposes a Client Selection Mechanism that selects the most performant client based on both Delay and Computational Resources.

SDN's role extends beyond dynamically updating routes; it also maintains a record of the delay profile of each client in the FL process round by round. This information is later delivered to the FL Orchestrator, which uses it in client selection.

The results of the thesis can be summarized as follows:

- Dynamic Routing enhances the performance of the FL process in terms of convergence time compared to static routing by avoiding congested routes. However, it has a limitation in that it cannot address the issue of all routes being completely congested or overloaded.
- Delay-based Client Selection addresses the limitations of Dynamic Routing by excluding clients that are heavily overloaded. However, it is most effective when the clients are homogenous in terms of computational and memory resources.
- Delay/Computational Resources Selection proves to be the best choice as it takes into account both route congestion and the computational and memory resources of the clients.

To assess the feasibility of the proposed framework and mechanisms, I conducted a testbed using a Fat Tree topology with various overloading profiles, as explained in detail in Chapter 4, to evaluate different aspects of the framework. The work encompasses multiple chapters, starting with the theoretical foundations of related topics and concluding with the testbed implementation and results.

Chapter 2 introduces the concept of SDN and compares it to traditional networks, followed by an explanation of the OpenFlow protocol, which connects the SDN Controller with network devices. The chapter also presents the OpenDaylight SDN Controller and the OpenV Switch, both of which were used in the testbed. It concludes with an assessment of the benefits and disadvantages of SDN.

In Chapter 3, the work introduces FL and highlights its differences compared to centralized Machine Learning. It explores various types of FL (vertical, horizontal, and transferred FL), provides insights into some FL applications, and discusses privacy mechanisms in FL. The chapter concludes by summarizing the benefits and challenges of FL.

Chapter 4 introduces some existing FL frameworks and showcases SDN implementations in FL. It identifies gaps in the existing literature that the proposed framework aims to address. The chapter then provides an in-depth introduction to the proposed framework.

Chapter 5 offers a step-by-step implementation of the testbed, starting with an explanation of the network's topology and insights into the choices made in the proposed topology. It then delves into the running applications on top of the SDN Controller, including Dynamic Routing, Delay-based Selection Strategy (DSS), and

Delay/Computational-Resources Selection Strategy (DCSS). The chapter also explains the FL Server and FL Client applications. Finally, it presents the Deep Learning Model in use (DenseNet121 with additional Dense Layers) and the dataset (CIFAR10).

Chapter 6 focuses on evaluating each of the proposed algorithms and provides a detailed analysis. It begins with an assessment of Dynamic Routing, followed by an evaluation of the Delay-based Selection Strategy (DSS) with a set of homogeneous clients. The chapter concludes with an evaluation of the Delay/Computational-Resources Selection Strategy (DCSS) with a set of heterogeneous clients. The chapter closes with a general conclusion based on the obtained results.

Chapter 2

Software Defined Network and OpenFlow

2.1 Introduction

Software Defined Networking (SDN) is an emerging approach that decouples the data plane (forwarding plane) and the control plane in network architectures. The data plane is responsible for forwarding packets in network devices, while the control plane defines how the data plane should handle packet forwarding. In traditional networks, network devices such as switches, routers, or access points typically integrate both the control and data planes, leading to increased complexity as the network scales. Figure 2.1 shows the difference between traditional networks and SDN.

SDN introduces a centralized logic that enhances network management programmability and flexibility by providing a global view of the network through a central device called the controller [2]. This migration of control to a central controller simplifies the network devices, transforming them into simple data-plane devices [3]. Various SDN controller platforms are available, including RYU [4], OpenDaylight [5], and ONOS [6].

The concept of SDN has evolved over the years, with initiatives such as SOFTNET (1988) [7], Active Networking (1990) [8], OPENSIG (1995) [9], and NETCONF (2006) [10] contributing to its development. However, the real birth of SDN was facilitated by the introduction of OpenFlow in 2006 [11]. OpenFlow defined a standardized method for managing the connection between network devices and the

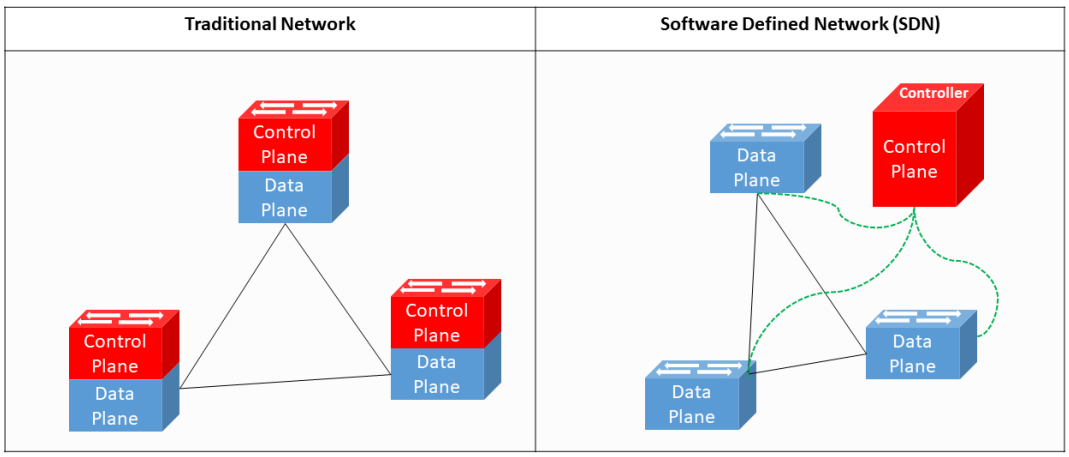

FIGURE 2.1: Traditional network versus SDN

controller, enabling the installation of control plane commands on these devices. Devices that support OpenFlow include OpenFlow tables that store forwarding rules and provide an Application Programming Interface (API) to expose device functionality to the controller.

Overall, SDN revolutionizes network management by separating the control plane from the data plane and centralizing control, leading to improved programmability, flexibility, and scalability in network architectures.

This chapter delves into the SDN architecture, OpenFlow Protocol, OpenDaylight SDN Controller, Open vSwitch OF Switch, SDN benefits, and SDN challenges to provide a comprehensive understanding of this programmable networks approach.

2.2 SDN Architecture

The basic architecture of SDN consists of three main planes: the Data Plane, Control Plane, and Application Plane. These planes interact with each other to deliver the required network services. The Data Plane includes all network devices (e.g., switches and routers) responsible for collecting network information, such as topology and traffic statistics, and forwarding data in the network based on the rules pushed by the control plane.

The Control Plane interfaces the Application Plane with the Data Plane, interacting with the application plane through the Northbound API and with the Data Plane through the Southbound API. Applications running on the Application Plane can receive network information provided by the network devices and make decisions in the form of rules, which they can then push to the network devices through the Control Plane. In wide networks, the Control Plane typically includes multiple controllers that interact through the Westbound and Eastbound APIs, enabling them to exchange network information and coordinate their decisions [12].

The Application Plane hosts running applications to provide the required services by configuring the network devices and receiving information from them through the Control Plane. Many typical applications can be run on the SDN Application Plane, such as firewalls, proxies, access control mechanisms, routing, and load balancing [13].

2.2.1 Data Plane

The main difference of the data plane network devices in SDN is that these devices have now migrated their control plane functionality to the centralized controller. In traditional networks, each network device includes its own control plane, which consists of a routing protocol to find paths to other devices and a forwarding algorithm to forward packets based on this routing information.

With SDN, the network devices are simplified and made more cost-effective because they only need to support basic forwarding mechanisms managed by the controller. The control plane intelligence is moved to the SDN controller, which makes the devices less complex and reduces their costs.

There are several ways to implement network devices in SDN:

- **Virtual switch on normal Operating System (OS)**: In this approach, virtual software switches, such as Open vSwitch, can be run on a regular PC OS like Linux. These virtual switches enable multiple Virtual Machines (VMs) on a single physical device to be seen as different switches in the control plane, providing flexibility and scalability.

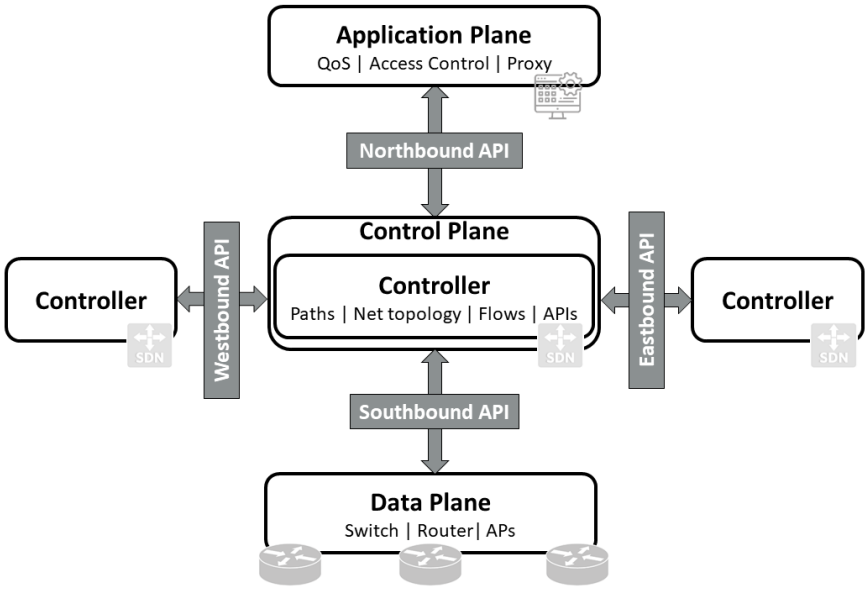

FIGURE 2.2: SDN Architecture

- **Switch over Open Hardware**: This implementation allows for vendor-independent functionality of network devices, making it suitable for experimental evaluations. It provides higher performance compared to software-based implementations.
- **Vendor-dependent switch**: Some vendors have started providing firmware updates for their traditional network devices to enable SDN functionality. This allows existing devices to support SDN without the need for hardware replacements.

These different ways of implementing SDN network devices offer various trade-offs in terms of performance, flexibility, and cost, allowing network administrators to choose the most suitable option based on their specific requirements and infrastructure.

2.2.2 Control Plane

The control plane plays a vital role in SDN, serving two main functions: controlling the network by forwarding rules to the network devices in the Data Plane and monitoring the network status through interactions with the network devices. High-Level Programming Languages, such as Python, are commonly used for programming the control plane.

Regarding rule management, rules can be updated either based on time or events. Time-dependent updates involve updating the rules for a specific duration, while event-dependent updates occur when specific network events happen. The Control Plane must maintain an updated and consistent view of the network status to provide this information to running applications. This includes the Global View of the network and various statistics about the traffic, such as time duration, packet size, and loss ratio.

In multi-controller SDN environments, one of the main challenges is synchronizing the distributed controllers. Various approaches have been proposed to facilitate communication between the controllers, such as publish-subscribe mechanisms and

the creation of a dedicated SDN for managing controller communication. These synchronization methods are essential to ensure that the controllers work coherently and efficiently, enabling them to collectively manage the network effectively. Overall, the control plane's role in SDN is crucial for providing centralized management and control of the network, leading to improved flexibility, scalability, and programmability compared to traditional network architectures.

2.2.3 Application Plane

The programmability of SDN provides applications with high flexibility in configuring the network by updating rules in network devices. This flexibility is made possible by the global view of the physical network and the ability to virtualize it according to the specific service requirements. Numerous applications can be implemented in the Application Plane, offering a wide range of network management and optimization capabilities:

- **Traffic Engineering**: This application optimizes routing, resource allocation, and energy consumption to ensure the desired Quality of Service (QoS). It can efficiently manage network traffic flows and ensure the best utilization of available resources.
- **Network Monitoring**: Network monitoring applications perform traffic inspection for troubleshooting, security assurance, and network analysis. By monitoring network traffic, administrators can detect and address issues promptly.
- **Virtual Network Slicing**: Virtual network slicing is particularly relevant in 5G networks, where it allocates different network resources for different services. It enables the efficient sharing of physical network resources among multiple virtual networks, each tailored to specific service requirements.
- **Load Balancing**: Load balancing applications distribute traffic across available links to provide optimal resource utilization and QoS. It ensures that no single link becomes overloaded, leading to a more efficient and reliable network.
- **Security and Access Control**: SDN can be used to detect security threats and manage access control policies. By centralizing security management, SDN applications can better protect the network from potential threats.

Many other application may be defined demonstrating the powerful of SDN approach supported by the global view, dynamic reconfiguration and virtualization of the network.

2.3 OpenFlow Protocol

OpenFlow Protocol [6] is one of the enabling approaches of SDN, which standardized the communication between the control plane and the data plane on the southbound interface. The network devices that support this protocol are called "OpenFlow Switches" where, the OpenFlow architecture consists of three main components: OpenFlow switch, Secure Channel, and Controller [7] as shown in figure 2.3.

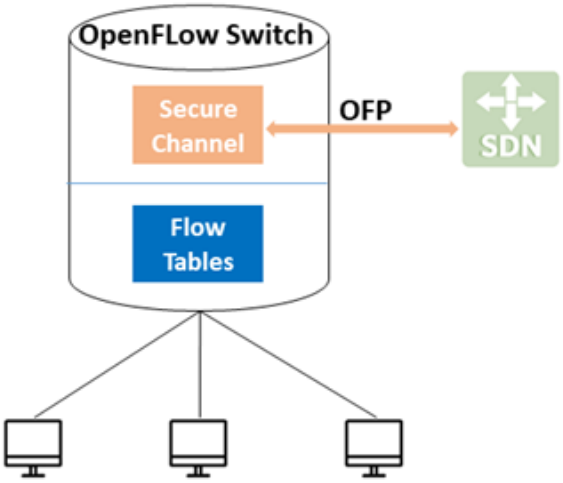

FIGURE 2.3: OpenFlow Components

2.3.1 OpenFlow Switch (OF Switch)

OpenFlow Switches (OF Switches) consist of two main components (as shown in figure 2.4): Flow Tables that have multiple rules called "Flow Entries," and a Secure Channel or Channels to communicate the data plane of the OF Switch with the control plane of the controller or controllers. The OpenFlow protocol defines different types of packets to be exchanged between the controller and OF Switch over the Secure Channel to control the entries (delete, modify, add, etc.) of the tables in the OF Switch. The tables are handled as a pipeline where the packet has to traverse one by one until finishing the pipeline. Each table consists of a set of entries (see 2.3.3), which includes matching fields, counters, and actions. A packet executes the actions of one entry when the packet matches the matching field of that entry. When a packet matches many entries of one table, the most prioritized entry wins. Eventually, the actions of different tables are collected in one "Action Set" and executed. An OF Switch may be in one of two types:

- **OF-only Switch**: support only the OF operations, i.e., the packet only go through the tables pipeline.
- **OF-hybrid Switch**: support the OF operations in addition to the normal Ethernet switches functions.

2.3.2 OpenFlow Ports

The ports enable the OF Switches to connect by sending messages over ingress ports and receiving over the output ports [14]. The ports are classified into three types:

- **Physical Ports**: These ports correspond to the physical network interfaces in the OF Switch.
- **Logical Ports**: These ports do not correspond to the physical network interfaces in the OF Switch. They may be defined by non-OpenFlow methods, such as Tunnels or loopbacks.
- **Reserved Ports**: These ports are reserved for specific forwarding actions, such as forwarding packets to the controller. The reserved ports are listed in table 2.1.

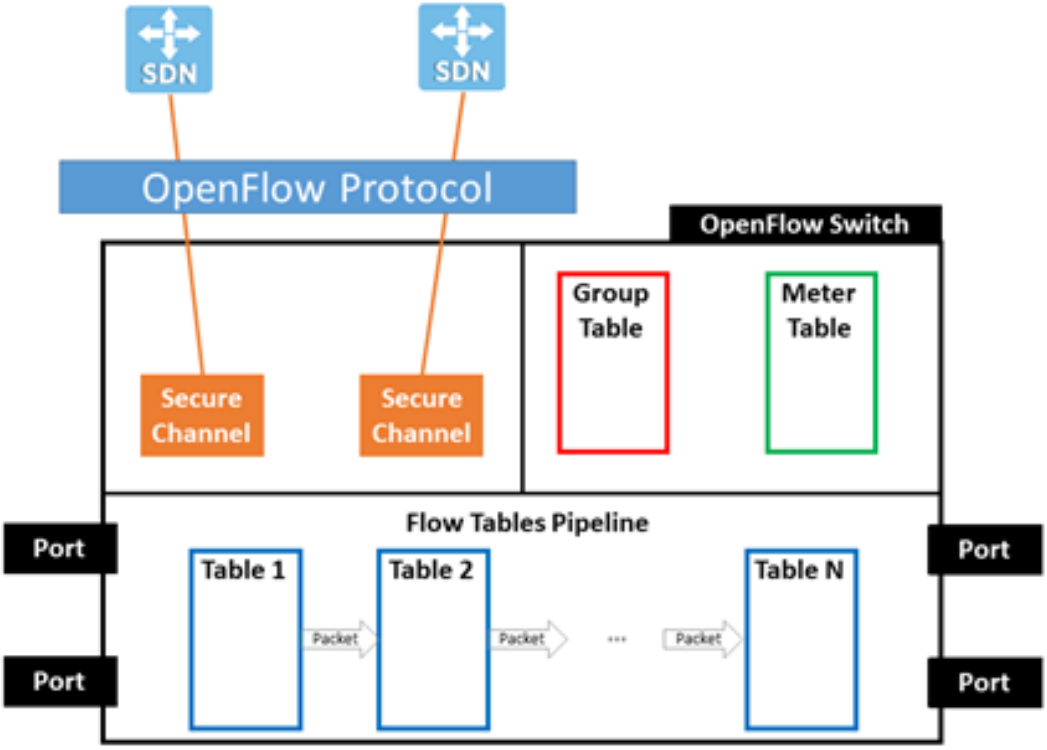

FIGURE 2.4: OF Switch Components

TABLE 2.1: OpenFlow Reserved Ports

Port	Function
ALL	Send to all port except input port
CONTROLLER	Send the packet to the Controller
TABLE	Send the packet to a specific table
INPORT	The packet ingress port
ANY	Send to any port of set of ports
UNSET	The output port has not been seen
LOCAL (optional)	The switch itself
NORMAL (optional)	Act as normal (non-OF) switch
FLOOD (optional)	Send to all switches

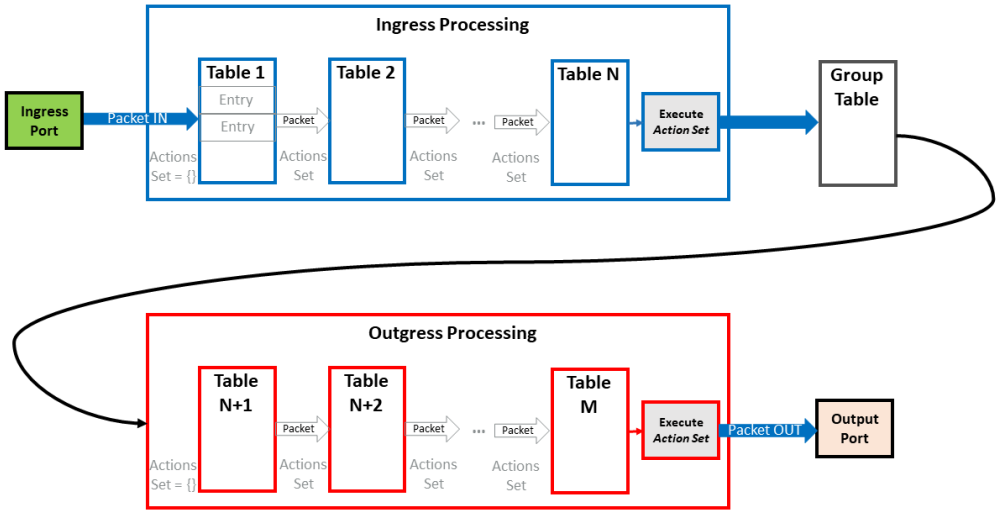

FIGURE 2.5: Flow Tables Pipeline

An OF Switch can modify the ports' configuration at any time, including adding, removing, and changing the port status.

2.3.3 OpenFlow Tables

OpenFlow has three types of tables:

- **Flow Table**: This table includes multiple entries that a packet may match against. Each entry consists of matching fields, counters, and actions. A packet traverses through multiple flow tables in a pipeline, and the actions of matching entries are executed accordingly.
- **Group Table**: After a packet is processed in a Flow Table, it may be forwarded to the Group Table to implement specific actions related to a group of packets. Group tables enable collective forwarding or processing of packets based on certain criteria.
- **Meter Table**: The Meter Table includes actions related to the performance of flows. It is used for traffic metering and enables the implementation of Quality of Service (QoS) and rate limiting mechanisms. The metering actions in this table help regulate the flow's bandwidth or enforce traffic rate limits.

Flow Tables

Flow Tables in OpenFlow are processed as a pipeline, and they are numbered starting from "0". The pipeline has two main processing phases: ingress processing and egress processing. These phases are separated by the Group Table, as illustrated in Figure 2.5.

Each Flow Table is composed of multiple entries, and each entry contains various components, including matching fields, priority, counter, timeout, instructions, and cookie [14], as shown in Table 2.2.

TABLE 2.2: Flow Entry

Matching Fields	Priority	Counter	Timeout	Instructions	Cookie

TABLE 2.3: OpenFlow Instructions

Instruction	**Function**
Outport *portNum*	Send the packet to a specific port
Group *groupId*	Direct the Packet to a specific Group Table
Drop *(empty instructions)*	Drop a packet
Meter *meterId*	Direct the Packet to a specific Meter Table
Set-Field *type, value*	Change a specific field in the packet's header
Copy-Field	Copy a value from the packet to the pipeline

- **Matching Fields**: are the conditions that indicate if a packet match an Entry of not. Some parameters used in Matching Fields:
  - *IPv4 source/destination address*
  - *IPv6 source/destination address*
  - *MAC source/destination address*
  - *ARP IPv4/IPv6/MAC source/destination address*
  - *Input Port*
- **Priority**: helps in choose one Entry in case of more than one matching a table. A number between 0 and 65,535, the higher the value the higher the priority.
- **Counter**: is incremented whenever a packet match the entry
- **Timeout**: to control the lifetime of an Entry
- **Instructions**: defines the actions that should be implemented in case of matching and are added to the Action Set of the packet. The instructions may add action, modify the Actions Set or change the control of the pipeline. Table 2.3 shows some of the instructions.
- **Cookie**: are used to have some statistics.

Group Tables

Group Tables were introduced in OpenFlow 1.1 and later versions to offer more flexibility and reduce duplication in the tables. A group table consists of group entries, where each group entry contains four fields (as shown in table 2.4):

- **Group Identifier**: This field uniquely identifies the entry within the group table.
- **Group Type**: Specifies the type of the group, such as all, select, indirect, and fast failover. Each type defines different behaviors for packet processing (table 2.5).
- **Counters**: These are incremented when packets match the corresponding group entry.

TABLE 2.4: Group Entry

Group Identifier	Group Type	Counters	Actions Buckets

TABLE 2.5: Group Table Entry Types

Group Type	**Function**
All	Execute all Actions Buckets
Indirect	Execute the only one bucket in the group
Select	Execute only one bucket from the group of defined buckets
Fast Failover	Execute the first live-port bucket

- **Actions Buckets**: Each group entry contains a set of action buckets. Each bucket has a set of actions that define how to process the packets. The buckets are ordered, and they may be executed totally or partially depending on the Group Type.

Group Tables provide more flexibility and enable dynamic packet processing, load balancing by defining a set of output ports to distribute the load, and help in reducing overhead in the flow tables. With Group Tables, network administrators can create more sophisticated forwarding behaviors and achieve better traffic management in SDN networks.

Meter Tables

Group Tables were indeed introduced in OpenFlow 1.3 and later versions. They offer Quality of Service (QoS) control of the flows, allowing for actions such as limiting the packet rate of flows. The metering functionality is flow-based, meaning that each flow may include a meter action in its Action Set. A meter measures the packet rate of the flow and controls its rate using Meter Bands, with each band handling a specific rate and defining how to process the matching packets (e.g., modify packet rate, DSCP remark, or drop).

Each Meter Entry in the Group Table consists of three fields (as shown in table 2.6):

- **Meter Identifier**: This field uniquely identifies the entry within the Meter Table.
- **Meter Bands**: Each Meter Entry includes one or more bands, and each band is responsible for handling packets within a specific rate range. The bands define the specific actions to be taken for the matching packets.
- **Counters**: The counters in the Meter Entry are incremented whenever packets match the corresponding metering rules.

With the introduction of Meter Tables, SDN controllers gain the ability to implement QoS policies, rate limiting, and other traffic shaping mechanisms, providing better control over the network's traffic and ensuring that different flows receive the desired quality of service.

TABLE 2.6: Meter Entry

Meter identifier	Meter Bands	Counters

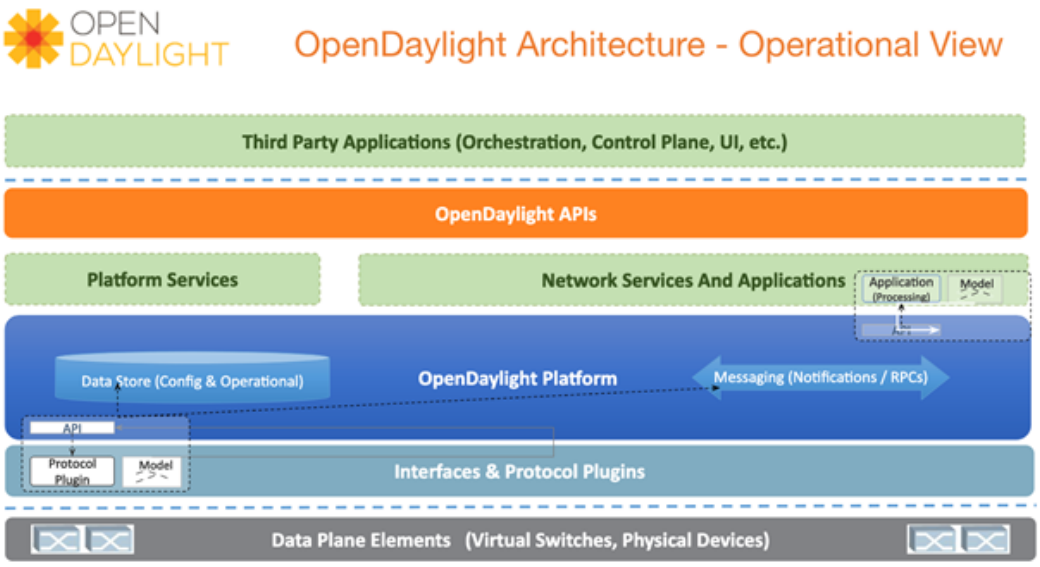

FIGURE 2.6: ODL General Architecture [15]

2.4 OpenDaylight Controller

OpenDaylight (ODL) Controller [15] is an SDN controller that enables agile and programmable control of SDN. ODL adopts a modular approach through *plugins* that define the protocols and applications/services running on the controller.

The general architecture of ODL (as depicted in Figure 2.5) consists of three main layers: the Southbound (SB) Layer plugins, the Service Adaptation Layer (SAL), and the Northbound (NB) Application/Service Functions. The SAL's primary task is to interface the SB plugins that connect the network's data plane (network devices) to the NB Functions plugins. On top of the ODL controller, third-party applications can be defined to consume data provided by the NB Functions, such as the Topology Exporter, through APIs to make decisions that control and manage the network [9].

Initially, the SAL was API-Driven (referred to as API-Driven SAL) but faced scalability challenges. As a result, a new Model-Driven SAL was introduced. In the Model-Driven approach, a data/service Provider offers services through APIs, and data/service Consumers consume one or more data/services through these APIs (e.g., OpenFlow plugin).

The Model-Driven approach has gained increased attention in the networking field due to its simplicity in defining network services, protocols, policies, and applications [16]. To support the Model-Driven approach in ODL, the NETCONF [17] and RESTCONF [18] protocols have been used:

- **NETCONF Protocol**: A key enabler of SDN, it allows the ODL Controller to manage and control network devices and retrieve information from them. NETCONF defines configuration and operational data stores and a set of operations to access these data stores for creation, retrieval, deletion, or modification. RPC (Remote Procedure Call) is used by NETCONF to deliver its notifications, and XML is used to encode/decode the data in data stores.

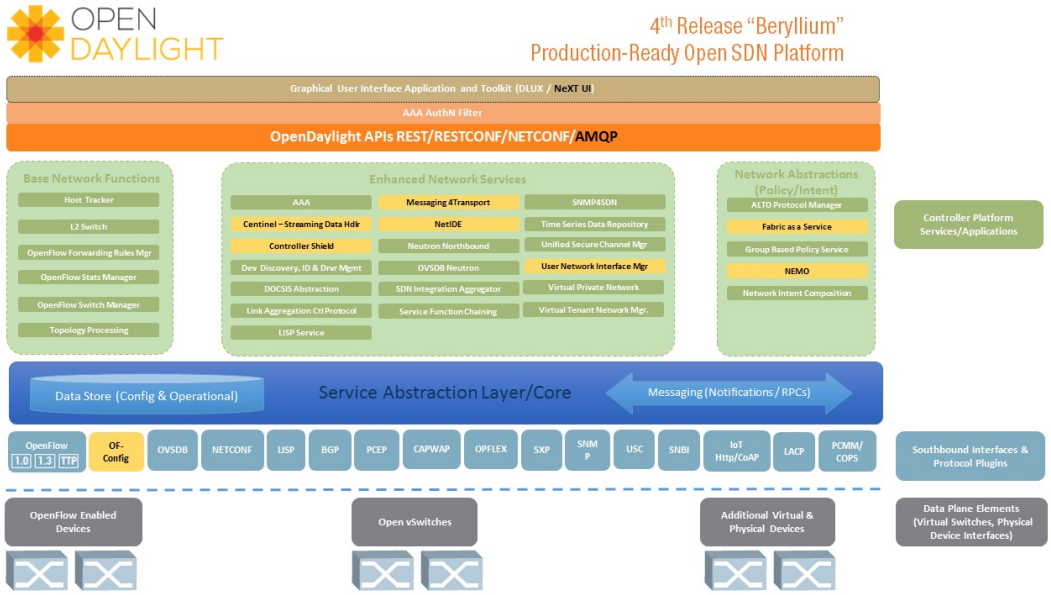

FIGURE 2.7: ODL Beryllium Architecture [15]

- **RESTCONF Protocol**: This is a REST-based protocol that leverages the HTTP protocol to expose network data stores defined by NETCONF through HTTP GET methods. It also enables the configuration and modification of data stores through HTTP DELETE, POST, and PUT methods. Both XML and JSON are used for data encoding/decoding.

ODL comes in different versions, and for our thesis, we use OpenDaylight Beryllium 0.4.3 (Figure 2.7), which is the fourth version of ODL and introduced many SB plugins (such as OF-Config) and NB service/application Functions.

2.5 Open Vswitch

Open vSwitch [19] is a versatile software switch capable of handling both Layer 2 and Layer 3 functionalities of the OSI model, enabling it to manage Ethernet switching and IP routing. It provides a software platform with standardized management and forwarding interfaces, allowing for traffic forwarding programmability and control. OVS is specifically designed as a virtual switch that can be seamlessly integrated into various Linux-based virtualization technologies such as KVM and VirtualBox. It efficiently connects multiple virtual machines (VMs) by managing virtual ports and also extends its management capabilities to physical ports. Moreover, OVS supports the OpenFlow protocol, which enables centralized SDN controllers to effectively manage the switch and modify its flow rules.

The architecture of OVS, illustrated in figure 2.8, comprises several essential components:

- **ovs-vswitch**: This daemon is responsible for implementing OVS in collaboration with the OVS Kernel Module, which operates within the Linux Kernel. The first packet is handled by ovs-vswitchd to define the rule for subsequent packets. The Kernel Module then efficiently processes the subsequent packets in the User Space, reducing the need for repetitive processing.

- **ovsdb**: OVS utilizes a database tool called ovsdb.

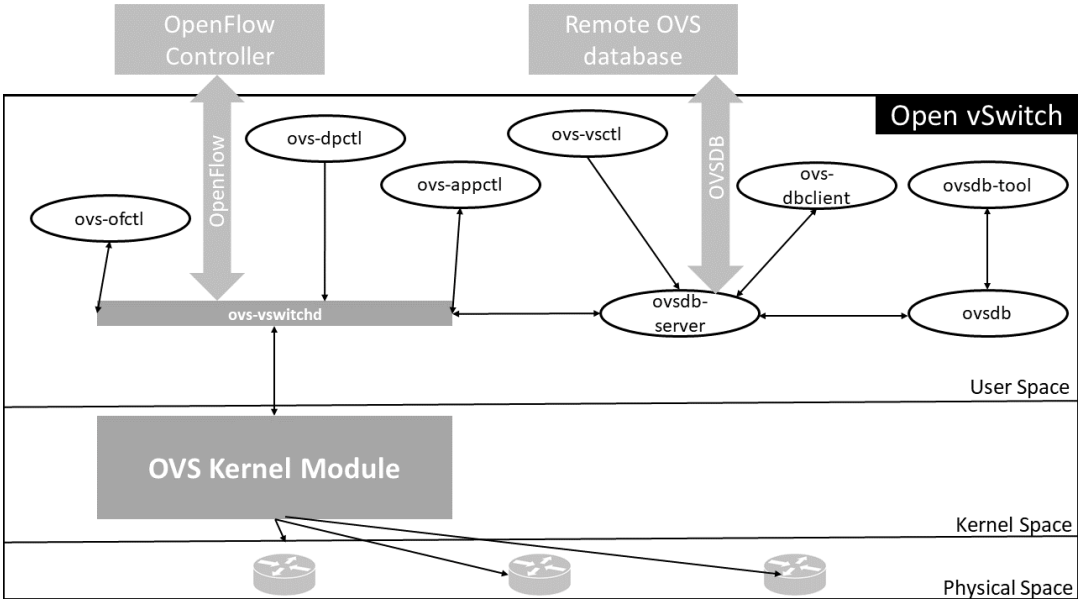

FIGURE 2.8: OVS Architecture

- **ovsdb-server**: This database server receives queries from ovs-vswitchd to obtain the necessary configuration information.
- **ovs-dpctl**: This tool is utilized to configure the OVS Kernel Module.
- **ovs-vsctl**: It is a tool used for sending and receiving configurations to and from ovs-vswitchd.
- **ovs-appctl**: This utility allows the sending of commands to the OVS daemon.
- **ovs-ofctl**: This utility is employed to send and receive control commands related to OpenFlow and Controllers.

OVS can identify multiple ovs-switchd instances, but only one instance may run at a time to manage all OVS instances. Each instance includes multiple bridges (datapaths), which may be defined within a single machine. The configuration of OVS is retrieved from ovsdb, which operates in a client-server model using RPC. OVS operates through two channels:

- **OpenFlow Channel**: It facilitates the control and configuration of flows and the exchange of network information.
- **OVSDB Channel**: This channel allows external management of OVS by configuring ovs-vswitchd through OVSDB.

OVSDB provides a range of functionalities, such as:

- Creating, editing, and deleting OpenFlow datapaths (bridges);
- Creating, editing, and deleting OpenFlow datapath ports;
- Creating, editing, and deleting datapath tunnel interfaces for OpenFlow;
- Creating, editing, and deleting ports' queues;
- Configuring a set of controllers to which the OpenFlow datapath connects;

- Configuring QoS policies;
- Collecting statistics.

2.6 SDN Benefits

The decoupling of the Data Plane and Control Plane in SDN has brought significant control and flexibility to the network through programming and reconfiguration. By centralizing control, SDN enables robust and real-time management of the network, resulting in optimized configuration and enhanced performance [20]. Here are some of the key benefits of SDN:

- **Configuration Enhancement**: In traditional networks, configuring new devices can be challenging and error-prone due to the distributed control. SDN, on the other hand, provides a unified and centralized control plane, offering greater flexibility, dynamicity, and control over network configuration.
- **Performance Improvement**: The heterogeneity of network devices in traditional networks makes overall management and control difficult. SDN's centralized control allows for integrated and centralized configuration, leading to improved network performance.
- **Innovation Encouragement**: SDN's programmability and software-based nature enable easy experimentation with new applications and services in testbed environments, fostering innovation and rapid development.
- **Security Improvement**: SDN enables centralized security policies and access control, making it easier to manage and enforce security measures across the network. Additionally, network segmentation and traffic isolation help mitigate security threats.
- **Global Network View**: With a centralized controller, SDN provides a comprehensive view of all network devices, surpassing the limited information available in traditional networks that are often limited to neighboring devices.
- **Virtualization and Network Slicing**: SDN allows for virtualization of network resources and segmentation based on traffic types, enabling network slicing to cater to the diverse requirements of different services, which is especially valuable in 5G networks and IoT environments.

Overall, the central control, programmability, openness, and automation introduced by SDN offer numerous benefits that surpass traditional approaches, which were often constrained by proprietary devices and distributed control mechanisms. SDN's capabilities pave the way for more efficient, flexible, and innovative network management and services.

2.7 SDN Challenges

The programmability and centralized control of SDN have introduced many benefits. However, on the other hand, they have also introduced some challenges in terms of security, scalability, reliability, and performance [20]. We will discuss them in this section.

- **Controller-Device Communication Overhead**: SDN requires continuous interaction between the controller and network devices in both directions. The network information has to be sent from the network devices to the controller to maintain an up-to-date network state and a global view. Conversely, the controller has to update the rules and flows of the network devices to maintain and match service requirements. The continuous communication in both directions introduces an overhead to SDN that may become a challenge in bandwidth-limited communication networks.

- **Single Point of Failure**: Having the control plane centralized in the controller makes the controller a vital component of the network, where any malfunctioning or error in the controller may bring down the entire network. To address this issue, the controller has to be resilient and reliable.

- **Security Threats**: Having security issues or threats in the central control plane will affect the whole network, making the controller a preferred point of attack for hackers. In addition, network programmability makes the network vulnerable to many new types of attacks.

- **Scalability**: As the network grows, the controller has to support a larger number of devices, rules, and information that may affect its performance. The capacity of the controller is limited, which may limit the network scalability. To address this issue, a multi-domain SDN can be used, having many controllers, one for each domain. Furthermore, having a multi-domain SDN makes the control and management more complex.

In conclusion, even though the new paradigm of SDN, supported by the programmability and centralized control, has brought revolutionary benefits, it has also introduced some challenges that need to be solved and are considered open issues and an active research gap.

Chapter 3

Federated Learning

3.1 Introduction

Federated Learning (FL) was initially proposed by Google [1] as a solution to address a critical problem in Machine Learning (ML) - the preservation of clients' privacy [21]. In many scenarios, data needs to be collected from multiple distributed sources, such as mobile devices, and then centrally used to train ML models. However, this centralized approach poses a security threat to user privacy as the data may be vulnerable to leaks. FL, on the other hand, allows each device, referred to as an FL Client, to locally train a version of the ML model using its own data. The locally-trained model's parameters are then sent to a central server, known as the FL Server, which aggregates the local models and updates the global model [22]. By training the data locally, FL ensures user data privacy and reduces the computational burden on the FL Server. As a result, FL is an efficient approach for enabling low-cost ML on edge devices like mobile phones and sensors [23]. Figure 3.1 illustrates the difference between traditional centralized ML training and FL training.

FL encompasses three main types, each depending on the type of local data among the FL Clients: Horizontal FL, Vertical FL, and Federated Transferred Learning.

While FL brings several enhancements to ML, certain challenges require further investigation and resolution. For instance, the heterogeneity of participating FL Clients [24] and the specifications of communication resources that connect the FL Clients to the FL Server need careful consideration. Additionally, there are concerns about the FL Server acting as a single point of failure and the potential security risks associated with it.

This chapter delves into the FL architecture, types, applications, benefits, and challenges to provide a comprehensive understanding of this cutting-edge approach.

3.2 Federated Learning Process

The Federated Learning (FL) process involves a group of N clients, denoted as $U = \{u_1, u_2, ..., u_n\}$ out of which M clients are selected to participate in the FL process using a Selection Strategy $S = \{s_1, s_2, ..., s_m\}$. A global model, denoted as w', is defined and initialized in the FL Server and then transmitted to all the selected clients S. These selected clients then train the global model on their respective local data, resulting in the generation of local models w_m, where m ranges from 1 to M. Consequently, the FL process can be summarized in the following steps (as depicted in figure 3.2):

1. Initialize the global model w'.

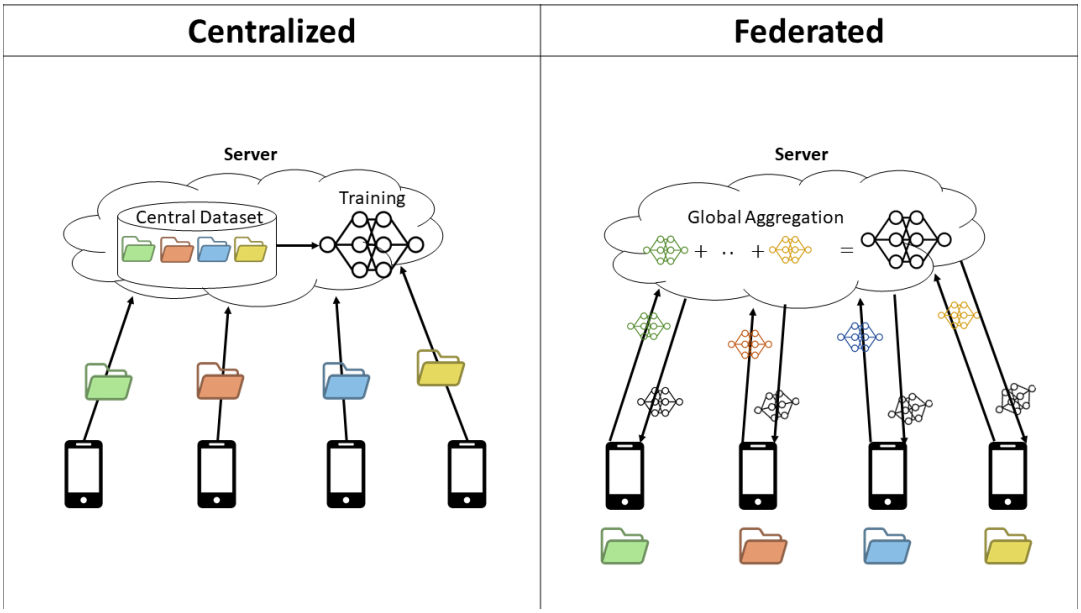

FIGURE 3.1: Centralized ML versus FL Architecture

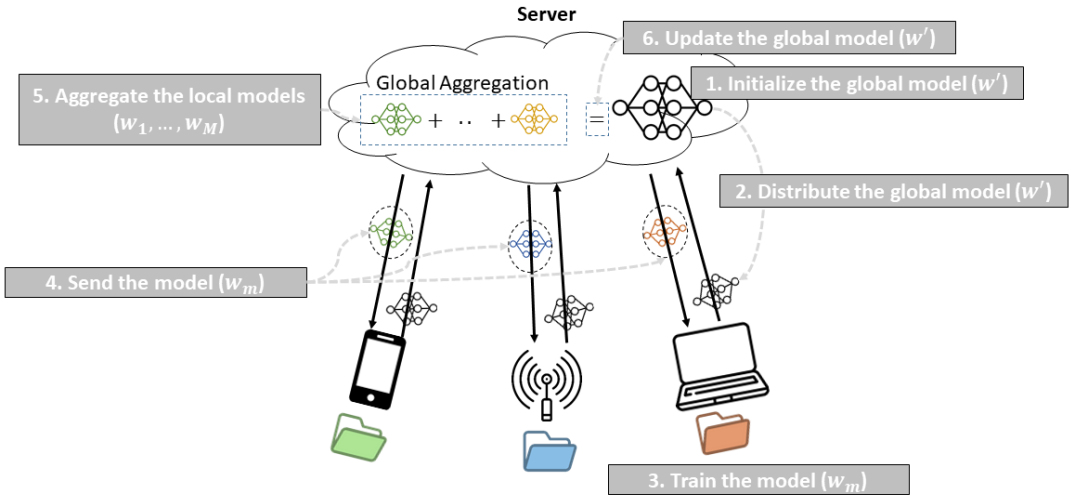

FIGURE 3.2: FL Process

2. Distribute the model to the selected clients S.
3. Train the model locally in each selected client w_m.
4. Send each local model back to the server.
5. Aggregate the local models at the server.
6. Update the global model and return to step 2 for further iterations.

In the FL process, several research works aim to find optimal solutions for different aspects, including the selection strategy to choose the clients, the aggregation methods to combine the local models, and the data preparation techniques for local training. These efforts are essential to enhance the efficiency and effectiveness of the FL process, making it more suitable for various practical applications.

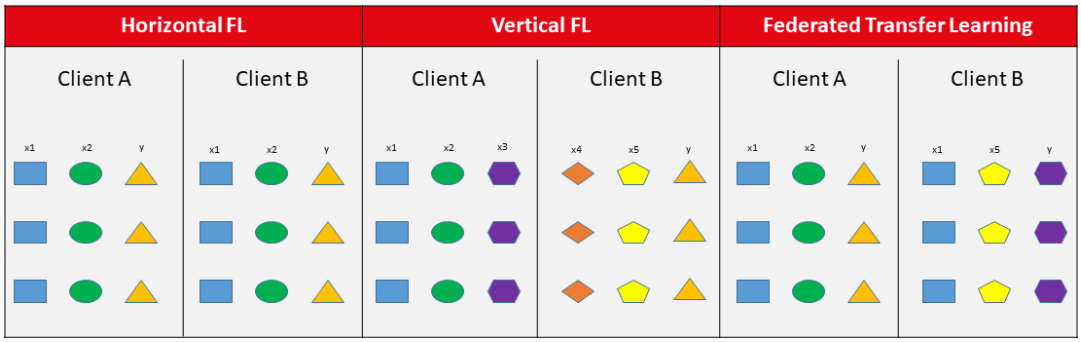

FIGURE 3.3: FL Types

3.3 Federated Learning Types

According to the data distribution between the clients, FL can be categorized into three main types [25]: Horizontal, Vertical, and Transferred FL, as depicted in figure 3.3.

Horizontal FL

In this type of FL, known as Horizontal FL, all clients share the same data features. For example, in a healthcare FL model, hospitals act as FL Clients, and they all have local data stored with the same types of features (e.g., medical image types). Horizontal FL allows the participant FL Clients to divide the total dataset among themselves. This approach expands the available dataset for training across multiple clients, enhancing the accuracy of the model.

An illustrative example of Horizontal FL is seen in Google's Android Model update. Each Android user, representing an Android Client, is allowed to train the model on its local data and then upload the model parameters to a central Android cloud, which serves as the FL Server. This way, the global model is updated by aggregating the locally trained models from multiple Android users.

Figure 3.4 depicts the architecture of Horizontal FL, showcasing how the data is distributed among the FL Clients and how the global model is updated through the aggregation of the locally trained models.

In Horizontal FL, the sharing of data features among clients enables collaborative learning while preserving data privacy and enhancing the overall performance of the FL process.

Vertical FL

In Vertical FL, the clients have different data features that are integrated together to enhance the data feature dimensions. For example, consider a movies purchase website and a movies evaluation website, both of which intersect in most of their visitors. Each website has data that may enhance the commerce of the other. The movies purchase website data about the purchases of different movies is a significant feature that the other website does not possess. On the other hand, the movies evaluation website may provide evaluations to the other website to make advertisements on the most demanded movies. The clients exchange their trained data in order to introduce the features of the others. Figure 3.5 depicts the architecture of Vertical FL.

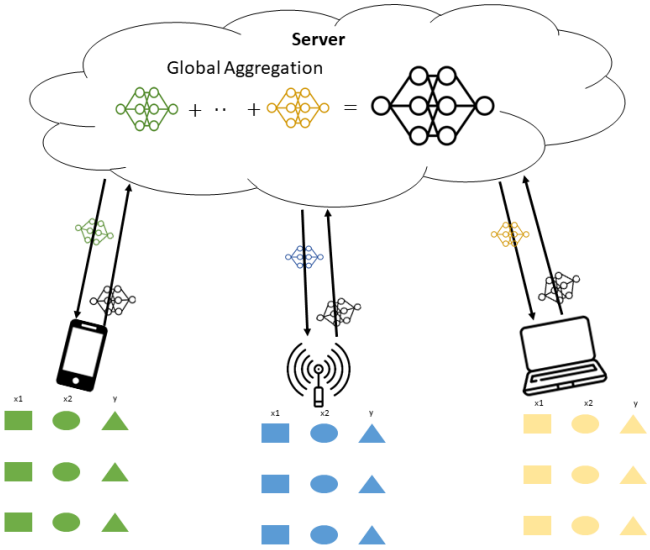

FIGURE 3.4: Horizontal FL

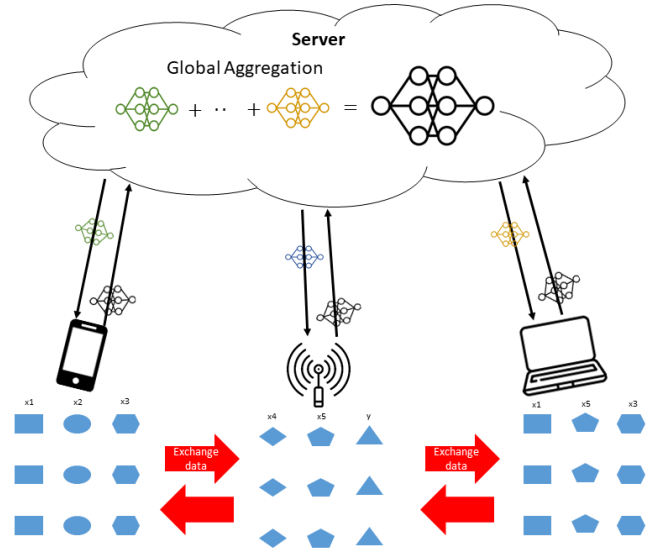

FIGURE 3.5: Vertical FL

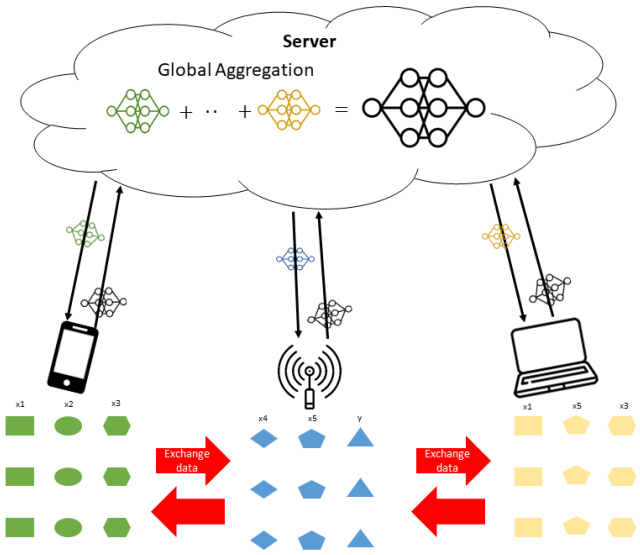

FIGURE 3.6: Federated Transfer Learning

Federated Transfer Learning

FTL (Federated Transferred Learning) has been proposed in [26] as a combined version of FL and Transferred Learning. In this approach, a pre-trained model is distributed among the FL Clients, which, in turn, fine-tune the model on their local dataset to customize it. This allows users with limited datasets to contribute with faster convergence and better generalization while maintaining privacy. For example, the movies evaluation website's general model can be transferred to a specific user who has his personal evaluations and preferences, and the general model can be customized to this user by training the model on the user's local data. In conclusion, FTL is used when the datasets have different features in addition to different sample spaces. Figure 3.6 depicts the architecture of FTL.

3.4 Federated Learning Applications

FL has been utilized in various applications to leverage the distributed nature of datasets in clients while ensuring user privacy. Here are three popular applications of FL:

Google Virtual Keyboard (Gboard)

Gboard is a virtual mobile keyboard that offers services such as next-word prediction, automatic correction, and word completion. In the FL-based approach proposed in [27], a server waits for a specific number of mobile clients before initiating the next round of training. Additionally, in another paper [28], Gboard's FL application was extended to predict emojis based on typed text using Recurrent Neural Networks (RNN).

Healthcare

FL is widely used in healthcare applications, where data collected by different hospitals or medical centers for specific diseases may be small. By implementing an FL server, hospitals act as FL clients, training their local version of the global model to preserve user privacy. Having multiple clients improves model accuracy, which is crucial in medical applications. Healthcare applications leveraging FL include predicting patient hospital stays [29] and detecting changes in the brain related to Parkinson's Disease using speech signals [30].

IoT Applications

The rapid growth of data in edge devices in IoT applications has posed challenges for centralized ML due to limited communication resources and increased delays. FL has emerged as a promising solution, allowing edge devices to train ML models locally without sharing their data with a remote/cloud server. FL ensures privacy, reduces network congestion and latency, and improves training accuracy. FL has found applications in areas such as FL for Unmanned Aerial Vehicles (UAVs) in 5G/6G networks [31] and FL for Intelligent Transportation Systems (ITS) [32].

3.5 FL Privacy Mechanisms

FL provides privacy preservation by keeping the data of FL clients locally and only sharing information about the training results. However, even these results may contain some private information. To address this concern, FL employs multiple privacy mechanisms, with the most common ones being Differential Privacy, Robust Aggregation, and Homomorphic Encryption and Anomaly Detection.

- **Differential Privacy (DP)**: DP prevents the prediction of FL client's data based on statistical characteristics by adding random noise to the transmitted data before aggregation [33]. Although DP may result in some statistical information loss, it ensures the security and privacy of the data.
- **Secure Multi-party Computation (SMC)**: SMC enables multiple parties to jointly compute a function while keeping their inputs private. In FL, SMC is used to preserve client privacy, especially during the aggregation phase. Google's approach [34] uses SMC to aggregate parameters from multiple clients by averaging.
- **Homomorphic Encryption (HE)**: HE allows data to be processed without decryption. In FL, it is used in the aggregation process, enabling the FL server to aggregate training parameters without decrypting the raw data. Xu et al. [35] proposed a double-masking protocol using HE to preserve FL client privacy.
- **Robust Aggregation**: Robust aggregation detects and removes faulty or unreliable training parameters from FL clients. This mechanism ensures that only reliable contributions are considered in updating the global model [36].
- **Anomaly Detection (AD)**: Anomaly detection identifies and discards malicious or poisoned training parameters from participating clients. AD techniques have been developed to detect deviation and honesty of clients' responses [37] [38].

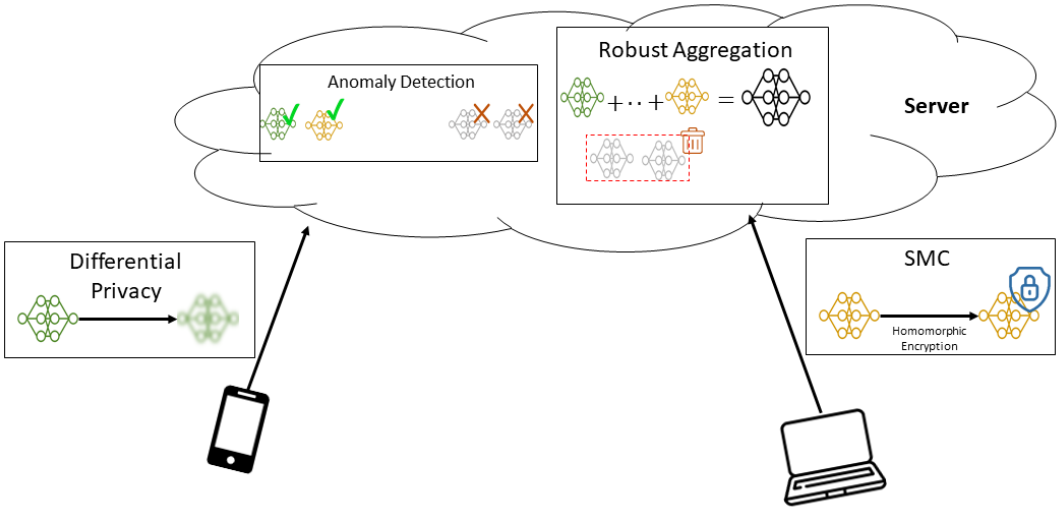

FIGURE 3.7: FL Privacy Mechanisms

Figure 3.7 illustrates how these privacy mechanisms are integrated in FL. Homomorphic Encryption can support Secure Multi-party Computation alongside Differential Privacy on the client side. On the server side, both Robust Aggregation and Anomaly Detection are utilized to ensure secure and privacy-preserving aggregation.

3.6 Federated Learning Benefits

After discussing the various aspects of Federated Learning (FL), it is evident that FL offers several benefits for machine learning applications, particularly in terms of security, learning efficiency, and energy and communication efficiency. The key advantages of using FL are as follows:

- **Privacy Preservation**: FL allows training models locally on client devices, ensuring that sensitive data is kept locally and not transmitted over the network. This approach protects the privacy of individual users' data and mitigates the risk of data breaches during transmission.
- **Data Security and Confidentiality**: Since FL only shares training parameters instead of raw data, the risk of security attacks during data transmission is significantly reduced.
- **Reduced Communication Bandwidth**: Transmitting training parameters instead of raw data significantly reduces the required communication bandwidth, making FL more efficient in communication-constrained environments.
- **Learning Generalization**: Training models on diverse client devices with different environments and characteristics enhances the generalization capabilities of the learning model, reducing bias and increasing its overall performance.
- **Real-time Learning and Adaptation**: With FL, local devices can perform real-time training on their own data, enabling rapid learning and adaptation to changing conditions.

- **Efficiency and Scalability**: Distributing the learning process among FL clients reduces the workload on the central server, enhancing the overall efficiency and scalability of the system. FL can handle a large number of clients without putting excessive strain on the server.
- **Eco-friendly Solution**: By reducing the load on central servers and cloud infrastructure, FL reduces the power consumption and carbon footprint, making it a more eco-friendly solution for machine learning applications.

In conclusion, Federated Learning provides a secure and efficient approach to machine learning by preserving privacy, reducing communication overhead, and promoting real-time learning and scalability. It is an ecologically friendly solution that addresses the challenges of data security and confidentiality in modern distributed environments.

3.7 Federated Learning Challenges

Federated Learning (FL) faces several challenges despite its benefits in security, communication, and privacy. These challenges include:

- **Data Heterogeneity**: Variability in data distributions, sizes, or qualities among FL Clients can impact the generalization of the global learning model. The disparities in data may lead to biased or less accurate models.
- **Unbalanced Participation**: Some FL Clients may be less active or have lower availability, leading to an imbalanced participation rate. This can affect the fairness and accuracy of the FL process.
- **Clients Heterogeneity**: FL Clients may differ in terms of CPU, memory, and communication resources. This heterogeneity poses challenges in efficiently delivering updates, as clients with powerful hardware may not be able to communicate effectively if the communication resources are limited.
- **Single Point of Failure**: Relying on a centralized FL server as the coordinator makes the server a critical point of failure. If the server is attacked or fails, the entire FL process may be disrupted.
- **Communication Overhead**: The need to maintain connections between clients and the central server introduces communication overhead. This can become burdensome, particularly in scenarios with many clients and limited communication resources.
- **Data and Model Poisoning**: FL is susceptible to attacks where malicious entities inject poisoning data into clients' local datasets or send malicious training parameters. These attacks can compromise the integrity of the global model.
- **Communication Resource Limitations**: Congestion or failure of communication links between FL clients and the server can disrupt the FL process, leading to incomplete learning rounds and potentially impacting the quality of the global model.

Addressing these challenges is crucial to ensure the effectiveness, fairness, and security of the FL process in various applications and environments. Researchers and developers continue to explore innovative solutions to overcome these obstacles and make FL more robust and reliable.

Chapter 4

Existing FL frameworks and proposed enhancements

4.1 Introduction

The Federated Learning (FL) approach has gained increased interest due to its ability to enable edge devices within networks to train Machine Learning Models locally, eliminating the need to transfer data to a centralized cloud server as required in traditional training models. FL allows edge devices to retain their data locally, thereby mitigating security threats. Additionally, it reduces the consumption of communication resources and alleviates congestion [39].

However, the heterogeneity of the clients participating in the FL process, particularly the Internet of Things (IoT) devices, has introduced a new challenge. These devices are constrained in terms of computational capacity, memory resources, and communication resources. This challenge stems from the unique nature of FL, which necessitates devices to train machine learning models locally and exchange the training models between the clients and the server. Careful consideration is required to address this issue as it can impact the Quality of Service (QoS) of the service, especially for delay-sensitive applications. The limitations in computational and memory resources also affect the time needed to train the model locally, while the limitations in communication resources affect the time needed to exchange the models.

Numerous research works have focused on enhancing the FL process from a Distributed Learning perspective. In the following sections, we will delve into some of these works and we will propose a novel SDN-based FL framework to enhance the overall system performances.

4.2 Reference works on FL client selection strategies

Tiansheng *et al.* [40] proposed an online client-selection strategy called *RBCS* to ensure fairness among clients in Federated Learning (FL). This strategy focuses on balancing the fairness factor with the efficiency of the FL process in terms of accuracy and convergence time. The motivation of the authors is the fact that a higher level of fairness leads to increased training accuracy because the model becomes generalized across a larger number of clients. However, the inclusion of poorly capable clients in terms of communication bandwidth in the proposed strategy negatively impacted the effectiveness of the FL process. The main part of *RBCS* is the Penalty Factor, denoted as V, which adjust the performance of the algorithm where, higher values of V indicate less fairness and longer convergence time and vice versa.
To validate their approach, the authors compared the results of *RBCS* (with different V values) with two other commonly used client selection methods: random selection

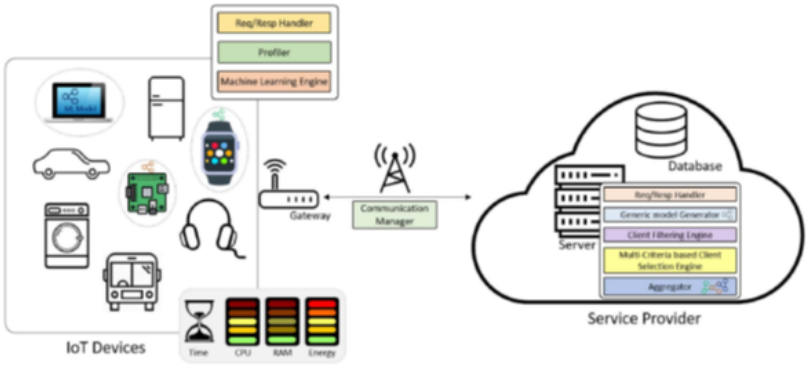

FIGURE 4.1: Framework architecture of FedMCCS [42]

and *FedCS* [41]. The results demonstrated that *RBCS* achieves comparable accuracy to the random selection method (considered the fairest method), while outperforming *FedCS*.
However, the simulation conducted by the authors overlooked the possibility of other applications' data being transferred through the network, which could significantly impact the transmission delay of the model. This could result in dropped rounds even when the bandwidth is larger compared to other clients. Additionally, the study did not establish any criteria to measure the relationship between fairness and the improvement of accuracy and convergence time. Such criteria could be useful in implementing a trade off between fairness and performance based on specific requirements.

The work of Sawsan *et al.* [42] introduces *FedMCCS*, a multi-criteria framework for IoT devices client selection in Federated Learning (FL) based on several factors of the clients such as time, energy, memory, and CPU, which are typically resource-constrained in IoT devices. The framework architecture, depicted in figure 4.1, consists of two main elements: the Service Provider and the IoT devices connected via any communication method. The Service Provider determines which ML model to use, and the clients train the model on their local data.
The authors propose enhancements to two baseline methods, *VanillaFL* [43] and *FedCS* [41], and demonstrate their effectiveness through four main points. Firstly, the target accuracy of the ML model is achieved in fewer rounds compared to *VanillaFL* and FedCS. The number of rounds required was 8.0 times less than *VanillaFL* and 8.4 times less than *FedCS* to reach 80% accuracy. Secondly, the clients' selection process shows higher efficiency, as evidenced by a higher number of client responses from the selected clients, which improves convergence time compared to [43] and [41]. Thirdly, *FedMCCS* reduces the number of discarded rounds due to a lack of client responses compared to the other two methods, where approximately half of the total rounds are discarded. Finally, there is less waste of network resources due to congestion and limited resources. This is achieved by selecting the most performing clients, despite the higher network traffic compared to *VanillaFL*, which can be overlooked due to the fewer convergence rounds.
FedMCCS outperforms the two state-of-the-art methods, but the paper suggests future work to develop the framework further. One area of improvement is studying the efficiency of selecting a client to participate in the training, which may reduce the need for the participation of other clients and lead to higher efficiency in client selection. Additionally, the paper primarily focuses on the feasibility of clients based on their computational resources, without taking into account the communication

resources, which can have a significant impact on the FL process, even if the clients are high-performing in terms of energy, CPU, and memory.

In their work [44], Wang *et al.* proposed *CMFL* (Client-based Model Feedback), a method aimed at reducing communication overhead in Federated Learning (FL). *CMFL* achieves this by preventing the transmission of irrelevant client updates, thereby reducing network usage and minimizing overhead. The idea behind *CMFL* is to provide clients with feedback containing the global tendency of the server model. This enables clients to evaluate how much their local data can contribute to this global tendency without impacting convergence time. By adopting this approach, updates are sent only by the most suitable clients, effectively preventing the transmission of updates that do not significantly contribute to the global model learning.
The validation of *CMFL* was conducted by comparing it to *VanillaFL* [43] and Gaia [45] using a CNN model (MNIST dataset) and an LSTM model (Next-Word-Prediction model). *CMFL* demonstrated a reduction in the number of rounds required to converge to a specific accuracy when compared to both methods with higher reduction in LSTM model due to the fact that it is more complicated.
To assess the efficiency of the selected clients without compromising convergence accuracy, *CMFL* was applied to a multi-task FL MOCHA framework [46]. *CMFL* demonstrated an enhancement in accuracy compared to the basic MOCHA framework, achieving similar accuracy with the same number of rounds.
Although the paper achieved promising results, *CMFL* has a weakness that lies in the threshold value, which needs to be optimized to strike a balance between overhead reduction and learning efficiency. A higher threshold value results in greater overhead reduction but potentially lower learning efficiency, and vice versa. Additionally, while the paper primarily focuses on overhead reduction, it does not consider the potential impact of communication resources on the communication between clients and the server. This aspect should be taken into account to further enhance the overall FL process.

In terms of secure and high performance FL process, in [47], Wang *et al*. proposed a client selection scheme to assess the feasibility of client contributions to the global model in Federated Learning (FL). The scheme, depicted in figure 4.2, assigns clients a reputation score and a record. The reputation score is utilized to exclude clients from the learning process if they are identified as malicious or if their data is poisoned or of poor quality, thus making a lesser contribution to the global model. The record score keeps track of the number of times a client has been excluded from participating in the global model. After a specific number of exclusions, the client is completely eliminated if proven to be involved in poisoning or making poor contributions.
The validation of the scheme was conducted by comparing it to a baseline FL scheme without client selection. Three different datasets, namely MNIST, Fashion MNIST, and CIFAR-10, were used along with two learning models. The proposed scheme outperformed the baseline scheme with varying accuracies for different models and datasets. The results showed that increasing the number of chances (the record values before eliminating the client) improved the test accuracy but resulted in longer convergence time. It was also observed that increasing the fraction of participating clients decreased the test accuracy.
Although the work demonstrated the effect of the number of chances and the fraction of participating clients, it did not propose a study to optimize their values or define their relation to learning accuracy. Furthermore, the focus of the work was on the efficiency of clients' data, while other factors such as communication resources between the server and clients, as well as the capacity of the clients, can significantly

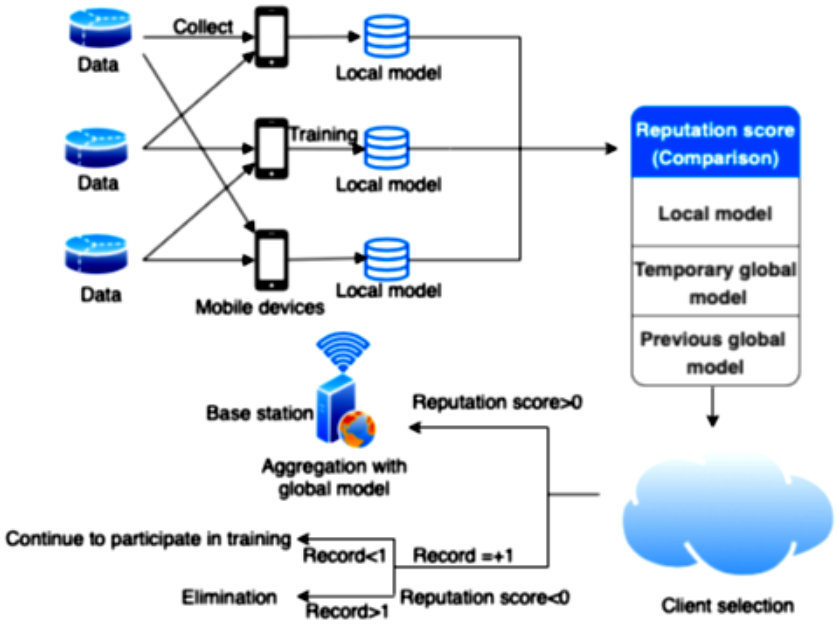

FIGURE 4.2: The proposed architecture of [47]

impact FL. These factors should be considered to comprehensively address the challenges and improve the overall FL process.

4.3 Reference works on SDN-based FL frameworks

By reviewing the previous papers, we can notice that most of the literature did not pay attention to the networking aspects of the FL process, which may have a significant performance effects especially for the time-sensitive applications. The importance of networking resources comes from the nature of FL, which in essence depends on exchanging the training data periodically between the clients and the server. On one hand, the communication resources have usually background data for many applications different to FL that introduce a congestion to the client-server connection. On the other hand, the traffic of the FL learning itself may congest each other where, the traffic of one client may be overload the shared resource with the other clients. This phenomenon will contribute to the delay of data exchange resulting in longer convergence time. For this reason, we can find that the capacity of the clients is not enough in term of computational and memory resources. The performed clients may finish the training in short time but if their communication resources suffer from a congestion or are limited, the training data may be lost or has long delay resulting in large number of rounds dropping or long rounds time. A new emerging paradigm "Network for AI" may be exploited which is complementary for the "AI for network" to fill this gap and enhance the performance of FL. One of the approach is the introduction of SDN programmable networks in FL, which still not much investigated and there are some initial interesting works. In the next section, we review some works in this field.

The emerging applications in 5G and 6G networks that use widely the FL must meet the needed QoS so, the studies now aim to obtain high quality FL in short time.

In their work [48], Eunil *et al.* proposed an auction-based strategy supported by SDN networks for FL. The system model consists of an FL Server, N FL Clients, and

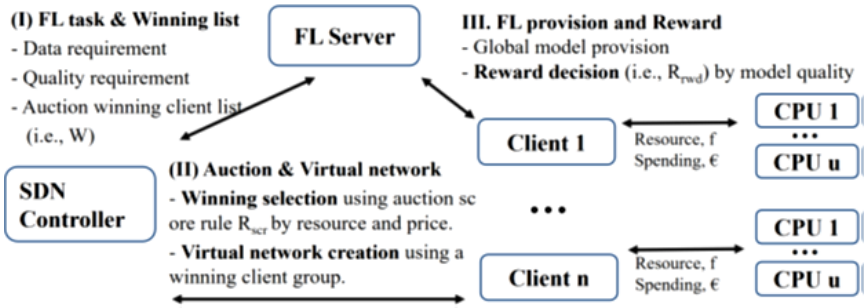

FIGURE 4.3: The system model of auction-based strategy in [47]

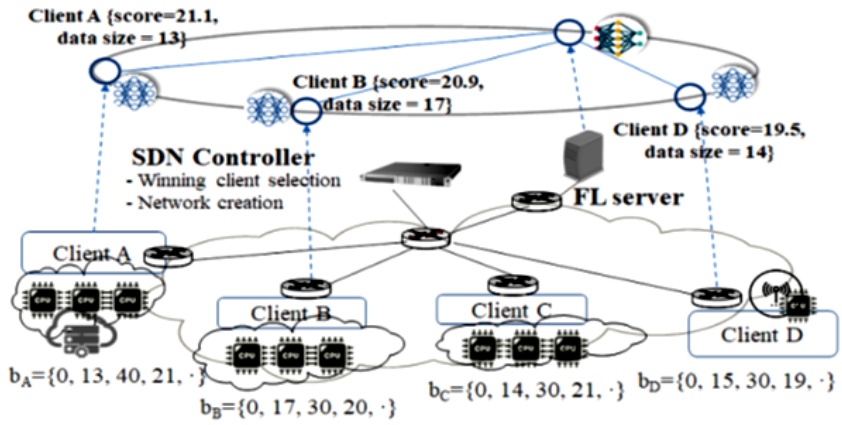

FIGURE 4.4: Overlay network of the winning clients [48]

an SDN Controller, as depicted in figure 4.3. The SDN Controller, managed by the FL Server, builds an overlay network that supports resource management. After the FL task is determined by the FL Server, the SDN Controller initiates an auction process, utilizing a Winning Function to select the best clients and a Rewarding Function to compensate the winning clients when they successfully complete the desired task. Each client sends a bid back to the server, containing its quality as per the requirements. The SDN Controller then selects the best clients and constructs an overlay network consisting of these clients, as illustrated in figure 4.4.

The proposed model was evaluated in three key areas. Firstly, the efficiency of the defined Quality Function was tested, ensuring that the quality increases as the dataset size increases and the data distribution among the clients is reduced. Secondly, the Server Utility, defined as the global model accuracy minus the reward cost, was examined. The goal was to increase accuracy while minimizing resource usage. The results showed that the Server Utility improved as the dataset size increased until reaching a specific threshold, beyond which larger data sets consumed intensive resources. Finally, the Client Utility, defined as the reward price minus the resource cost, was considered. The paper demonstrated that reducing the number of local iterations of the clients enhanced the Client Utility by lowering resource usage and associated costs.

While the usage of SDN networks in FL presents opportunities to control network and communication resources, the paper mainly focused on computational and memory resources. Further exploration could leverage SDN to incorporate communication resources into the client selection process. Despite the use of the SDN Controller, the primary focus remained on computational and memory resources.

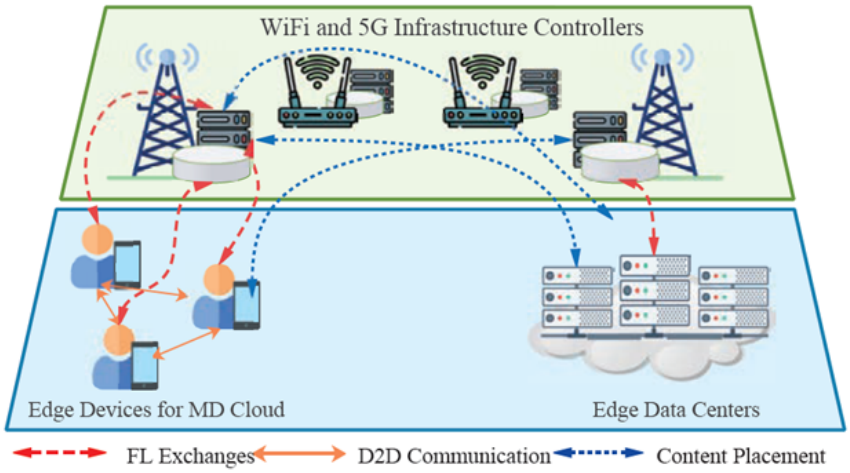

FIGURE 4.5: *FedCO* system architecture [49]

FedCO is a work proposed in [49], an SDN-based Federated Learning (FL) framework designed for IoT devices served by 5G networks. The goal of *FedCO* is to optimize data placement to ensure the required Quality of Service (QoS) by enabling seamless communication. The system structure, depicted in figure 4.5, consists of SDN Controllers collocated with each 5G NR BS (Base Station) and interconnected through an SDN architecture. The framework's controller has two main tasks: selecting the appropriate device (edge or cloud) to store the data to meet QoS requirements and aggregating the locally trained models from these edge and cloud devices. A Deep Neural Network (DNN) model is employed to update and cache the locations permanently, delivering them to the controller.
In the evaluation, the authors compared their work with two other methods: random caching [50] and *EdgeBoost caching* [51]. They measured the Cache Hit Rate (CHR), which represents the ratio of served requests to the total number of requests, and the Average Delay, which measures the time interval between sending a request and receiving the first packet of the response. The results showed that increasing the cache size improved the CHR for all methods, but *FedCO* outperformed the other methods for a specific cache size. Moreover, increasing the number of caching devices reduced the Average Delay at the expense of increased overhead in all methods, but FedCo still outperformed the other two methods due to the DNN updates and SDN Controller monitoring.
However, *FedCO* has several limitations. Firstly, the optimal placement of edge caching points needs to be further optimized to achieve the best performance. Additionally, the framework is heavily dependent on 5G networks, which are not yet widely deployed in many cities. Lastly, similar to previous papers, the potential of leveraging SDN for networking optimization has not been fully explored in this work.

In their work [52], the authors proposed an efficient slicing scheme for Federated Learning (FL) in IoT networks, supported by SDN controllers for orchestration and Network Function Virtualization (NFV) location decisions. The main concept of the scheme involves introducing IoT slicing based on the application requirements. The system scheme, depicted in figure 4.6, consists of four main elements: the centralized DNN global model, the Support Vector Regression (SVR) algorithm used for managing decentralized FL model training, slicing orchestration, and a long-term

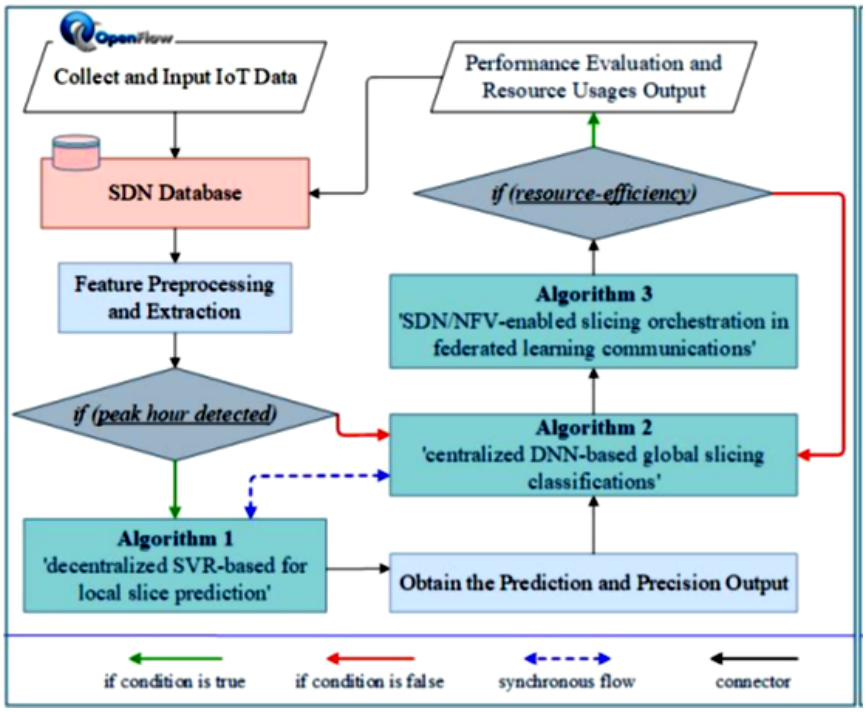

FIGURE 4.6: System architecture proposed in [52]

database for storing slicing decision efficiencies. The decentralized approach employs multiple distributed SDN controllers, which are utilized during peak hour scenarios to reduce congestion resulting from multiple connections to a central entity. During non-peak hour scenarios, the centralized model is used for prediction and heavy computational tasks. Additionally, the centralized global model periodically updates the weights of distributed devices during off-peak hours to match the most up-to-date model.
In the evaluation, the authors compared their proposed work with two approaches: Single SDN Controller (SSDNC) and Multiple SDN Controllers (MSDNC). The results demonstrated a reduction in delay and packet loss compared to MSDNC, with even greater reduction when compared to SSDNC. Furthermore, the proposed scheme exhibited improved learning performance in terms of accuracy and loss, outperforming both MSDNC and SSDNC.
Although the work leveraged SDN as an NFV orchestrator for slicing and finding the shortest paths, it did not investigate the potential use of SDN features to optimize client-to-server communication. This could involve checking congestion and applying load balancing techniques, which might introduce further enhancements to the FL process.

An SDN-based Distributed IoT architecture is proposed in [53], as illustrated in figure 4.7. The primary objective is to enable IoT gateways to implement distributed learning and Federated Learning (FL) as an application of distributed learning at the network edge while maintaining the Quality of Service (QoS) for IoT devices. The architecture comprises four main layers: the Cloud Layer, the Distributed SDN Controllers in the Control Plane, the Data Plane with network devices (e.g., switches and routers) performing tasks such as caching, executing Control Plane commands through the Forwarding Engine, and implementing distributed intelligence via the

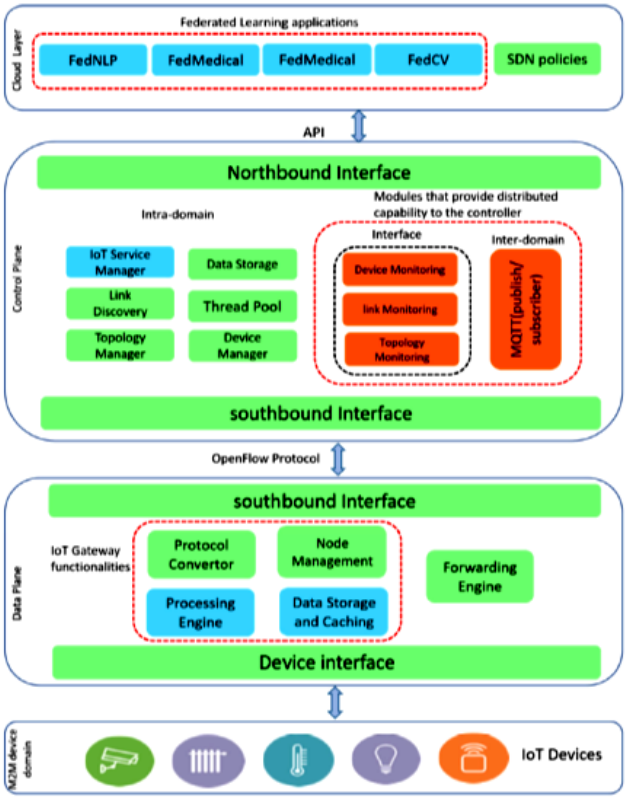

FIGURE 4.7: Distributed IoT architecture proposed in [53]

Processing Engine. Lastly, the M2M Plane includes the IoT devices. The SDN Controllers are synchronized and connected through the MQTT protocol to define forwarding rules. Additionally, a client selection algorithm is proposed to find the shortest path from selected clients to the server based on link utilization.
The results have been compared with other approaches, including those without SDN, with centralized SDN, and Distributed SDN without client selection. The proposed work outperforms all other approaches in terms of delay. Throughput and scalability are also tested, and the results demonstrate that as the number of distributed controllers increases, so does the throughput and scalability.
However, the evaluation is conducted in a limited number of controller domains, and it is important to validate the feasibility of using this architecture in larger networks. Furthermore, the selection of clients is solely dependent on their communication resources, while in many cases, clients are heterogeneous, and their computational and memory resources significantly affect the QoS of applications. Considering these factors would be essential to ensure the effectiveness of the architecture in real networks.

The FL in the context of e-Health is introduced for the first time in [54], which proposed an integration of Federated Learning (FL) with energy-constrained Mobile Edge Computing (MEC) for e-Health applications. The SDN network is integrated with MEC to create *SMEC*, as depicted in figure 4.8. The SDN Controller manages the Edge Nodes (ENs), which act as Base Stations collecting Patients' Health Information (PHI) and include OpenFlow Switches. The proposed FL mechanism, Fully Decentralized Federated Learning (*FDFL*), is used to aggregate trained models from the ENs. The approach also incorporates a privacy-preserving algorithm. The SDN Controller plays a role in monitoring the energy consumption of the ENs, indicating

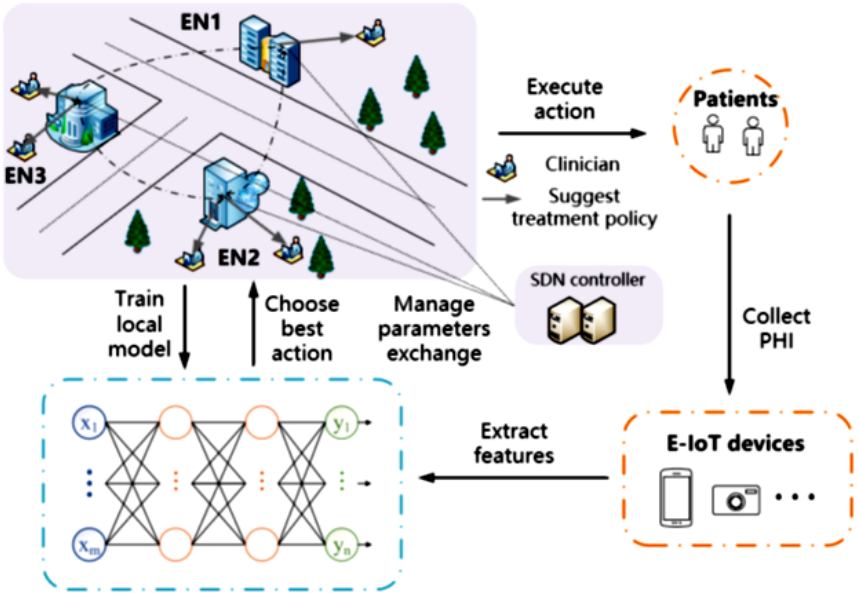

FIGURE 4.8: *SMEC* system architecture in [54]

the aggregation time limit, and adjusting communication parameters to achieve a tradeoff between consensus, energy consumption, and convergence time. The work introduces a novel Reinforcement Learning algorithm called DDQN, which demonstrates competitive accuracy in e-Health applications.
The validation of the results involves comparing the impact of different numbers of ENs, showing that increasing the number of ENs enhances the performance of the FL process even when some ENs fail.
The focus of the approach is primarily on the training model and its effect on the performance of e-Health applications. The role of the SDN Controller is limited to facilitating communication among the ENs and estimating convergence time. However, there is potential to further exploit the capabilities of SDN in various aspects, such as dynamic routing and client selection, which could enhance the overall FL process in e-Health scenarios.

An SDN-based Federated Learning (FL) scheme is introduced for intrusion detection in Vehicular Ad Hoc Networks (VANET) in [55]. The scheme, depicted in figure 4.9, involves multiple distributed SDN Controllers serving as FL clients. These controllers collect data from Base Stations, which aggregate data from vehicles. The data is stored in a Database Server combined with each controller. A single cloud server acts as the FL server responsible for maintaining the intrusion detection global model by aggregating locally trained models from the SDN controllers and redistributing the global model to the clients. Local training in the SDN controllers is achieved through two SDN applications: the first application collects VANET network communication information, which is then fed into another application running the intrusion detection model to detect abnormal network behavior. This approach enables real-time training and higher scalability.
For validation, two data sets are used to compare the performance of the proposed scheme with other state-of-the-art works in intrusion detection for VANET networks using FL, Machine Learning, or Deep Learning [56], [57], [58], and [59]. The proposed scheme demonstrates better performance in terms of achieved accuracy compared to these works.
The scheme leverages the SDN approach from a learning perspective by utilizing

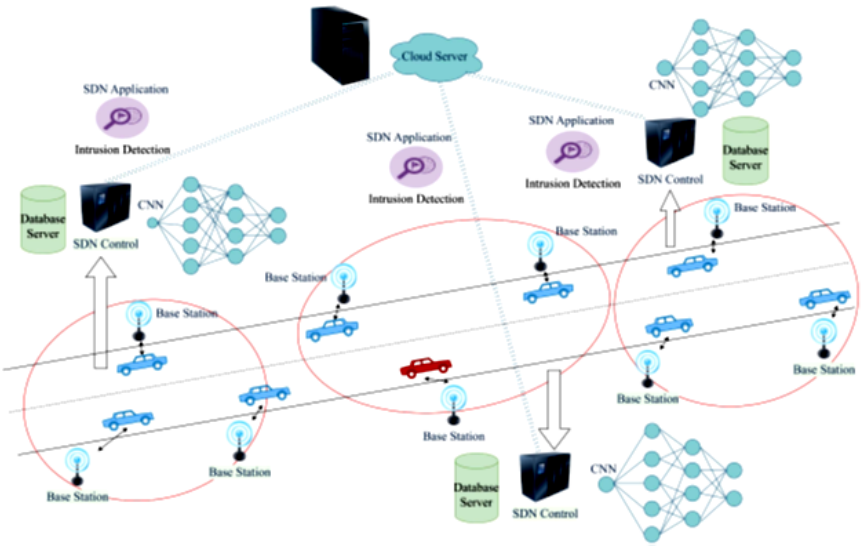

FIGURE 4.9: The scheme proposed for VANET in [55]

SDN controllers to aggregate network information and detect intrusions in a distributed manner. However, it does not fully exploit the networking capabilities of SDN, such as routing, communication resource management, and client selection. Additionally, the validation is conducted by comparing the final accuracy with the state-of-the-art, disregarding the time needed to achieve the target accuracy, which can be crucial in VANET intrusion detection applications.

4.4 Open Issues and Novel Contributions

The works reviewed so far have identified some gaps and issues that could be addressed by leveraging SDN networks. These works have missed the potential cooperation between SDN Controllers and FL servers to dynamically select clients based on their communication resources, link congestion, computational capacity, and memory. This dynamic client selection could enhance the performance of FL, particularly for time-sensitive QoS applications in 5G and 6G networks that require high accuracy and low loss within the shortest possible time.

The dynamic nature of the links used to connect FL clients to the FL server introduces challenges due to the lack of a dedicated network for FL. The network may transmit other applications' data, and clients themselves may transmit data related to other applications. Moreover, the clients' CPU and memory specifications significantly impact the training process. Higher CPU and memory result in faster training, but in real cases, FL clients perform various operations in addition to FL model training, leading to dynamically changing residual CPU and memory. Therefore, a dynamic client selection mechanism is necessary to select the most capable clients round by round, enabling the FL process to be completed as quickly as possible.

The contribution of our work can be summarized as follows:

- Study the feasibility of introducing SDN in the FL process by dynamically updating routes between FL clients and the FL server based on link congestion.
- Introduce a dynamic client selection mechanism based on the communication resource states between the FL homogeneous clients and FL server.

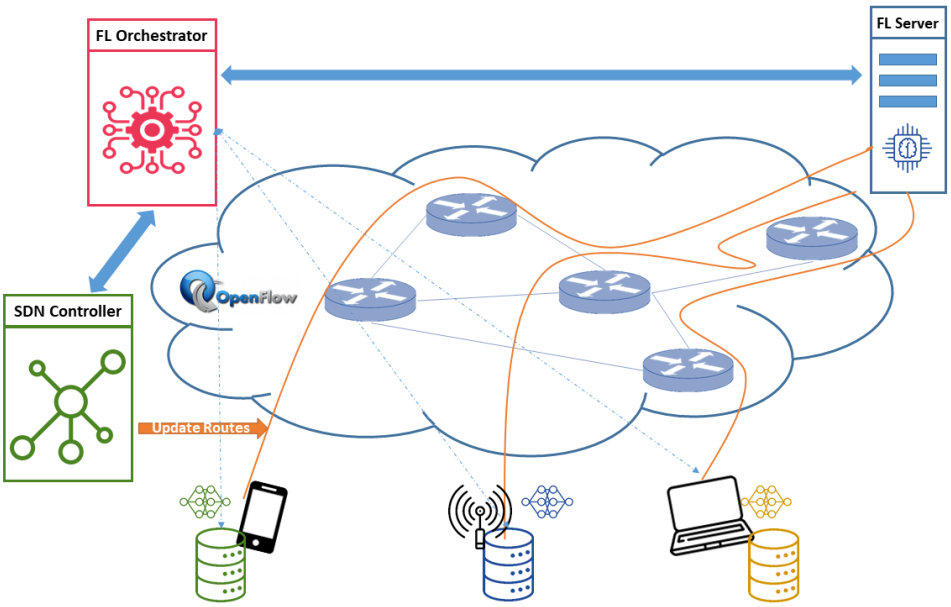

FIGURE 4.10: The proposed framework

- Introduce a dynamic client selection mechanism considering the communication resource states as well as the computational and memory resources of FL heterogeneous clients.
- Validate the proposed frameworks using a testbed network and compare the results with FL frameworks without the proposed approaches to demonstrate their performance improvements.

4.5 The Proposal: SDN-Assisted Client Selection to Enhance the Quality of Federated Learning Processes

In the proposed framework, we introduce several components to implement the SDN-based FL process. These components include an SDN Controller, FL server, FL clients, and a new entity called the FL Orchestrator, as shown in figure 4.10. The FL Orchestrator plays a crucial role in managing the networking and client selection aspects of the FL process.

In the proposed framework, we introduce several components to implement SDN-based FL Client Selection. These components include an FL Server, FL Clients, as in the conventional FL process, and a new entity called the FL Orchestrator, responsible for Client Selection. The SDN Controller provides measurements that assess the FL Orchestrator in selecting clients. Figure 4.10 shows the proposed architecture.

The task of the FL Server is as in the basic FL process, managing the FL process through aggregating the models from the participating clients and then updating the global model, which is then sent back to the participating clients in the next round.

The SDN Controller, in turn, has two main tasks. Firstly, the controller implements a Dynamic Routing and Load Balancing Mechanism that is typically used to guarantee fast and reliable data delivery by choosing the less congested route for each FL Client to the FL Server. Secondly, it is responsible for implementing a mechanism that measures the experienced delay of each client, even those which did not participate in the last training round. By using this mechanism, the FL Orchestrator

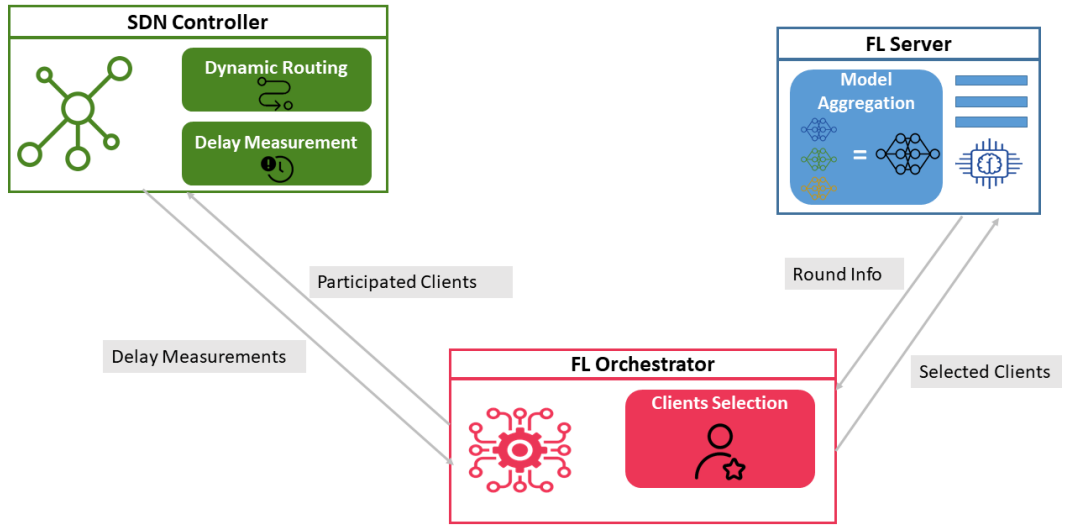

FIGURE 4.11: Functionalities and Interaction of Framework's Elements

can have a database about the network conditions of all available clients (N) that offer participation in the FL process. The main contribution of the SDN Controller is that it not only adjusts data routes dynamically but also enables the FL Orchestrator to dynamically select the FL Clients.

The FL Orchestrator's main task is to select the best clients to participate in each FL training round to ensure an efficient and effective FL process that achieves a specific accuracy as soon as possible. To achieve this task, the FL Orchestrator builds a two-way connection to both the SDN Controller and FL Server to exchange different information necessary for efficient Client Selection.

The FL Orchestrator sends a list of the participated clients from the previous round to the SDN Controller to enable the controller to perform different management, control, and monitoring of the traffic related to those clients through Dynamic Routing and Load Balancing techniques. On the other hand, the SDN Controller measures the delay of the FL Clients continuously and delivers a report of those measurements to the FL Orchestrator. On the other side, the FL Orchestrator sends, round by round, the selected clients to the FL Server, which, in turn, delivers information about the FL process, such as round number, round start, and round end. The functionalities and interactions between the components are shown in Figure 4.11."

The proposed framework establishes a close collaboration between the FL Orchestrator and SDN Controller, continuously and round by round, to select the most efficient clients and optimize the FL process.

In the next chapter, we provide a detailed explanation of all the applications and algorithms utilized in the proposed framework.

Chapter 5

Testbed Design and Configuration

5.1 Introduction

I discuss in detail the configuration of the different elements of the implemented testbed to validate the proposed SDN-based FL framework and obtain the main performance metrics I used to measure the effectiveness of this approach. In this section, I introduce the network topology and the reasons behind some assumptions. Then, I present the configuration of the SDN Controller and its tasks, along with the running applications. Additionally, I discuss the configuration of the OVSs. Following that, I provide a detailed explanation of the integration of the FLOWER Framework [60] into the proposed project, including the selection strategies and the client and server applications. Finally, I delve into some details about the learning model and the dataset in use.

5.2 Network Topology and Infrastructure Design

The proposed SDN-based FL framework has been implemented using the GNS3 [61] virtual environment emulator, which offers an easy way to design, build, emulate, configure, test, and troubleshoot virtual and real networks without the need for hardware. The network utilized a three-level Fat-Tree architecture (figure 5.1) and consisted of an ODL (OpenDaylight) SDN controller, labeled as *"CONTROLLER"* located at the top of the tree. Ten OVSs (Open vSwitches) were distributed across three levels, with eight FL Clients connected to the third-level switches and eight network users. The switches were configured (as shown in figure 5.1) so that *"S1"* and *"S2"* were located at level-1; *"S3," "S4," "S5,"* and *"S6"* at level-2; and *"S7," "S8," "S9,"* and *"S10"* at level-3. Cross-connections were established between the switches in levels 2 and 3 to provide more route choices for the clients. All links connecting the switches had a capacity of 100 Mbps.

The network hosted a Federated Learning (FL) server (*"SERVER"*) and eight clients (*"C1," "C2," "C3," "C4," "C5," "C6," "C7,"* and *"C8"*) that used the FLOWER (A Friendly Federated Learning Framework) framework [60] to conduct an FL process. Additionally, several virtual machines (*"D1," "D2," "D3," "D4," "D5," "D6," "D7,"* and *"D8"*) were included in the topology to apply the overloading profiles, which will be explained later.

All components were built on virtual machines using Oracle VirtualBox [62] and exported into GNS3. The specifications and roles of these machines are shown in table 5.1.

To achieve a heterogeneous routing scheme from the clients to the server, the server was connected to *S2*. As a result, the number of hops between the clients and

TABLE 5.1: Virtual Machines Specifications

Type	Virtual Machine	CPU Cores	RAM (GB)
SDN Controller	*CONTROLLER*	8	16
FL Server	*SERVER*	8	8
OVS Switches	*S1, S2, S3, S4, S5, S6, S7, S8, S9 and S10*	4	4
FL Clients	*C1, C2, C3, C4, C5, C6, C7 and C8*	Variable	Variable
Network Users	*D1, D2, D3, D4, D5, D6, D7, and D8*	2	4

the server varied; for instance, *C1* had a route with three hops, while *C8* had a route with two hops.

5.3 SDN Controller Configuration

After installing ODL on the controller, a set of features was installed to meet the desiderd requirements, as shown in table 5.2.

The SDN controller plays a vital role in controlling all switches and running the required applications through interaction with the FL Orchestrator. The controller builds a virtual graph of the network that includes switches, links, and connected clients using the Networkx library [63] (as shown in figure 5.2). This graph is used to find the optimal routes from each client to the server. The controller is responsible for two main tasks within the proposed architecture: **Dynamic Routing Application** and **Delay Collection**.

5.3.1 Dynamic Routing Application

Dynamic Routing is an application that runs on top of the controller. It retrieves information about the participated OVSs, clients, links, and connection ports from the controller. This information is then used to build a congestion-weighted virtual network graph. The application determines the best route from each client to the server using Dijkstra's shortest path algorithm [64] preiodically each 5 seconds, with the cost of congestion (delay) of the link taken into account. This congestion data can be retrieved using the provided RESTCONF API from the `odl-restconf-all` plugin of the ODL controller. Once the best path for each client is found, multiple flows are pushed to the switches that make up the path using the `odl-openflowplugin-all` plugin. This ensures that data can be routed efficiently from the FL client to the FL server and vice versa. The flowchart of the Dynamic Routing Application is shown in figure 5.3. This application is constructed using the Python programming language and helps in avoiding congested routes by selecting less congested ones.

5.3.2 Delay Collection Algorithm

The controller continuously measures the delay of the clients using a `ping` mechanism running in the background. This delay information is then transmitted to the server at the start of each training round, allowing the server to incorporate it into the selection mechanism. The flowchart for Delay Collection is displayed in figure 5.6.

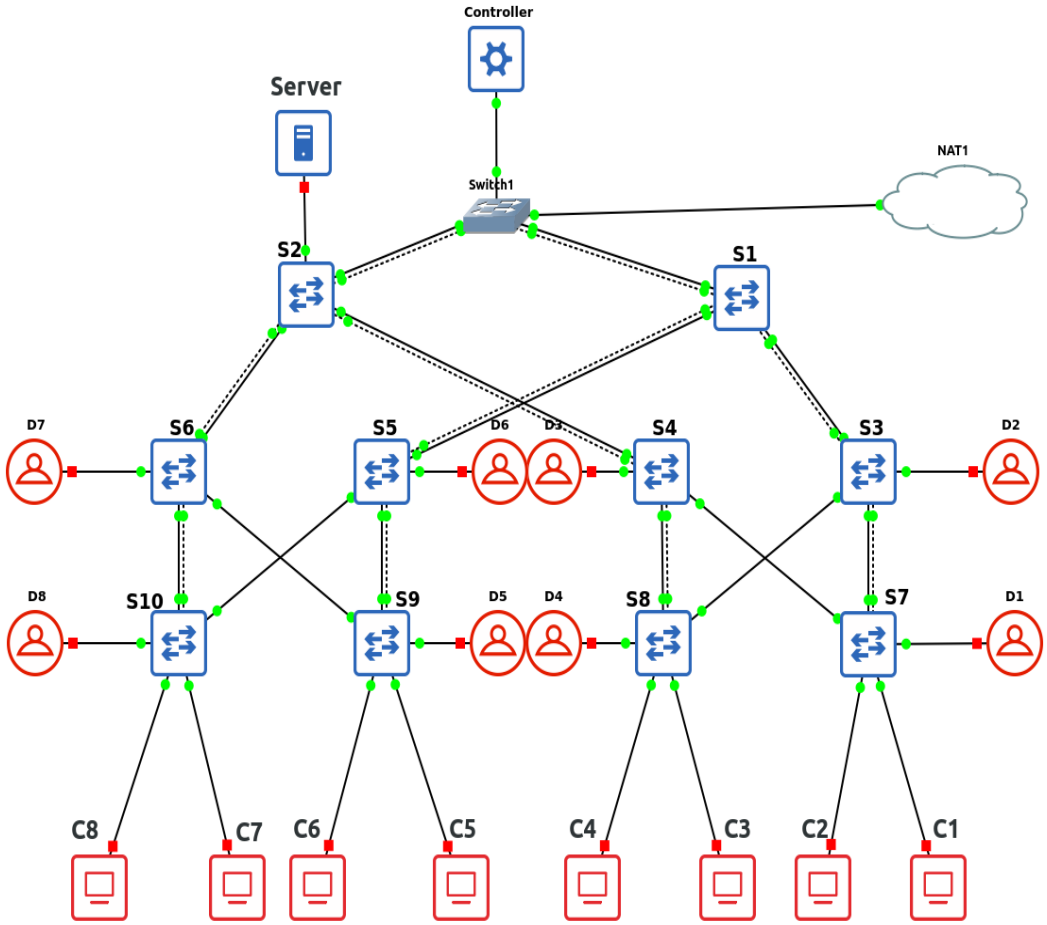

FIGURE 5.1: Network Topology

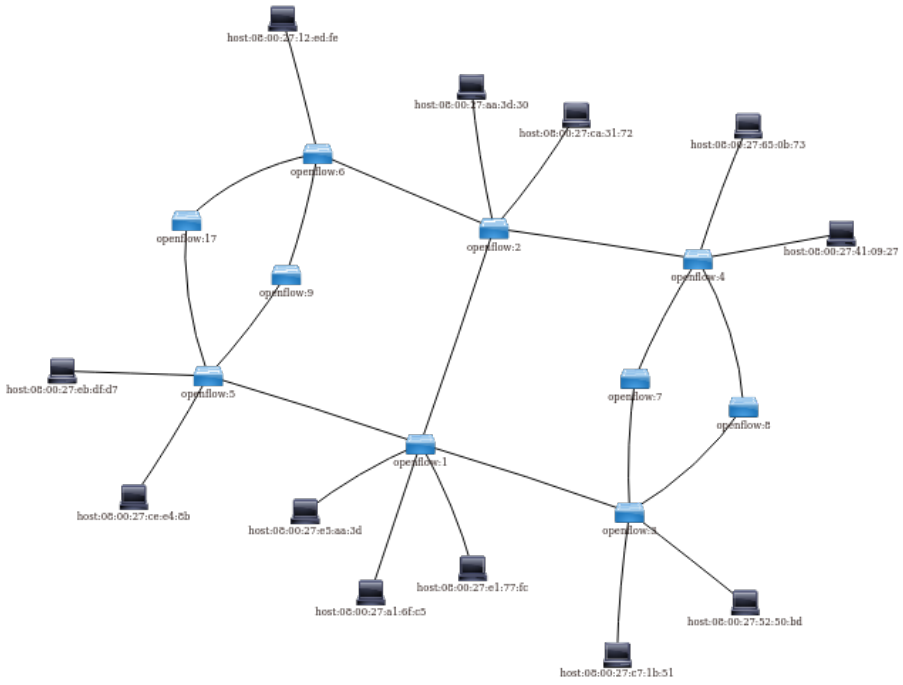

FIGURE 5.2: Virtualized Network Topology in Networkx

TABLE 5.2: ODL Installed Features

Feature	Function
`odl-restconf-all`	Enables the ODL controller to offer a programmable and configurable network management solution.
`odl-openflowplugin-all`	It allows the controller to communicate with OpenFlow-enabled network devices, such as switches and routers, and provides an abstraction layer that allows network administrators to configure and manage network flows through a high-level interface.
`odl-l2switch-all`	This feature provides support for Layer 2 switching functionality in software-defined networks (SDNs). It enables the controller to manage the forwarding of Ethernet frames at Layer 2, including MAC address learning, forwarding, and flooding.
`odl-mdsal-all`	It enables the controller to manage and manipulate network data models and provides a set of APIs for interacting with network devices and services.
`odl-yangtools-common`	It enables the controller to work with the YANG data modeling language, which is used to define the structure and behavior of network devices and services.
`odl-dlux-all`	It enables the controller to provide a user-friendly interface for configuring and managing network resources using a web-based interface.

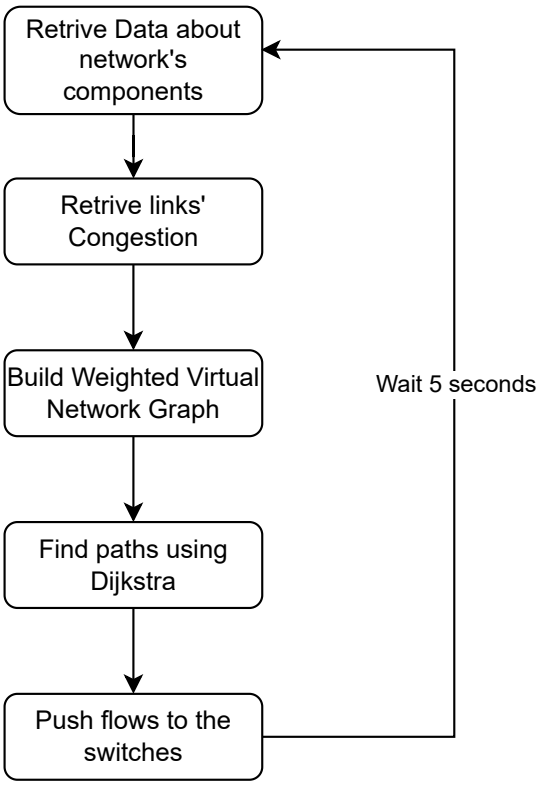

FIGURE 5.3: Dynamic Routing Application Flowchart

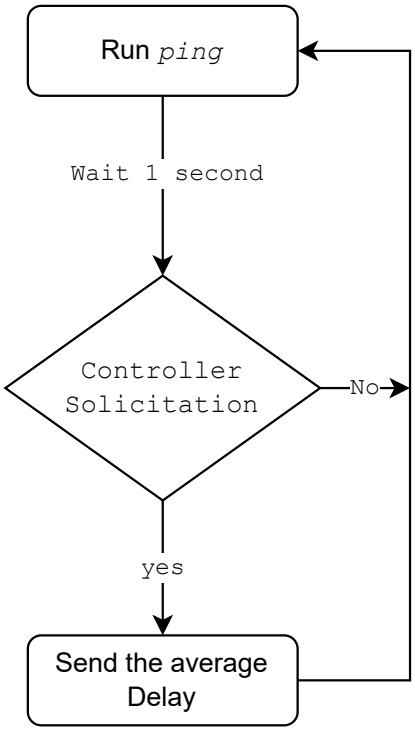

FIGURE 5.4: Delay Collection Flowchart

5.4 Open vSwitch Configuration

Virtual machines running the Ubuntu OS were employed to host Open vSwitch (OVS) instances. Following the download of the OVS source code from GitHub, the ovsdb Database was constructed. The ovsdb-server was configured to employ a previously created database, listen on a Unix domain socket, establish connections to any managers specified within the database, and apply the SSL configuration defined in the database.

```
$ mkdir -p /usr/local/var/run/openvswitch
$ ovsdb-server --remote=punix:/usr/local/var/run/
   openvswitch/db.sock \
    --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
    --private-key=db:Open_vSwitch,SSL,private_key \
    --certificate=db:Open_vSwitch,SSL,certificate \
    --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
    --pidfile --detach --log-file
```

LISTING 5.1: OVS Configuration

After that, I build a bridge and add the ports of the switch; in addition, I connect that bridge to the controller using the proper IP address and port number.

```
$ ovs-vsctl add-br <bridge>
$ ovs-vsctl add-port <bridge> <port>
```

LISTING 5.2: OVS Connection

Finally, I enable the out-of-band mode to listen only to the controller's flows and avoid install "hidden flows" automatically and each switch is assigned a name to be distinguished from others.

```
$ ovs-vsctl set controller br0 connection-mode=out-of-band
```

LISTING 5.3: OVS Mode

5.5 Flower Framework Integration

The Flower framework is utilized to facilitate the execution of the Federated Learning (FL) process within both the server and clients. Several modifications have been introduced into the core source code to enable seamless integration between the server, clients, and the controller. These modifications encompass three primary functions: **Selection Strategy**, **Server Application**, and **Client Application**.

5.5.1 Selection Strategy

The Flower framework comes with a default Random Selection strategy, where users can specify the minimum number of participating clients required to start the simulation (N) and the number of sampled clients for each round (M) where:

$$(M; M \leq N) \tag{5.1}$$

However, I have developed two custom Selection Strategies: the Delay-dependent Selection Strategy (DSS) and the Delay/Computational-Resources Selection Strategy (DCSS).

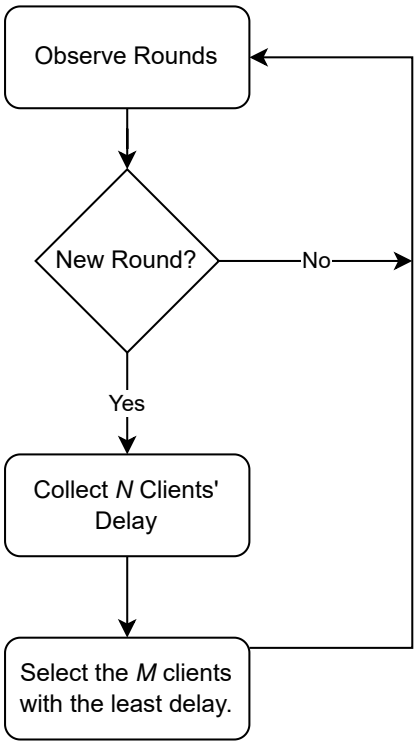

FIGURE 5.5: DSS Strategy

5.5.2 Delay-dependent Selection Strategy (DSS)

The Delay Selection (DSS) Strategy collects the delay measurements from all the N participated clients and then samples M clients with the least delay for each round. This process is repeated at the start of each round, considering the updated delay information of clients to ensure dynamic client selection based on the delay profile within the network. You can refer to figure 5.5 for the flowchart of this strategy.

5.5.3 Delay/Computational-Resources Selection Strategy (DCSS)

The new selection strategy takes into account both the delay and the computational resources available in each client to select clients with the least delay and the best computational resources. To achieve this, an Evaluation Index (EI) is defined for each client, representing a weighted selection metric that includes both *Normalized Delay (ND)* and *Computational Index (CI)*, each with a modified weight, denoted as α.

$$EI = \alpha * CI + (1 - \alpha) * ND \tag{5.2}$$

ND represents the Normalized Delay of the Client:

$$ND = \frac{Client'sDelay}{MaxDelay} \tag{5.3}$$

CI defines the Computational Index of the clients that incorporate both the CPU Index and Memory Index:

$$CI = \beta * CPUI + (1 - \beta) * MEMI \tag{5.4}$$

The *CPUI* is used to normalize the processing capability of the client with the other clients according to the number of CPU cores and the CPU usgae.

$$CPUI = \frac{\#\,of\ CPU_{Cores}}{Max\ CPU_{Cores}} * (1 - CPU_{usage}); 0 \leq CPU_{usage} \leq 1 \tag{5.5}$$

Finally, the *MEMI* normalize the free available memory with the client comparing to the other clients.

$$MEMI = \frac{FreeRAM}{MaxRAM} \tag{5.6}$$

5.5.4 Server Application

The server application is responsible for managing the Federated Learning (FL) process through multiple steps. During server initialization, the following parameters are defined:

- Number of rounds
- Server IP address
- Type of selection strategy (Delay or Delay/Computational-Resources)

Once the server is initialized, the Aggregation Strategy, based on the FedAvg scheme [65], is selected and the following parameters are specified:

- Minimum number of participating clients (to start the training)
- Number of sampled clients for each training round
- Number of sampled clients for each evaluation round
- Initial weights source

After the server and the aggregation strategy are initialized, the training model is defined, and the application is run with the defined initialization. For the first round, the server sends the initial weights of the model to each sampled client and waits for the results from all sampled clients unless the *round_timeout* is over. For the subsequent rounds, the server sends the weights of the previous round.

After receiving the weights from each client adjusted by training on its local dataset, the aggregation strategy is used to update the weights at the end of each round that is used and sent to the sampled clients in the subsequent round.

Finally, after finishing all rounds, the server saves the data of performance metrics (Accuracy and Loss) and the time of each round in a text file to be used for evaluation. figure 5.6 illustrates the main blocks of the server application.

5.5.5 Client Application

The client-side application is responsible for receiving the global model from the server, training it with the local dataset, and sending the locally trained model back to the server. The client application starts by defining the model to be used and preparing the local dataset. The client builds a TCP connection to the server on a predefined port. After that, the client delivers its computational resources (RAM and CPU capacities and their usage) to the server, which is then used in the selection phase.

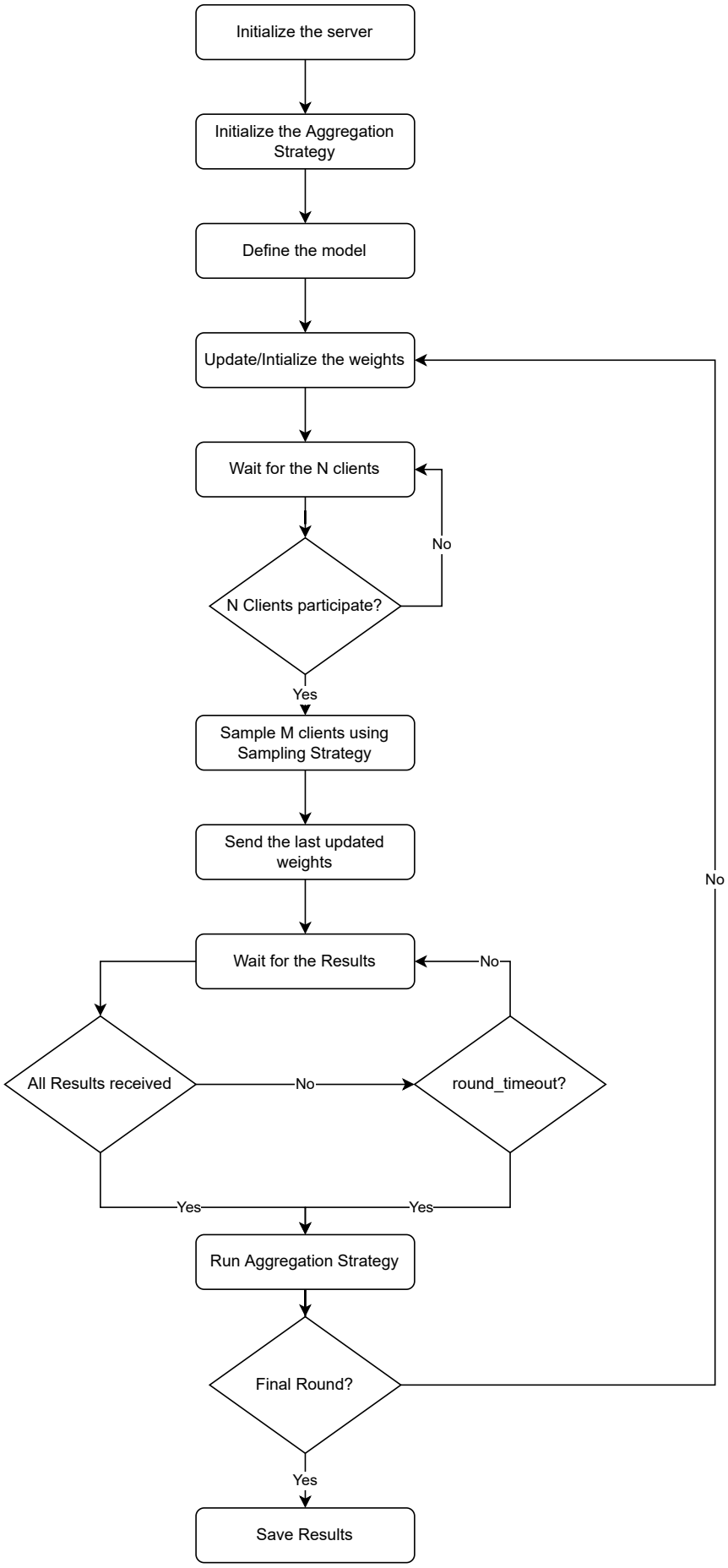

FIGURE 5.6: Server Application

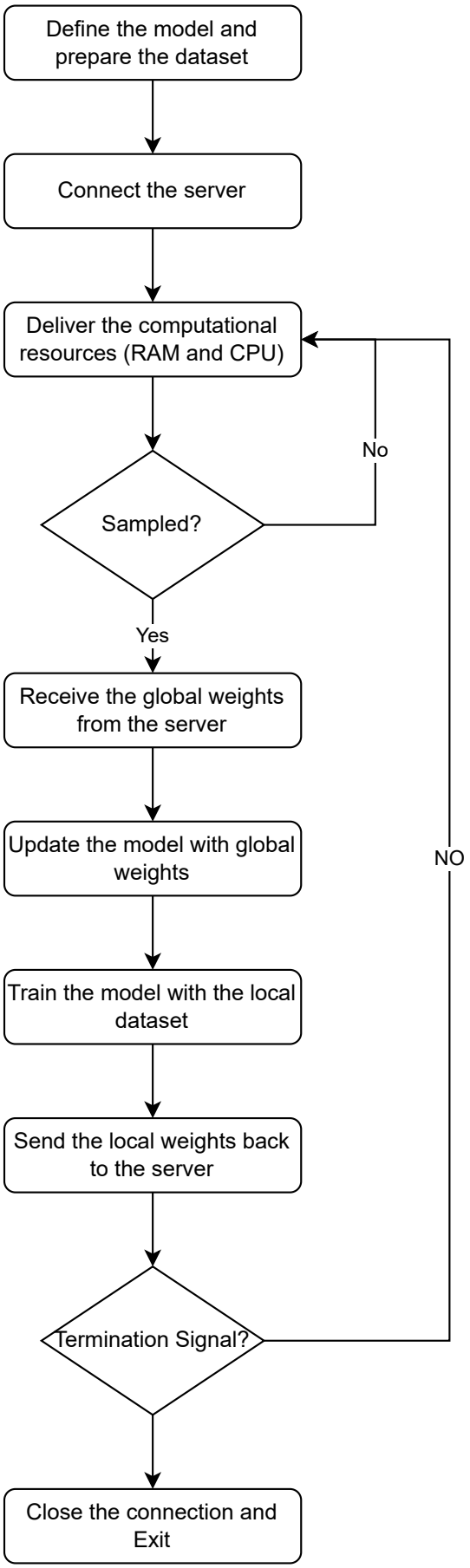

FIGURE 5.7: Client Application

If the server samples the client, it starts training on the local dataset after receiving the up-to-date global model's weights. Once the training is done, it sends the local model's weights back to the server and repeats the procedure for each round. Figure 5.7 shows the client application flowchart.

5.6 Overloading Mechanism

The simulation requires overloading some links with different data rates in different scenarios. I decided to use the *iperf* tool [66] between different network users (D1, D2, ..., D8 in figure 5.1) along the links to be loaded, creating a UDP connection. One side acts as an *iperf* server, and the other acts as an *iperf* client, initiating the connection towards the *iperf* server.

5.7 Deep Learning Model

Densenet121 is a convolutional neural network architecture proposed by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger [67]. It comprises 121 layers and is trained on the ImageNet dataset for image classification tasks. The key innovation of Densenet is the concept of "dense connectivity," where each layer in the network is connected to every other layer in a feed-forward fashion. This approach significantly reduces the number of parameters needed to train the network while improving accuracy. Densenet121 is one of the smaller Densenet models, with only 7.98 million parameters, making it more efficient and faster to train compared to larger models like Densenet169 and Densenet201. It has achieved state-of-the-art results on many image classification benchmarks, including the ImageNet dataset. Densenet121 is implemented in various deep learning frameworks, including TensorFlow [68] and PyTorch [69], and pre-trained models are available for use in transfer learning applications.

In my FL process, I employ a transfer-learning mechanism by using the DenseNet121 pre-trained model defined in the TensorFlow framework. I add a forward network on top of it, consisting of three dense layers with 128, 64, and 10 units, respectively. The first two layers use the ReLU activation function, and the last one uses the softmax activation function. I use the *Adam* [70] optimizer with a learning rate of 0.005 and the *categorical_crossentropy* loss function. Figure 5.8 provides a summary of the model.

5.8 CIFAR-10 Dataset

The CIFAR-10 dataset [71] is a collection of 60,000 color images, each measuring 32x32 pixels, categorized into 10 classes, with 6,000 images per class. It is divided into 50,000 training images and 10,000 test images. The ten classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Images in the CIFAR-10 dataset are characterized by their low resolution (32x32 pixels) and a wide range of colors.

The CIFAR-10 dataset serves as a popular benchmark for evaluating machine learning algorithms, particularly deep learning algorithms designed for image classification tasks. Numerous researchers have developed models that have achieved state-of-the-art performance on this dataset. Figure 5.9 showcases sample images from the CIFAR-10 dataset.

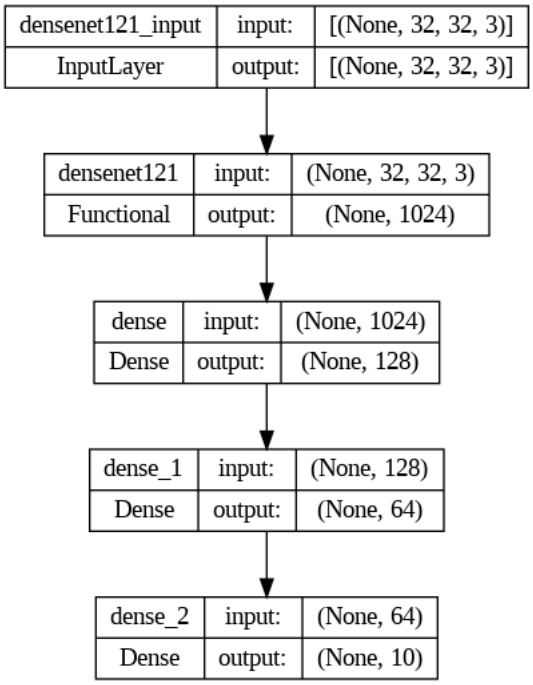

FIGURE 5.8: Deep Neural Network Model

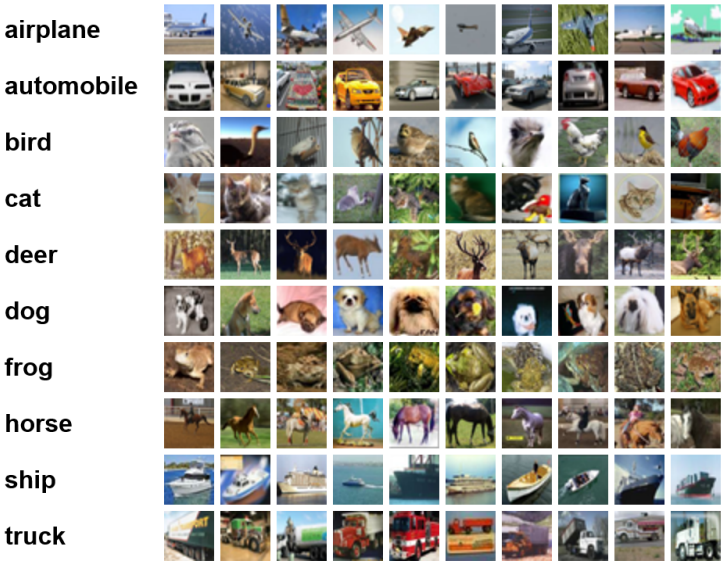

FIGURE 5.9: CIFAR-10 Dataset [71]

Chapter 6

Results and Performance Evaluation

6.1 Introduction

We have observed a lack of integration of SDN into FL processes. Most of the literature focuses on client resources related to training performance, while the network connecting the server and clients can significantly affect performance. This is especially true when other applications are running on the same network, introducing congestion and causing delays in the connection between clients and the server. Therefore, we made the decision to introduce the use of SDN to leverage its capabilities in enhancing network performance for delivering FL services, in addition to considering client specifications.

This chapter will discuss and present the proof-of-concept for the ideas used to enhance FL performance using SDN. We have implemented three different approaches: Dynamic Routing, Delay-dependent Selection, and Delay/Computational-Resources-dependent Selection. For each approach, we demonstrate its feasibility by implementing a specific network scenario that showcases its superiority in that context. In the following sections, we will explain the scenario implementations and discuss the results.

6.2 Dynamic Routing

In this specific scenario, we compared the time required to complete the FL process using static and dynamic routes. The static routing approach, specifically Distance-vector routing, uses the number of hops between the clients and the server as a metric for route selection. In contrast, the proposed dynamic routing algorithm relies on link congestion (delay) as the metric for route selection. Table 6.1 defines the chosen static routes using the hop counts from each FL Client to the FL Server and figure 6.1 show the static routes on the topology (dashed-lines).

To validate the Dynamic Routing, we defined overload Profile-1, which overloads one link of the static routes for each client (as shown in table 6.2 and figure 6.2), with different data rates: *No Overload*, *10 Mbps*, *30 Mbps*, and *50 Mbps*.

To compare the results, we overload the links with each data rate then measure the performance metrics (Accuracy and Loss) against time for both cases Static and Dynamic Routing as depicted in figures from 6.3 to 6.10.

We observed a significant improvement in performance, particularly in terms of the time required to complete the FL process, when we introduced Dynamic Routing as compared to using static routing. This time reduction became more pronounced

TABLE 6.1: Static Routes of the Clients

Client	Route
C1	*S10, S6, S2*
C2	*S10, S6, S2*
C3	*S9, S6, S2*
C4	*S9, S6, S2*
C5	*S8, S4, S2*
C6	*S8, S4, S2*
C7	*S7, S4, S2*
C8	*S7, S4, S2*

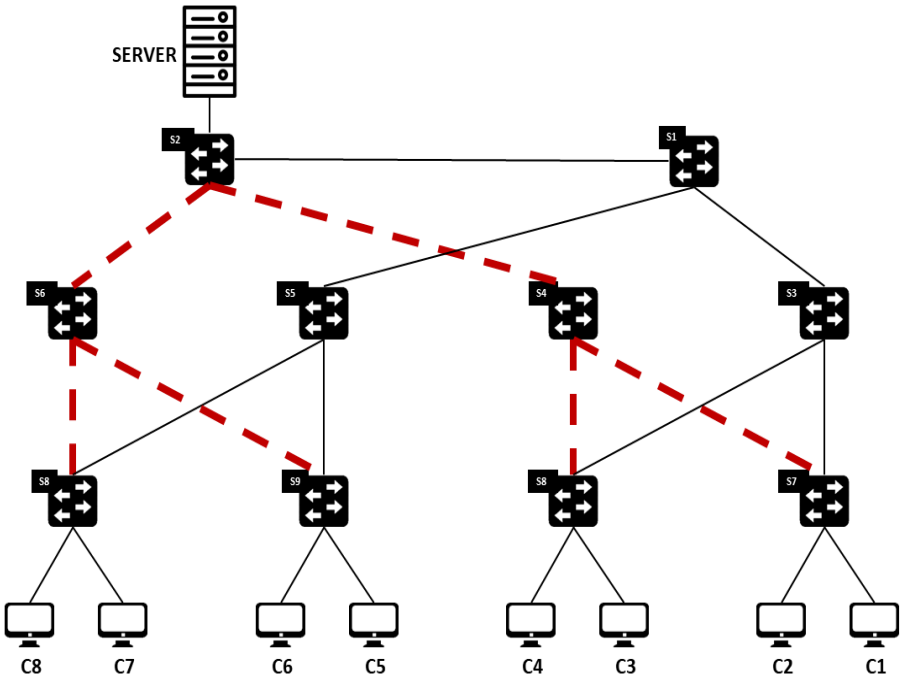

FIGURE 6.1: Static Routes of the Clients (dashed-line)

TABLE 6.2: Overloading Profile-1

Client	Overloaded Links
C1	*S10, S6*
C2	*S10, S6*
C3	*S9, S6*
C4	*S9, S6*
C5	*S8, S4*
C6	*S8, S4*
C7	*S7, S4*
C8	*S7, S4*

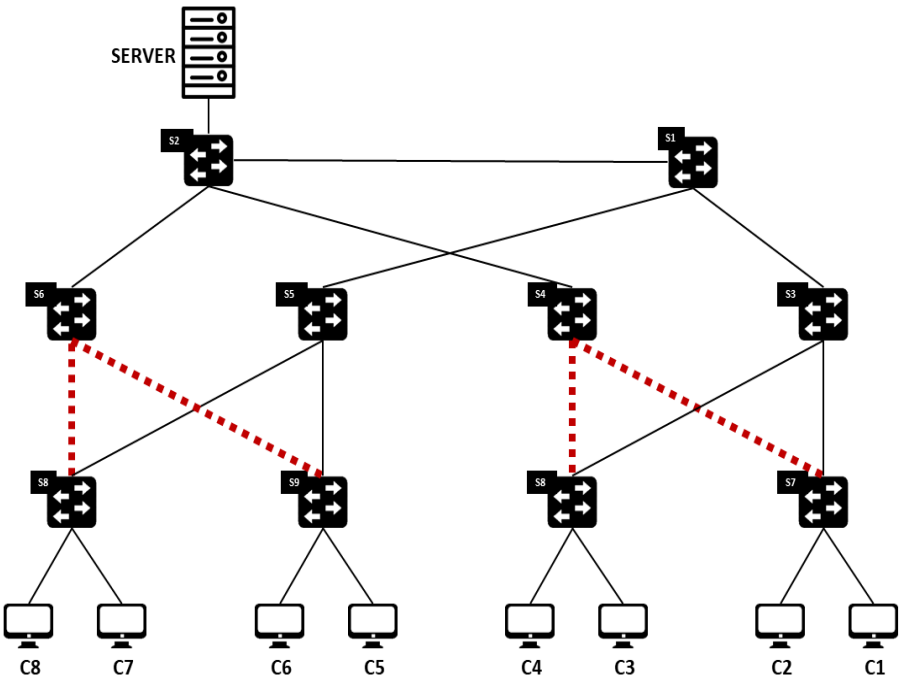

FIGURE 6.2: Overloaded Links of Profile-1 (dotted-line)

TABLE 6.3: Convergence Time Reduction of Profile-1

Scenario (Overloading Rate)	Convergence Time Reduction (sec)
No Overload	*465.74*
10 Mbps	*1091.16*
30 Mbps	*1490.84*
50 Mbps	*3346.7*

as the data rate for overloading increased. This suggests that as network congestion increased, the advantages of dynamic routing, in terms of time savings, became more evident. The specific difference in convergence time reduction between static and dynamic routing to achieve the best accuracy will be provided in table 6.3.

The primary limitation of Dynamic Routing is its inability to alleviate congestion when all available links are congested. To address this drawback, we implemented a second approach that prioritizes selecting clients with the lowest delay. In essence, the main goal of this work is to avoid clients whose routes are consistently congested. The results and findings from the simulation of the first approach, Dynamic Routing, are summarized in table 6.4. This table offers a comprehensive overview of the performance and outcomes of the Dynamic Routing approach.

6.3 Delay-dependent Client Selection

In the previous section, we determined that Dynamic Routing could avoid congested links only when an alternative route was available for the client, which was not congested. In many real-life scenarios, congested links are common over time. Therefore, we propose a selection strategy DSS (explained in Chapter 5) that chooses clients experiencing the least delays among the available clients, assuming that all clients have the same (homogeneous) computational resources (RAM: 4GB, CPU: 2

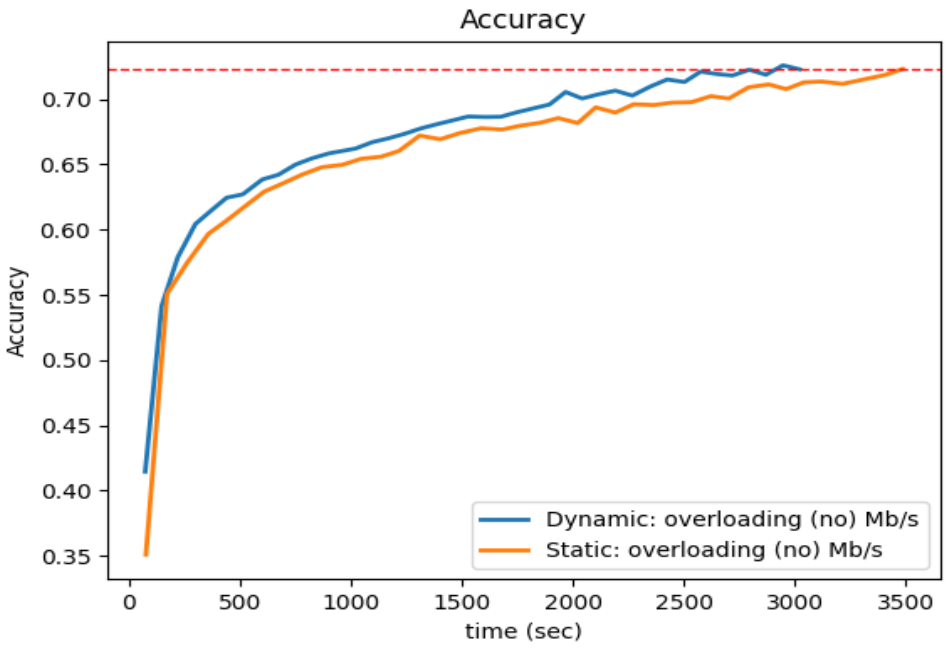

FIGURE 6.3: Dynamic Routing: Accuracy of "No Overload" Profile-1

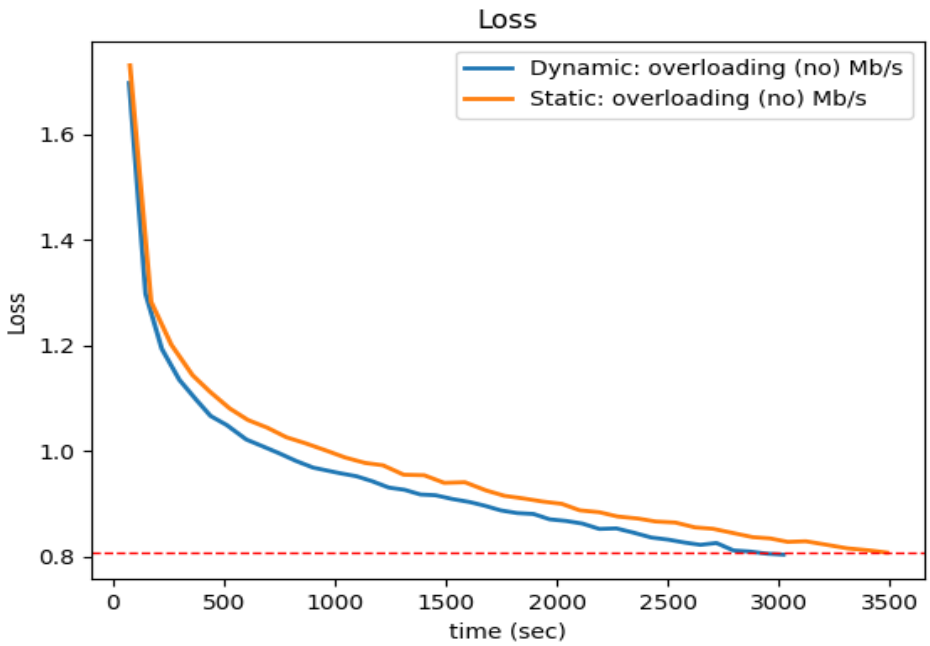

FIGURE 6.4: Dynamic Routing: Loss of "No Overload" Profile-1

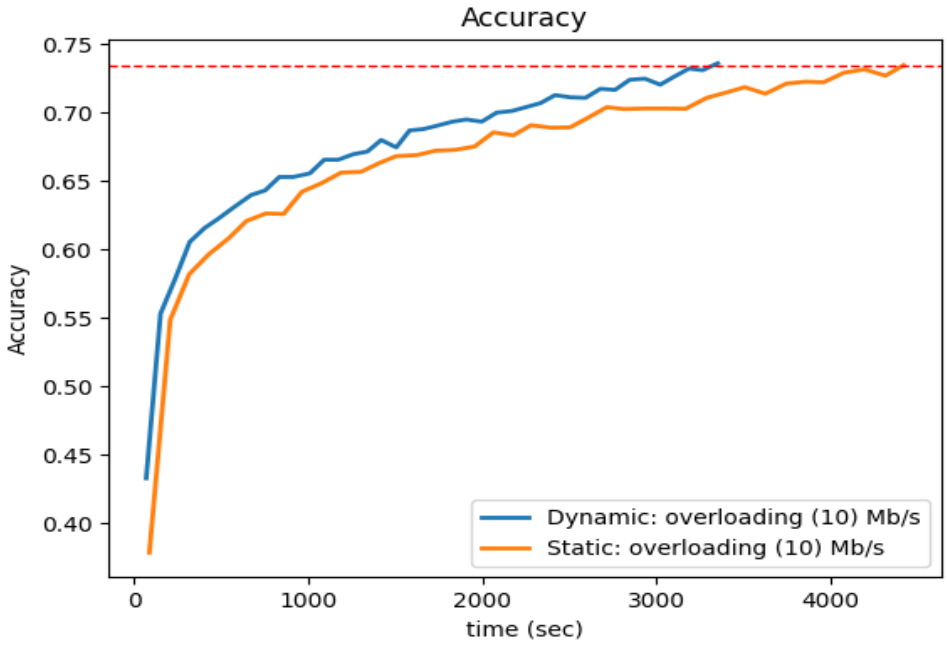

FIGURE 6.5: Dynamic Routing: Accuracy of "10 Mbps" Overload Profile-1

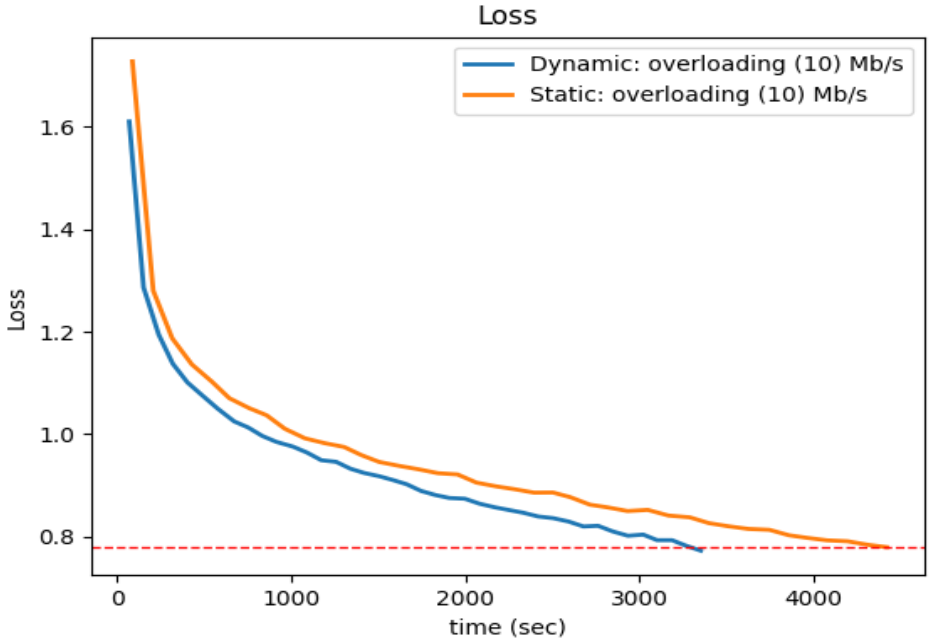

FIGURE 6.6: Dynamic Routing: Loss of "10 Mbps" Overload Profile-1

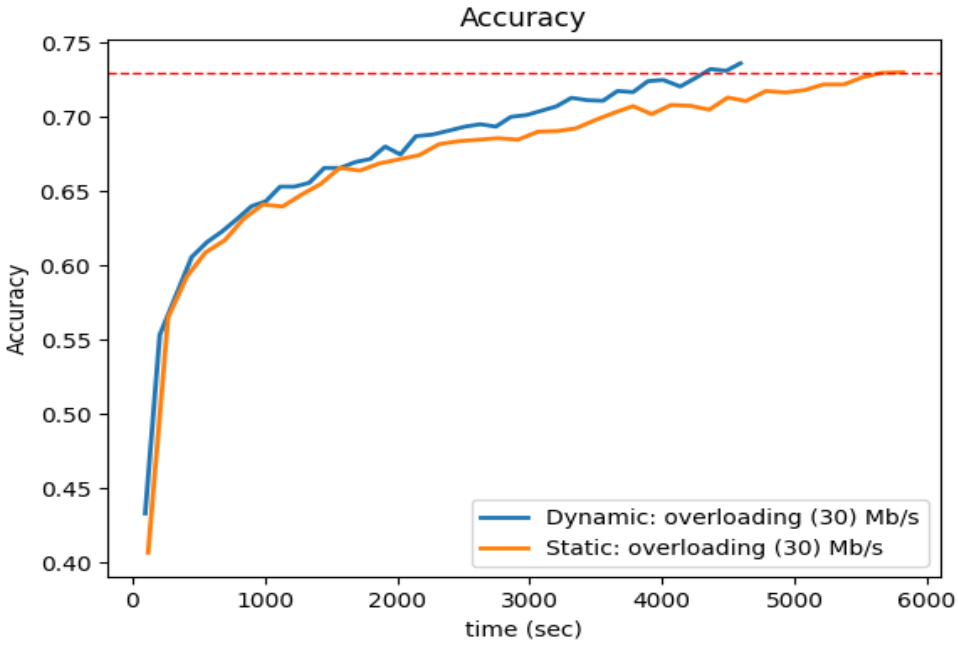

FIGURE 6.7: Dynamic Routing: Accuracy of "30 Mbps" Overload Profile-1

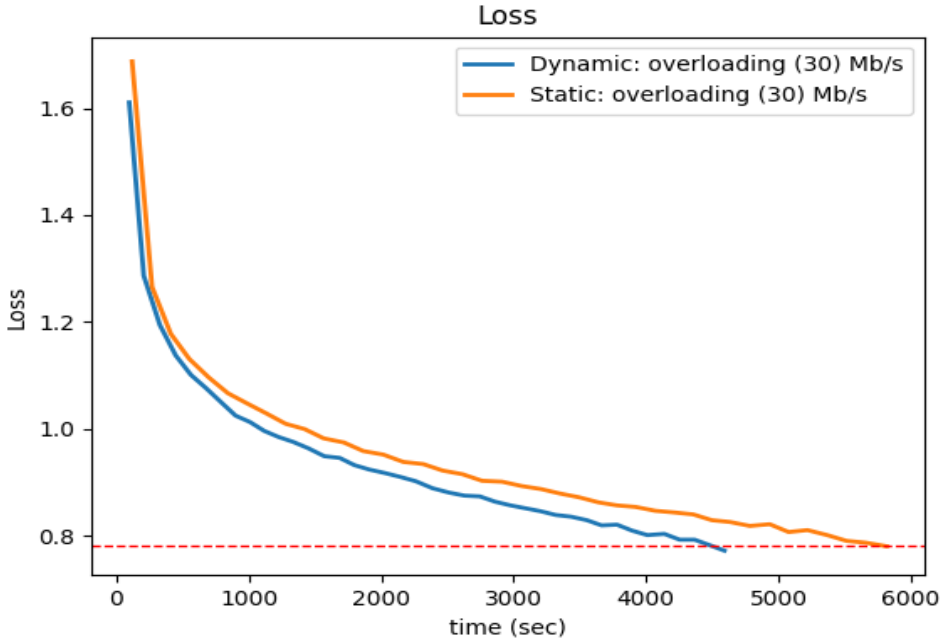

FIGURE 6.8: Dynamic Routing: Loss of "30 Mbps" Overload Profile-1

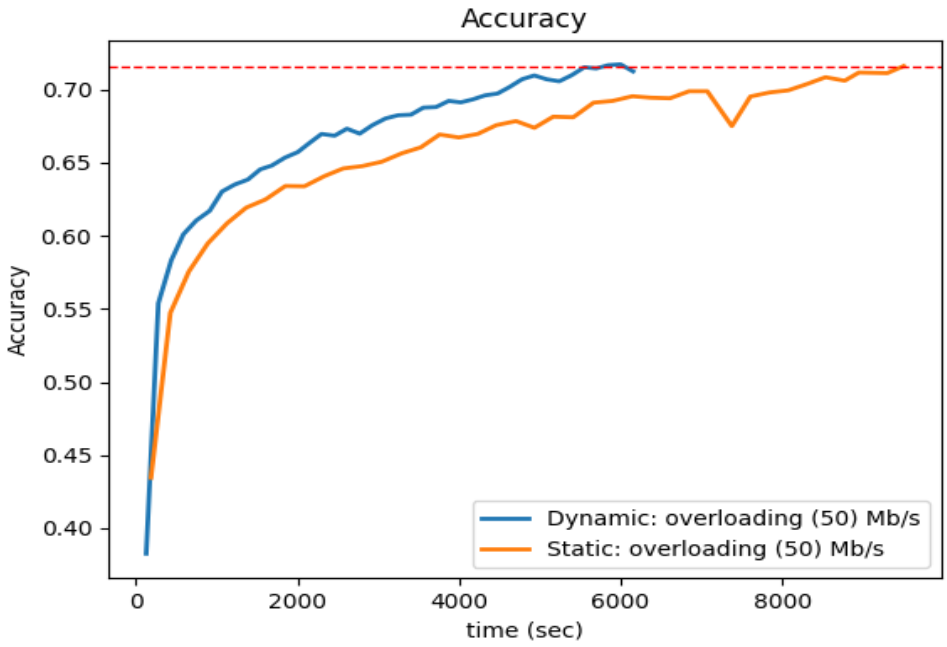

FIGURE 6.9: Dynamic Routing: Accuracy of "50 Mbps" Overload Profile-1

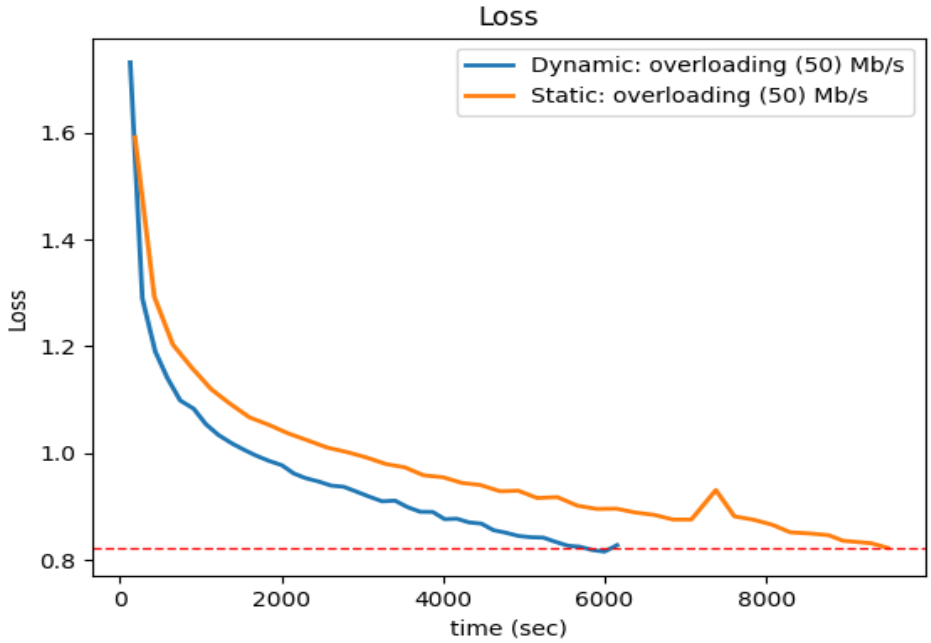

FIGURE 6.10: Dynamic Routing: Loss of "50 Mbps" Overload Profile-1

TABLE 6.4: Dynamic Routing Conclusion

Problem	*Avoidance of congested routes*
Goal	*Prove that Dynamic Routing needs less time to finish the FL process than Static Routing.*
Methods	*Overload one link of the static route for each client.*
Validation Scenarios	*Overload the links according to Profile-1 with different data rates:* • *No overload* • *10 Mbps* • *30 Mbps* • *50 Mbps*
Metrics	*Accuracy and Loss versus time.*
Results	• *Dynamic routing reduces the time needed to complete a FL process.* • *The reduction is enhanced when the overloading data rate is increased.*
Cons (Disadvantages)	*Dynamic Routing cannot avoid the congestion when all available links are overloaded.*
Solution	*Use Client Selection*

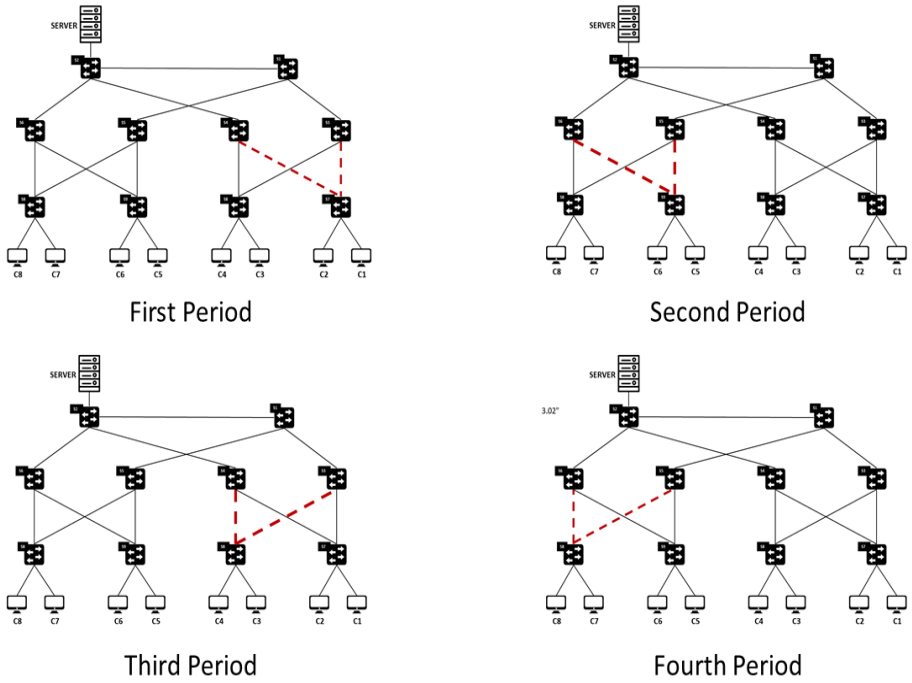

FIGURE 6.11: Overloaded Links of Profile-2 (dashed-line)

cores). To ensure the dynamism of the selection, we defined an overloading profile, namely Profile-2, explained in figure 6.11. In this profile, overloading is periodically altered. Profile-2 periodically overloads the two available links of each switch in the third level, inducing congestion for clients connected to these switches in all available routes to the server. Dynamic Routing continues to run in the background. Table 6.5 defines the overloaded links periodically.

TABLE 6.5: Overloaded Links of Profile-2

Period	**Overloaded links**	
First	*S3,S7*	*S4,S7*
Second	*S5,S9*	*S6,S9*
Third	*S3,S8*	*S4,S8*
Fourth	*S5,S10*	*S6,S10*

To validate the Delay-dependent Client Selection, Profile-2 overloading is used with different datarates: *30 Mbps*, *50 Mbps* and *70 Mbps* and period equal to 500 seconds. To compare the results, we overload the links with each data rate then measure the performance metrics (Accuracy and Loss) against time for both cases Random and Delay-dependent Client Selection as shown in figures from 6.12, to 6.18.

We noticed that the performance, in terms of the time needed to complete the FL process, is improved when selecting clients that experience less overloading using the Delay Selection method, as compared to random selection, which includes routes that are consistently overloaded. Table 6.6 illustrates the time reduction for each data rate of Profile-2.

To consolidate all the proposed mechanisms, we introduce Profile-3, which combines elements from both Profile-1 and Profile-2. This profile allows us to validate

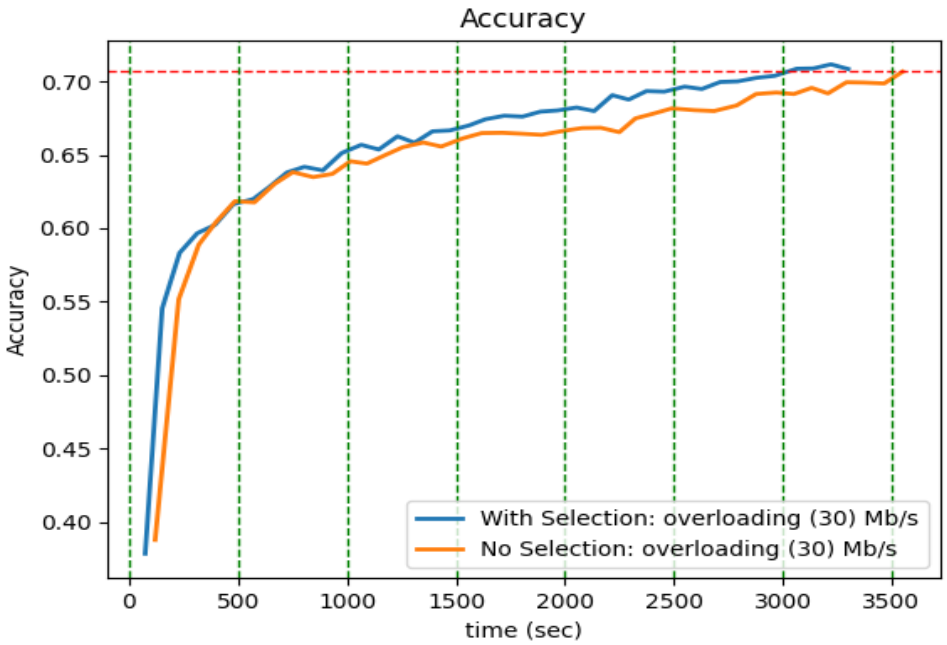

FIGURE 6.12: Delay-based Selection: Accuracy of "30 Mbps" Overload Profile-2

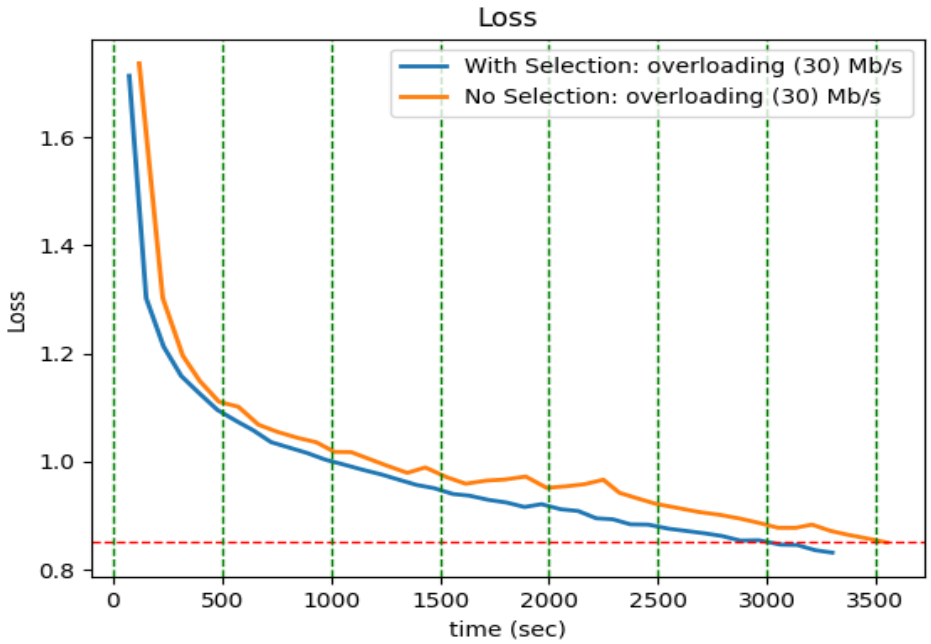

FIGURE 6.13: Delay-based Selection: Loss of "30 Mbps" Overload Profile-2

TABLE 6.6: Convergence Time Reduction of Profile-2

Scenario (Overloading Rate)	Convergence Time Reduction (sec)
30 Mbps	*528.18*
50 Mbps	*802.82*
70 Mbps	*1049.55*

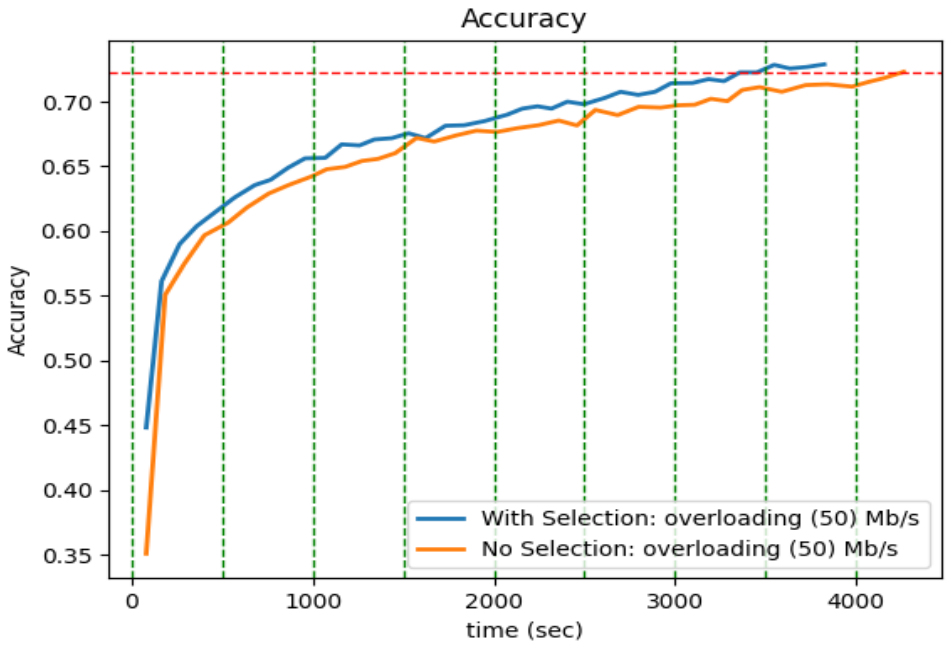

FIGURE 6.14: Delay-based Selection: Accuracy of "50 Mbps" Overload Profile-2

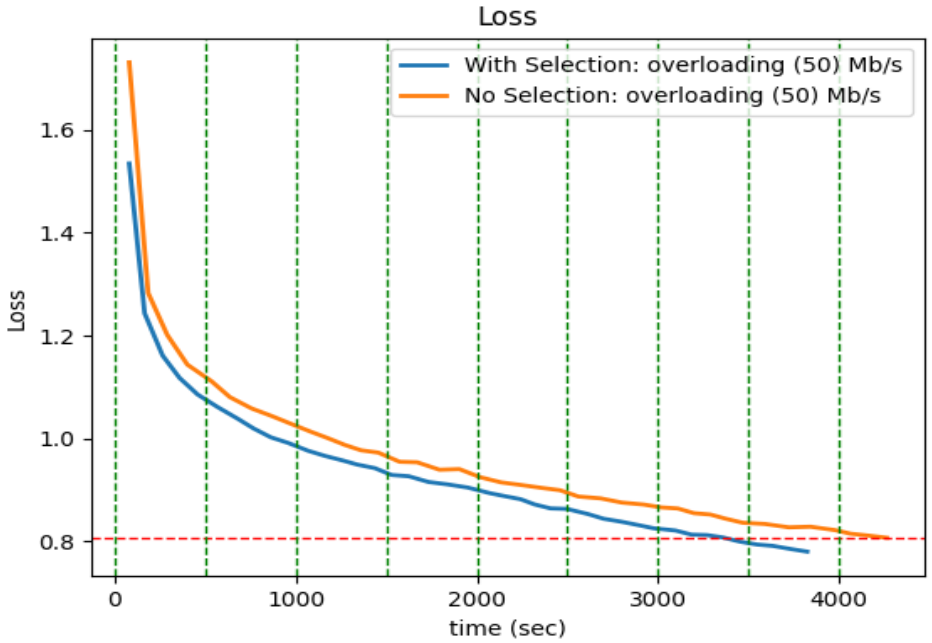

FIGURE 6.15: Delay-based Selection: Loss of "50 Mbps" Overload Profile-2

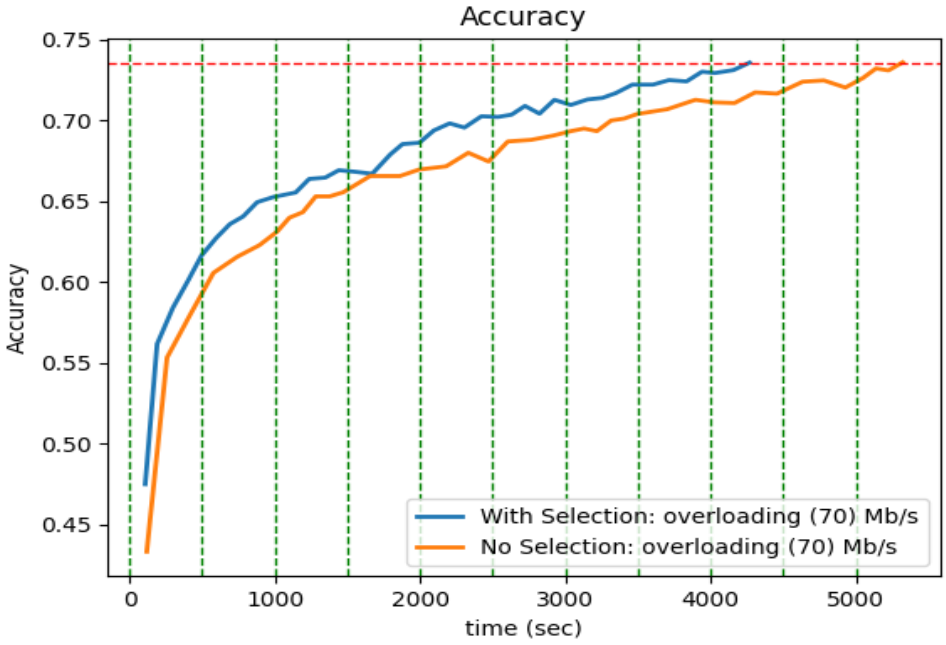

FIGURE 6.16: Delay-based Selection: Accuracy of "70 Mbps" Overload Profile-3

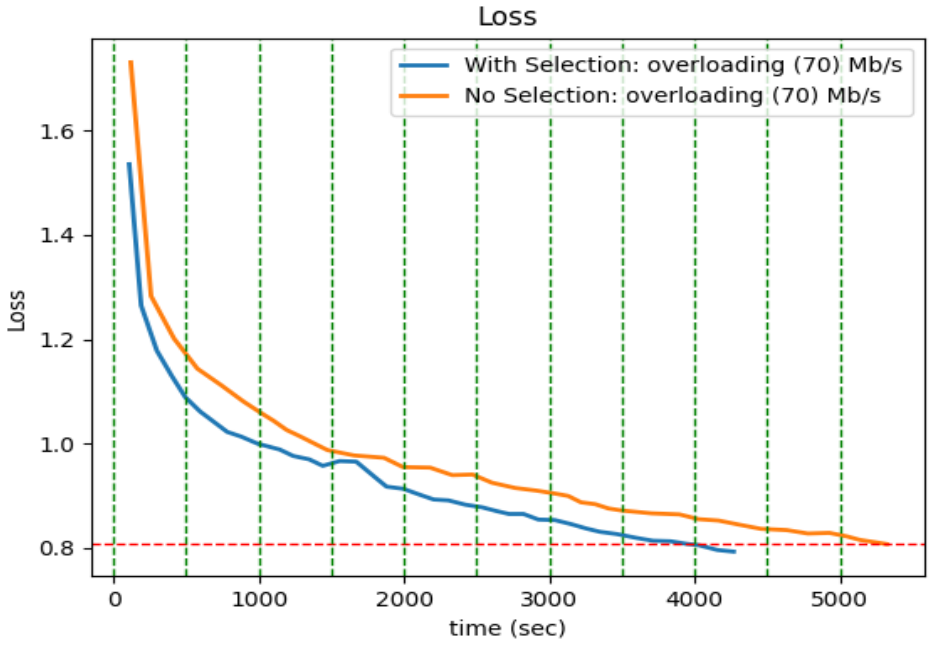

FIGURE 6.17: Delay-based Selection: Loss of "70 Mbps" Overload Profile-3

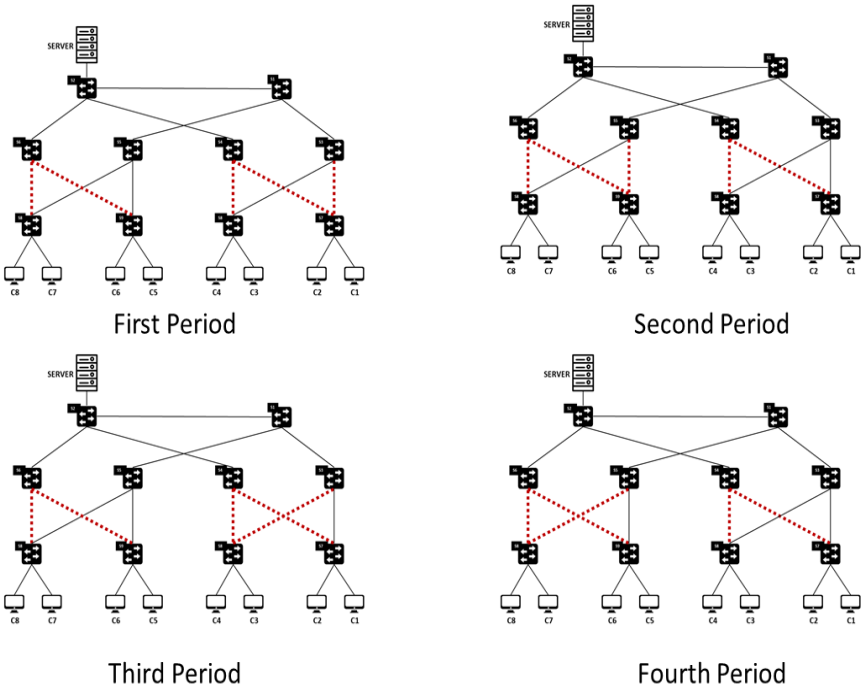

FIGURE 6.18: Overloaded Links of Profile-3 (dashed-line)

TABLE 6.7: Convergence Time Reduction of Profile-3

Scenario (Overloading Rate)	Convergence Time Reduction (sec)
10 Mbps	*352.93*
25 Mbps	*2957.44*
40 Mbps	*4591.72*

the simultaneous use of all these mechanisms. In Profile-3, static routes are overloaded, similar to Profile-1, and in addition, alternative paths for two of the clients are periodically overloaded, akin to Profile-2. Figure 6.18 provides a detailed explanation of the Profile-3 overloading mechanism.

In this scenario, we implement Profile-3, where the clients have homogeneous computational resources (2GB RAM and 4 CPU cores) and an overloading data rate of *25 Mbps*. Figures 6.19 and 6.20 illustrates that Dynamic Routing outperforms Static Routing, while Delay Selection further improves the performance together with Dynamic Routing when applied in conjunction.

To validate the concept, we implemented the scenario with lower and higher data rates: *10 Mbps* and *40 Mbps*, respectively as depicted in figures 6.21, 6.22, 6.23, and 6.24. In both cases, the combination of Dynamic Routing and Delay Selection with homogeneous clients outperformed static routing, leading to a significant reduction in time as the overloading data rate increased. Table 6.7 provides a clear breakdown of the time reduction for each scenario.

One of the main drawbacks of the Delay Selection approach is its limited effectiveness when the clients exhibit heterogeneity in terms of computational resources, such as varying RAM memory and CPU cores. The computational resources of the clients can significantly impact the time required for the FL process. To overcome this issue, we propose the Delay/Computational-Resources Selection mechanism,

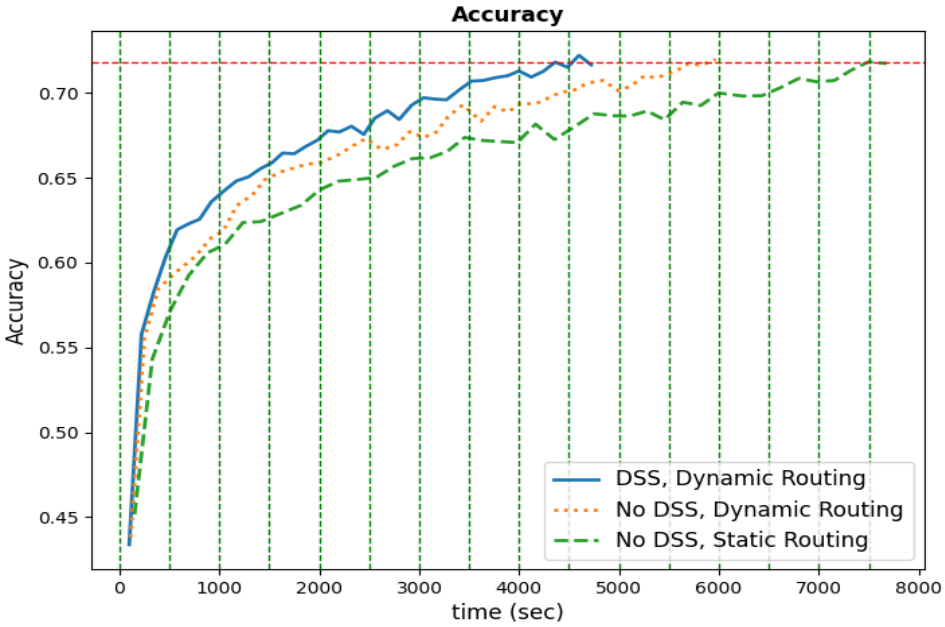

FIGURE 6.19: Delay-based Selection: Accuracy of "25 Mbps" Overload Profile-3

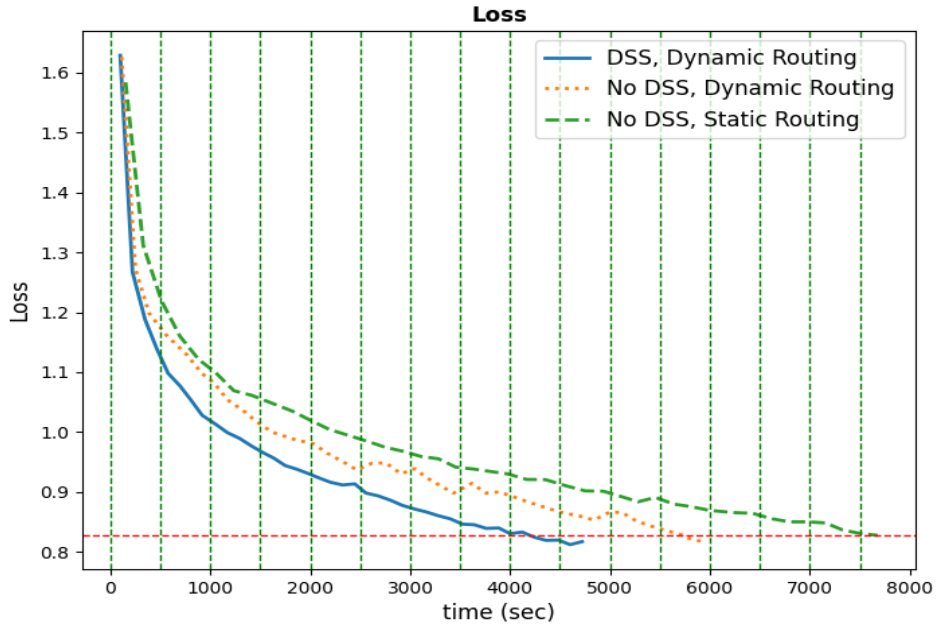

FIGURE 6.20: Delay-based Selection: Loss of "25 Mbps" Overload Profile-3

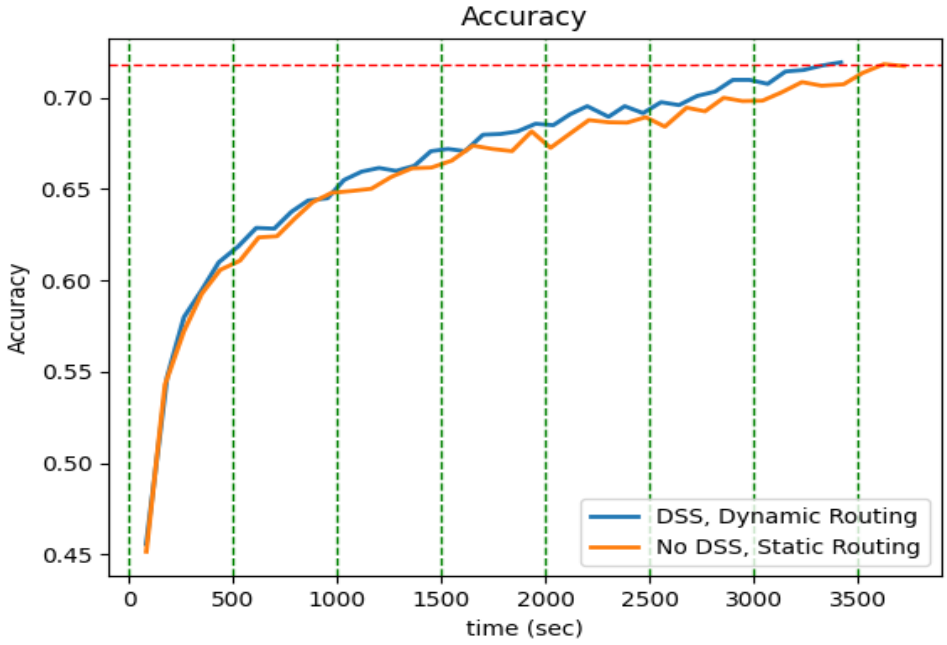

FIGURE 6.21: Delay-based Selection: Accuracy of "10 Mbps" Overload Profile-3

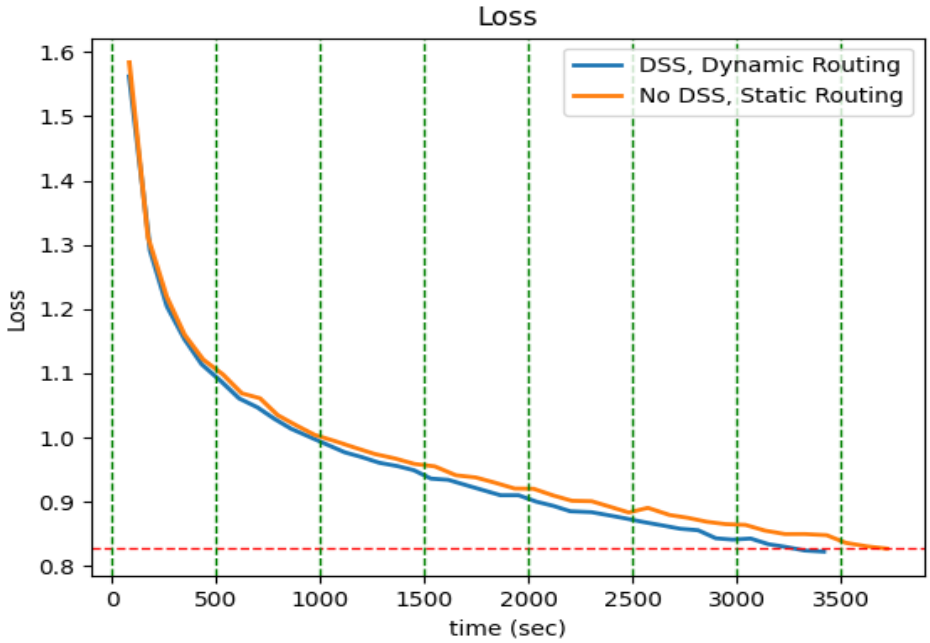

FIGURE 6.22: Delay-based Selection: Loss of "10 Mbps" Overload Profile-3

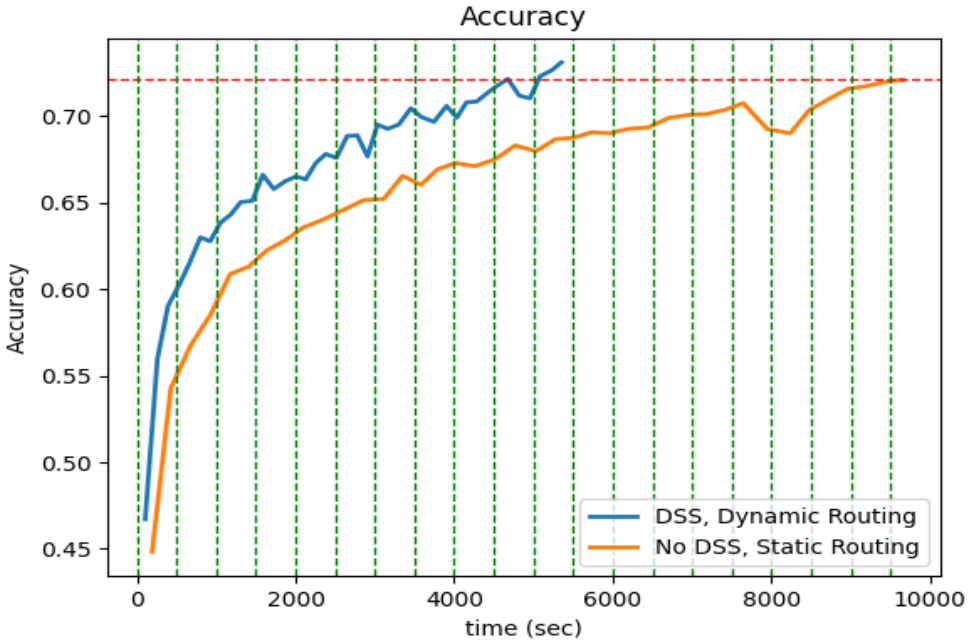

FIGURE 6.23: Delay-based Selection: Accuracy of "40 Mbps" Overload Profile-3

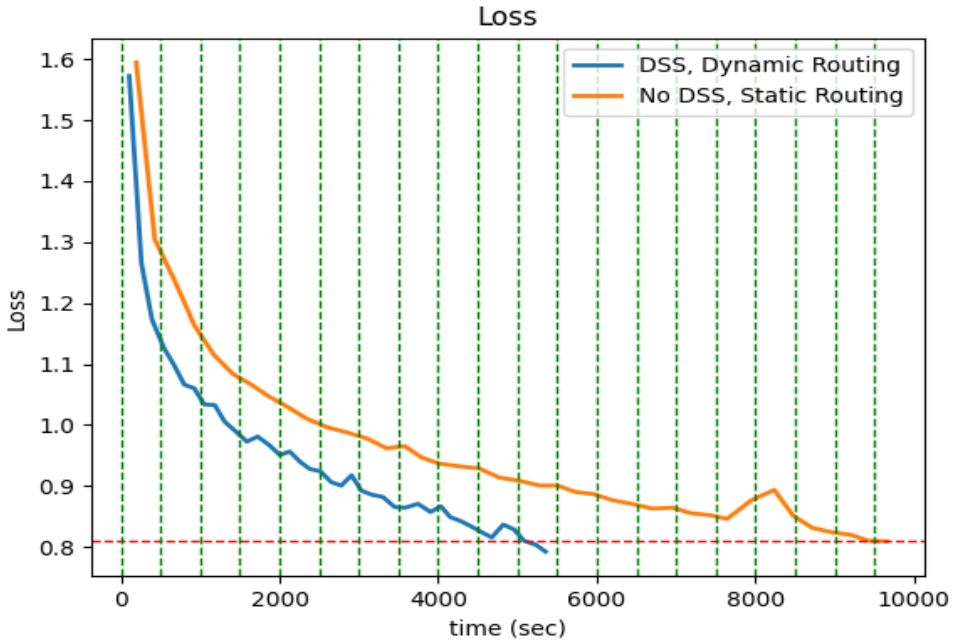

FIGURE 6.24: Delay-based Selection: Loss of "40 Mbps" Overload Profile-3

TABLE 6.8: Delay-based Client Selection Conclusion

Problem	*Some clients have congested routes more than others*
Goal	*Prove that Delay Selection helps in reducing the time to finish the FL process than Static or Random Selection.*
Methods	*Overload all links of each of the two clients periodically.*
Validation Scenarios	*Overload the links according to Profile-2 with different data rates:* • *30 Mbps* • *50 Mbps* • *70 Mbps*
Metrics	*Accuracy and Loss versus time.*
Results	• *Delay Selection reduces the time needed to complete a FL process.* • *The reduction is enhanced when the overloading data rate is increased.*
Cons (Disadvantages)	*Delay Selection is sufficient only when the clients are homogeneous.*
Solution	*Use Delay/Computational-Resources-dependent Selection*

which selects clients that experience the least delay and possess the best computational resources. This combined selection approach aims to optimize both network performance and computational capabilities. Table 6.7 concludes the Delay-dependent Client Selection.

6.4 Delay/Computational-Resources Client Selection

We found that when clients are heterogeneous, it's necessary to use Delay/Computational-Resources Selection (DCSS) instead of Delay Selection. This is because communication resources affect the delay between FL Clients and the FL Server, and computational resources affect the training time needed in each training round. To prove this concept, we will implement Profile-2 but with heterogeneous clients, and the DCSS strategy (explained in Chapter 5) will be used to select the clients. The computational resources of the clients are divided into three categories (Cat1, Cat2, and Cat3) as shown in table 6.9. The category of each client declared in table 6.10

The feasibility of selecting clients has been evaluated according to the equation 5.2. Three overloading rates have been used (40 Mbps, 60 Mbps, and 80 Mbps), and the results are depicted in figure 6.25, 6.26, 6.27, 6.28, 6.29, 6.30. Different values of α (the weight of computational resources), namely 0.0, 0.35, 0.7, and 1.0, have been used. When α=0, it means that we revert to the Delay Selection Strategy (DSS),

TABLE 6.9: Heterogeneous Clients Categories

Category	Computational Resources	
	RAM (GB)	*CPU (Cores)*
Cat1	2	2
Cat2	4	4
Cat3	6	6

TABLE 6.10: Heterogeneous Clients Categories's Assignements

Client	Category
C1	*Cat1*
C2	*Cat3*
C3	*Cat3*
C4	*Cat2*
C5	*Cat3*
C6	*Cat2*
C7	*Cat1*
C8	*Cat2*

which is based solely on delay. On the other hand, when α=1.0, we select clients solely based on their computational and memory resources, disregarding delay.

It is evident that scenarios considering both delay and computational/memory resources (α=0.35 and α=0.7) consistently outperform those that focus solely on delay (α=0.0) or computational/memory resources (α=1.0). However, as the overloading data rate increases, we observe that it becomes necessary to increase the weight of delay (reduce α) because the delay in communication between the FL Server and FL Clients becomes more significant in comparison to the delay introduced by the training process. Table 6.11 show the convergence time comparing to worst case for each scenario.

TABLE 6.11: Delay/Computational-Resources Selection: Convergence Time Reduction of Profile-2

Scenario (Overloading Rate)	Convergence Time Reduction (sec)			
	α=***0***	α=***0.35***	α=***0.7***	α=***1.0***
40 Mbps	0	558.43	**1035.40**	366.47
60 Mbps	282.64	**1760.86**	1091.52	0
80 Mbps	85.04	**1140.57**	729.87	0

The results and findings from the simulation of the first approach, Delay/Computational Resources Selection, are summarized in table 6.12. This table offers a comprehensive overview of the performance and outcomes of the Dynamic Routing approach.

The weights of the selection equation (α and β) must be optimized where, the the optimal weights change with different overloading delays and computational resources of the clients. One of the promising techniques of the selection of these weights is the AI, specifically Reinforcement Learning, that can used in future works.

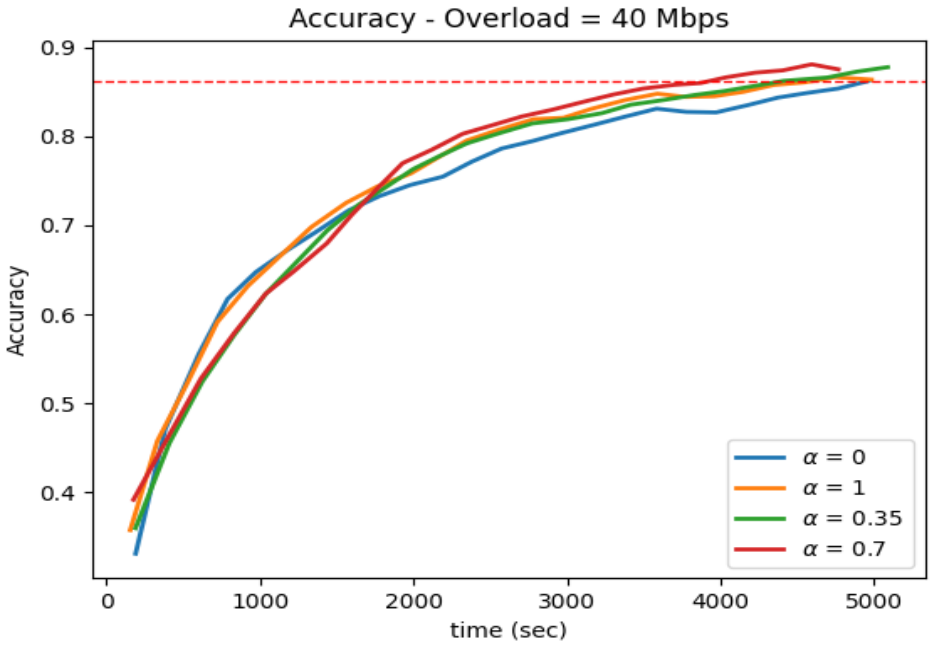

FIGURE 6.25: Delay/Computational Resources Selection: Accuracy of "40 Mbps" Overload Profile-2

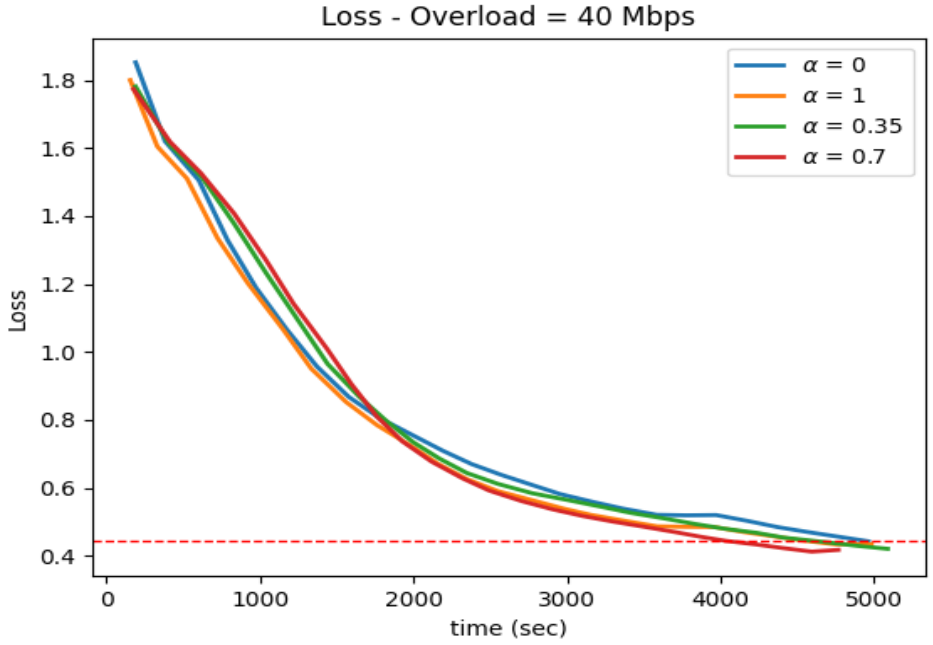

FIGURE 6.26: Delay/Computational Resources Selection: Loss of "40 Mbps" Overload Profile-2

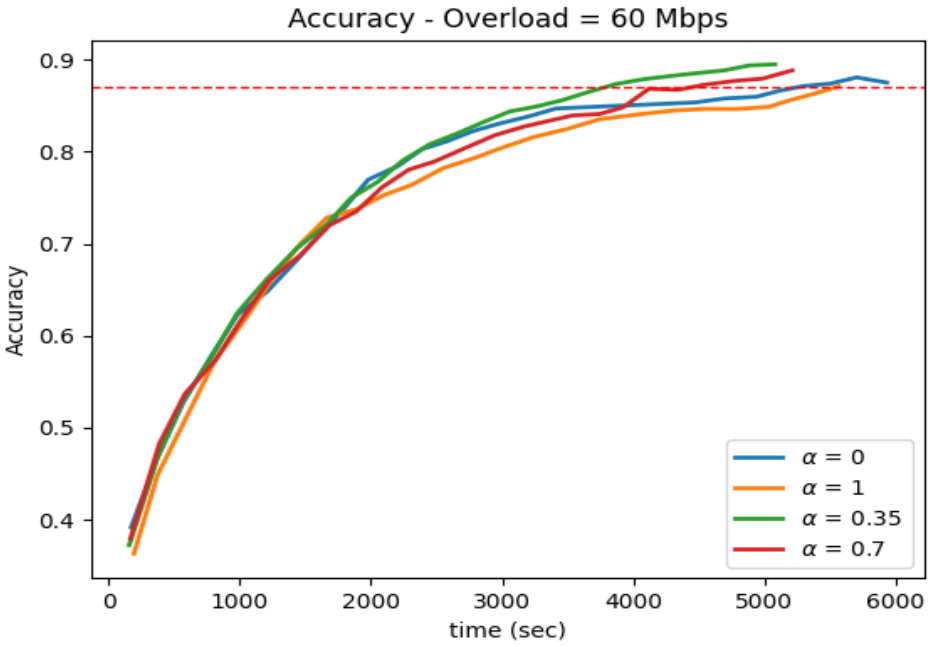

FIGURE 6.27: Delay/Computational Resources Selection: Accuracy of "60 Mbps" Overload Profile-2

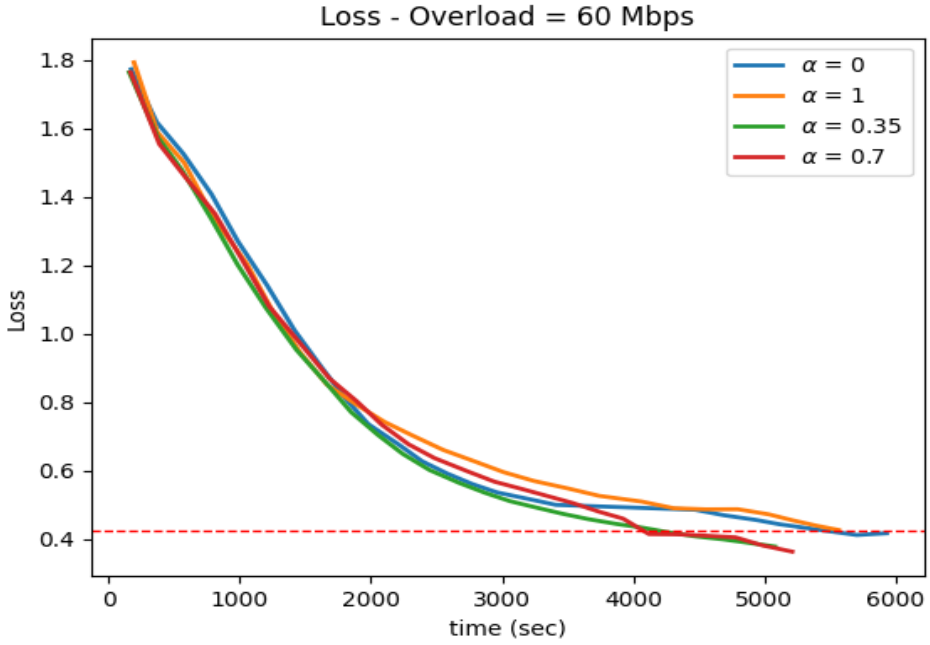

FIGURE 6.28: Delay/Computational Resources Selection: Loss of "60 Mbps" Overload Profile-2

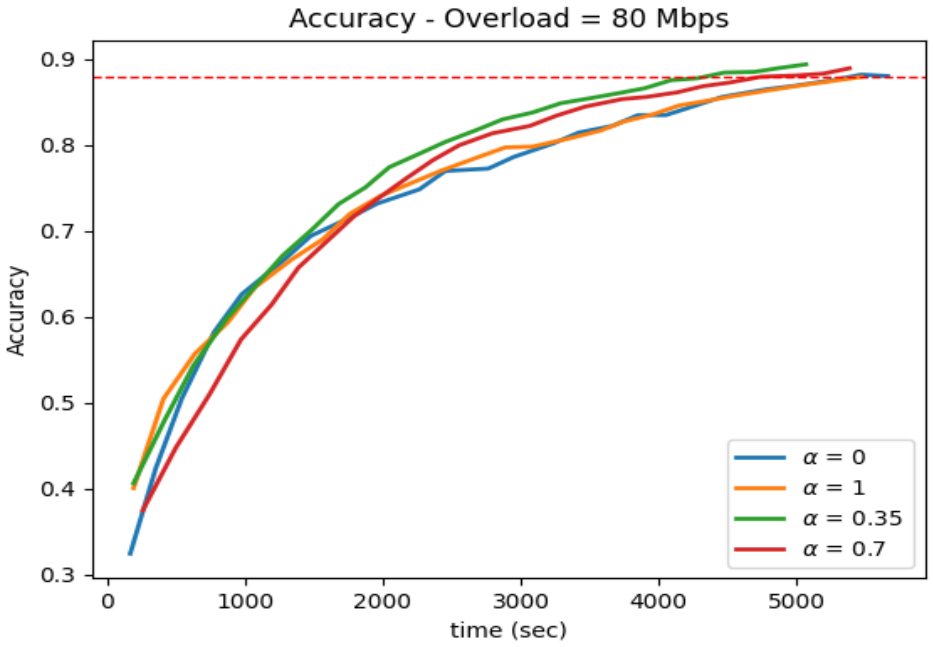

FIGURE 6.29: Delay/Computational Resources Selection: Accuracy of "80 Mbps" Overload Profile-2

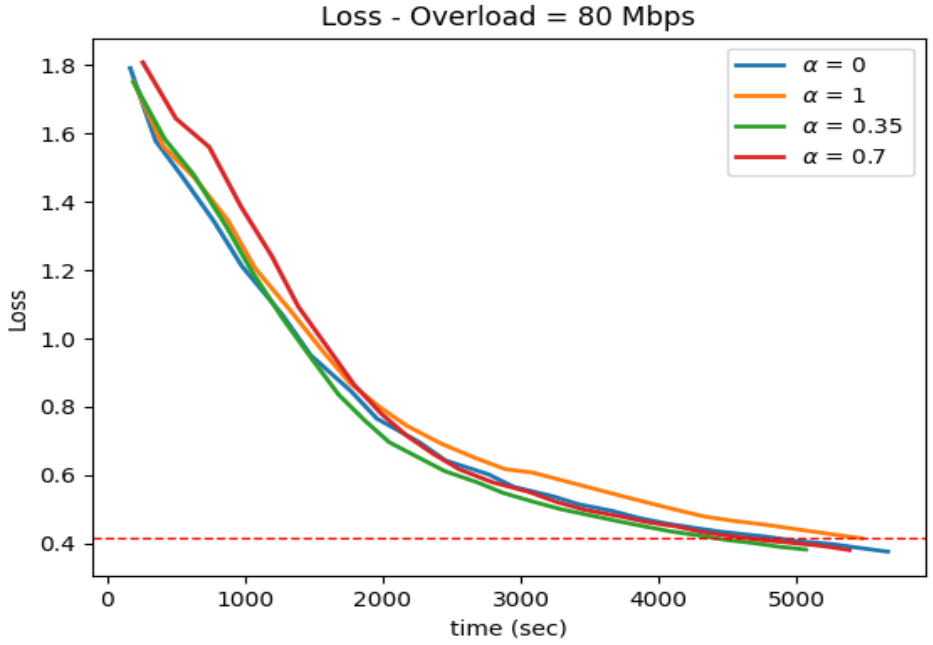

FIGURE 6.30: Delay/Computational Resources Selection: Loss of "80 Mbps" Overload Profile-2

TABLE 6.12: Delay/Computational Resources Selection Conclusion

Problem	*Heterogeneity of FL Clients*
Goal	*Prove that both the communication delay and computational resources must be taken into account in client selection.*
Methods	*Use Profile-2 Overload*
Validation Scenarios	*Overload the links according to Profile-2 with different data rates:* • *40 Mbps* • *60 Mbps* • *80 Mbps*
Metrics	*Accuracy and Loss versus time.*
Results	• *Delay/Computational Resources Selection compensates for the limitation of Delay-based Selection, which cannot perform well when the clients are heterogeneous.* • *The weights of the delay and computational resources must be optimized according to the overloading level and the heterogeneity of the clients.*
Cons (Disadvantages)	*Optimization of weights (α and β)*
Solution	*AI is one of the promising solutions*

6.5 Conclusion

In this thesis, the feasibility of using SDN in Federated Learning was investigated, focusing on the importance of communication resources between FL Clients and the FL Server to improve FL performance in terms of convergence time. The thesis proposed three different algorithms to enhance FL performance: *Dynamic Routing*, *Delay-based Selection Strategy (DSS)*, and *Delay/Computational Resources Selection Strategy (DCSS)*. Validation was conducted using a set of overloading profiles (*Profile-1, Profile-2, and Profile-3*) with varying data rates. The results can be summarized as follows:

- Dynamic Routing helps avoid overloaded routes between the FL Server and FL Clients by selecting less congested routes. However, when congestion exists in all possible routes, client selection strategies are needed to choose the best-performing clients.
- In scenarios with homogeneous clients in terms of computational and memory resources, performance is primarily affected by the delay faced by each client since training times are similar among all clients. Therefore, client selection can be based solely on the delays experienced by clients.
- In heterogeneous-client scenarios where clients have varying computational and memory resources, performance is influenced by two factors: delay due to communication resources and delay due to training time. In such cases, client selection must consider both communication and computational/memory resources. The significance of each resource depends mainly on the level of background application overloading that shares communication resources with FL elements.

6.6 Future Work

The work has shown an enhancement of FL performance using SDN and client selection strategies, especially when both communication delay and computational resources in terms of CPU and Memory have been used to select the clients. It has also been shown that the weight of the delay and computational resources has to be adaptive according to the specific training scenario. As a future work, further investigation of the selection weights has to be done with a larger number of clients. Then, Reinforcement Learning can be exploited to choose the best combination of the selection parameters that achieve the best performance according to the application's needs. In terms of learning aspects, more complex deep learning models may be tested with various types of datasets.

Bibliography

[1] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: 10.1145/3133956.3133982.

[2] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. "Software-defined networking (SDN): a survey". In: *Security and Communication Networks* 9.18 (2016), pp. 5803–5833. DOI: https://doi.org/10.1002/sec.1737.

[3] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. "Opportunities and Research Challenges of Hybrid Software Defined Networks". In: *SIGCOMM Comput. Commun. Rev.* 44.2 (2014), pp. 70–75. ISSN: 0146-4833. DOI: 10.1145/2602204.2602216.

[4] *Ryu SDN Framework*. https://ryu-sdn.org/. Accessed on 07 04, 2023. 2017.

[5] *OpenDaylight Project*. https://www.opendaylight.org/. Accessed on 07 04, 2023. 2016.

[6] *Open Networking Foundation*. https://opennetworking.org/onos/. Accessed on 07 04, 2023. 2023.

[7] J. Zander and R. Forchheimer. "The SOFTNET project: a retrospect". In: *8th European Conference on Electrotechnics, Conference Proceedings on Area Communication*. 1988, pp. 343–345. DOI: 10.1109/EURCON.1988.11172.

[8] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The Road to SDN: An Intellectual History of Programmable Networks". In: *Queue* 11.12 (Dec. 2013), pp. 20–40. ISSN: 1542-7730. DOI: 10.1145/2559899.2560327.

[9] Andrew T. Campbell et al. "Open Signaling for ATM, Internet and Mobile Networks (OPENSIG'98)". In: *SIGCOMM Comput. Commun. Rev.* 29.1 (Jan. 1999), pp. 97–108. ISSN: 0146-4833. DOI: 10.1145/505754.505762.

[10] Rob Enns. *NETCONF Configuration Protocol*. RFC 4741. Dec. 2006. DOI: 10.17487/RFC4741. URL: https://www.rfc-editor.org/info/rfc4741.

[11] *Open Networking Foundation Security Working Group*. https://opennetworking.org/sdn-resources/customer-case-studies/openflow/. Accessed on 07 04, 2023. 2023.

[12] Bruno Astuto A. Nunes et al. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks". In: *IEEE Communications Surveys Tutorials* 16.3 (2014), pp. 1617–1634. DOI: 10.1109/SURV.2014.012214.00180.

[13] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. "Software-defined networking (SDN): a survey". In: *Security and communication networks* 9.18 (2016), pp. 5803–5833.

[14] *Open Networking Foundation (ONF) (2015) OpenFlow Switch Specification*.

[15] *OpenDaylight SDN Controller*. https://www.opendaylight.org/.

[16] Jan Medved et al. "OpenDaylight: Towards a Model-Driven SDN Controller architecture". In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. 2014, pp. 1–6. DOI: 10.1109/WoWMoM.2014.6918985.

[17] R. Enns et al. *Network Configuration Protocol (NETCONF)*. Tech. rep. 2011.

[18] A. Bierman and et al. *RESTCONF Protocol*. Tech. rep. Feb. 2014.

[19] *Open vSwitch*. https://www.openvswitch.org/.

[20] Wenfeng Xia et al. "A Survey on Software-Defined Networking". In: *IEEE Communications Surveys Tutorials* 17.1 (2015), pp. 27–51. DOI: 10.1109/COMST.2014.2330903.

[21] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: 10.1145/3133956.3133982. URL: https://doi.org/10.1145/3133956.3133982.

[22] Tian Li et al. *Fair Resource Allocation in Federated Learning*. 2020. arXiv: 1905.10497 [cs.LG].

[23] Ronald Doku, Danda B. Rawat, and Chunmei Liu. "Towards Federated Learning Approach to Determine Data Relevance in Big Data". In: *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. 2019, pp. 184–192. DOI: 10.1109/IRI.2019.00039.

[24] Mohammed Aledhari et al. "Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Access* 8 (2020), pp. 140699–140725. DOI: 10.1109/ACCESS.2020.3013541.

[25] Qiang Yang et al. "Federated Machine Learning: Concept and Applications". In: *ACM Trans. Intell. Syst. Technol.* 10.2 (Jan. 2019). ISSN: 2157-6904. DOI: 10.1145/3298981.

[26] Yang Liu, Tianjian Chen, and Qiang Yang. "Secure Federated Transfer Learning". In: *CoRR* abs/1812.03337 (2018). arXiv: 1812.03337. URL: http://arxiv.org/abs/1812.03337.

[27] Timothy Yang et al. "Applied Federated Learning: Improving Google Keyboard Query Suggestions". In: *CoRR* abs/1812.02903 (2018). arXiv: 1812.02903. URL: http://arxiv.org/abs/1812.02903.

[28] Andrew Hard et al. "Federated Learning for Mobile Keyboard Prediction". In: *CoRR* abs/1811.03604 (2018). arXiv: 1811.03604. URL: http://arxiv.org/abs/1811.03604.

[29] Md. Mahbubur Rahman et al. "Hospital patients' length of stay prediction: A federated learning approach". In: *Journal of King Saud University - Computer and Information Sciences* 34.10, Part A (2022), pp. 7874–7884. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2022.07.006. URL: https://www.sciencedirect.com/science/article/pii/S1319157822002336.

[30] Soroosh Tayebi Arasteh et al. *Federated learning for secure development of AI models for Parkinson's disease detection using speech from different languages*. 2023. arXiv: 2305.11284 [eess.AS].

[31] Mingzhe Chen, Walid Saad, and Changchuan Yin. "Liquid State Machine Learning for Resource and Cache Management in LTE-U Unmanned Aerial Vehicle (UAV) Networks". In: *IEEE Transactions on Wireless Communications* 18.3 (2019), pp. 1504–1517. DOI: 10.1109/TWC.2019.2891629.

[32] Hao Ye et al. "Machine Learning for Vehicular Networks: Recent Advances and Application Examples". In: *IEEE Vehicular Technology Magazine* 13.2 (2018), pp. 94–101. DOI: 10.1109/MVT.2018.2811185.

[33] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: http://dx.doi.org/10.1561/0400000042.

[34] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: 10.1145/3133956.3133982.

[35] Guowen Xu et al. "VerifyNet: Secure and Verifiable Federated Learning". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 911–926. DOI: 10.1109/TIFS.2019.2929409.

[36] Yanyang Lu and Lei Fan. "An Efficient and Robust Aggregation Algorithm for Learning Federated CNN". In: *Proceedings of the 2020 3rd International Conference on Signal Processing and Machine Learning*. SPML 2020. Beijing, China: Association for Computing Machinery, 2020, pp. 1–7. ISBN: 9781450375733. DOI: 10.1145/3432291.3432303.

[37] Brandon Tran, Jerry Li, and Aleksander Madry. "Spectral Signatures in Backdoor Attacks". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[38] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. "The Limitations of Federated Learning in Sybil Settings". In: *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. San Sebastian: USENIX Association, Oct. 2020, pp. 301–316. ISBN: 978-1-939133-18-2.

[39] Mohammed Aledhari et al. "Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Access* 8 (2020), pp. 140699–140725. DOI: 10.1109/ACCESS.2020.3013541.

[40] Tiansheng Huang et al. "An Efficiency-Boosting Client Selection Scheme for Federated Learning With Fairness Guarantee". In: *IEEE Transactions on Parallel and Distributed Systems* 32.7 (2021), pp. 1552–1564. DOI: 10.1109/TPDS.2020.3040887.

[41] Takayuki Nishio and Ryo Yonetani. "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761315.

[42] Sawsan Abdulrahman et al. "FedMCCS: Multicriteria Client Selection Model for Optimal IoT Federated Learning". In: *IEEE Internet of Things Journal* 8.6 (2021), pp. 4723–4735. DOI: 10.1109/JIOT.2020.3028742.

[43] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 20–22 Apr 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[44] Luping WANG, Wei WANG, and Bo LI. "CMFL: Mitigating Communication Overhead for Federated Learning". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, pp. 954–964. DOI: 10.1109/ICDCS.2019.00099.

[45] Kevin Hsieh et al. "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 629–647. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh.

[46] Virginia Smith et al. "Federated Multi-Task Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf.

[47] Yuwei Wang and Burak Kantarci. "A Novel Reputation-aware Client Selection Scheme for Federated Learning within Mobile Environments". In: *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 2020, pp. 1–6. DOI: 10.1109/CAMAD50429.2020.9209263.

[48] Eunil Seo, Dusit Niyato, and Erik Elmroth. "Auction-based Federated Learning using Software-defined Networking for resource efficiency". In: *2021 17th International Conference on Network and Service Management (CNSM)*. 2021, pp. 42–48. DOI: 10.23919/CNSM52442.2021.9615554.

[49] Venkatraman Balasubramanian et al. "Intelligent Resource Management at the Edge for Ubiquitous IoT: An SDN-Based Federated Learning Approach". In: *IEEE Network* 35.5 (2021), pp. 114–121. DOI: 10.1109/MNET.011.2100121.

[50] Changqiao Xu et al. "Optimal Information Centric Caching in 5G Device-to-Device Communications". In: *IEEE Transactions on Mobile Computing* 17.9 (2018), pp. 2114–2126. DOI: 10.1109/TMC.2018.2794970.

[51] Venkatraman Balasubramanian et al. "Edge-Boost: Enhancing Multimedia Delivery with Mobile Edge Caching in 5G-D2D Networks". In: *2019 IEEE International Conference on Multimedia and Expo (ICME)*. 2019, pp. 1684–1689. DOI: 10.1109/ICME.2019.00290.

[52] Prohim Tam, Sa Math, and Seokhoon Kim. "Efficient resource slicing scheme for optimizing federated learning communications in software-defined IoT networks". In: *Journal of Internet Computing and Services* 22.5 (2021), pp. 27–33.

[53] Ramin Firouzi and Rahim Rahmani. "A Distributed SDN Controller for Distributed IoT". In: *IEEE Access* 10 (2022), pp. 42873–42882. DOI: 10.1109/ACCESS.2022.3168299.

[54] Zeyue Xue et al. "A Resource-Constrained and Privacy-Preserving Edge-Computing-Enabled Clinical Decision System: A Federated Reinforcement Learning Approach". In: *IEEE Internet of Things Journal* 8.11 (2021), pp. 9122–9138. DOI: 10.1109/JIOT.2021.3057653.

[55] Jie Cui et al. "Collaborative Intrusion Detection System for SDVN: A Fairness Federated Deep Learning Approach". In: *IEEE Transactions on Parallel and Distributed Systems* 34.9 (2023), pp. 2512–2528. DOI: 10.1109/TPDS.2023.3290650.

[56] Jiangang Shu et al. "Collaborative Intrusion Detection for VANETs: A Deep Learning-Based Distributed SDN Approach". In: *IEEE Transactions on Intelligent Transportation Systems* 22.7 (2021), pp. 4519–4530. DOI: 10.1109/TITS.2020.3027390.

[57] Yunlong Lu et al. "Blockchain and Federated Learning for Privacy-Preserved Data Sharing in Industrial IoT". In: *IEEE Transactions on Industrial Informatics* 16.6 (2020), pp. 4177–4186. DOI: 10.1109/TII.2019.2942190.

[58] Gunasekaran Raja et al. "SP-CIDS: Secure and Private Collaborative IDS for VANETs". In: *IEEE Transactions on Intelligent Transportation Systems* 22.7 (2021), pp. 4385–4393. DOI: 10.1109/TITS.2020.3036071.

[59] Akhil Krishna et al. "Intrusion Detection and Prevention System Using Deep Learning". In: *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*. 2020, pp. 273–278. DOI: 10.1109/ICESC48915.2020.9155711.

[60] Daniel J. Beutel et al. *Flower: A Friendly Federated Learning Research Framework*. 2022. arXiv: 2007.14390 [cs.LG].

[61] *GNS3 Software*. https://www.gns3.com/. Accessed on 09 04, 2023.

[62] *Oracle Virtual Box*. https://www.virtualbox.org/. Accessed on 09 04, 2023.

[63] *Networkx Library*. https://networkx.org/. Accessed on 09 04, 2023.

[64] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[65] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 20–22 Apr 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[66] *Iperf Tool*. https://iperf.fr/. Accessed on 09 04, 2023.

[67] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[68] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[69] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: Curran Associates, Inc., 2019. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[70] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[71] *CIFAR-10 (Canadian Institute for Advanced Research)*. https://www.cs.toronto.edu/~kriz/cifar.html, note = Accessed on 09 04, 2023,