NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Proof-of-Concept solution for RE-CENT service method design

Ilias C Tziviskos

Supervisor(or supervisors): Dr. Nikos Passas, Prof. Dionysis Xenakis

ATHENS

MAY 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Λυση Αποδειξης Ιδεας Για Σχεδιαση Μεθοδου Υπηρεσιας Re-Cent

Ηλίας Χ Τζιβίσκος

Επιβλέπων (Επιβλέποντες): Δρ. Νίκος Πασσάς, Καθ. Διονύσης Ξενάκης

ΑΘΗΝΑ

ΜΑΪΟΣ 2023

BSc THESIS

Proof-of-Concept solution for RE-CENT model service

Ilias C Tziviskos
S.N.: 1115201500253

SUPERVISORS: Dr. Nikos Passas, Prof. Dionysis Xenakis

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Λυση Αποδειξης Ιδεας Για Σχεδιαση Μεθοδου Υπηρεσιας Re-Cent

Ηλίας Χ Τζιβίσκος
Α.Μ.: 1115201500253

ΕΠΙΒΛΕΠΟΝΤΕΣ Δρ. Νίκος Πασσάς, Καθ. Διονύσης Ξενάκης

ABSTRACT

With the increase in mobile devices and simultaneously the volume of data received and transmitted by them, the current mobile network architecture faces challenges in accommodating them. In recent years, innovative network architectures have emerged, providing solutions to the issues present in the current network architecture. One such method is the RE-CENT service design approach. In this thesis, we present a proof-of-concept solution based on the RE-CENT service method, by utilizing widely available hardware and software. We analyze i) the architecture of this solution by breaking it down to its main components as well as the technologies used for both the network and application layer, ii) the steps of the protocol designed for their communications and iii) the test cases that measure the effectiveness of the solution. Through our results we showed the viability of the proof-of-concept solution, having no penalty in performance no matter the number of concurrent mobile users and amount of data requested and transmitted through the network.

SUBJECT AREA: Telecommunications

KEYWORDS: SDN, IoT, OpenFlow, video streaming, Wi-Fi

ΠΕΡΙΛΗΨΗ

Με την αύξηση των κινητών συσκευών και παράλληλα του όγκου δεδομένων που λαμβάνονται και μεταδίδονται από αυτές, η τωρινή αρχιτεκτονική του κινητού δικτύου αντιμετωπίζει προκλήσεις στην προσαρμογή τους. Τα τελευταία χρόνια, εμφανίζονται καινοτόμες αρχιτεκτονικές δικτύου που παρέχουν λύσεις στα προβλήματα που υπάρχουν στην τωρινή αρχιτεκτονική δικτύου. Μία τέτοια μέθοδος είναι η προσέγγιση σχεδίασης υπηρεσιών RE-CENT. Σε αυτήν τη διατριβή, παρουσιάζουμε μία λύση προσέγγισης απόδειξης βασισμένη στη μέθοδο υπηρεσίας RE-CENT, χρησιμοποιώντας ευρέως διαθέσιμο υλικό και λογισμικό. Αναλύουμε i) την αρχιτεκτονική αυτής της λύσης, διαχωρίζοντας τα κύρια της συστατικά καθώς και τις τεχνολογίες που χρησιμοποιούνται τόσο στο επίπεδο του δικτύου όσο και της εφαρμογής, ii) τα βήματα του πρωτοκόλλου που σχεδιάστηκε για την επικοινωνία τους και iii) τις περιπτώσεις δοκιμών που μετρούν την αποτελεσματικότητα της λύσης. Μέσω των αποτελεσμάτων μας, αποδείξαμε την εφικτότητα της λύσης, χωρίς καμία ποινή στην απόδοση, ανεξαρτήτως αριθμού ταυτόχρονων κινητών χρηστών και ποσότητας δεδομένων που αιτούνται και μεταδίδονται μέσω του δικτύου.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τηλεπικοινωνίες

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: SDN, IoT, OpenFlow, video streaming, Wi-Fi

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION AND MOTIVATION

Over the last decade, mobile technology has had an incredible impact on society and business. From a recent report [1], mobile subscriptions have surpassed 8 billion and it is forecasted that will reach 9 billion in the coming years. The commercialization of 5G by service providers globally has brought an influx of subscriptions by 98 million, bringing the total number to 570 million during the third quarter of 2021, while it is estimated that this number will reach 660 million by the end of 2021. In parallel, 4G subscriptions are still dominant, totaling more than 4.7 billion, but projected to decline to around 3.3 billion by the end of 2027 as subscribers migrate to 5G. The same principle applies for 3G and 2G subscriptions as well.

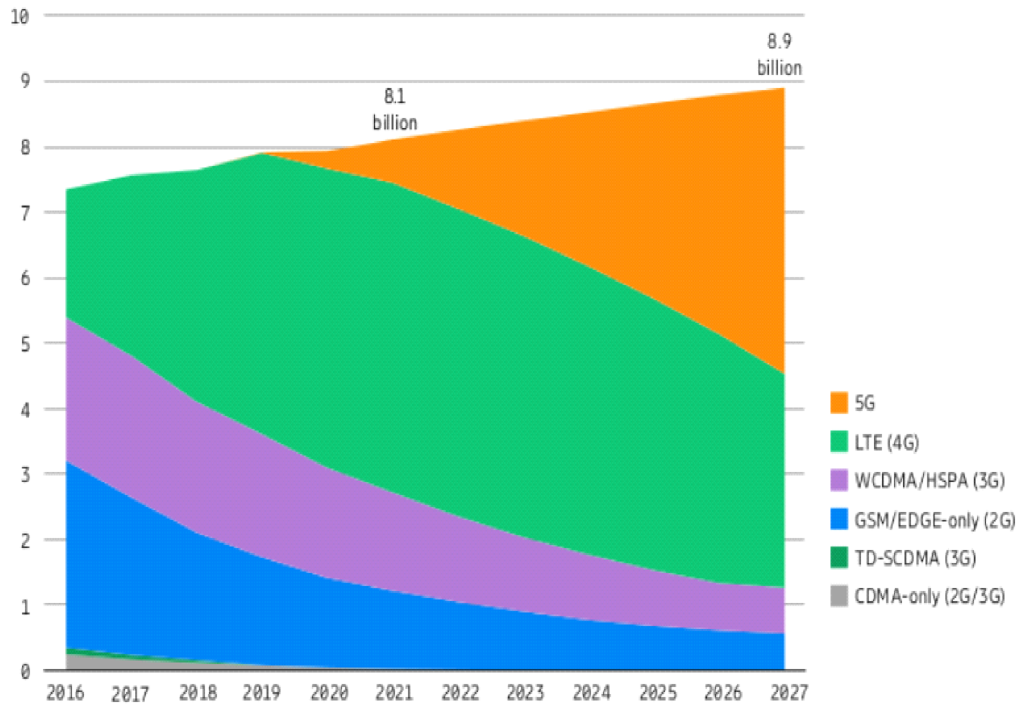


Figure 1: Mobile subscriptions by technology (billion) [2]

At the same time, Internet-of-Things (IoT) finds use on applications related to large-scale smart metering, smart buildings, transportation, logistics and security and surveillance. Most of those applications operate under several broadband cellular network generations, with 2G and 3G being dominant. As 5G New Radio (NR) [2] is being introduced in old and new spectrum, and applications becoming more demanding in throughput and latency, more and more IoT devices connected via 2G and 3G have been in slow decline since 2019 as it is speculated that by the end of 2027 40 percent of cellular IoT connections will be broadband IoT (4G/5G), with 4G connecting the majority.

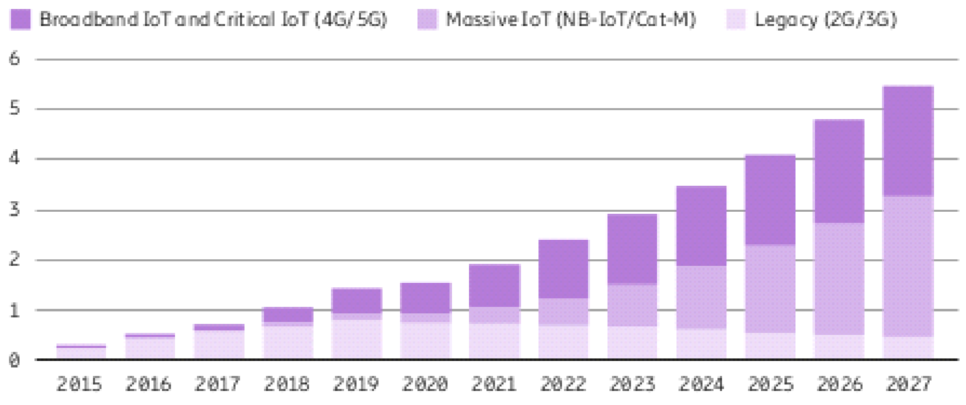


Figure 2: Cellular IoT connections by segment and technology (billion) [2]

Connected devices will exceed 50 billion by 2025, including IoT devices. A major driving factor for this imminent explosion in devices can be attributed to the low cost of production as well as new application opportunities. For mobile cellular devices, over 400 5G enabled smartphones to have been launched, accounting for 23 percent of global volume. This is significantly higher than 4G's volume for the same cycle, being 8.3 percent. For the later, IoT devices have been rising due to their low-cost and ease of adaptation of several technologies, e.g., broadband IoT with 4G and 5G, Massive IoT with NarrowBand-IoT (NB-IoT) or Cat-M, as well as Legacy with 2G and 3G.

As the number of devices grows, data usage follows in similar pattern, estimated to surpass the 160 Exabytes per month by the year 2024. Smartphones continue to be at the epicenter of generating most of the mobile data traffic, calculated to be around the 97 percent. This growth in mobile data traffic per smartphone can be attributed to i) the improved device capabilities, ii) an increase in data-intensive content, with video streaming services being the most popular among users, and iii) more data consumption due to continued improvements in the performance of deployed networks.

In view of that, the telecommunication industry has put considerable effort towards the consolidation of the fifth generation (5G) of mobile data networks. Firstly, with the release of a standalone version of the NR, 3GPP has specified the architectural options towards the evolution of LTE-compliant networks to fully-fledged 5G systems, incorporating forward-thinking radio access technologies, such as multi-GHz spectrum communications, dynamic beamforming, and small-sized cellular hotspots. Secondly, via the means of network softwarization and virtualization, the mobile network operators (MNOs) will be in position to dynamically isolate network resources towards serving specific vertical applications with guaranteed quality of service (QoS), or quality of experience (QoE), a.k.a. network slicing [3], or even open up their core network functionality to other third-party (OTT) service providers, e.g. authentication, authorization, accounting (AAA).

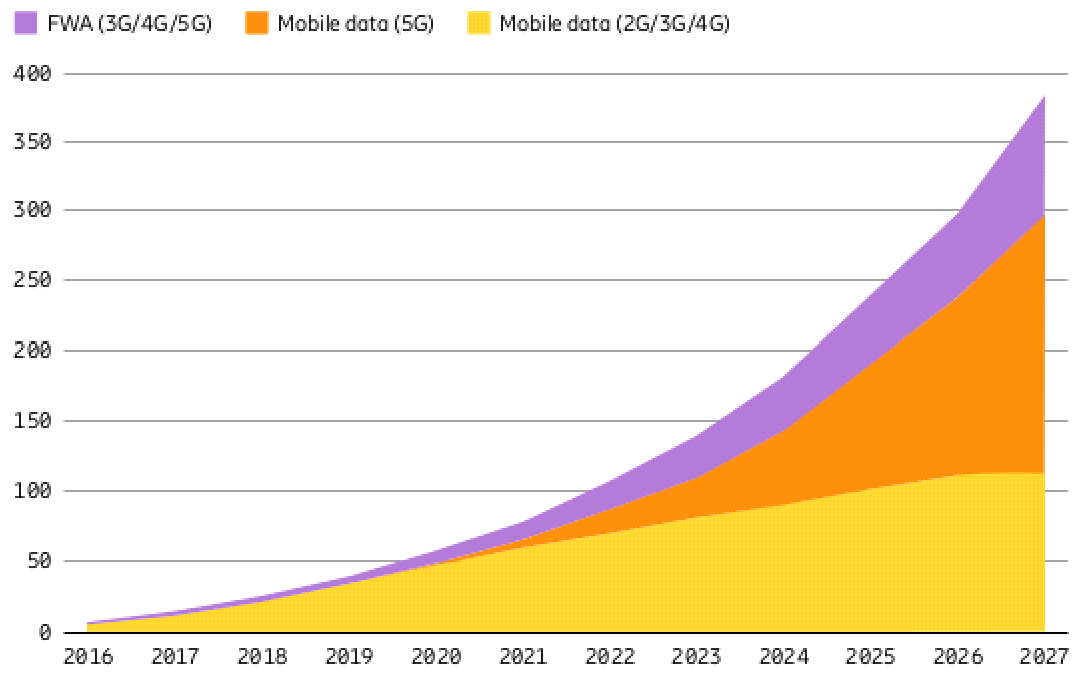


Figure 3: Global mobile network data traffic (EB per month) [2]

The current network architecture for mobile data access is carried out through heterogenous, multi-tier, multi-domain, multi-owned RAN islands, ranging from user-installed access points for private use (e.g., Wi-Fi routers), to large-scale mobile data networks enabling nation-wide coverage and world-wide roaming. Regardless, for clients to access the cellular mobile data networks there need to be a predetermined data usage plan, agreed with an MNO in the form of fixed term contracts. For non-cellular fixed wireless access service for local area networks (WLANs), the option for long-term access based on leasing backhaul connectivity to the Internet exists, yet again governed by an internet service provider (ISP) in the form of service license agreements (SLAs). This allows a set number of users to access the Internet by using given credentials. For short-term access to the Internet, users are forced either to use money payments to acquire a set amount of data or a limited time window, or with “free-of-charge” and open networks by registering to a web server (e.g., captive portal) via a social media account, with which the users agree on having their personal data harvested.

Recently, contract-less mobile data access models have seen an increase in popularity. Contract-less models open up the currently proprietary network core of telecommunication standards, allowing for more flexible service while on the same time, giving the opportunity for OTT vendors to be enter today’s market. One of many proposed solutions is the REsource sharing model for user-CENTric digital content delivery over beyond 5G mobile data networks (RE-CENT) model [4].

In heterogeneous wireless networks, end users utilize different radio access technologies (RATs) to access the available network tiers, having as an entry key pre-cached access credentials provided by their home network operator, e.g., subscriber ID, IMSI, social media account, network keys and passwords. Depending on the RAT and the access rights of end users, mobile data access is governed by factors like i) the coverage provided by the accessible network tiers in the near area, ii) the status of nearby attachment points (considering the additional load offered by other users and the backhaul connectivity available), and iii) the mobile data usage plan agreed with the home operator per user.

In contrast, the RE-CENT model enables end users, access points and cellular base stations to share, trade and consume their network assets (backhaul links, Internet connectivity, cached content, etc.) in real-time and without the need for SLAs to be established beforehand. Instead, end users (termed as RE-CENT clients) and service providers (termed as RE-CENT servers) can set up on-the-fly service agreements and implement blockchain-backed service charging on a per delivered video chunk basis in line with their current service requirements, coverage, available assets and preferences.

The implementation of the RE-CENT model can be broken down in three phases, i) service discovery and pairing, ii) service negotiation and parameterization and iii) online service management and charging.

During the service discovery and pairing phase, RE-CENT clients communicate their service requests to nearby RE-CENT servers, by specifying necessary parameters regarding the target content, such as URL, author, keywords etc., and the target QoE key performance indicators (KPIs). Some of the KPIs specified by RE-CENT users are min/average video bitrate, delay tolerance, packet loss rate, available buffer, target screen resolution. The service discovery and pairing can be implemented by either using a client-driven or server-driven approach.

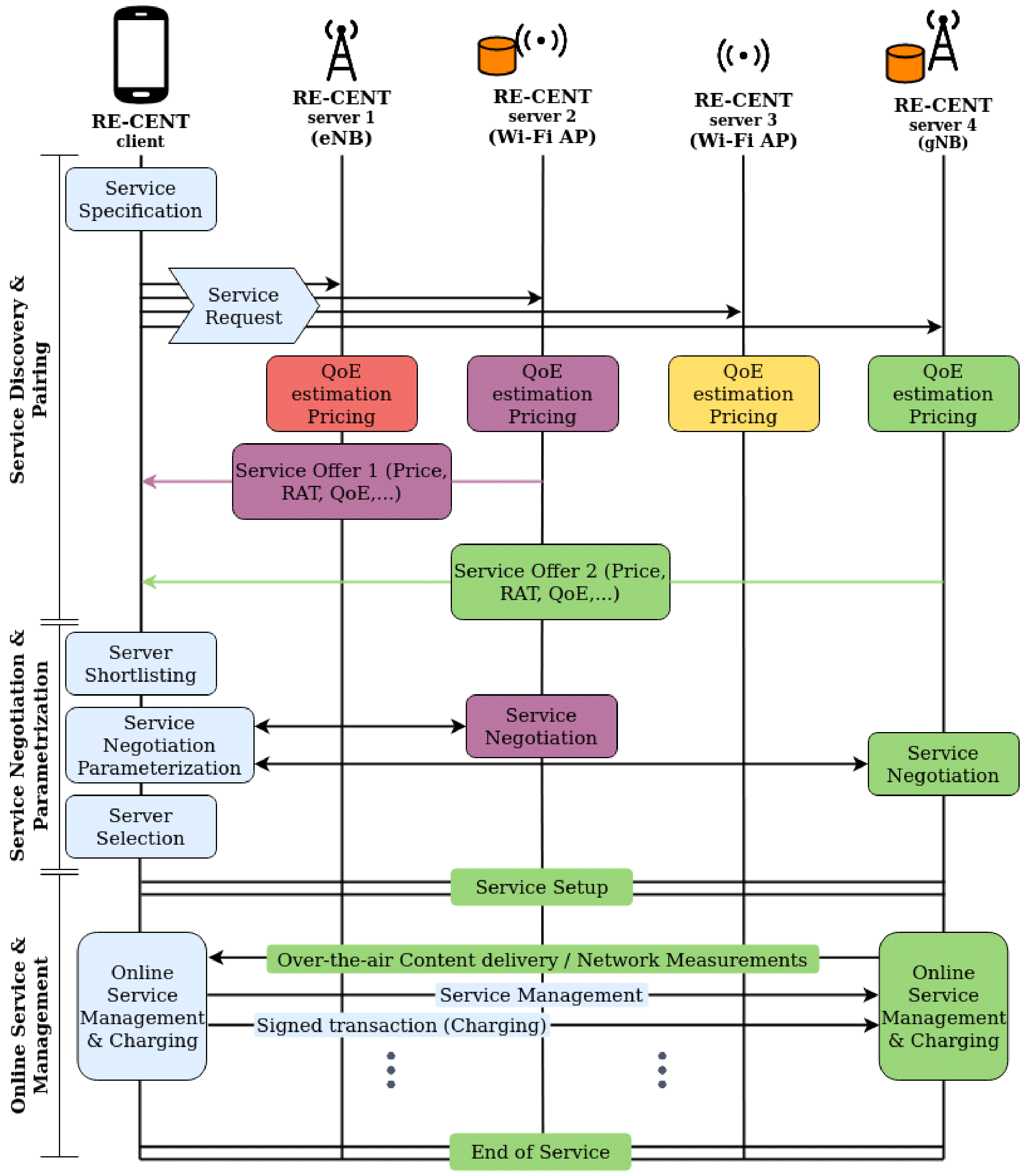

RE-CENT client
RE-CENT server 1 (eNB)
RE-CENT server 2 (Wi-Fi AP)
RE-CENT server 3 (Wi-Fi AP)
RE-CENT server 4 (gNB)
Service Discovery & Pairing
Service Negotiation & Parametrization
Online Service & Management
Service Specification
Service Request
QoE estimation Pricing
QoE estimation Pricing
QoE estimation Pricing
QoE estimation Pricing
Service Offer 1 (Price, RAT, QoE,...)
Service Offer 2 (Price, RAT, QoE,...)
Server Shortlisting
Service Negotiation Parameterization
Service Negotiation
Service Negotiation
Server Selection
Service Setup
Online Service Management & Charging
Over-the-air Content delivery / Network Measurements
Service Management
Signed transaction (Charging)
Online Service Management & Charging
End of Service

Figure 4: user-driven RE-CENT service phases [4]

Under the client-driven approach, RE-CENT clients broadcast their service requests to nearby RE-CENT servers. In turn, RE-CENT servers should be able to estimate its capability to carry out a service request on the basis of the parameters specified by the RE-CENT client and the locally available asset pools (content, spectrum, Internet connectivity, processing and storage capacity, RAT interfaces, etc.). After the QoE estimation, RE-

CENT servers respond with a targeted service offer, showing its interest on servicing the user's request.

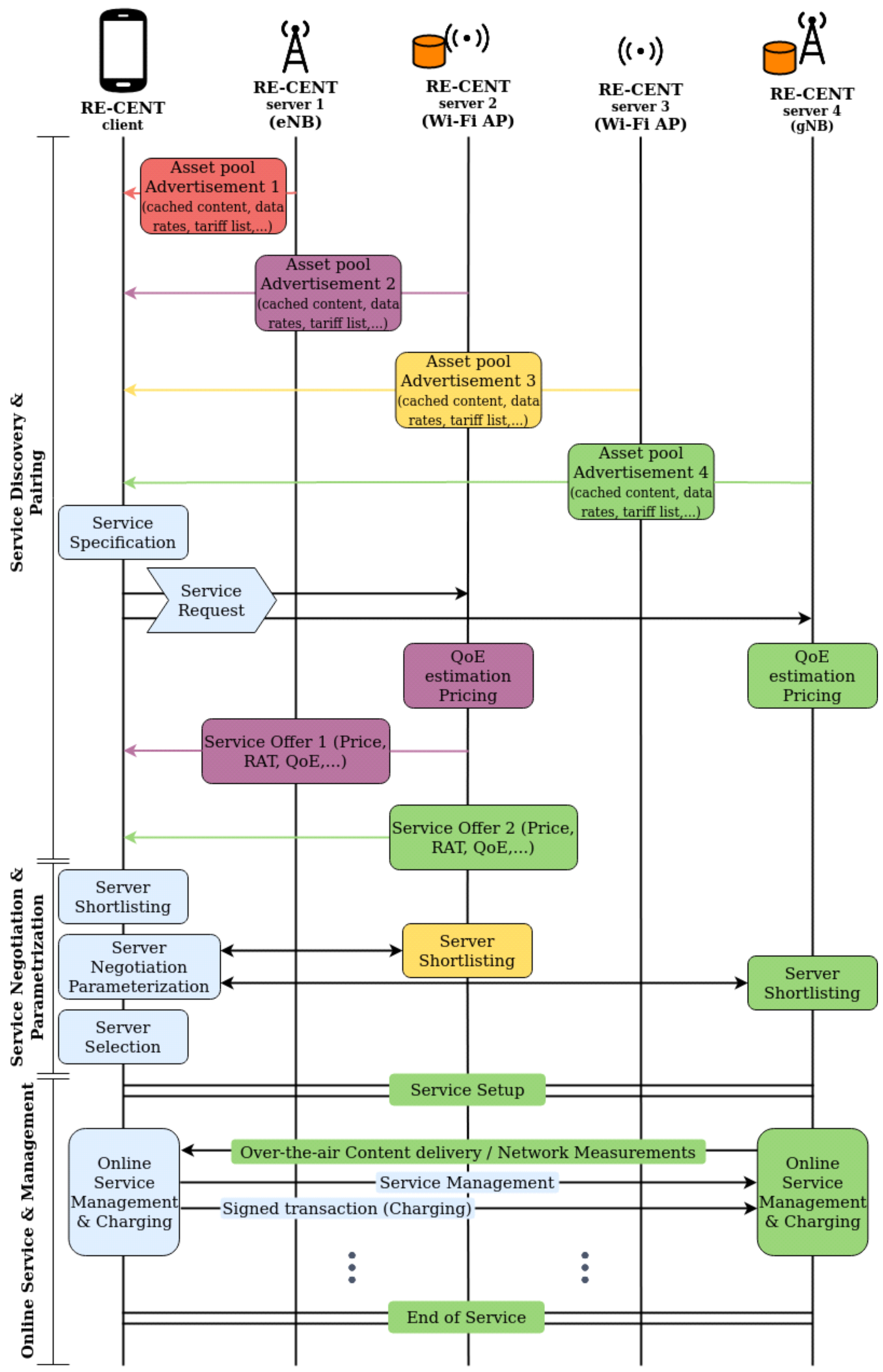

RE-CENT client
RE-CENT server 1 (eNB)
RE-CENT server 2 (Wi-Fi AP)
RE-CENT server 3 (Wi-Fi AP)
RE-CENT server 4 (gNB)
Service Discovery & Pairing
Asset pool Advertisement 1 (cached content, data rates, tariff list,...)
Asset pool Advertisement 2 (cached content, data rates, tariff list,...)
Asset pool Advertisement 3 (cached content, data rates, tariff list,...)
Asset pool Advertisement 4 (cached content, data rates, tariff list,...)
Service Specification
Service Request
QoE estimation Pricing
QoE estimation Pricing
Service Offer 1 (Price, RAT, QoE,...)
Service Offer 2 (Price, RAT, QoE,...)
Service Negotiation & Parametrization
Server Shortlisting
Server Negotiation Parameterization
Server Shortlisting
Server Shortlisting
Server Selection
Online Service & Management
Service Setup
Online Service Management & Charging
Over-the-air Content delivery / Network Measurements
Service Management
Signed transaction (Charging)
Online Service Management & Charging
End of Service

Figure 5: server-driven RE-CENT service phases [4]

The server-driven approach has the RE-CENT servers advertise their available asset pools by broadcasting to users in close proximity. Advertised assets include locally cached content, data rates for Internet connectivity, tariff list, etc. RE-CENT users receiving these advertisements yet again need to target RE-CENT users to specify their service request according to the servers' assets and receive service offers and select the most suitable server (pairing).

The service negotiation and parameterization phase resolves around the server selection from a RE-CENT user and parameter negotiation between RE-CENT users and the shortlisted servers necessary for the over-the-air content delivery. RE-CENT clients, having received offers from nearby RE-CENT servers, deploy their own server selection strategies, taking into consideration criteria with regards to i) the offer's included price, ii) the RAT options available by the RE-CENT server, iii) QoE KPIs, iv) how reputable the server is and v) other service implementation options provided by the server (e.g. support of selected codec, use of minimum encryption). With the offers shortlisted, clients initiate negotiations with the remaining RE-CENT servers around service parameters spanning the entire protocol stack. The target QoE KPIs are concluded having as basis the initial request/offers. Following, parameters regarding the over-the-air content delivery are specified such, but not limited to, the RAT technology to be utilized, the spectrum bands through which the delivery will take place and the encryption protocol to be used. Finally, the service peers should also agree on the payment relay service, if that is desired, for reducing the on-chain costs attributed to service charging implementations or use micro-payments.

After the negotiation and parameterization phase has concluded, a server has been selected and the online service and management phase is initiated. Throughout this phase, the RE-CENT client and server are responsible for establishing, maintaining, and terminating the mobile video service at network-level. From the blockchain-level perspective, the agreed payment timeplan from the negotiation and parameterization phase is followed by the service peers, to implement blockchain-backed charging by utilizing either a direct on-chain P2P payments, or off-chain through SC-certified payment methods. According to the trust a peer has, the agreed payment plan can be tight in case of service delivery among untrusted peers, or a single transaction can be implemented for assured trustworthy peers. To ensure service continuity, network-level interactions are necessary for handling mobility management and QoE-driven service provisioning deployment. Under the RE-CENT mobile data access model, service management logic is transferred to the client, assuming user-driven network-assisted service provisioning. The client is responsible for any sudden service discontinuity (e.g., either due to reliability issues with the selected RE-CENT server, or due to user's mobility), taking actions to control for such unforeseen events by incorporating network measurements provided by the server.

Focusing on the advantages and disadvantages the two approaches of the service discovery and pairing are modeled, the server-driven RE-CENT service discovery and pairing has the RE-CENT servers advertise their generic service offers in timed intervals without the need of RE-CENT users being present. This has the disadvantage of using up precious network resources to broadcast a large number of messages. In addition, mobile devices belonging to RE-CENT clients have to "listen" to said broadcasts, further straining the battery usage of their devices. On the other hand, the user-driven service discovery and pairing has the advantage of having the RE-CENT client first go through a service specification and broadcasting its request to nearby RE-CENT servers reactively. The targeted nature of this procedure uses up less network resources, by using the medium less freely, allowing for a more reactive service control. Although, this procedure can still put a strain on the battery life of a device, it will still be noticeably less than the server-driven approach.

In this paper, we give a proof-of-concept implementation of the user-driven RE-CENT service model analyzed above, due to its resourcefulness of network resources and the targeted nature of the protocol towards mobile terminal users, by utilizing modern and broadly accessible technologies in development tools, the software and hardware.

The remainder of the paper is structured as follows. In Section 2 we list key technologies that can aid in the modernization of the current stagnant network architecture as well as used for the development of the aforementioned protocol. In Section 3 we give a description of the proof-of-concept solution by enumerating the components that take part, focusing on their functional split. Section 4 goes over the implementation of programs used for each component. Section 5 we present the performance evaluation of the proposed proof-of-concept implementation. Finally, we conclude in Section 6 and present directions for future.

2 KEY TECHNOLOGIES

In this section, the technologies aiding to the transition of the network's stagnant state are enumerated. SDN technology was chosen due to its ability on separating the control and forwarding plane, allowing for a more flexible approach to already existing implementations. OpenFlow protocol allows for more thorough packet inspection, by not being limited to the conventional ways of routing and switching already available to the current network architecture. A big number of network gear is available in the market have built-in support for the OpenFlow protocol. For devices that do not support this kind of functionality the Open vSwitch software can be utilized to enable the OpenFlow protocol functionality. Multi-Access Edge Computing (MEC) aims to enhance the network performance and reliability by leveraging edge resources residing within coverage of 5G cellular networks. O-RAN aims at expanding the ecosystem, with more vendors providing the building blocks, there is more innovation and more options for OTTs, adding new services. Android gives a development platform for creating applications on a variety of devices, by accessing embedded functionalities ranging from Bluetooth through Wi-Fi protocols. The blockchain technology has gained popularity over the last decade with the emergence of cryptocurrencies. The technology can be used for fast payment plans, outgrowing the fixed monthly payment plans and converting to a pay-as-you-go chunk based content servicing.

2.1 SDN

Conventional networks utilize special algorithms implemented on dedicated devices to control and monitor the data flow in the network, managing routing paths and determining how different devices are interconnected in the network. In general, these routing algorithms and sets of rules are implemented in dedicated hardware components such as Application Specific Integrated Circuits (ASICs), designed for performing specific operations. As an example, the packet forwarding process, in a conventional network, upon the reception of a packet, an ASICs uses a set of rules embedded in its firmware to find the destination device as well as the routing path for that packet. This operation takes place in inexpensive routing devices. More expensive routing devices can treat different packet types in different manners based on their nature and contents. Such a customized local router setup allows more efficient traffic congestion and prioritization control.

As the number of connected devices is increasing, the amount of data applications transmit and receive increases as well. The current network hardware poses severe limitations as the demand for scalability, security, reliability, and network speed increases, thus hindering their performance. Due to the hardwired nature of routing rules, they lack flexibility in dealing with different packet types. In addition, the aforementioned network devices make up the backbone of the Internet; have to be adaptable to changes both in hardware or software without being strenuous.

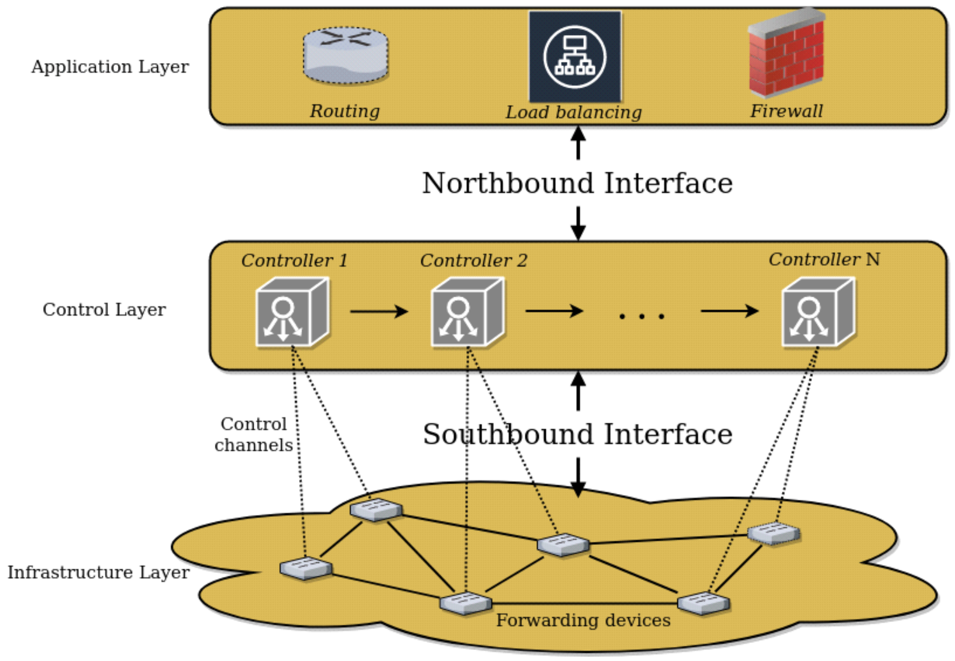


Figure 6: Software Defined Network Architecture

A solution to the above problem is known as Software-Defined Networking (SDN), a dynamic, manageable, cost-effective, and adaptable architecture. As it is illustrated in Figure 6, the control and forwarding planes are separated, giving a more flexible architecture than the traditional network architecture. SDN controllers located in the control layer are a logically centralized entity, overviewing demands from the application layer, via the Northbound Interface, and transferring them to the Infrastructure layer via the Control to Data-Plane Interface (CDPI), also known as Southbound Interface. Between a SDN Controller and forwarding devices, a controller has programmatic control over a device's forwarding operations, advertisement of capabilities, statistics reporting and event notification, in an open and vendor-neutral interoperable way.

Focusing on the control layer of the architecture and specifically the controller component, several SDN-Controller software has been developed for the aforementioned functionality, providing an ample selection for developers. The software available can be categorized in i) open and community-driven initiatives and ii) vendor specific. Some community-driven initiatives are Open Network Operating System (ONOS) [5], OpenDayLight [6], Faucet [7], RYU controller framework [8] and Floodlight [9] to name a few, while Nuage Virtualized Services Controller (VSC) [10], by Nokia, lightly.io [11], by PANTHEON.tech, and VortiQa Open Network Director [12], by Freescale Semiconductor all fall in the vendor specific software category.

OpenFlow

In the SDN architecture, the Southbound interface is used to interconnect SDN controllers with the forwarding devices on the Infrastructure layer. Although this interface uses several protocols to establish this communication, with some being the Forwarding and Control Element Separation (ForCES) [12], Protocol Oblivious Forwarding (POF) [13], OpFlex [14], OpenState [15], the protocol that has seen more extensive use is the OpenFlow protocol [16]. With the adoption of the protocol from the Open Networking Foundation [17], a big number of vendors have included OpenFlow in their devices. This justifies the major companies and organizations deploying OpenFlow include Google, Alibaba, AT&T, the U.S. National Security Agency (NSA) and Microsoft among many others.

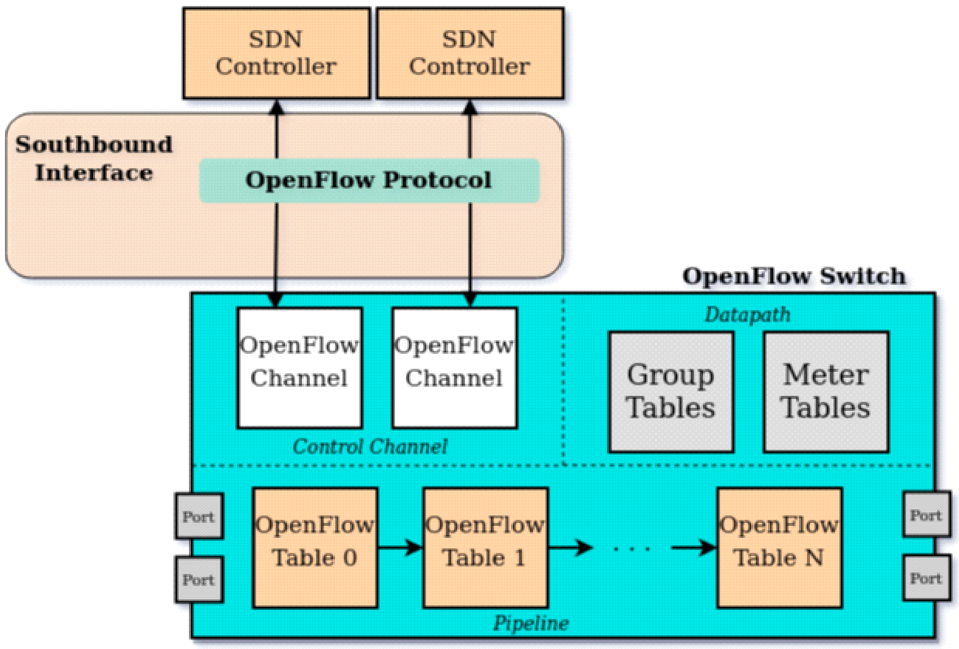


Figure 7: OpenFlow Switch Architecture

OpenFlow-compliant switches come in two types: OpenFlow-only, and OpenFlow-hybrid. OpenFlow-only switches support only OpenFlow operation, in those switches all packets are processed by the OpenFlow pipeline and cannot be processed otherwise. OpenFlow-hybrid switches support both OpenFlow operation and normal Ethernet switching operation, i.e., traditional L2 Ethernet switching, VLAN isolation, L3 routing (IPv4 routing, IPv6 routing...), ACL and QoS processing. Those switches should provide a classification mechanism outside of Open-Flow that routes traffic to either the OpenFlow pipeline or the normal pipeline. An OpenFlow-hybrid switch may also allow a packet to go from the OpenFlow pipeline to the normal pipeline through the NORMAL and FLOOD reserved ports.

The OpenFlow pipeline is part of every OpenFlow-compliant Logical Switches, consisting of a set of flow tables and a group table, performing packet lookups and forwarding. Matching begins at the first flow table, in priority order, and may continue to additional flow tables down the pipeline.

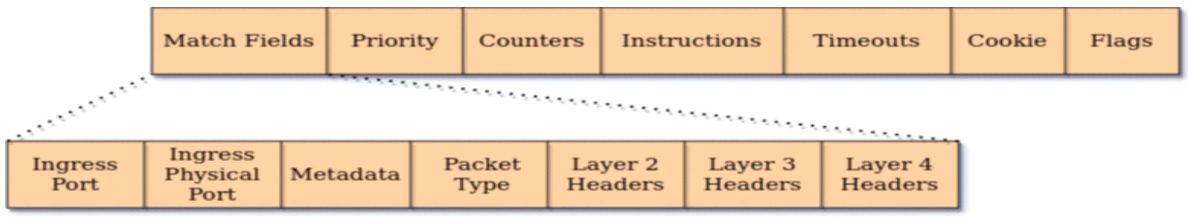


Figure 8: OpenFlow Entry specification

Each flow table on the switch contains a set of flow entries, consisting of:

- match fields: to match against packets. It consists of the ingress port, protocol packet headers from Layers 2, 3 and 4 (e.g., Ethernet, IPv4, IPv6, TCP, UDP, ARP, ICMP, MPLS), a packet type value, as well as optional fields such as metadata taken from the previous table.
- counters: updated when packets are matched. In particular, the number of bytes and packets matched against a flow entry and duration the entry has been established to the switch's table tracked in second and nanosecond precision.
- instructions: to modify the action set or pipeline processing.
- timeouts: maximum amount of time or idle time before flow is expired by the switch.
- cookie: opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.
- flag: alter the way flow entries are managed.

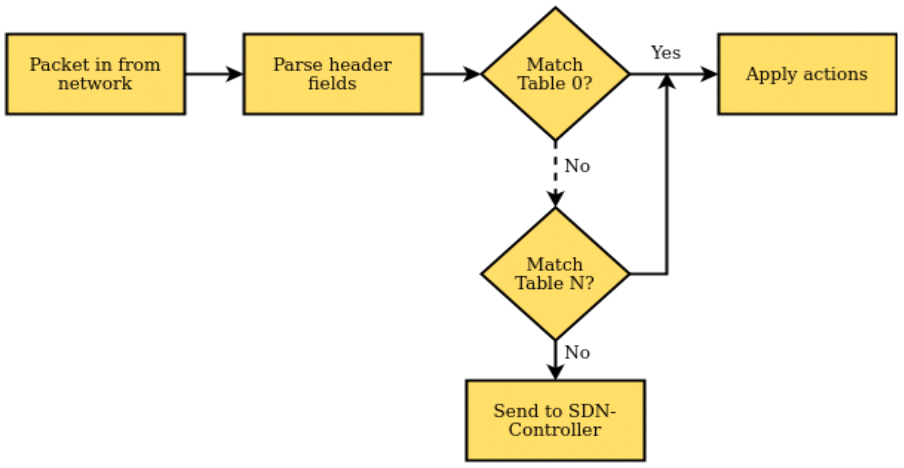


Figure 9: OpenFlow Matching Procedure

As depicted in Figure 9, when an incoming packet is matched against a table’s flow entry, a set of actions are implemented, found in the instructions field of the matched flow entry. If no matched entry is found in a flow table, the outcome depends on the configuration of the table-miss flow entry. Most of the time, the packet is sent to the connected SDN-Controller for further inspection.

Actions can either describe packet forwarding to a physical or logical port, packet modification and group table processing. Actions are split into required and optional.

Required actions are necessary for all OpenFlow-hybrid switches, while OpenFlow-hybrid switches can use the later type. Some of the basic actions are:

- Required Action: Forward. An OpenFlow switch must support forwarding packets to physical ports as well as the following virtual ports:
    - ALL: Send the packet out all interfaces, not including the incoming interface.
    - CONTROLLER: Encapsulate and send the packet to the controller.
    - LOCAL: Send the packet to the switch's local networking stack.
    - TABLE: Perform actions in flow table. Only for packet-out messages.
    - IN PORT: Send the packet out the input port.
- Optional Action: Forward. The switch optionally supports the below physical ports:
    - NORMAL: Process the packet using the traditional forwarding path supported by the switch (i.e., traditional L2, VLAN, and L3 processing).
    - FLOOD: Flood the packet along the minimum spanning tree, not including the incoming interface.
- Required Action: Drop. A flow-entry with no specified action indicates that all matching packets should be dropped.

Every OpenFlow-compliant Logical switch is connected to a SDN-Controller via an OpenFlow Channel. The OpenFlow Channel operates over the Transport Connection Protocol (TCP) or the Transport Layer Security (TLS). The SDN-Controller installs and maintains the switch using this link handles events and transmits data to the connected switch. Multiple channels can be established with many SDN-Controllers sharing a switch's management. SDN-Controllers can add, update, and delete flow entries in flow tables, both reactively, in response to packets, and proactively. Specifically, there are three message types supported by the OpenFlow protocol, controller-to-switch, asynchronous and symmetric.

The controller-to-switch messages are initiated by the SDN-Controller and used to directly manage or inspect the state of the switch. Some of the messages are:

- Features: Request for the identity and basic capabilities of a switch; the switch must respond with a features reply specifying the identity and basic capabilities of the switch. This is commonly performed upon establishment of the OpenFlow channel.
- Configuration: The controller is able to set and query configuration parameters in the switch. The switch only responds to a query from the controller.
- Modify-State: State management messages. Their primary purpose is to add, delete and modify flow/group entries and insert/remove action buckets of group in the OpenFlow tables and to set switch port properties.

- Read-State: Information collection from the switch, such as current configuration, statistics, and capabilities. Most requests and replies are implemented using multipart message sequences.
- Packet-out: Used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages. Packet-out messages must contain a full packet or a buffer ID referencing a packet stored in the switch. The message must also contain a list of actions to be applied in the order they are specified; an empty list of actions drops the packet.
- Barrier: Used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.
- Role-Request: Used by the controller to set the role of its OpenFlow channel, its Controller ID, or query these. This is mostly useful when the switch connects to multiple controllers

Asynchronous messages are initiated by the switch for notifying the SDN-Controller about network events and changes to a switch's state. Asynchronous messages supported by a switch are:

- Packet-in: Transfer the control of a packet to the controller. For all packets forwarded to the CONTROLLER reserved port using a flow entry or the table-miss flow entry, a packet-in event is always sent to controllers. Other processing, such as Time-To-Live (TTL) checking, may also generate packet-in events to send packets to the controller.
- Flow-Removed: Inform the controller about the removal of a flow entry from a flow table. Flow-Removed messages are only sent for flow entries with the OFPFF_SEND_FLOW_REM flag set. They are generated as the result of a controller flow delete request, the switch flow expiry process when one of the flow timeouts is exceeded, or other reasons.
- Port-status: Inform the controller of a change on a port. The switch is expected to send port-status messages to controllers as port configuration or port state changes. These events include change in port configuration events, for example if it was brought down directly by a user, and port state change events, for example if the link went down.
- Role-status: Inform the controller of a change of its role. When a new controller elects itself master, the switch is expected to send role-status messages to the former master controller
- Controller-Status: Inform the controller when the status of an OpenFlow channel changes. The switch sends these messages to all controllers when the status of the OpenFlow channel of any switch's changes. This can assist failover processing if controllers lose the ability to communicate among themselves.
- Flow-monitor: Inform the controller of a change in a flow table. A controller may define a set of monitors to track changes in flow tables

Lastly, symmetric messages are initiated from either entityor sent without solicitation.

- Hello: Messages exchanged between the switch and controller upon connection startup.
- Echo: Messages that can be sent from either the switch or the controller and must return an echo reply. Mainly used to verify the liveness of a controller-switch connection and may as well be used to measure its latency or bandwidth.
- Error: Used by the switch or the controller to notify problems to the other side of the connection. Mostly used by the switch to indicate a failure of a request initiated by the controller.
- Experimenter: Experimenter messages provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

2.2 Open vSwitch

Open vSwitch is a multilayer software switch licensed under the open-source Apache 2 license [18]. It can implement a production quality switch platform that supports standard management interfaces and opens the forwarding functions to programmatic extension and control.

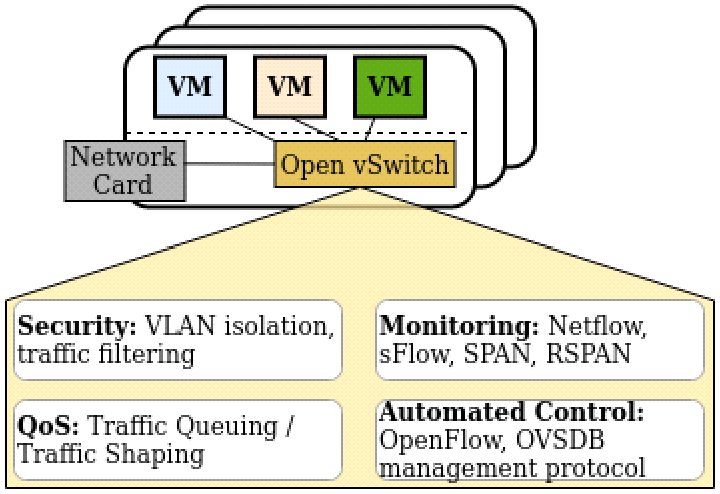

VM
VM
VM
Network Card
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: Traffic Queuing / Traffic Shaping
Automated Control: OpenFlow, OVSDB management protocol

Figure 10: Open vSwitch basic Architecture

Open vSwitch is well suited to function as a virtual switch in Virtual Machine (VM) environments. In addition to exposing standard control and visibility interfaces to the virtual networking layer, it was designed to support distribution across multiple physical servers. Open vSwitch supports multiple Linux-based virtualization technologies including Xen/XenServer, KVM, and VirtualBox.

The bulk of the code is written in platform-independent C and is easily ported to other environments. The current release of Open vSwitch supports the following features:

- Standard 802.1Q VLAN model with trunk and access ports
- NIC bonding with or without LACP on upstream switch

- NetFlow, sFlow(R), and mirroring for increased visibility
- QoS configuration, plus policing
- Geneve, GRE, VXLAN, STT, and LISP tunneling
- 802.1ag connectivity fault management
- OpenFlow 1.0 plus numerous extensions
- Transactional configuration database with C and Python bindings
- High-performance forwarding using a Linux kernel module

The included Linux kernel module supports Linux 3.10 and up.

Open vSwitch can also operate entirely in userspace without assistance from a kernel module. This userspace implementation should be easier to port than the kernel-based switch. OVS in userspace can access Linux or DPDK devices. Note Open vSwitch with userspacedatapath and non DPDK devices is considered experimental and comes with a cost in performance.

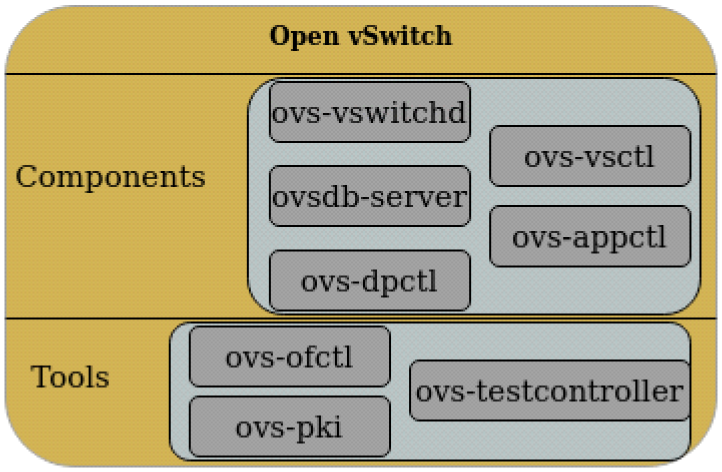


Figure 11: Open vSwitch Basic Components and Tools offered

The main components and provided tools of this distribution, as shown in Figure 11 are:

Components:

- ovs-vswitchd: a daemon program implementing the main functionality of the switch, along with a companion Linux kernel module for flow-based switching.
- ovsdb-server: a lightweight database server that ovs-vswitchd queries to obtain its configuration.
- ovs-dpctl: a tool for configuring the switch kernel module.
- ovs-vsctl: a utility for querying and updating the configuration of ovs-vswitchd.

- ovs-appctl: a utility that sends commands to running Open vSwitch daemons.

Tools:

- ovs-ofctl: a utility for querying and controlling OpenFlow switches and controllers.
- ovs-pki: a utility for creating and managing the public-key infrastructure for OpenFlow switches.
- ovs-testcontroller: a simple OpenFlow controller that may be useful for testing.

MEC

In recent years, there have been major changes to the Telco world on mobile communications technologies. With highly capable end-devices, users have access to services such as video, music, social networking, gaming, and other interactive applications as well as emerging services such as augmented reality, resulting in a huge data traffic volume, straining the current network infrastructure. Additionally, IoT networks, Machine-Type-Communications will add a large number of devices that are less tolerant of time delays. With this wide range of new and diverse services it becomes an integral part of the mobile users' entertainment and social life, increasing the expectations towards immersive QoE as well. The related performance requirements include the support of 100 times higher data volumes, data rates equal to 10 Gb/s, vary low service level latency just under 5ms and mass connectivity of up to 300,000 devices within a single cell, with ultra-high reliability and reduced energy consumption by 90%.

The European Telecommunications Standards Institute (ETSI) has initiated Mobile Edge Computing (MEC) standardization to promote and accelerate the edge-cloud computing in mobile networks, by launching the MEC Industry Specification Group (ISG) in December 2014. Its objective is to create an open environment across multi-vendor cloud platforms located on the Radio Access Network (RAN). By transferring data intensive tasks towards the edge and locally processing data in proximity to the users, mobile network operators can reduce traffic bottlenecks in the core and backhaul networks and assist in the offload of heavy computational tasks from power constrained User Equipment (UE) to the edge. This can provide the potential for developing a plethora of new applications, bringing innovation and new business opportunities. By collecting contextual information and specific content, proximity and location awareness can offer a tailored mobile broadband experience.

2.3 O-RAN

As it has already been analyzed, the emergence of IoT networks, M2M communications, and data demanding mobile applications are straining the current Radio Access Network (RAN). The co-existence of such a diverse ecosystem of applications requires a more flexible network that satisfies all needed features. A possible solution is to design a separate network for each type of application. No matter how enticing this concept is, it is not feasible from operating expenses (OPEX) aspect. This has turned both the academia and industries on making the mobile network more software driven, flexible, virtualized, and intelligent and energy efficient while being reliable and cost-effective.

Given the above requirements, it would be feasible to split the RAN into multiple parts based on the functionality, making the architecture smarter and more versatile. This is known as Open RAN (O-RAN). O-RAN approaches the current infrastructure in a more software-oriented infrastructure, making it easier for the network to act according to the QoS requirements of each application.

The architecture of O-RAN is shown in Figure 12. Many of the functions from the traditional RAN architecture that were aggregated in a single node are now disaggregated. As a result, this action increases the reliability by avoiding single points of failure. Additionally, this allows the separation of control and user plane. Consequently, the function of the control plane can be implemented on all server platforms, while real time functions can be implemented on hardware level.

In a more in depth look at the architecture, trained models and real time control functions are included in the RAN Intelligent Controller near-Real Time (RIC near-RT) for run time execution. It utilizes the Radio Network Information database which tracks the state of the underlying network via E2 and A1 interfaces. The E2 interface interconnects the RIC near-RT and Control and Distributed Unit (CU/DU) that feeds data that include various RAN measurements for radio resources management tracked by Artificial Intelligence/Machine Learning (AI/ML). The latter interface is responsible for passing on the AI enable policy and ML based training models to the RIC non-RT. In turn, the RIC non-RT is affiliated with the Orchestration and Automation (OA) layer and controls functions for non-real time intelligence radio resource management as well as supporting operations of RIC near-RT functions.

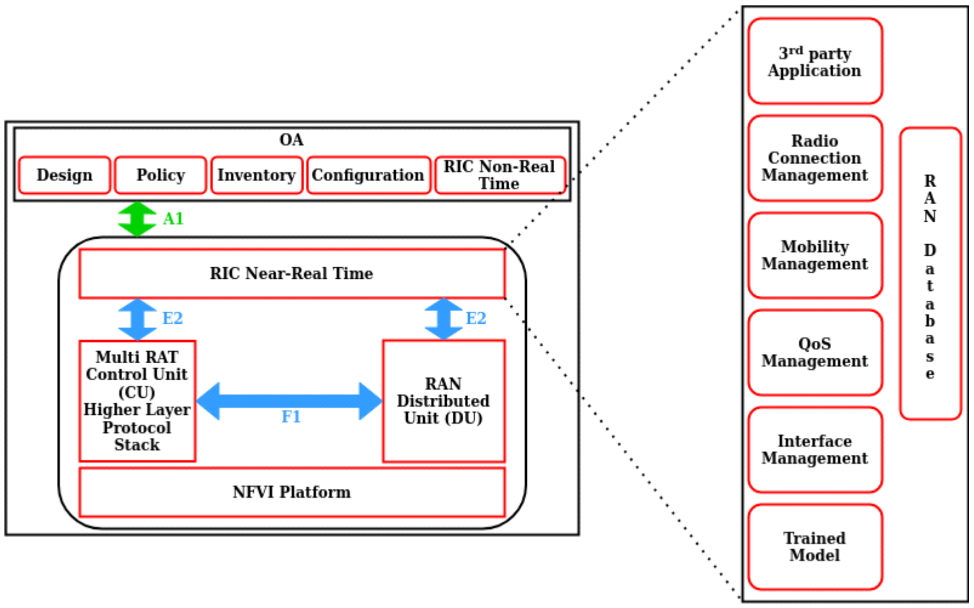


Figure 12: O-RAN Architecture

The adoption of O-RAN architecture can greatly influence the market, giving opportunities to public network operators to achieve their core network technology. It is estimated that the ease of use of O-RAN and modern learning methods may reduce the maintenance cost up to 80%.

2.4 802.11 standard

IEEE 802.11, commercially known as Wi-Fi, is part of the IEEE 802 set of local area network (LAN) technical standards and specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) computer communication. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand and are the world's most widely used wireless computer networking standards. IEEE 802.11 is used in most home and office networks to allow devices to communicate with each other and access the Internet without the necessity of wires. It uses various frequencies, some of them being 2.4 GHz, 5 GHz, 6 GHz, and 60 GHz frequency bands.

An important feature included in this protocol is the advertisement of information elements. An information element is transported in IEEE 802.11 management frames and a single frame contains multiple information elements. Those frames are sent between Wi-Fi access points and a mobile device, e.g., smartphones, tablets and any other device capable of connecting to the Wi-Fi. Information Elements (IE) [19] have a common general format consisting of a 1 octet Element ID field, a 1 octet Length field, an optional 1 octet Element ID Extension field, and a variable-length element-specific Information field. Each element is identified by the contents of the Element ID and, when present, Element ID Extension fields as defined in this standard. An Extended Element ID is a combination of an Element ID and an Element ID Extension for those elements that have a defined Element ID Extension. The Length field specifies the number of octets following the Length field.

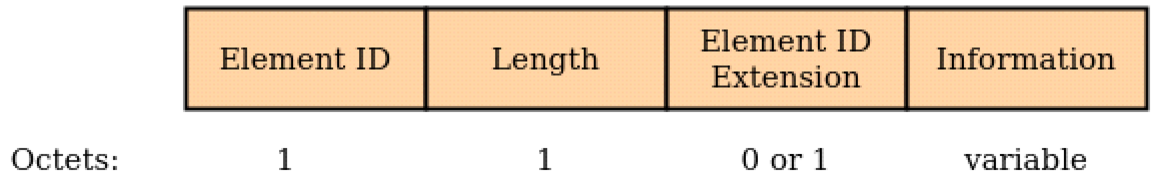


Figure 13: Information Element general format

2.5 Android

In the past decade, mobile devices, such as cellphones and tablets, have become an integral part of everyday life. They have made communicating, content consumption and automation of everyday tasks effortless. Even though there has been no shortage of manufacturers and operating systems (OS), since 2011, Android devices have been the best-selling OS worldwide, with over three billion monthly active users and as of January 2022 it holds 69.74% of the market share [20]. Its openness and ease of use is a driving factor for its massive adoption both in the development area and from normal users. Without any license fee, the source code can be used to create applications which, after following the rule and condition in the license’s terms and conditions, can be uploaded to the Google Store for others to download and use.

The OS running on every Android device is based on a modified version of the Linux kernel in conjunction with other open software, designed primarily for touchscreen mobile devices. Android is developed by a consortium of developers known as the Open Handset Alliance (OHA)[21], and commercially sponsored by Google. Its architecture is comprised of several components, ready to aid any need an Android device may have. The Android OS architecture is mainly categorized in the i) System Applications, ii) Java API Framework, iii) Native C/C++ Libraries, iv) Android Runtime, v) Hardware Abstraction Layer and vi) the Linux Kernel.

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. So, a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app). The system apps function both as apps for users and to provide key capabilities that developers can access from their own app.

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

- A rich and extensible View System you can use to build an app’s UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack
- Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

The Application Runtime environment contains core libraries and either the Dalvik Virtual Machine (DVM) for devices with Android 4.4 “KitKat" installed or the Android Runtime (ART) environment. For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the ART. ART is written to run multiple virtual machines on low-memory devices by executing Dalvik Executable (DEX) files, a bytecode format designed specifically for Android that's optimized for minimal memory footprint. Build tools, such as d8, compile Java sources into DEX bytecode, which can run on the Android platform.

Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- On Android 9 (API level 28) and higher, conversion of an app package's Dalvik Executable format (DEX) files to more compact machine code.
- An improved debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields.

On the same layer, many core Android system components and services, such as ART and Hardware Abstraction Layer (HAL), are built from native code that requires native libraries written in C and C++. They provide support for applications regarding audio, access to display subsystem, graphics (OpenGL ES), database access, functionality on displaying web content (Mediakit) as well as security for establishing safe sessions across the web.

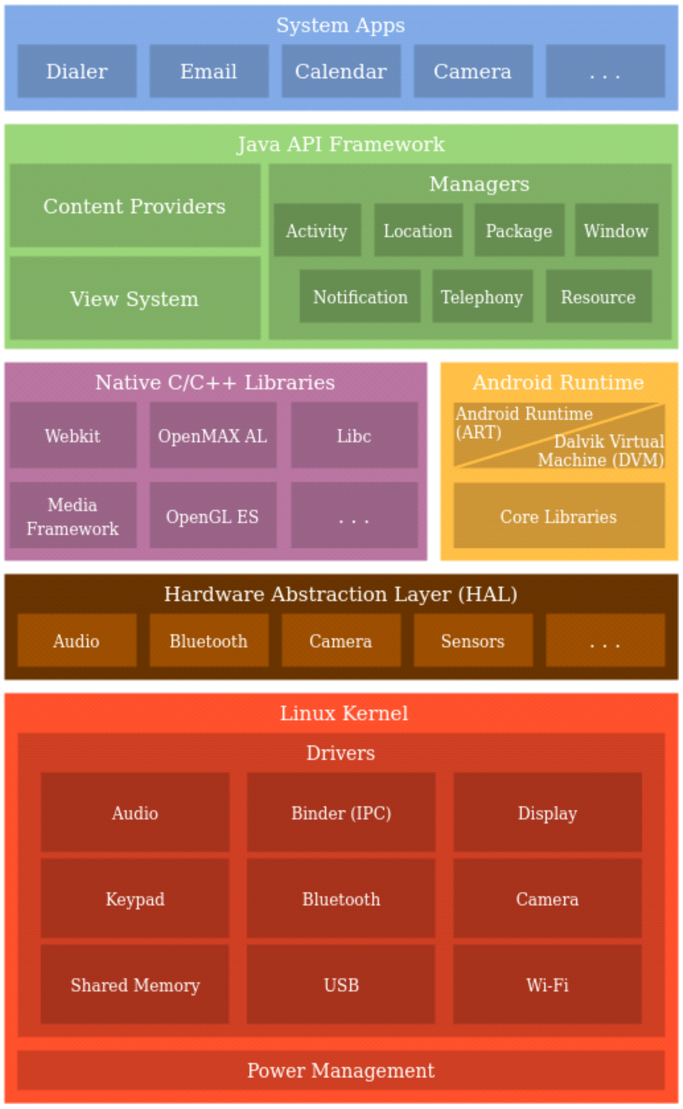


Figure 14: Android OS Architecture

The HAL provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. It consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

The foundation of the Android platform is the Linux kernel. It provides an abstraction layer between the device hardware and the aforementioned layers. Its features include memory management, allocation and management of processes, handling of the network stack as well as ensuring the proper function of applications according to the installed drivers.

2.6 Blockchain

The rise of cryptocurrencies over the last years, specifically Bitcoin (BTC) [22] and Ethereum (ETH) [23], have solidified blockchain-based systems and cryptocurrency technologies, opening up new possibilities towards the standardization of secure distributed consensus networks. The transactions that take place on a cryptocurrency network are all based on a public digital ledger implemented in a distributed manner (without the presence of a central repository) and usually without a central authority (i.e. a bank, company, or government). This ledger is called blockchain. It has enabled the tracking of said transactions by a network of non-trusted peers, while eliminating the double-spending of digital assets by enforcing distributed consensus on the sequence of legitimate blocks of the blockchain in the long run.

Blocks record transactions of digital coins across end users containing metadata that are necessary for sustaining, extending, and safeguarding the distributed ledger structure. As it is shown in Figure 15a, a block consists of its block header and a data block. The block header contains a hash value from the previous block header, a timestamp, the block number, also known as blocks height in some networks, a hash representation of the block data, e.g. generated by a Merkle tree, or by utilizing a hash of all combined block data and a nonce value. In blockchain networks utilizing mining, the previous value is a number used to solve a hash puzzle. Other networks see it as an optional field or giving it another purpose other than solving a hash puzzle.

The block data contains a list of transactions and ledger events as well as other data. Transactions, which summarize the transfer of digital coins (or balance updates) across the end users, are disseminated across the consensus network nodes to enable them to fill up and seal new blocks. In more recent blockchain systems, a coin transfer, known as a call, to a blockchain user can be initiated by a smart contract (SC). SCs are self-executing scripts that reside on the blockchain. They consist of publicly verifiable code that enables general-purpose computations to occur in a deterministic on-chain fashion.

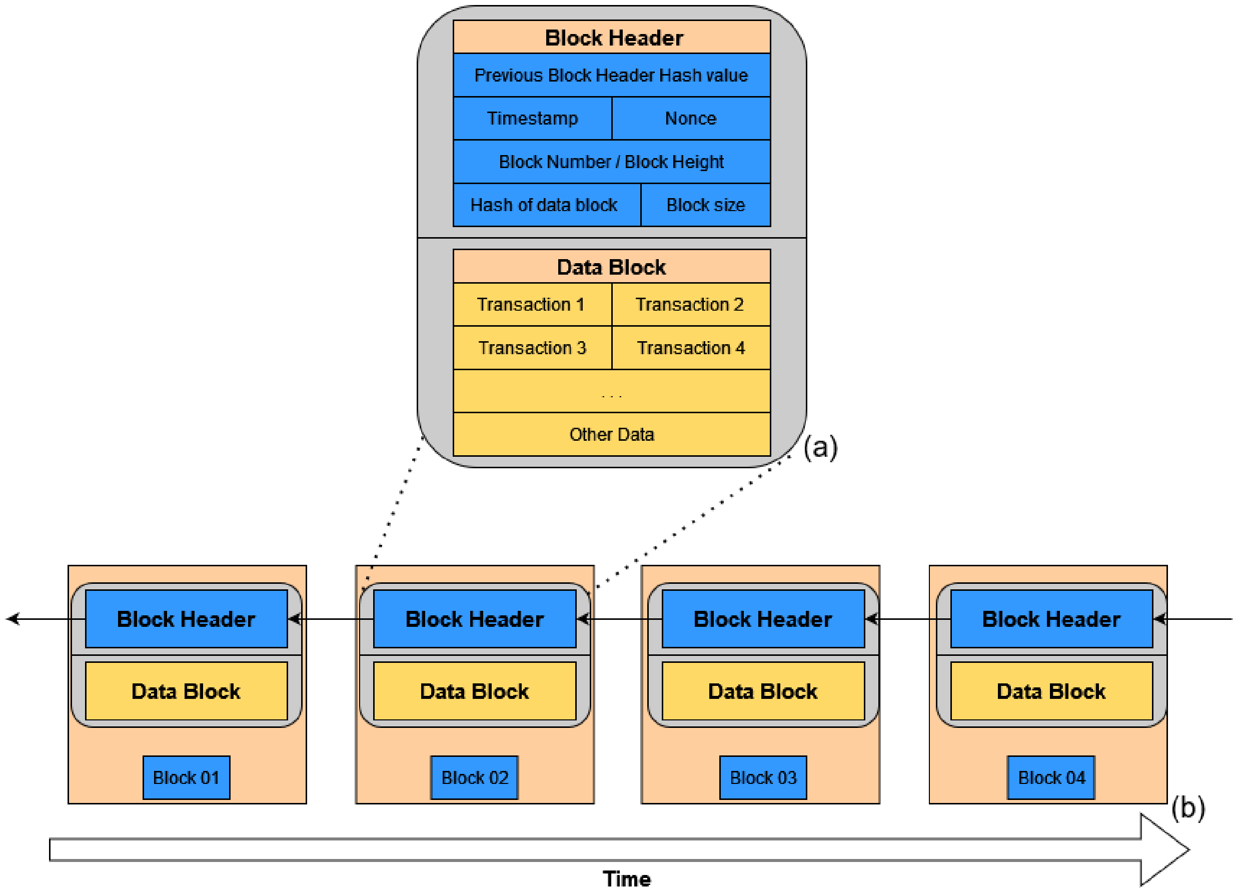


Figure 15: Block (a) and Blockchain Architecture (b)

A consensus protocol is the core component of the crypto-currency system. It ensures a unanimous agreement between all the participating nodes on a common transaction history that is serialized and crystallized due time into sequential blocks, forming the blockchain, as shown in Figure 15b. Well established consensus protocols, named Proof-of-Work (PoW) protocols, are based on the participation of consensus nodes in a solution process. Each node must identify a hash function output, termed nonce, that uses a hash value of previous blocks and the payload of the current block. The structure of the hash function makes the identification of the nonce difficult, as it can only be done by using brute-force. This comes with substantially large amounts of energy, long-term security concerns due to the decreasing mining rewards embedded in the consensus emission policy as well as low transaction throughput.

For the above reasons, the blockchain community has put effort in the replacement of PoW consensus protocol. New protocols have emerged based on Proof-of-Useful-Work [24], Proof-of-Space [25], Proof-of-Storage [26], Proof-of-Elapsed-Time [27], Proof-of-Stake (PoS) [28] and Proof-of-Authority (PoA)[29]. From the previously mentioned protocols, PoS and PoA have stood out due to their low operational costs regarding their energy and processing efficiency. In PoS consensus, block sealers are selected according to their stakes in the crypto-currency platform. For the latter consensus, the generation and appending of new blocks is set to a small set of nodes, called validators, providing proof of their identity publicly. Although PoA consensus is available in the ETH, integration of PoS is still undergoing.

2.7 Linux related programs

2.7.1 hostapd

The HOST Access Point Daemon (hostapd) [30]is a user space daemon for access point and authentication servers that runs in the background and acts as the backend component controlling authentication. The IEEE 802.11 access point management, IEEE 802.1X/WPA/WPA2/EAP Authenticators, RADIUS client, EAP server and RADIUS authentication server can all implemented by it.

For the access point configuration there are 2 different available schemas. A routed access point can be created within an Ethernet network and can be used as a wireless access point, creating a secondary standalone network, as shown in Figure 17. The resulting new wireless network is entirely managed by the wireless interface of the device. The second available configuration is a bridged access point. In this a device can be used as a bridged wireless access point within an existing Ethernet network, as shown in Figure 18. This will extend the network to wireless computers and devices.

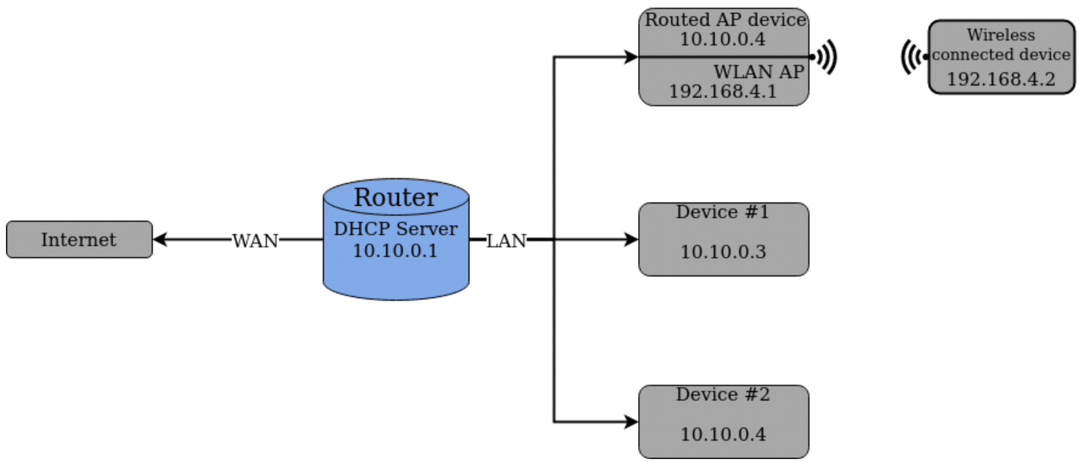


Figure 16: Routed AP General Configuration

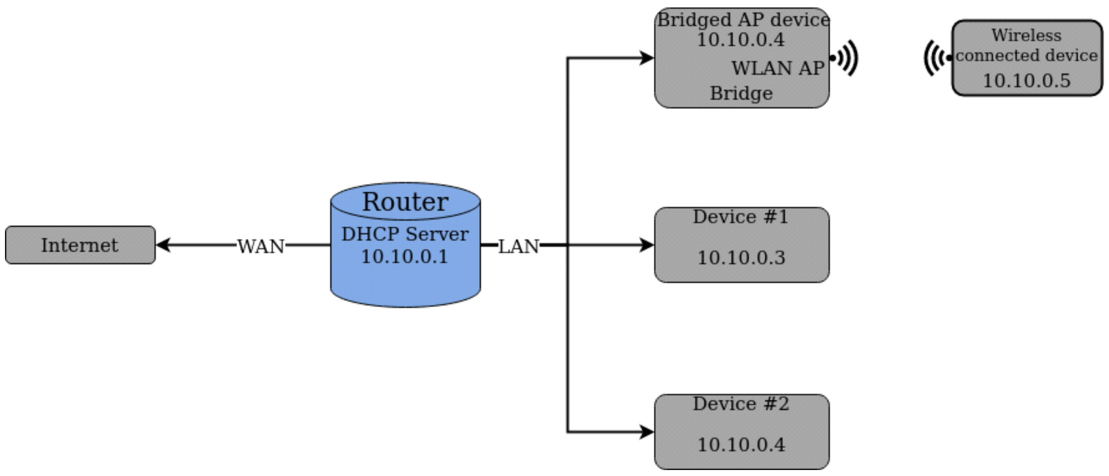


Figure 17: Bridged AP general configuration

2.7.2 dnsmasq

Designed to be lightweight and have a small footprint, suitable for resource constrained routers and firewalls, it provides network infrastructure for small networks: DNS, DHCP, router advertisement and network boot [31]. It has also been widely used for tethering on smartphones and portable hotspots, and to support virtual networking in virtualization frameworks. Supported platforms include Linux (with glibc and uclibc), Android, *BSD, and Mac OS X. It is included in most Linux distributions and the ports systems of FreeBSD, OpenBSD and NetBSD. Dnsmasq provides full IPv6 support.

The DNS subsystem provides a local DNS server for the network, with forwarding of all query types to upstream recursive DNS servers and caching of common record types (A, AAAA, CNAME and PTR, also DNSKEY and DS when DNSSEC is enabled). The DHCP subsystem supports DHCPv4, DHCPv6, BOOTP and PXE. The Router Advertisement subsystem provides basic autoconfiguration for IPv6 hosts. It can also be used stand-alone or in conjunction with DHCPv6.

2.7.3 dhcpcd

An implementation of the DHCP client specified in RFC 2131 [32]. It gets the host information (IP address, routes, etc) from a DHCP server and configures the network interface of the machine on which it is running. It daemonises and waits for the lease renewal time to lapse. It will then attempt to renew its lease and reconfigure if the new lease changes when the lease begins to expire or the DHCP server sends a message to renew early. The dhcpcd program provides functionalities of the programs BOOTP (RFC 951 [33]), IPv6 Router Solicitor (RFC 4861 [34], RFC 6106 [35]), IPv6 Privacy Extensions to Autoconf (RFC 4941 [36]), DHCPv6 client (RFC 3315 [37]).

2.7.4 FFmpeg

FFmpeg is a free and open-source software project consisting of a suite of libraries and programs for handling video, audio, and other multimedia files and streams[38]. At its core is the command-line ffmpeg tool itself, designed for processing of video and audio files. It is widely used for format transcoding, basic editing (trimming and concatenation), video scaling, video post-production effects and standards compliance such as the Society of Motion Picture and Television Engineers (SMPTE) and International Telecommunication Union (ITU).

FFmpeg also includes other tools: ffplay, a simple media player and ffprobe, a command-line tool to display media information. Among included libraries are libavcodec, an audio/video codec library used by many commercial and free software products, libavformat (Lavf), an audio/video container mux and demux library, and libavfilter, a library for enhancing and editing filters through a Gstreamer-like filtergraph.

The transcoding process in ffmpeg for each output can be described by the following diagram.

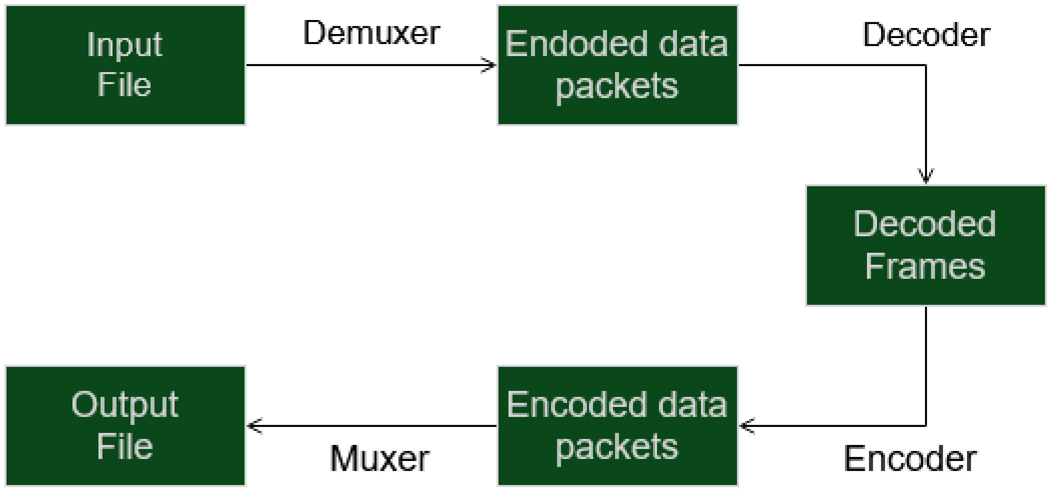


Figure 18: FFmpeg transcoding process

ffmpeg calls the libavformat library (containing demuxers) to read input files and get packets containing encoded data from them. When there are multiple input files, ffmpeg tries to keep them synchronized by tracking lowest timestamp on any active input stream.

Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering.

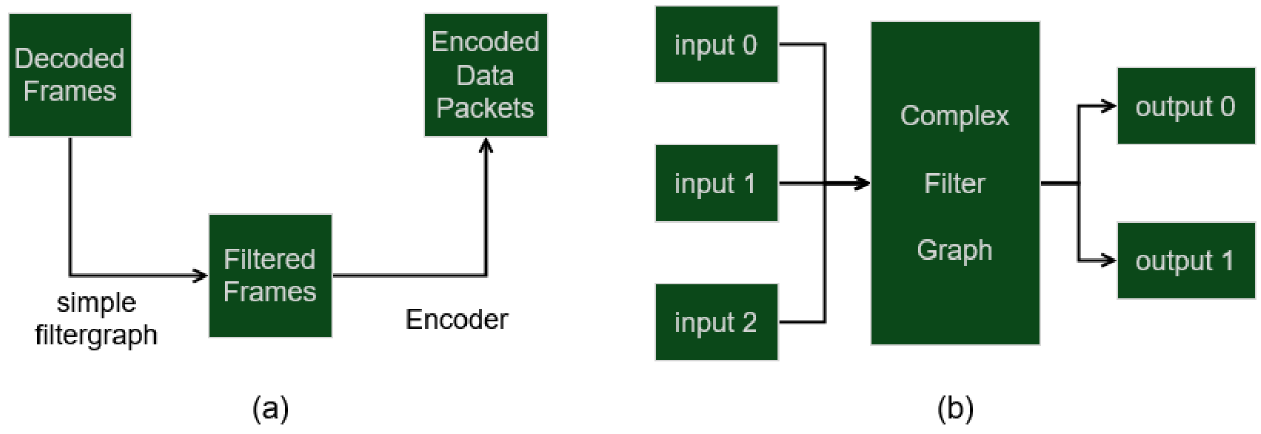


Figure 19: FFmpeg simple (a) and complex (b) filtergraphs

After filtering, the frames are passed to the encoder, which encodes them and outputs encoded packets. Finally, those are passed to the muxer, which writes the encoded packets to the output file.

FFmpeg is part of the workflow of many other software projects, and its libraries are a core part of software media players such as VLC and have been included in core processing for YouTube and Bilibili. Encoders and decoders for many audio and video file formats are included, making it highly useful for the transcoding of common and uncommon media files.

FFmpeg is published under the LGPL-2.1-or-later or GPL-2.0-or-later, depending on which options are enabled.

2.7.5 Nginx

Nginx is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, originally written by Igor Sysoev. For a long time, it has been running on many heavily loaded Russian sites including Yandex, Mail.Ru, VK, and Rambler. It was also used by: Dropbox, Netflix, Wordpress.com, and FastMail.FM.

The sources and documentation are distributed under the 2-clause BSD-like license.

Basic HTTP server features:

- Serving static and index files, autoindexing,open file descriptor cache
- Accelerated reverse proxying with caching,load balancing and fault tolerance
- Accelerated support with caching of FastCGI, uwsgi, SCGI, and memcached servers,load balancing and fault tolerance
- Modular architecture. Filters include gzipping, byte ranges, chunked responses, XSLT, SSI, and image transformation filter. Multiple SSI inclusions within a single

page can be processed in parallel if they are handled by proxied or FastCGI/uwsgi/SCGI servers

- SSL and TLS SNI support
- Support for HTTP/2 with weighted and dependency-based prioritization

TCP/UDP proxy server features:

- Generic proxying of TCP and UDP
- SSL and TLS SNI support for TCP
- Load balancing and fault tolerance
- Access control based on client address
- Executing different functions depending on the client address
- Limiting the number of simultaneous connections coming from one address
- Access log formats, buffered log writing, fast log rotation, and syslog logging
- IP-based geolocation
- A/B testing
- njs scripting language

Architecture

Nginx uses multiplexing and event notifications heavily and dedicates specific tasks to separate processes. Connections are processed in a highly efficient run-loop in a limited number of single-threaded processes called workers. Within each worker Nginx can handle many thousands of concurrent connections and requests per second.

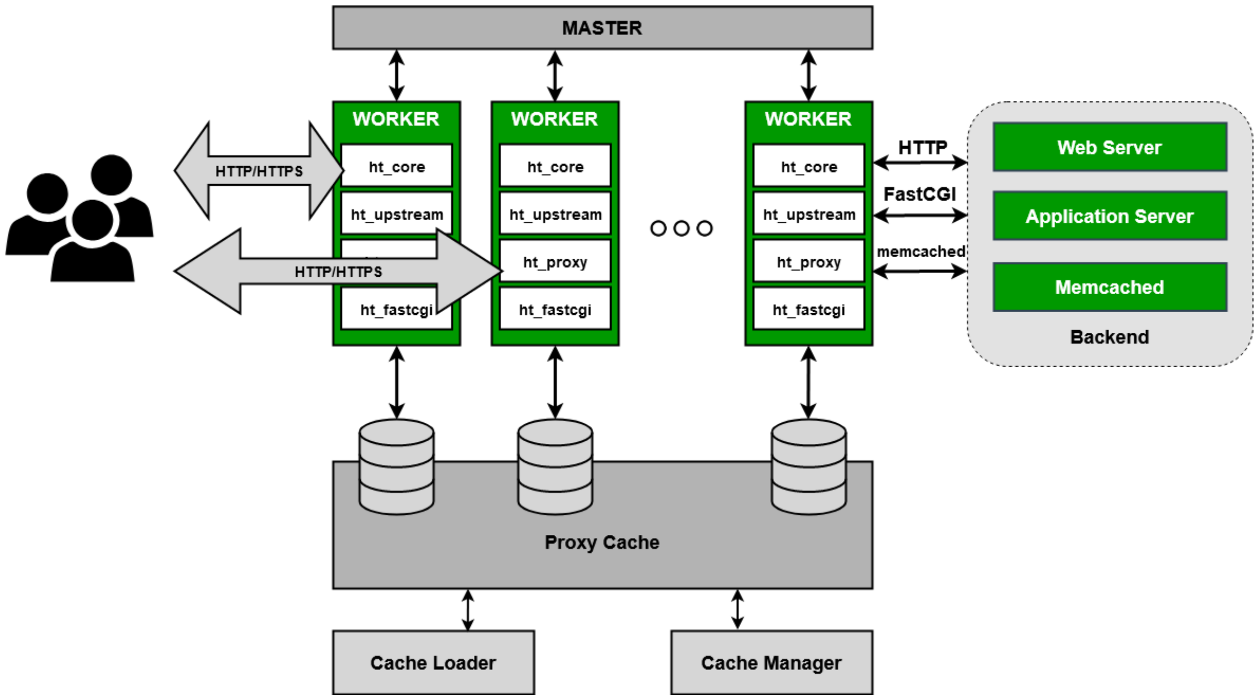


Figure 20: NGINX Architecture

There's no specialized arbitration or distribution of connections to the workers in nginx; this work is done by the OS kernel mechanisms. Upon startup, an initial set of listening sockets is created. Workers then continuously accept, read from, and write to the sockets while processing HTTP requests and responses.

The run-loop includes comprehensive inner calls and relies heavily on the idea of asynchronous task handling. Asynchronous operations are implemented through modularity, event notifications, extensive use of callback functions and fine-tuned timers, so to be as non-blocking as possible. The only situation where nginx can still block is when there's not enough disk storage performance for a worker process.

nginx conserves CPU cycles as well because there's no ongoing create-destroy pattern for processes or threads. What nginx does is check the state of the network and storage, initialize new connections, add them to the run-loop, and process asynchronously until completion, at which point the connection is deallocated and removed from the run-loop. Combined with the careful use of syscalls and an accurate implementation of supporting interfaces like pool and slab memory allocators, nginx typically achieves moderate-to-low CPU usage even under extreme workloads.

nginx runs several processes in memory; there is a single master process and several worker processes. There are also a couple of special purpose processes, specifically a cache loader and cache manager. All processes primarily use shared-memory mechanisms for inter-process communication. The master process is run as the root user. The cache loader, cache manager and workers run as an unprivileged user.

The master process is responsible for the following tasks:

- reading and validating configuration
- creating, binding, and closing sockets
- starting, terminating, and maintaining the configured number of worker processes
- reconfiguring without service interruption
- controlling non-stop binary upgrades (starting new binary and rolling back if necessary)
- re-opening log files
- compiling embedded Perl scripts

The cache loader process is responsible for checking the on-disk cache items and populating nginx's in-memory database with cache metadata. Essentially, the cache loader prepares nginx instances to work with files already stored on disk in a specially allocated directory structure. It traverses the directories, checks cache content metadata, updates the relevant entries in shared memory and then exits when everything is clean and ready for use.

The cache manager is mostly responsible for cache expiration and invalidation. It stays in memory during normal nginx operation, and it is restarted by the master process in the case of failure.

2.7.6 MySQL

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access, and facilitates testing database integrity and creation of backups.

MySQL is free and open-source software under the terms of the GNU General Public License and is also available under a variety of proprietary licenses. It has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often, MySQL is used with other programs to implement applications that need relational database capability.It is a component of the LAMPweb applicationsoftware stack, which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python. It is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress. In addition, a number of popular websites, including Facebook, Flickr, MediaWiki,Twitter, and YouTube use MySQL.

2.7.7 Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require tools or libraries[39]. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, and upload handling, various open authentication technologies and several common framework related tools.

Applications that use the Flask framework include Pinterest and LinkedIn.

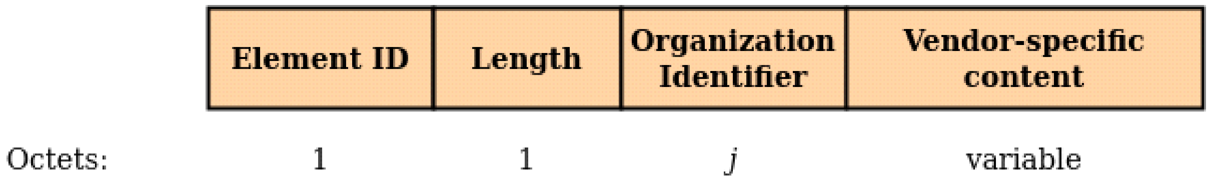
3 PROBLEM STATEMENT AND SOLUTION PROPOSITION

In this section, the proposed solution developed is analyzed from a more technical standpoint. The main components that take place in the protocol are demonstrated, emphasizing on their functional split. The phases of the proposed protocol are analyzed before technologies that were used in both software and hardware levels are reviewed along with their functionality.

I focused on developing a proof-of-concept implementation guided by the user-driven RE-CENT service method. In short, a mobile client, termed RE-CENT client, broadcasts its desired content to RE-CENT servers in its near vicinity. Following, RE-CENT servers able to provide the client's demand, reply with a targeted offer. After some parameters are set through negotiation, the user picks a server, and the content delivery is initiated.

This approach can be implemented with an application developed for devices a mobile client can use to gain access to this protocol, e.g., tablets, smartphones. By the RE-CENT protocol standards, RE-CENT clients, i.e., mobile users, can access the wireless medium by broadcasting and listening for offers for various wireless protocols (e.g., Wi-Fi, broadband cellular networks). Applications for mobile device have limited access to such functionality and are not yet on par with the functionality portrayed above and on the introductory section.

For the proposed implementation, we used the Wi-Fi technology, as it is the most accessible via an ample selection of devices that support it. A problem weencountered from the beginning was the way mobile users can distinguish an AP. The SSID and information elements are the main ways for mobile devices to distinguish APs apart from one another. The information elements in particular, detail over a broad spectrum of capabilities an AP has configured. Information about the SSID selected for the AP, the country code the AP is set to operate, power constraints the equipment may have, as well as reserved values for future use. The most configurable element is the vendor specific element. It can carry information in a formatted way, so reserved element IDs can keeptheir standard purpose. In turn, custom information is set in a beacon frame, being a code word or a URL, notifying the user for the uniqueness of the AP. The element's format is shown inFigure 21.



	Element ID	Length	Organization Identifier	Vendor-specific content
Octets:	1	1	*j*	variable

Figure 21: Vendor Specific element definition

In short, a vendor specific element is comprised of an Element ID, always set to 221, the length of the rest of the entry, an Organization Identifier as well as the desired content. The Organization Identifier contains a public unique identifier assigned by IEEE. Its minimum length (*j*) is the minimum number of octets required to contain the entire organizationally unique identifier.

3.1 Preview of the Solution

The protocol we developed can be broken down in three distinct phases, as it is depicted in Figure 222. Those are i) network discovery and pairing, ii) service negotiation and registration, iii) online service and monitoring.

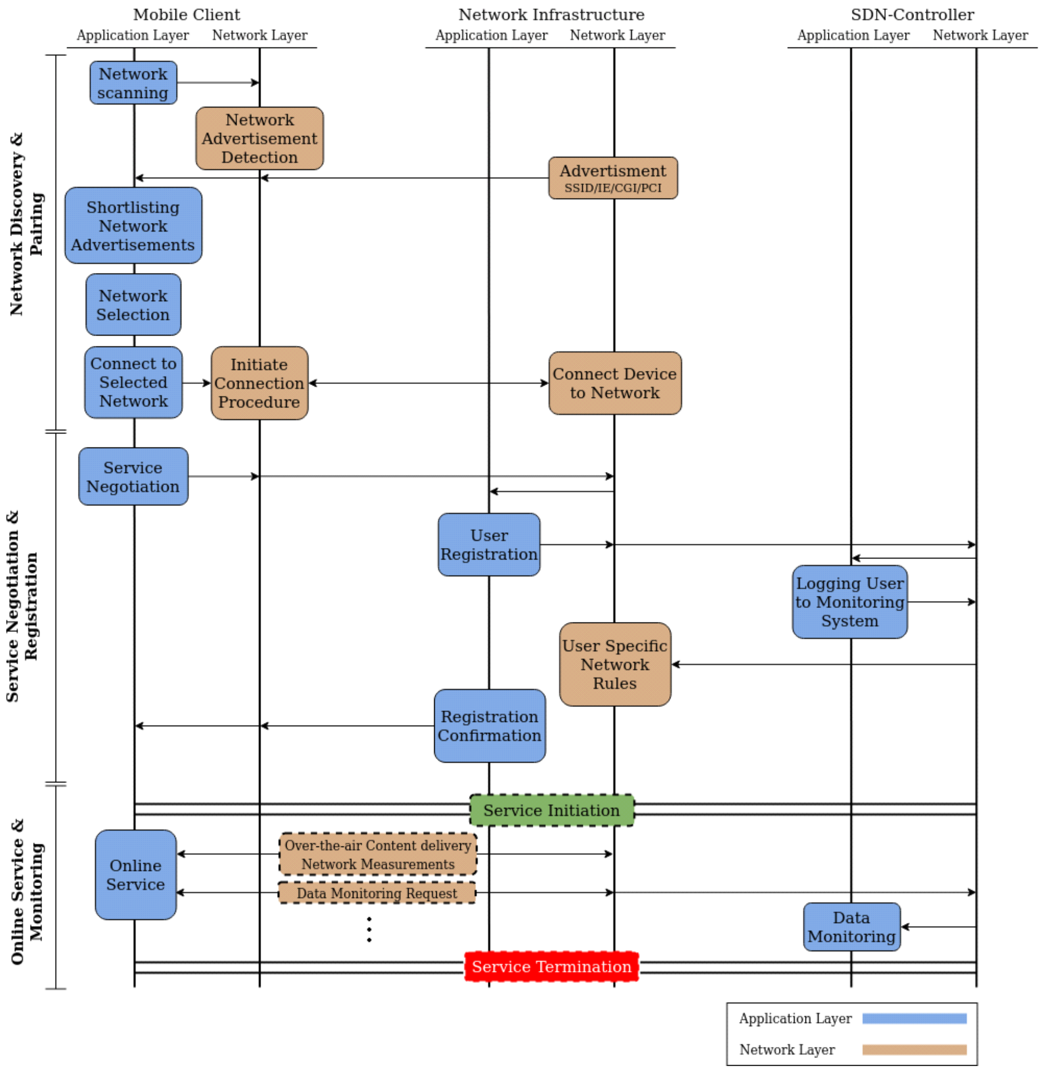

Mobile Client
Application Layer
Network Layer
Network Infrastructure
Application Layer
Network Layer
SDN-Controller
Application Layer
Network Layer
Network Discovery & Pairing
Network scanning
Network Advertisement Detection
Advertisment SSID/IE/CGI/PCI
Shortlisting Network Advertisements
Network Selection
Connect to Selected Network
Initiate Connection Procedure
Connect Device to Network
Service Negotiation & Registration
Service Negotiation
User Registration
Logging User to Monitoring System
User Specific Network Rules
Registration Confirmation
Online Service & Monitoring
Service Initiation
Online Service
Over-the-air Content delivery Network Measurements
Data Monitoring Request
Data Monitoring
Service Termination
Application Layer
Network Layer

Figure 22: Protocol Phases

3.2 Network Discovery and Pairing

The mobile client's device boots up and listens, on network level, for network advertisement messages emitted from network devices located in its vicinity, making their presence known to the device. Different wireless technologies provide their set of identifiers, broadcasted to mobile clients. For 802.11 based networks, the Specified Service Set Identifier (SSID) and information elements included in a beacon frame, while for cellular networks the Cellular Global Identifier (CGI) and the Physical Cell ID (PCI) are noteworthy examples. After the device has a set of information regarding nearby network devices, it shortlists them according to the criteria, and from the remaining ones, selects a network and initiates the appropriate connection procedure.

3.3 Service Negotiation and Registration

After gaining access to the selected network, the mobile client is requests access to the device’s broadband connection, by connecting to an onboard server operating on the device. The request is transferred in turn to an external network controller overseeing the network device. The user is logged to the system by the controller and user specific network rules are established to the device’s network layer. Following this outcome, a confirmation is sent back to the connected mobile client.

3.4 Online Service and Monitoring

The mobile client has successfully gained access to the system and is ready to start using network resources while it receives network measurements related to the chosen service. In parallel, the network controller issues requests to the network infrastructure the client is connected to regarding statistics collected from the established network rules, so its data usage can be monitored with the purpose of taking actions when necessary if it sees fit.

The main components present in the above protocol are i) the mobile client, ii) the network infrastructure and iii) the network controller. For a better understanding of each component, their building blocks are split into the application network layer.

The Mobile Client device, as shown in Figure 233, has functions on the application layer for registering the client to the system, QoE estimation functions, for shortlisting advertisements and selecting networks to initiate a connection with, as well as capabilities for video streaming and file transferring. On its network layer, network discovery for advertisement reception and network connection for initiating a connection procedure towards a specific network infrastructure, as well as content delivery to an application.

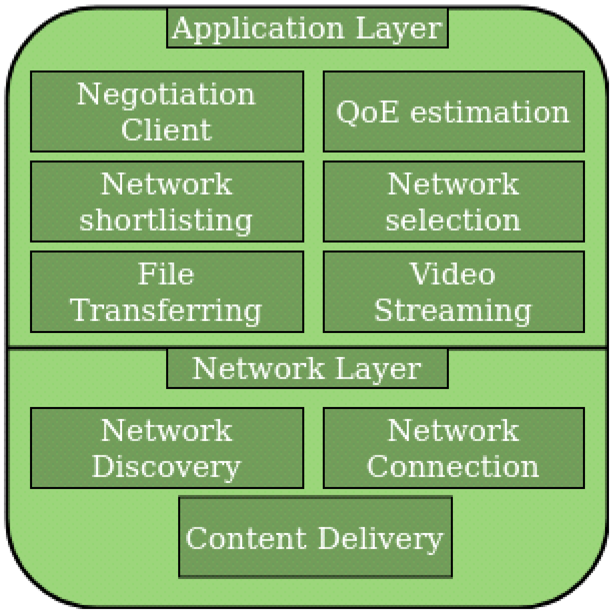


Figure 23: Mobile Client

The Network Infrastructure, as shown in Figure 244, its application layer is responsible for QoE Estimation of the connections established with mobile clients, the Negotiation Server responsible for registering the user to the network controller's monitoring system, and a function for providing information related to connection sessions initiated by connected mobile clients to the network controller.

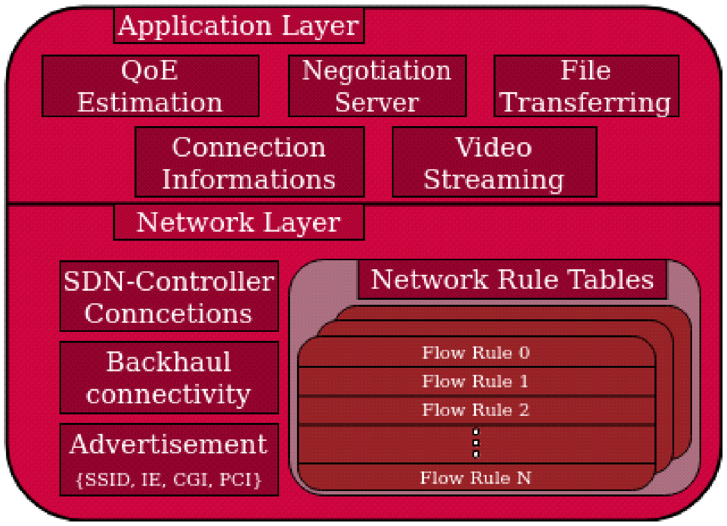


Figure 24: Network Infrastructure

On its network layer, it advertises several identifiers to nearby mobile clients, connects the device to the backhaul network to allow users to access the Internet and maintains connections to an external SDN-Controller.

Lastly, the network controller, as shown in Figure 255, on the application layer is responsible for registering mobile clients to the monitoring system, supervising their network resources usage and managing all associated network infrastructure, by responding to incoming events as well as issuing commands regarding their network routing capabilities. On the network layer, it maintains connections to all associated network infrastructure so the previously described application functionalities can be delivered to one or more associated network infrastructure.

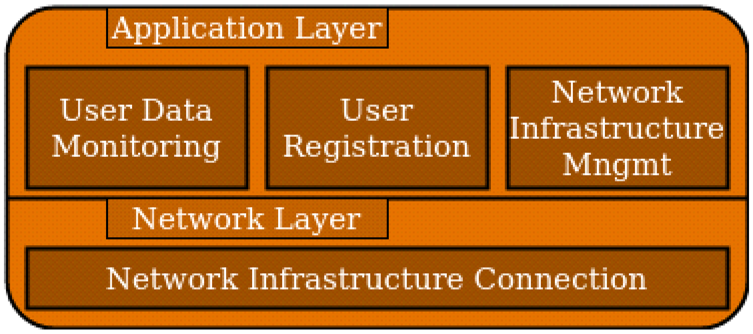


Figure 25: Network Controller

3.5 Detailed Architecture

For the implementation of the above protocol and the already described protocols, several technologies were utilized.

For the mobile client, an Android application was developed with the help of Java version 11. The application tackling the building blocks for both application and network layer has different approaches and so it was developed and tested for devices with Android OS 7.0, 8.0 and 11.0, and will be explained in the coming subsections.

For the network infrastructure a Raspberry Pi 3 B+ was used. It was configured as an AP with the use of an assortment of Linux programs, them being the hostapd, dnsmasq, dhcpcd and iptables, all described in the Key Technologies section. The AP running on the wireless interface (wlan0) was configured as a bridged AP, as shown in Figure 17. Furthermore, the Raspberry Pi device is configured as an OpenFlow hybrid switch, with the use of the Open vSwitch program to manage by the SDN-Controller entity.

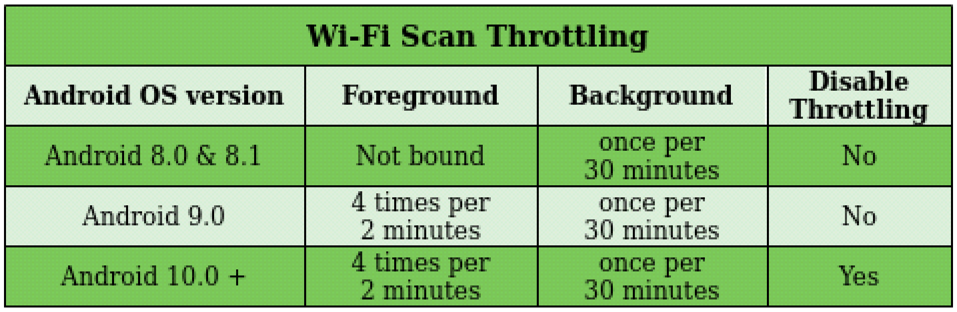
The network controller, in turn, was developed with the use of the RYU framework. For both AP configurations, bridged and routed, there is a different controller application that monitors the data usage of a connected mobile user.

With the aforementioned technologies for the implementation as well as the protocol phases, there are several problems encountered in i) Access Point Discovery, ii) Access Point Filtering & Shortlisting, iii) Access Point Connection.

3.6 Access Point Discovery

This phase, as shown in Table 1, requires the module Network Detection. For an Android application to detect APs in their near vicinity, they passively scan by listening on different frequency bands for beacons emitted from APs in a short time frame. Depending on the OS version installed on a device, initiating a passive Wi-Fi scan differs. Up until Android 8.0, developers had the ability to trigger passive Wi-Fi scanning by utilizing an API enabling this functionality [40]. With the release of Android 9.0 and higher, Google restricted the use of said API, effectively deprecating its use. Instead, OS initiated passive Wi-Fi scanning was introduced after fixed timed intervals. Additionally, due to battery consumption concerns on applications using network functions, Google throttled the frequency of Wi-Fi scanning from Android 8.0 up until Android 10 and higher (Table 1).

Table 1: Wi-Fi Scan Throttling



Wi-Fi Scan Throttling			
Android OS version	**Foreground**	**Background**	**Disable Throttling**
Android 8.0 & 8.1	Not bound	once per 30 minutes	No
Android 9.0	4 times per 2 minutes	once per 30 minutes	No
Android 10.0 +	4 times per 2 minutes	once per 30 minutes	Yes

It should be noted that for applications running Android 10.0 and higher, a setting for disabling Wi-Fi scan throttling has been introduced under the Developer options for local testing.

3.7 Access Point Filtering and Shortlisting

For this phase, as shown from Figure 23, the building blocks Network Shortlisting and Network Selection are both used from the application layer. After the passive scanning time period is over, the information is passed from the network layer to application layer in the form of a list, containing information about APs in the client's area, to the first for the two building blocks mentioned, Network Shortlisting. The information contained in the received list range from an AP's SSID, MAC address, supported 802.11 standards and information elements included in the beacon frame, to name a few. For Android devices with OS until version 10 (SDK version 29), the only way to distinct APs from one another was their SSIDs. From Android 11 and onwards, Google gave access to previously restricted Information Elements (IE) to developers, and in extent to applications. With them, applications have another means of identifying significant APs, as it is already described in Section 3. With the candidate APs filtered out with the aforementioned criteria, the remaining APs are shorted out according to their Received Signal Strength Identifier (RSSI) and the connection phase is initiated, under the functionality contained in the Network Selection.

3.8 Access Point Connection

Having selected an AP to connect to, the application must initiate the connection procedure. For this, the building blocks, illustrated on Figure 23, Network Selection, from the application layer, and Network Connection, from the network layer were utilized. Following the functionality described above for the Network Selection block, the approach with which this action is executed differs for different OSes installed in the device. Devices having OS version up until 9.0, applications have control over connecting [41], disconnecting [42] and reconnecting [43] to an AP via APIs provided by Google. Since the introduction of Android 10.0 OS, Google deprecated the aforementioned APIs, deeming them too intrusive to the device's system functionalities. Instead, developers, and in extensions the applications developed, issue a list of Wi-Fi network suggestions to the system to connect to, along with other parameters that affect the behavior of the API [44]. The system will then connect to one of the provided APs whenever it sees fit.

3.9 Execution Scenarios

In this subsection, a list of scenarios that can occur in both solutions is analyzed.

3.9.1 No Credentials

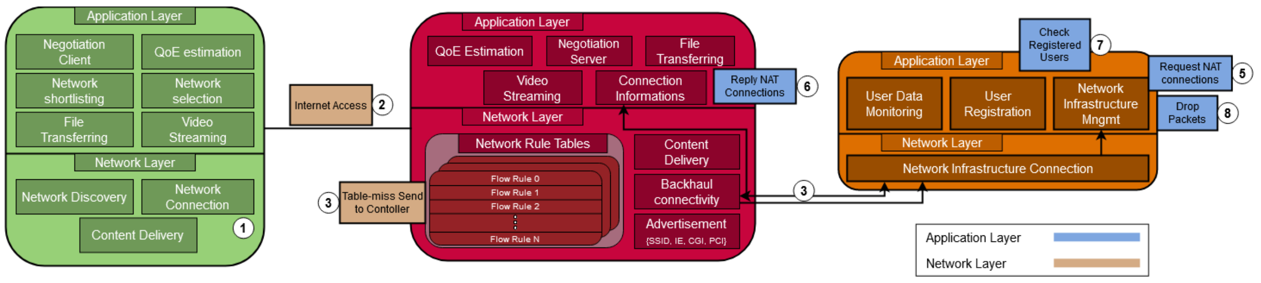


Figure 26: No credentials

- A mobile client connects to the AP without the use of the Android application.
- Tries to access the internet.
- No table entry was matched (table miss entry). The packet is sent to the controller for further inspection.
- The controller drops the packet.

3.9.2 Registering a user

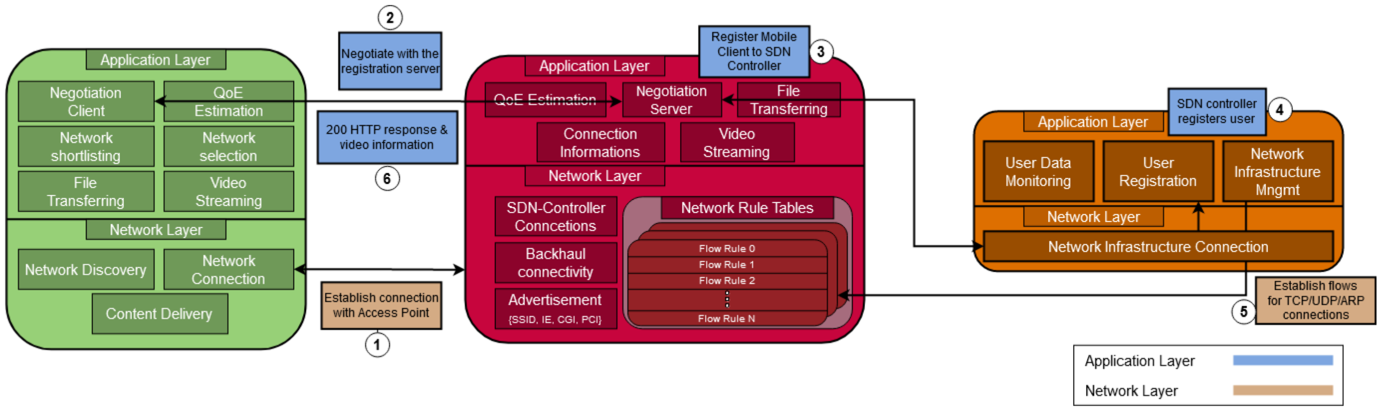


Figure 27: Registration Process

1. The mobile client connects to the AP via the Android application.
2. The Android application initiates a HTTP session towards the onboard server.
3. The Android application negotiates parameters related to the video streaming with the server.

4. The SDN-Controller receives the information about the newly connected user as well as the information for the video streaming server. It assigns a cookie value and makes a new entry to a dictionary containing the received information.
5. Flows are sent to Table 0 for packets for protocols TCP, UDP and ARP. Flows for TCP and UDP packets originating from a registered user (1a), the ingress port is the wlan0 NIC of the switch and the IP protocol number is set to 6 and 17 respectively. The same values apply for packets destined for a connected user, with the exception being the ingress physical port is the eth0 NIC.
6. Server responds with a 200 OK HTTP response, with all the necessary information for the video streaming server.

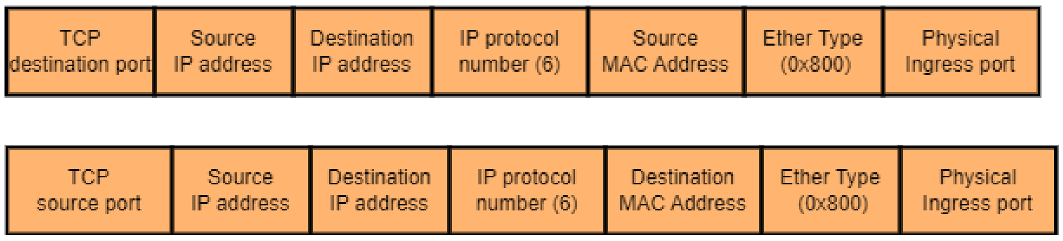


Figure 28: Flows Assigned per User for TCP, UDP (1a, 1b) and ARP (2a, 2b) Connections

3.9.3 Monitoring a user

The following Figure describes the procedure of data monitoring.

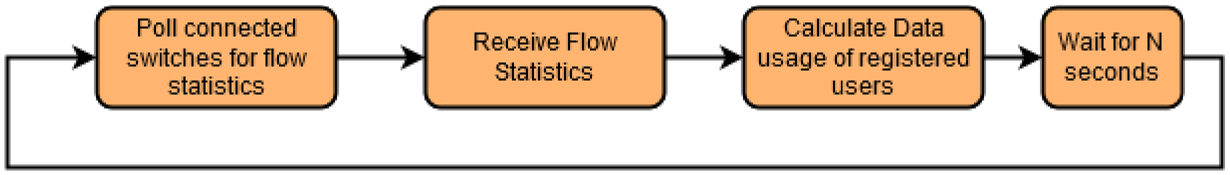

Poll connected switches for flow statistics
Receive Flow Statistics
Calculate Data usage of registered users
Wait for N seconds

Figure 29: Monitoring Users' data

- The SDN-Controller application requests flow statistics from each connected switch.
- An event from each switch is received.
- Controller calculates the data usage of each registered users, by utilizing the cookie value assigned to them.
- Wait N amount of seconds before re-polling the connected switches.

3.9.4 Un-registering a user

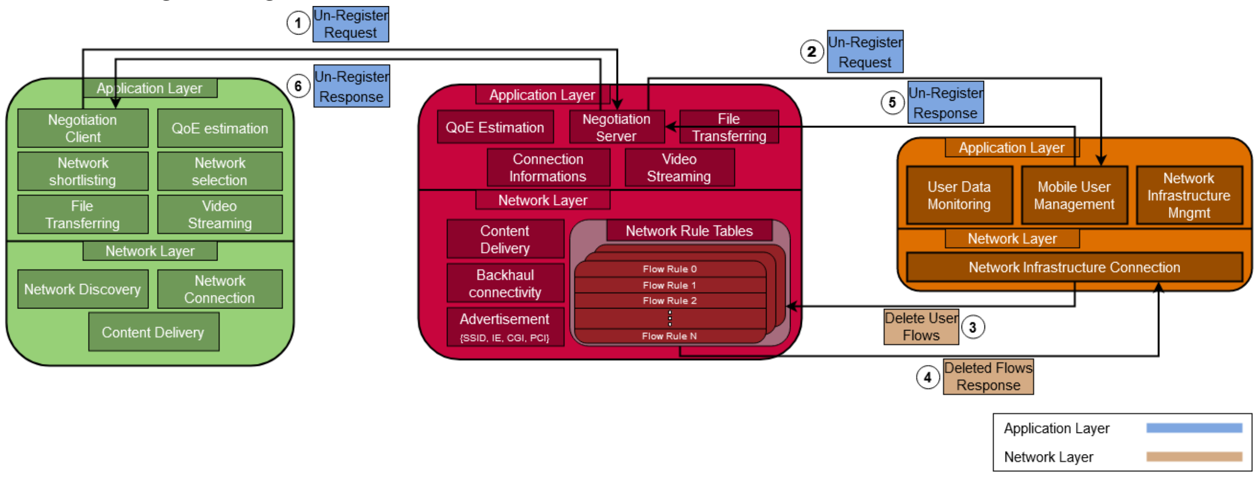


Figure 30: Un-Registering Procedure

1. The mobile user initiates an HTTP request to unregister from the monitoring system when the video reaches its end.
2. The negotiation server forwards it to the SDN controller.
3. Upon reception, the SDN controller request the deletion of the dedicated flows established for having access to the video streaming server.
4. The OpenFlow switch responds with messages about the successful deletion of the user flows.
5. The user management server responds to the negotiation server with a 200 OK HTTP response.
6. The negotiation server in turn responds to the user for its successful un-registration.

3.10 Problems and Possible Bugs

In this subsection, problems, and possible bugs that I have encountered during the proof-of-concept solution are outlined.

3.10.1 DNS and ARP packet tracking for mobile registered users

DNS packets originating from mobile users connected to an AP that runs on an OpenFlow switch are not sent on time and so the Android OS believes there is no connection in the connected Wi-Fi AP. For that, in each switch a pair of flows is installed upon connecting with the SDN controller, allowing requests and replies of DNS messages.

3.10.2 Routed Access Point configuration

From the beginning of the development of the proof-of-concept solution, there was a severe problemwe encountered. As described in section 2, subsection hostapd, there can be two different AP configurations. The routed AP, in short, creates a new network, with its own IP and uses the Linux kernel to map incoming and outgoing traffic to and from mobile users. This was found to be challenging when it came to the development of the SDN Controller application and its connection with the Open vSwitch software.

The problem that occurred was that there was no easy way on discerning the different mobile userswhen they were connected to the AP and having a video streamed to them via the local video streaming server. For this reason, the development of this solution could not be found, and the bridged AP configuration, as shown in Figure 17, was developed.

3.10.3 Android application on untested operating systems

The Android application is developed and tested for devices with operating systems ranging from 7.0 to 11.0, though it was tested for systems 7.0, 8.0, 9.0 and 11.0. The application can be unpredictable for devices with Android operating systemversion 10.0.

3.10.4 Android Wi-Fi connection

For devices with Android 10.0 OS and later, an application no longer can programmatically connect to a specific Wi-Fi AP. Instead, it suggests a Wi-Fi AP to the system to connect to. This can happen in an arbitrary point in time. In some cases, the connection to a suggested AP enters a loop. To avoid this, the user can turn off or on the Wi-Fi from the Wi-Fi settings menu. Another problem is the connection of a suggested AP. If the user connects to another AP and wants to switch to the suggested one, this action can only be done manually from the Wi-Fi settings menu ones again. In general, Google opted on making the Wi-Fi connect actions less automated in Android versions 10.0 and onwards.

3.10.5 Streaming ready videos

The video streaming server starts transmitting byte sections of a video to a connected mobile user. In my case, the video is an mp4 file type. This file type contains a header section containing information about the video a player needs before starting to play the content. This header can be in any part of the file. Any player that receives the video must have this information before starting to play the video. This is solved by moving this header to the front of the video.

3.10.6 Streaming server logging

When developing the video streaming, we tried logging the bytes that the server was sending to a mobile client. The bytes sent were not in accord with the bytes measured to the android application. I tried using synchronous server configuration, using the flask [39] library, asynchronous configurations, using the aiohttp [45], but the results are all the same.

3.10.7 Video streaming chunk sizes

In my test, I tried measuring the throughput of the video streaming server. I used different chunk sizes when reading the video file before sending it to a user. I found out that there was no difference in the throughput, both from the client’s and SDN controller’s perspective. Once again, I developed the server in both synchronous and asynchronous configurations, as well as using the NGINX HTTP server for enabling and disabling the HTTP server-side buffering, though no change found.

3.10.8 Hostapd service AP initialization

When booting the Raspberry Pi device, there is a chance that the hostapd service might not initialize the AP correctly. This problem is fixed by running the below command

```
sudo systemctl restart hostapd.service
```

4 IMPLEMENTATION

In this subsection, a brief explanation of the programs that were developed for this solution is displayed.

This subsection is segmented in the following paragraphs:

- Android application
- Python
- Bash programs

Each one outlines the application's classes, most of the times, these are separate files, and in turn, their functions are drilled down.

4.1 Android

Classes developed for this application:

- MainActivity
- Metrics
- WiFiReceiver
- WiFiScanResults
- WiFiConnection
- HTTPSession
- VideoPlayer
    - ExoVideoPlayer
    - ExoEventListener
    - ExoAnalyticsListener
    - ExoTransferListener
    - ThroughputHandlerThread
    - ThroughputHandler

MainActivity:

Class facilitating the main functionality of the application. It is responsible for managing the User Interface (UI) of the application, initializing variables related to networking actions, implementing filters for intercepting events originating either from other application classes or from the device's OS as well as requesting permissions from the user which are vital for the correct functionality of the application.

Table 2: MainActivity class

onCreate	Called when the activity is first created. Permissions necessary for the proper operability of the application are checked and requested from

	the user if missing, filters are created for intercepting events.
onResume	Called after your activity has been stopped, prior to it being started again. The views are set up to the device’s UI.
configureLocalViewFilter	Function called for setting up an Intent filter to capture messages from classes of this application.
configureWiFiReceiver	Function called for registering a class so Wi-Fi scan results can be received by the application. This function is used by devices with Android OS 8.0 and lower.
initializeViews	Function called to set several views, so a user can operate application functionalities, the views being, a button for initiating Wi-Fi scanning and connection phases. The operations are only used by devices with Android OS 8.0 and lower installed.
registerGPSConnectionIntentFilter	Function called for registering a filter for intercepting events related to the state of the GPS Location.
unregisterGPSConnectionIntentFilter	Function called to unregister the GPS Location receiver to avoid memory leakage.
setWiFiConnectionIntentFilter	Function setting up a filter for catching Wi-Fi state changes.
wifiConnectionFilter	A receiver variable used to receive Intents regarding Wi-Fi state changes, scan availability and post-connection of a suggested AP.
unregisterScanReceiver	Function called for unregistering a class for receiving Wi-Fi scan results. This class is exclusive to devices with Android OS 11.0 and above.
registerScanReceiver	Function called for registering a class for receiving Wi-Fi scan results. This function is exclusive to devices with Android 11.0 OS installed.
gpsStateChangeReceiver	Variable used for listening state alterations of the

	device’s GPS Location module. For devices with Android 9.0 OS and above installed.
checkWiFiPermissions	Function called to check on permissions to ensure proper application functionality. If any necessary permission is missing, it is requested by the user on runtime.
viewChanger	Variable used to receive Intent messages from other classes related to the application’s UI.
startActivityIntent	Variable user for initiating another activity outside of the scope of the main activity. Specifically, the video streaming activity. Upon its completion, the user disconnects from the connected Wi-Fi AP.
onRequestPermissionsResult	Function used for receiving the permission request results.
checkLocationStatus	Function used to check if the GPS Location is turned on.
cancelNotification	Function used to cancel any active notifications.
createChannel	Function called to create a notification channel before issuing any notifications to the user.
makeLocationNotification	Function used to generate a new notification, informing the user that the GPS Location is inactive. The user is then prompted to turn it on to ensure the proper functionality of the application. Implemented by devices with Android 8.0 OS and above installed.
checkWiFiThrottling	Function used to check the state of the Wi-Fi scan throttling option. If disabled, the user is prompted with a notification. Implemented by devices with Android 11.0 OS and above installed.
makeThrottlingNotification	Function that generates a notification suggesting to the user to turn off the Wi-Fi scan throttling option.
onDestroy	Final call you receive before your activity is destroyed by the operating system. Intent message receiver variables are unregistered, and

	any present notifications are released.

HTTPSession

Class used for generic HTTP operations, GET and DELETE in this case. The class runs on a thread different from the UI to avoid any freezes to the application.

Table 3: HTTPSession class

HTTPSession	Constructor of the class. It initializes a dictionary containing values for the upcoming session.
onPreExecute	Function called before the session starts so several variables related to the HTTP request is initialed.
doInBackground	Session creation and connection to the negotiation server.
ccreateJSONResponse	Helper function used for converting the headers and extra information received from an HTTP response to a JSON object.
readStream	Helper function used for parsing the HTTP response and converting it to a JSON object.

WifiReceiver:

Class responsible for receiving Wi-Fi scan results. It sorts out APs with specific SSID set to them, showcases the list to the application’s UI and logs scan related metrics.

Table 4: WifiReceiver class

WifiReceiver	Constructor function called when a new instance of this class is instantiated. It initializes variables for the list of available APs, metrics collector class and a context to showcase the received list to the user.
onReceive	Function called after a Wi-Fi scan has ended and a list of APs is available. The list is displayed to the user and then is examined for any APs with a specific SSID set to them and added to a new list, to later be used to connect to one of them. Finally, a message is sent to the MainActivity, signaling the transition to the connection phase.
sendMessage	Function used to signal the MainActivity that the user can proceed to the connection phase.

WiFiScanResults:

Class that receives available Wi-Fi scan results in the form of a list. It is used to sort out the received APs by either their SSID or vendor specific information elements and suggest to the OS any APs to connect to. The list is displayed to the user via the applications UI. The class also provides methods for disconnecting from a suggested AP and also wiping out any saved network suggestions. This class is only used by devices with Android 10.0 and above installed.

Table 5: WiFiScanResults class

WiFiScanResults	Constructor of the class. Called when an instance is created. Initializes class variables related to the list of suggested Wi-Fi APs and the context with which the collected APs are shown.
calculateWiFiLevel	Function labeling the RSSI of a Wi-Fi AP as Weak, Moderate and Strong.
onScanResultsAvailable	Function used to receive Wi-Fi scan results. For each Wi-Fi AP included in the received list, its Vendor Elements are examined and for APs with specific Vendor Elements value, it is suggested to the system so the device can connect to it. Finally, the results are displayed to the user.
showScanResults	Function used for displaying to the user all the received Wi-Fi APs.
checkVendorElement	Function used to go through the information elements of a received Wi-Fi scan result. The value of the Vendor Specific Element is inspected for a specific string value. The function is called only by devices with Android 11.0 and above OS installed.
getSuggested_ssids	Getter function that returns all the suggested networks' SSIDs.
unregisterCurrentNetwork Suggestion	Function used to disconnect from the current connected Wi-Fi network. The network must be one of the suggested networks.
unregisterAllSuggestions	Function used to unregister all suggested Wi-Fi APs from the system.

WiFiConnection:
Class used to initiate a connection towards a Wi-Fi network as well as registering the user to an onboard local server so he/she can access the Internet.

Table 6: WiFiConnection class

WiFiConnection	Constructor function of class WiFiConnection. Used to initialize class

	variables used on the following functions.
setMode	Function used for setting the mode for connecting to or disconnecting from a selected Wi-Fi Access Point.
run	Function used to either initiate a connection to an AP or to disconnect the device from an already connected AP. Implemented for devices with Android 8.0 OS and below.
sendMessageVisibleOFF	Function used to send a message to MainActivity to make the Connect and Scan buttons to change visibility states. This function is implemented only by devices with Android 8.0 OS and below installed.
findBestRSSI	Function used on finding the best RSSI from a list of available Wi-Fi networks. This function is implemented only by devices with Android 8.0 OS and below installed.
findAccessPoints	Function used to sort out a given list of Wi-Fi networks via their SSID. For each acceptable network, the best RSSI is calculated and is used to initiate the connection phase. This function is implemented only by devices with Android 8.0 OS and below installed.
getExistingNetworkId	Function used to check if the Wi-Fi network is included to the device's saved networks. If it is included, its ID is returned. This function is implemented only by devices with Android 8.0 OS and below installed.
connectToNetwork	Function used to connect to a selected Wi-Fi network via its SSID. Before connecting, if the device is already connected to a Wi-Fi network, the user is prompted to accept on disconnecting from it. This function is implemented only by devices with Android 8.0 OS and below installed.
isNetworkConnected	Function used to check if the connected Wi-Fi network has internet connection. This function is implemented only by devices with Android 8.0 OS and below installed.
startRegistration	Function that makes a GET HTTP request to a server so the user can be registered and monitored for the chosen data, as well as get information about the video streaming server.

fillSessionVariables	Helper function creating a dictionary of header values.
removeNetwork	Function used to disconnect from the currently connected Wi-Fi network. This function is implemented only by devices with Android 8.0 OS and below installed.
readHTTPResponse	Helper function used for extracting response information and sends them to the MainActivity for the video streaming activity to be started.
startUnregistration	Function used for unregistering the device from the monitoring system and disconnecting from the connected Wi-Fi Access Point.
sendMessage	Sends the outcome of the registration procedure to the MainActivity class to display it to the user via the application's UI.

Metrics:
Class used to calculate the elapsed time of procedures related to Wi-Fi scanning and Wi-Fi connection.

Table 7: Metrics class

Metrics	Class constructor function used to initialize variables used for the following functions.
setWiFiStartScanTime	Function that sets a timestamp for the beginning of Wi-Fi scanning.
setWiFiEndScanTime	Function that sets a timestamp for the end of Wi-Fi scanning.
setWiFiStartConnectTime	Function that sets a timestamp for the beginning of the Wi-Fi connection phase.
setWiFiEndConnectTime	Function that sets a timestamp for the end of the Wi-Fi connection phase.
logWiFiScanTime	Function used to calculate the elapsed time for the Wi-Fi scanning procedure.
logWiFiConnectTime	Function used to calculate the elapsed time for the Wi-Fi connection procedure.

getAverageScanTime	Function used to calculate the mean elapsed time for the Wi-Fi scanning procedure.
getTotalScanTime	Helper function used for calculating the sum of the scan times array elements.
getStandardDeviationScanTime	Helper function used for calculating the standard deviation of the logged Wi-Fi scan timestamps.
getAverageConnectTime	Function used to calculate the mean elapsed time for the Wi-Fi connection procedure.
getStandardDeviationConnectTime	Helper function used for calculating the standard deviation of the logged Wi-Fi connect timestamps.

VideoPlayer
Package containing classes related to video streaming over HTTP, while listening to events that may occur while the session is active.

ExoVideoPlayer:
Activity class responsible for initializing an ExoPlayer instance, connecting to an HTTP video streaming server and logging information related to the session’s metrics.

Table 8: ExoVideoPlayer class

onCreate	Function called upon creation of the activity. It receives information from the MainActivity, initializes event listeners for streaming session information and metrics and creates an ExoPlayer video player class instance.
registerExoEventListener	Registers a video player related event listener.
unregisterExoEventListener	Helper function un-registering the video player related event listener.
onStop	Function called when the activity is stopped and put in the background. It un-registers listeners and returns to the MainActivity.
onDestroy	Called before the activity is destroyed. Un-registers all the video player related event listeners before returning to the MainActivity.

ExoEventListener:
The inner class ExoEventListener is used for capturing events from the Exo player.

Table 9: ExeEventListener class

ExoEventListener	Constructor of the class.
onPlayWhenReadyChanged	Callback function signaling that the player class instance is prepared.
onPlaybackStateChanged	Callback function catching a change of the video player’s state

ExoAnalyticalListener:
Class used for capturing events originating from an instance of an Exo Player. Specifically, events related to buffering, starting, and stopping loading bytes from a remote source, as well as errors regarding the connection between the player and the server.

Table 10: ExoAnalyticalListener

ExoAnalyticalListener	Constructor of the class.
onBandwidthEstimate	Callback function called on the end of the loading of the video from the remote source printing the estimated bitrate.
onLoadStarted	Callback function called when the video started loading from the remote source.
OnLoadCompleted	Callback function called when the loading of the video from the remote source reached its end.
OnIsLoadingChanged	Callback function called when the loading state is changed.
OnLoadError	Callback function called when an error occurred on playback.
OnDroppedVideoFrames	Callback function called when video frames were dropped.
SendMessage	Function creating a message that is sent to the ThroughputHandler class.

ExoTransferListener:
A class responsible for logging transferring events that can occur during the loading of the video from the remote source.

Table 11: ExoTransferListener class

ExoTransferListener	Constructor of the class.
OnTransferInitializing	Function called when a transfer is being initialized.
OnTransferStart	Function called when the transfer of bytes has started from the remote source.

onBytesTransferred	Function called incrementally during a transfer of bytes from the remote source, while it also calculates the total bytes that are transferred.
OnTransferEnd	Function called upon the video was received in its total from the remote source.
SendMessage	Function creating a message that is sent to the ThroughputHandler.

ThroughputHandlerThread:
Class handling messages from different event handlers of an ExoPlayer instance. It calculates the bytes loaded from the remote source between intervals.

Table 12: ThroughputHandlerThread class

ThroughputHandlerThread	Constructor of the class.
onLooperPrepared	Callback function indicating the Looper is prepared.
getHandler	Function for returning the handler instance.

ThroughputHandler:
The inner ThroughputHandler class extends that inherits the Handler class. Receives incoming messages from both the analytics and transfer listeners.

Table 13: ThroughputHandler class

FormatTime	Function formatting the time as HH:MM:SS
handleMessage	Handler function for the incoming messages from the analytics and transfer listeners.
CalculateThroughput	Calculate the average throughput.

4.2 Python
In this subsection, programs developed on the Python language have their functionality described. For each file, the classes present are firstly described, followed by functions included in them.

The programs are segmented as follows:

- SDN Controller application
- Servers

SDN Controller application:
The SDN controller related files are listed below.

- controller_application.py

- metrics_class.py
- mobile_users_classes.py
- switch_classes.py
- constant_values.py
- params.conf

controller_application.py:
This file facilitates the SDN-Controller functionality. The file has two classes.

- ControllerMonitor: Manages OpenFlow enabled switches by assigning flows and monitoring registered users.
- RegistrationController: A WSGI server running on the configured IP address and port listed in the run file. Its role is to receive requests from the negotiation server for registering and unregistering mobile clients to the monitoring system of the first class.

ControllerMonitor:
Following are the function contained in the ControllerMonitor class with a short description of their purpose.

Table 14: ControllerMonitor class

_make_config_dictionary	Function used to make an OSLO configuration object used to parse the configuration file holding values used by the SDN-Controller application.
_find_config_flag	Inline function used for searching the configuration file name from the arguments of the execution command.
__init__	Constructor class of the application. Called when the class is created and used for initializing variables used by the application
_get_IP	Inline function used for seeking and retrieving the flag and given value of the IP from which the SDN-Controller application listens for events from connected OpenFlow enabled switches.
_print_app_parameters	Inline function used for displaying some of the application’s variables.
_switch_state_change_handler	Function used for listening on events regarding state

	changes of OpenFlow enabled switches.
_add_user_flows	Inner function of '_register_users', used to add user specific flows for a newly registered user.
_register_users	Function used for registering mobile users to the monitoring system. Essentially, a TCP socket server is created that listens for requests on a specific port, that runs on a separate from the main event thread.
_make_new_entry	Function used for creating an instance of class userDataClass, in order for the user's data to be monitored. A unique value, termed cookie, is given to the user and is an opaque controller identifier.
send_features_request	Function used for requesting features from a recently connected OpenFlow enabled switch.
send_port_desc_stats_request	Function used for sending a request regarding port information to a newly connected OpenFlow enabled switch.
_monitor_flows	Function requesting flow statistics from every connected OpenFlow enabled switch registered to this controller application. This function runs on a separate from the main event thread.
_request_switch_stats	Function making requests for flow and wlan0 port information from a specific OpenFlow switch.
_flow_stats_reply_handler	Function used for catching flow statistics replies from connected OpenFlow switches, displaying the established flows to the terminal window, and monitors the data usage of registered mobile users.
_make_cookie_dictionary	Inner function of '_flow_stats_reply_handler', used for making a dictionary of available user cookies.
_del_flows	Function used on deleting a set of flows formerly added for a registered mobile user.
port_desc_stats_reply_handler	Function used for receiving port description replies from connected OpenFlow enabled switches. The received information are displayed to the terminal window in a formatted way.

switch_features_handler	Function used for receiving switch features replies from any connected OpenFlow enabled switch. The information are displayed to the user in a formatted way. In addition, initial flows for DHCP, ARP, DNS and TCP protocols are sent to the switch.
redirect_table_flow	Method sending a flow for sending unmatched packets from the mobile user specific table to the general rules table.
unregister_user	Function used for removing users from the monitoring system. It is called when a user has exhausted the provided data. All the flows related to the user are deleted and the cookie value is added back to the available pool of cookies.
flow_removed_event	Function used for catching a flow removal event originating from an OpenFlow enabled connected switch. It keeps track of the amount of deleted flows dedicated to a formerly registered mobile user.
send_flow	Function used to create a flow addition request to a connected OpenFlow enabled switch.
packet_in_handler	Function used to receive missed packets from any connected OpenFlow enabled switch.

RegistrationController:
Following are the functions contained under the RegistrationController class, with a short description.

Table 15: RegistrationController class

__init__	Constructor of the class
register_mobile_user	Function used for responding to requests from the registration server to register a mobile user via the ControllerMonitor class.
unregister_mobile_user	Function used for responding to requests from the registration server to unregister a mobile user via the ControllerMonitor class.

metrics_class.py:
File containing a class, named ControllerMetrics, logging controller related metrics. Specifically, the elapsed time between flow statistics requests, the elapsed time for establishing flows to a newly connected OpenFlow enabled switch as well as the difference in bytes monitored from connected users.

Table 16: ControllerMetrics class

__init__	Class constructor initiating variables used by the below functions.
_connect_to_sql_server	Function used for registering to a local mySQL database.
new_user	Function creating a new entry in a dictionary used to monitor the byte difference of each registered user, whenever a flow statistics response is handled by the SDN-Controller application. The key value is the cookie value given to the newly registered user.
log_port_stats	Function logging informations taken from the 'wlan0' interface of an OpenFlow switch. Information logged are collisions, errors upon reception and transmission and dropped packets.
update_net_throughput	Function used to measure the Network Throughput from flows related to streaming HTTP videos
remove_user	Function called to remove any entry related to dictionaries tracking information about the to-be unregistered mobile user. Called when a user is unregistered from the monitoring system.
_write_switch_metrics	Function called for inserting network throughput data recorded for a switch
_write_switch_metrics_to_database	Inline function used for inserting the recorded data to the local mySQL database

_write_switch_metrics_to_file	Inline function appending logged throughput metrics from a user's temp file to the general net throughput file
_write_user_metrics	Function called for inserting the data recored of a user that is about to be un-registered from the monitoring system
_write_user_metrics_to_database	Inline function called for uploading the metrics logged on a user's temporary file
_write_user_metrics_to_file	Inline function appending logged throughput metrics from a user's temp file to the general net throughput file
_remove_temp_file	Function removing a user's temporary file
set_start_timestamp_flow_stats	Function used to log the time stamp after a request for flow statistics to a switch is made.
set_end_timestamp_flow_stats	Function used to log the time stamp after a reply for flow statistics from a switch is received.
_log_flow_stats_elapsed_time	Function called for calculating the elapsed time between flow statistics request and reply.
set_start_timestamp_flows_establish	Function used to log the time stamp from the beginning of sending flows to a connected OpenFlow enabled switch.
set_end_timestamp_flows_establish	Function used to log the time stamp after sending flows to a connected OpenFlow enabled switch.
_log_established_flows_elapsed_time	Function called for calculating the elapsed time of flow establishment to an OpenFlow enabled switch.

mobile_users_classes.py:
File containing a class, named userDataClass, holding information about any registered mobile user. The class contains methods for calculating the amount of data a user has

consumed, checking if the user has exceeded the amount of data chosen, as well as methods for receiving and logging information about the user.

Table 17: userDataClass class

__init__	Class constructor, used for initializing variables with user information such as the IP and MAC address, the video server address and port, the chunk size that the server will send data to the mobile client and the cookie value given by the SDN-Controller application.
__str__	Function returning a string containing several values in a formatted way.
getIP	Function that returns the IP address of a mobile user.
getMAC	Function that returns the MAC address of a mobile user.
getCookie	Function that returns the cookie value of a mobile user.
getServerAddress	Function that returns the video streaming server address.
getServerPort	Function that returns the video streaming server port.
getChunk	Function that returns the video streaming server transmission chunk size.
getUserId	Function that returns a unique user ID.

switch_classes.py:
File containing classes related to storing information about the connected switches to the SDN-Controller application. The classes contained in this file are:

- switchInfo
- switchInfoList

switchInfo:
The switchInfo class holds information of an OpenFlow enabled switch connected to the SDN-Controller application. Information such as its datapath class, its IP address and its datapath id.

Table 18: switchInfo class

__init__	Constructor of the class.
get_datapath_class	Function returning the datapath class of a switch.
get_ip_address	Function returning the IP address of a switch.
get_datapath_id	Function returning the datapath ID of a switch.

switchInfoList:
The switchInfoList class is a list containing instances of the switchInfo class.

Table 19: switchInfoList class

__init__	Constructor of the class.
findIP	Function seeking the index of a switch given an IP address.
findDatapathID	Function seeking the index of a switch given a datapath ID.
get_datapath_class	Function that returns the datapath class of a connected switch from a given switch id.
removeSwitchViaID	Function removing a switchInfo instance from the list given a datapath ID.
get_switch_ids	Function that returns all the connected switch IDs.

constant_values.py:
File containing constant variables used from the aforementioned files and their included classes and functions.

params.conf:
Configuration file containing configuration parameters regarding the controller application, such as its address and port to communicate with connected OpenFlow switches, switch related parameters, like the polling intervals the controller requests information from switches and general parameters.

Servers:
Two servers were developed for this proof-of-concept solution.

- Negotiation server: A flask server used for communicating with mobile users and registering and unregistering them to the monitoring system of the SDN controller's application.
- Video streaming server: A server able to transmit videos to multiple mobile users.

Negotiation server:
Files, classes, and functions used for this application are described below

- flask_test.py
- shared_functions.py
- constant.py
- servers.conf

flask_test.py:
File that facilitates the functionality of the negotiation server.

Table 20: negotiation_server.py functions

_register_user	Internal function used for registering a mobile user to the SDN Controller's monitoring application.
register_user	Function listening for GET requests from HTTP clients. Upon reception, the user is registered from the monitoring application.
_unregister_user	Internal function used for communicating with the SDN controller's server for requesting the un-registration of a connected mobile user to the monitoring system.
unregister_user	Function listening for DELETE requests from HTTP clients. Upon reception, the user is unregistered from the monitoring application.
main	Main function where the Flask server is initialized and started

shared_funtions.py:
File containing functions that can be used by both the negotiation and video streaming server.

Table 21: shared_funtions.py functions

make_dictionary	Function used for taking information from a configuration file regarding registration and HTTP servers.
log_event	General function used to write an event to a dedicated log file. The event is in CSV format.

set_controller_IP	Set the controller ID by executing a shell command and parsing the output
set_switch_ID	Function used for acquiring the datapath ID of the OpenFlow enabled switch. This is done via the ovs-ofctl to query the OvS system for switch information. The output is concatenated, and the ID is returned.

constants.py:
File that contains constant variables used by both the negotiation and video streaming servers.

servers.conf:
Configuration file containing information used by the negotiation server, such as the server's port and SDN controller negotiation server, and video streaming server, like the address and port and the quality of the video to be streamed.

Video Streaming server:
Responsible for streaming a video to a mobile user.

- flask_video_streaming.py
- shared_functions.py
- constants.py
- servers.conf

flask_video_streaming.py:

Table 22: flask_video_streaming.py functions

__init__	Constructor of the class.
throughput_monitor	Function used for monitoring the number of bytes read from the video file being sent to a client.
get_chunk	Function returning a set of values for the byte chunk to be sent to the client, and a boolean value showing if the file has reached its end.
_read_from_file	Function returning the chunk size to be sent to the client.
_move_file_reader	Function used for moving the file descriptor of the file. The function returns a boolean value indicating the end of file.
get_file_path	Getter function returning the full file path of the video file
generate_video_chunk	Function that reads chunks from a selected video file.
stream_video	Function serving clients making requests.

shared_functions.py:

File containing functions that can be used by both the negotiation and video streaming server.

Table 23: sharder_functions functions

make_dictionary	Function used for taking information from a configuration file regarding registration and HTTP servers.
log_event	General function used to write an event to a dedicated log file. The event is in CSV format.
set_controller_IP	Set the controller ID by executing a shell command and parsing the output
set_switch_ID	Function used for acquiring the datapath ID of the OpenFlow enabled switch. This is done via the ovs-ofctl to query the OvS system for switch information. The output is concatenated, and the ID is returned.

constants.py:

File that contains constant variables used by both the negotiation and video streaming servers.

servers.conf:

Configuration file containing information used by the negotiation server, such as the server's port and SDN controller negotiation server, and video streaming server, like the address and port and the quality of the video to be streamed.

4.3 Bash

In this subsection, scripts written in Bash language have their functionality is described. Specifically:

- generatevendorelement.sh
- run

generatevendorelement.sh:

A script used to generate a vendor element, as described inFigure 21. It takes as an input a string value, it checks for its length to be less than 255 bits, otherwise the user is prompted to rerun the program with a shorter value. If no input string is given, the user is prompted to give a value. This value can be used by the hostapd to be broadcasted to mobile devices run:

Script used to initiate the SDN-Controller application. The port and IP in which the application will be ran are set and the application is executed via the ryu-manager command.

5 NUMERICAL RESULTS

In this subsection, we evaluate the performance of the proof-of-concept solution for all its components. Referencing the main components of the solution from section 4.4, for the network infrastructure, as shown in Figure 24, we used a Raspberry Pi 3B+, the mobile clients, as shown in Figure 23, were 3 Android smartphones, with Android OS versions 7.0, 9.0 and 11.0, and lastly for the network controller, as shown on Figure 25, the RYU SDN controller framework was used.

Table 24: Raspberry Pi 3B + Specifications

CPU	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
RAM	1GB LPDDR2 SDRAM
Wireless Protocols	• 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac Wi-Fi • Bluetooth 4.2 • BLE (Bluetooth Low Energy)
Ethernet	Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
IO Ports	4 USB 2.0 ports
Power	5V/2.5A DC

Table 25: Mobile Devices Specifications

Model Names / Specifications	One Plus Nord N100	Xiaomi Redmi Note 4	Xiaomi Redmi Note 6
Processor	Snapdragon 460	Snapdragon 625	Snapdragon 636
RAM	4 GB	3 GB	3 GB
OS Version	11.0	7.0	9.0
Wi-Fi Versions	a/b/g/n/ac	a/b/g/n	a/b/g/n/ac

Table 26: Big Buck Bunny video information

Resolution@ Frame Rate	1920x1080@30fps	1280x720@30fps	640x480@30fps
File Size (bytes)	342.416.819	197.601.700	94.671.927
Duration (minutes, seconds)	10.34	10.34	10.34
Bitrate (bits/sec)	4.316.631	2.491.039	1.193.468

The streaming and negotiation server was configured with the Flask framework. For the streaming server the Gunicorn python HTTP server was used for managing the number of workers serving a video to a connected client and the NGINX server the routing and buffering of data between the Flask streaming server and a mobile client’s device. For the video, we chose the Big Buck Bunny in 3 different resolutions, 1080, 720 and 480, with frame rate of 30 fps (frames per second) to achieve compatibility with the devices used.

From the mobile client's side, the ExoPlayer [45] media player was used and configured for progressive HTTP video download.

5.1 Mobile Client Performance

The performance of APIs for scanning for and connecting to a Wi-Fi AP was evaluated. For this purpose, three different smartphones were used, all with a different Android OS version, as they are described in Table 25.

For the devices with Android OS until 9.0, the scanning time was measured from the moment the user issues a command to the system, by pressing a button, to collect any APs in its vicinity until the results are received as a callback to a class. For devices though with OS version 10 and onwards, the scanning procedure has been automated and the application waits from the system to scan the device's area. A callback class was created to capture these results each time.

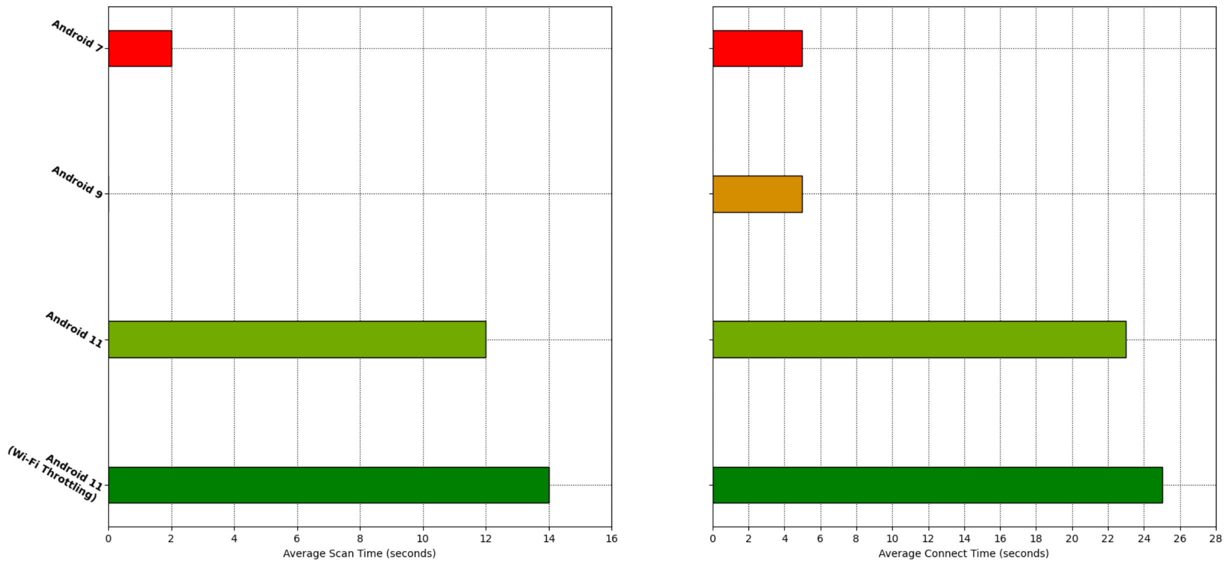


Figure 31: Average Scanning (left) and Connection (right) time

For the average scan time, devices with Android OS 7.0 and 9.0, the connect time was measured from the moment the user issues a connect command to the application, again by pressing a button present on the screen, until its successful connection to the selected AP. In similar fashion with the scanning procedure, devices with OS version 10.0 and onwards had the connection procedure automated. The connection time was measured after the reception of the first batch of Wi-Fi APs scan results up to the successful connection to a suggested AP, signaled with the reception of a callback event from the system. For the device with Android 11 OS, the average scan of Wi-Fi APs and connection time to a suggested AP was measured with the Wi-Fi throttling option was enabled.

As shown in **Error! Reference source not found.**, devices with Android OS 7.0 and 9.0 showed more favorable results in scanning for Wi-Fi APs from the device with OS version 11. It is noticeable that the device with OS version 9.0 gave instantaneous scan results related to the device with Android 7.0 version. The average scanning time for the Android 11 OS version, both with and without Wi-Fi throttling, were worse. This behavior is translated to the Wi-Fi connection average time as well. Lastly, the performance of the scanning, and subsequently connecting, time for the Android 11 device with and without the Wi-Fi scan throttling option is negligible, with a difference of 2 seconds.

5.2 SDN Controller data monitoring performance

In this subsection, the performance of the developed proof-of-concept solution is showcased for different scenarios of video streaming between the streaming server and one or more mobile clients. With varying parameters in both the streaming server and mobile client, the network throughput was measured as well as the responsiveness of the controller to events generated by a connected network infrastructure.

5.2.1 Throughput for different server chunk sizes and client player buffer sizes

We analyzed the average network throughput of the video streaming server behavior, measured on the SDN controller monitoring application. The video streaming server was developed to be a HTTP progressive video server, meaning there was a fixed bitrate for each video used.

Table 27: Variables for chunk size based video streaming

Chunk Sizes (kilobytes)	16	64	256
Video Resolutions (height)	480p	720p	1080p
Min/Max Buffer Sizes (milliseconds)	30/50	50/100	100/150

For the video streaming server, three different chunk sizes were used for reading the video file before sending the data to the registered mobile client. The video player of the client's device had three minimum and maximum buffers configurations for receiving a video. Finally, three different video resolutions were used, with varying file sizes. All the parameters are described in Table 27.

Error! Reference source not found. shows the network throughput performance for the above variables. From the figure, we deduce that the throughput for the different video resolutions increases when increasing the resolution from 480p to 1080p. This was

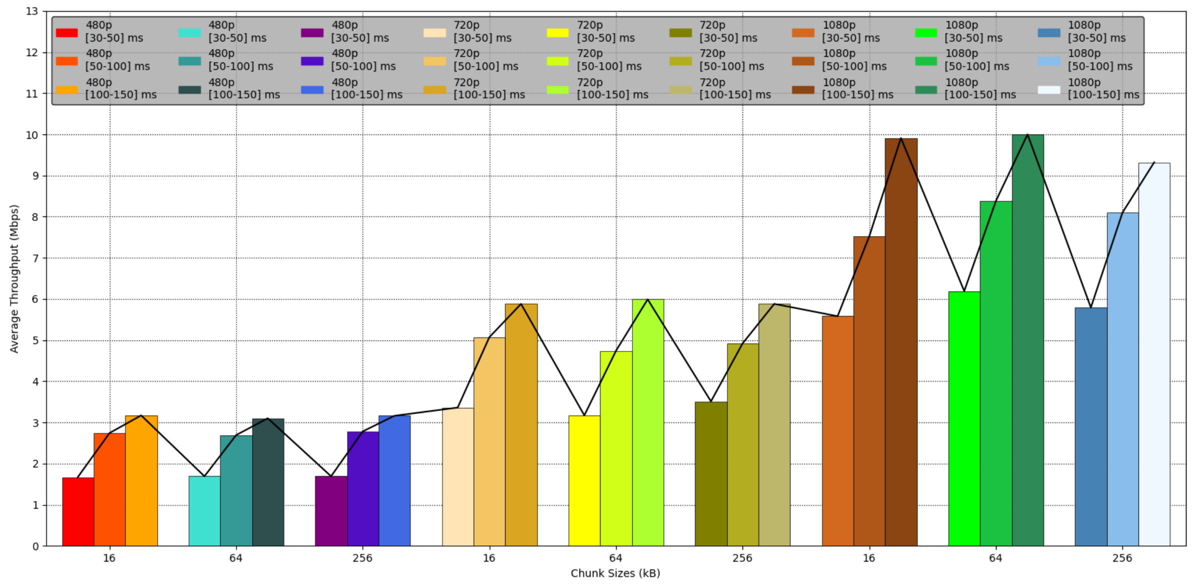


Figure 32: Average Throughput for Varying Chunk Sizes (Mbps)

expected, due to the increase of the video file size as well as the bitrate for those resolutions, as shown on Table 26. We also observe the increase of the network throughput per resolution due to the increase of the min and max buffers with which the client's video player loads data from the video streaming server. By increasing the buffers, the client demands more data from the server for the video to be played to the user. However, different chunk sizes for the same buffer configurations yielded the same results. At first those results don't seem logical, though in the case of the current video server, the client can only select the amount of bytes to be sent to its video player, as it can be seen from the throughput for different min and max buffers.

5.2.2 Throughput of video streaming to multiple users

Figure **33** compares the average network throughput for three different video resolutions. In addition, the scenario was repeated for three min and max buffer sizes configurations on a mobile client's video player, and for up to three concurrent users, with a fixed chunk size transmitted from the HTTP progressive video server.

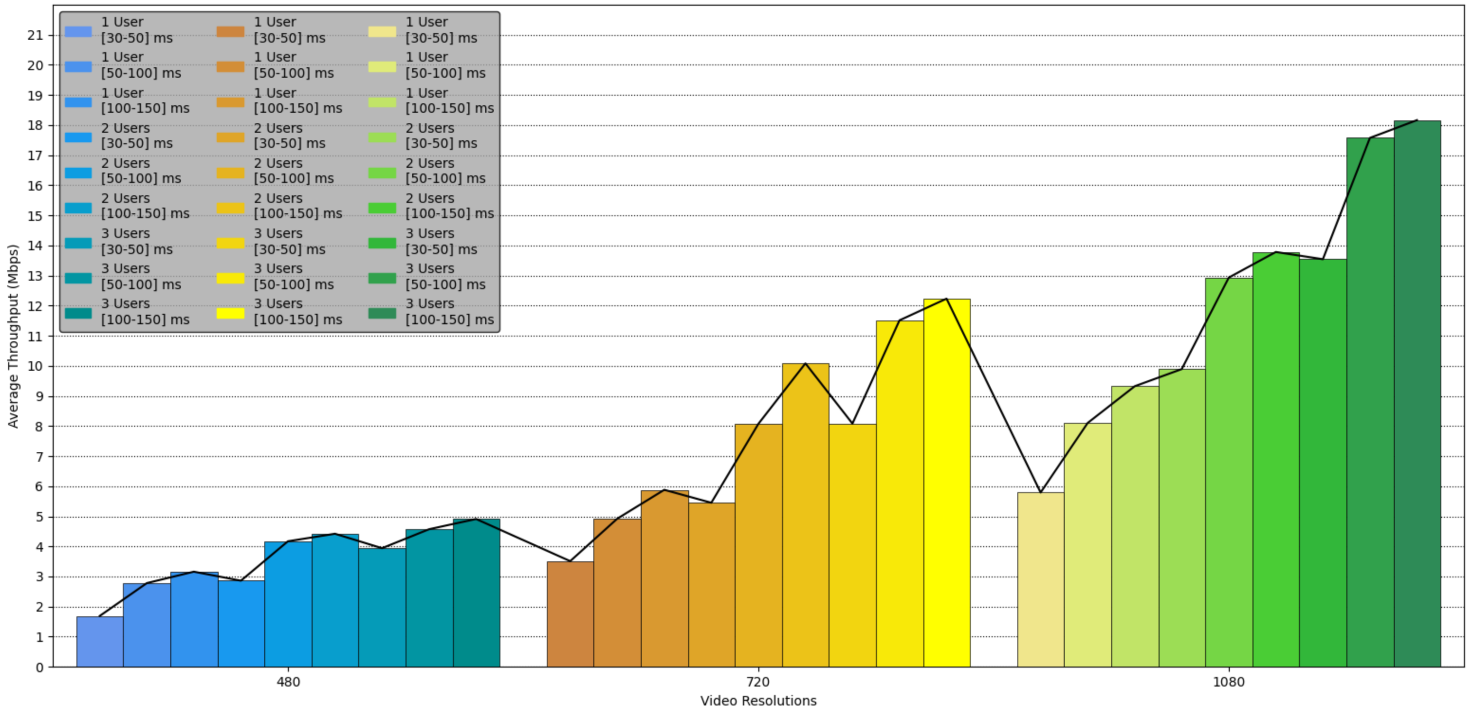


Figure 33: Average Network Throughput for Multiple Concurrent Users

We observe that by increasing the boundaries on the min and max buffers, the throughput increases per video quality. This is expected, since the user demands lengthier video segments, to which a greater number of bytes are transmitted. In addition, the throughput increases with both the improvement of the streamed video's resolutions and the increase of concurrent mobile users. This is yet expected, as the previous result can be applied to multiple users cumulatively, thus the throughput increases as the number of concurrent users is increased.

5.2.3 Runtime Events and Response Time

For the execution of this scenario, one mobile client was used, with the 1080p version of the Big Buck Bunny video, the minimum and maximum buffers of the player are set to 50 and 100ms. The SDN Controller requests every 5 seconds flow and port statistics from the OpenFlow switch. For the event timestamps, UNIX time was used to ensure uniformity among devices. The starting point of this scenario is calculated from the connection of the OpenFlow switch and ends when the mobile user successfully unregisters from the monitoring system.

Figure 34**Error! Reference source not found.** shows the requests and responses of different events that can occur during runtime, on an Android mobile device, the SDN controller application and the negotiation server. We observe that, the SDN Controller application has little to no latency between requests and responses. Looking in the registration and un-registration of a mobile user through all three devices, again, the responses between devices have time difference in the scale of milliseconds.

Figure 35 shows the data transmitted by the video streaming server and captured from a mobile client’s device. From the first two graphs, the transmission and reception of segments of the video are in the same data ranges, with a small deviation of 1.46 megabytes because a mobile client counts only the bytes that it receives without any headers while the SDN Controller counts them. The last plot graph shows the network throughput as it is measured from the SDN Controller. The controller program polls the connected switch for flow statistics and calculates the network throughput. The throughput is in-line with the bytes measured from the two plots. During the playback of the video, the player encountered no errors related to buffering or lost packages.

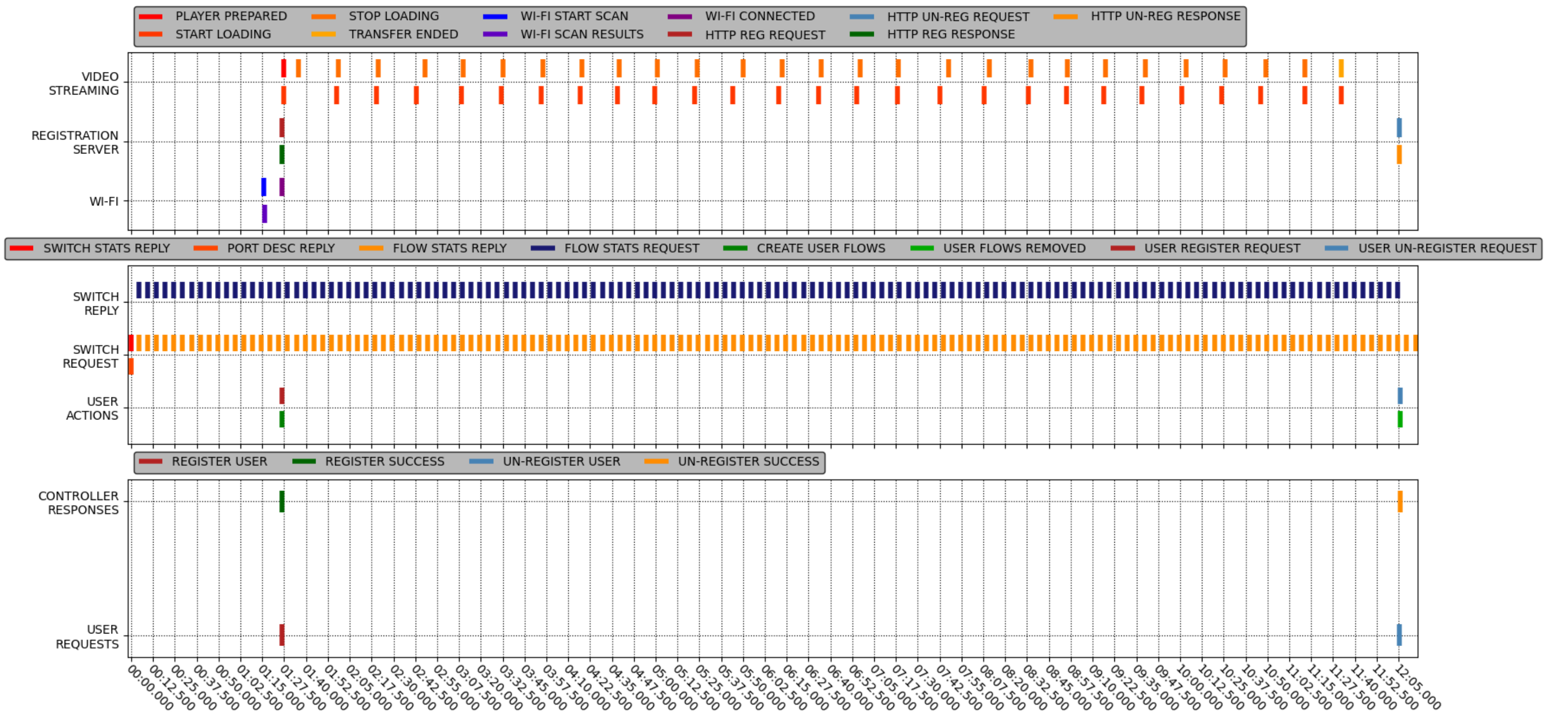


Figure 34: Events from mobile client, SDN Controller and Negotiation Server

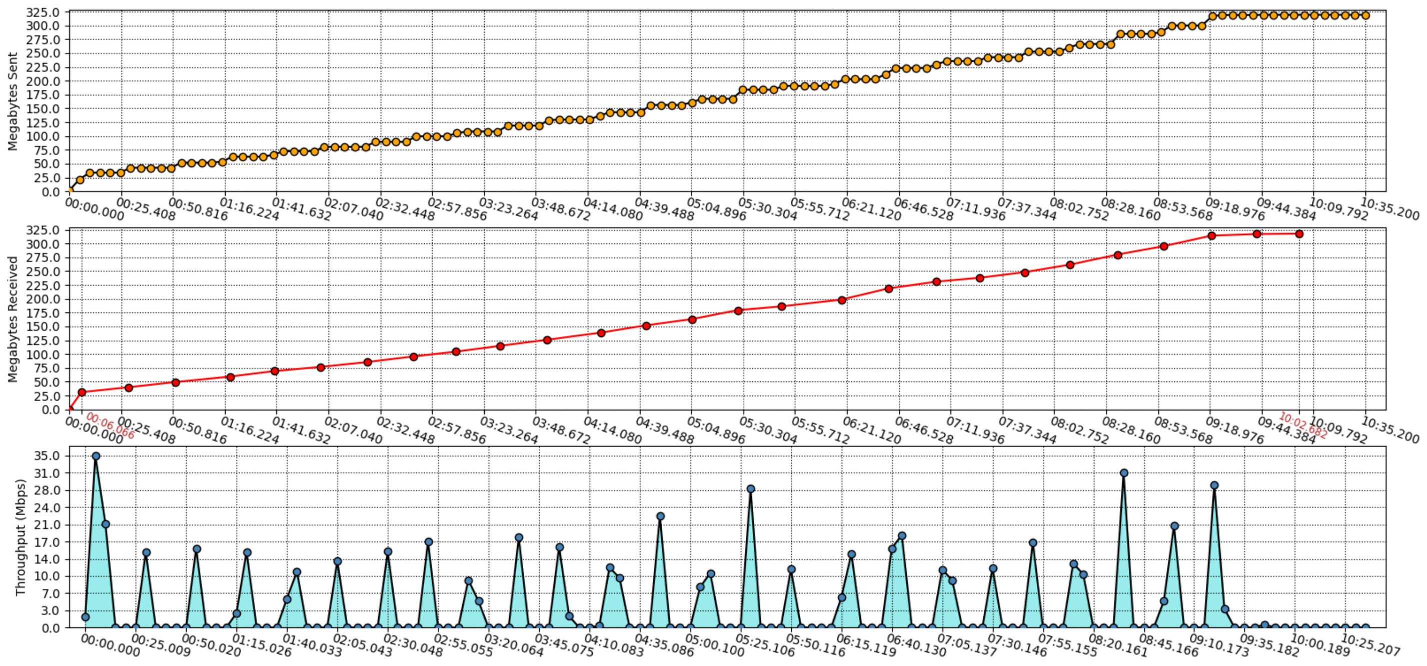


Figure 35: Bytes and throughput measured during a video session

6 INSTALLATION AND CONFIGURATION

In this section a thorough tutorial on how to install and configure any software used is highlighted.

6.1 Installing Python 3

This part is optional if the system already has a version of python 3 installed.

Before installing any program to the Raspberry Pi device, it is advised to execute the below commands to check the system is updated:

```
sudo apt update
sudo apt upgrade
```

All programs written in the Python language are run in the third version:

```
sudo apt install python3
sudo apt-get install python3-pip
```

6.2 Python 3.9 virtual environment

This part is optional if the system already has the Python 3.9 as its default version.

The SDN controller application utilizes packages that can only function properly on python 3.9. Although there are many ways to change the default python version installed in ones system, I opted on creating a virtual environment.

The venv module supports creating lightweight virtual environments, each with their own independent set of Python packages installed in their site directories. A virtual environment is created on top of an existing Python installation, known as the virtual environment's base Python, and may optionally be isolated from the packages in the base environment, so only those explicitly installed in the virtual environment are available.

To create a new environment, execute the following command

```
python3.9 -m venv<path-to-new-virtual-environment>
```

, where the path is the folder the SDN controller application is located.

6.3 OpenFlow switch and Access Point

6.3.1 Installing Raspbian OS

To install the image firmware you either the Raspberry Pi imager software [48], or if you are using Linux based systems, use the dd [49] command.

Using the Raspberry Pi imager

Launch the Raspberry Pi imager, shown in Figure 36.

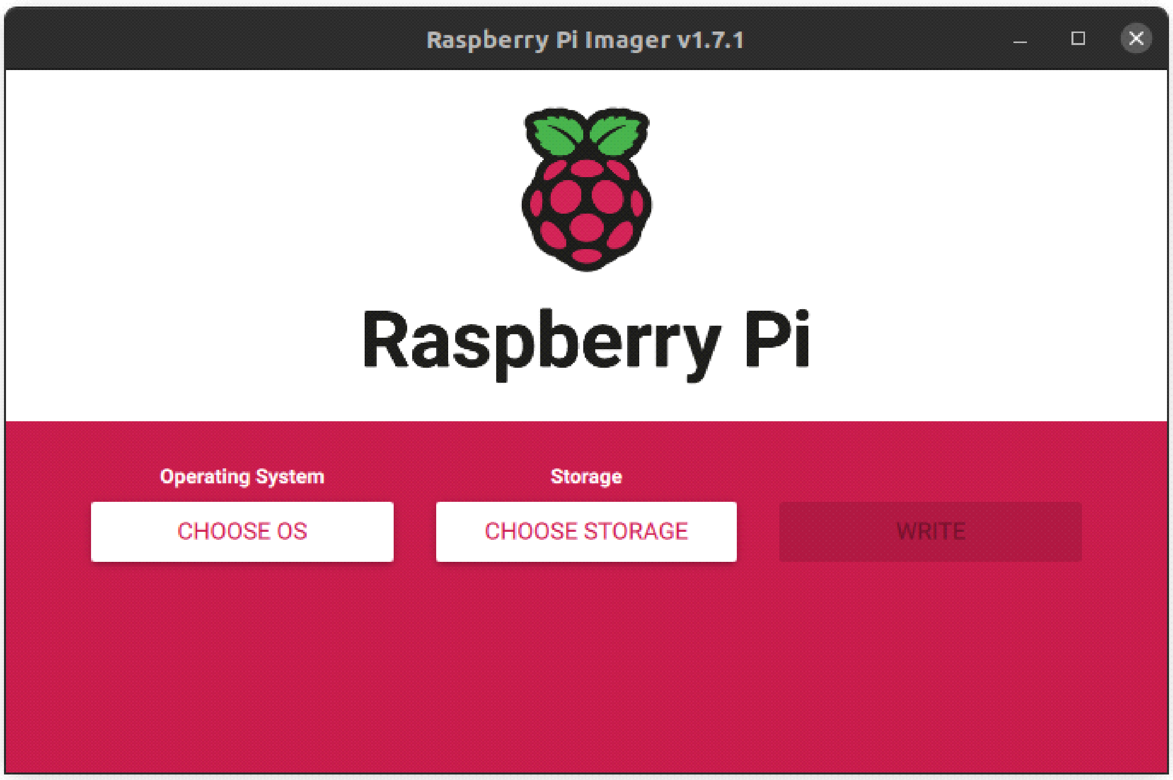
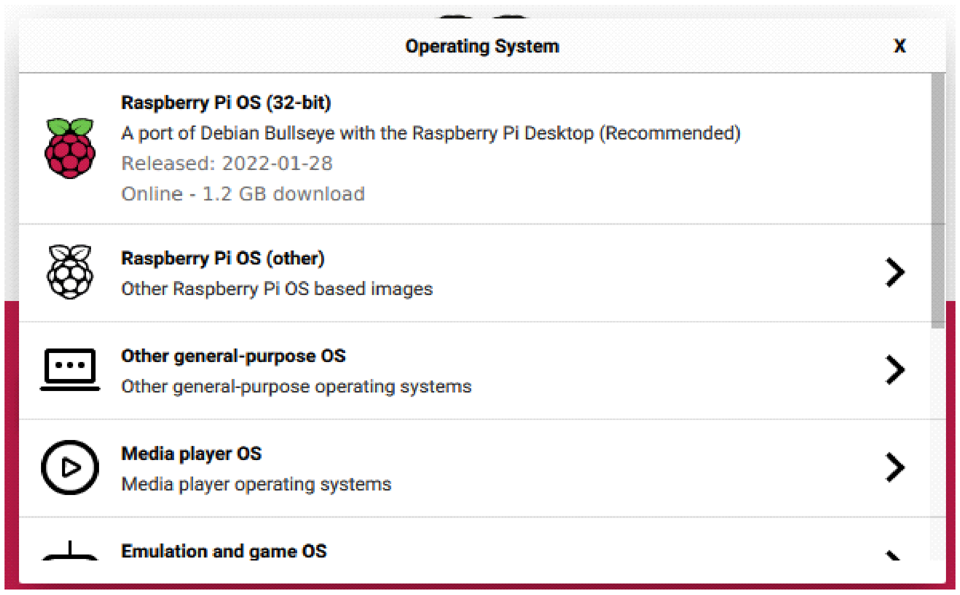


Figure 36: Raspberry Pi imager software

Press the CHOOSE OS button. For the latest version of Raspbian OS, choose the first selection, otherwise, choose the second selection, and navigate to the directory where you have installed the preferred image.



Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2022-01-28
Online - 1.2 GB download

Raspberry Pi OS (other)
Other Raspberry Pi OS based images

Other general-purpose OS
Other general-purpose operating systems

Media player OS
Media player operating systems

Emulation and game OS

Figure 37: Raspberry Pi imager OS selection list (Latest OS)

Press the CHOOSE STORAGE button and select the micro-SD card you chose to flash the firmware.

Afterwards, the button WRITE should be ready to choose. By pressing it, the installation will begin, and you will be prompted upon completion.

Using the dd command for Linux-based systems

Plug the SD card in the computer you have downloaded the image firmware. To identify the micro-SD card, run the following command as super user

```
sudodmesg
```

The most recent message should give you the SD card device name, such as sdb or sdf or similar. Ensure the drive was not auto mounted by your OS.

From root, use dd command to copy the image file to the device you identified

```
dd if=<imagename>.img of=/dev/sdX bs=2M conv=fsync
sync
```

Remove the X part of sdX with the letter the previous step has shown for your SD card.Remove the card and install it to the Raspberry Pi. Plug the charger to boot it up.

6.3.2 Accessing the device

To access the device, use either SSH from your device or attach a keyboard and mouse to the device and connect to a monitor with an HDMI cable. I opted for the second option as there are networking related configuration steps following and SSH connections will be lost upon execution.

6.3.3 Configuring the Access Point

Following is the procedure of configuring the Raspberry Pi device to an AP. The hostapd software is used for that purpose:

```
sudo apt install hostapd
```

Enable the access point service and set it to start when the device boots:

```
sudosystemctl unmask hostapd
sudosystemctl enable hostapd
```

For providing DNS and DHCP services to mobile users, install the dnsmasq software:

```
sudo apt install dnsmasq
sudo nano /etc/dhcpcd.conf
```

Go to the end of the file and add the following:

```
interface eth0
        noipv4
interface wlan0
        noipv4
interface br0
```

To ensure Wi-Fi radio is not blocked on your Raspberry Pi, execute the following command:

```
Sudorfkill unblock wlan
```

and is automatically restored upon boot time.

Create the hostapd configuration file, located at /etc/hostapd/hostapd.conf, to add the various parameters for your new wireless network:

```
sudo nano /etc/hostapd/hostapd.conf
```

and add the following lines:

```
country_code=GR
interface=wlan0
ssid=MyRaspberryPiAP
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=0
vendor_elements=dd0b6c656173655f74696d650a
```

For the changes to be established, the device needs to be rebooted.

6.3.4 OpenFlow Installation and Configuration

Go to directory /home/pi:

```
cd ~
```

From the official site of Open vSwitch [] choose a version to be installed. For this implementation, version 2.13.3 was chosen as it was the more stable at the moment:

```
wgethttps://www.openvswitch.org/releases/openvswitch-2.13.3.tar.gz
```

Upon completion, a tar.gz file will be shown in the directory. Execute the following command to extract a folder and enter to it:

```
tar -xvf openvswitch-2.13.3.tar.gz
cd openvswitch-2.13.3
```

Install an assortment of prerequisite programs in super user mode:

```
sudosu
```

```
apt-get install python-simplejson python-qt4 libssl-dev python-twisted-conch
automakeautoconf
```

Next, install a Linux header. Use the command:

```
uname -r
```

and find the most recent header version for linux-header-x.x.x-x-rpi and install it. For this implementation, version 4.9.0-6 was most recent:

```
apt-get install linux-headers-4.9.0-6-rpi -y
```

Upon completion, execute file configure by specifying the full path of the recently downloaded linux header:

```
./configure –with-linux=/lib/modules/4.9.0-6-rpi/build
```

The Configuration procedure may take some time to complete. Afterwards, execute the following commands in order the openvswitch program to be executed when the device is booted up:

```
cd datapath/linux
modprobeopenvswitch
echo “openvswitch” >> /etc/modules
cd ../../
```

For keeping any configured items between boots of the system, a database is created

```
touch /usr/local/etc/ovs-vswitchd.conf
mkdir -p /usr/local/etc/openvswitch/
ovsdb-tool create /usr/local/etc/openvswitch/conf.dbvswitchd/vswitch.ovsschema
```

Create a new file:

```
nano script
```

and add the following instructions inside:

```
#!/bin/bash
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
ovs-vswitchd --pidfile --detach
ovs-vsctl --no-wait init
ovs-vsctl show
```

After saving the file, change the credentials of the file to be an executable:

```
chmod +x script
```

The file needs to be executed when the device is booted up. For
that, go to home directory and open the .bashrc file:

```
cd ~ &&vim .bashrc
```

and add the following line at the end of the file

```
sudo openvswitch-2.13.3/script
```

This will make the system execute the file, loading any configuration a user has made.

Create a new OVS bridge

```
ovs-vsctl add-bridge br0
```

Set the OpenFlow version for the bridge

```
ovs-vsctl set Bridge br0 protocols=OpenFlow13
```

The Ethernet and wireless NICs will not be given any IP address from the dhcpcd program, since it is added to the bridge br0, while the OVS bridge br0 will automatically be given an IP address.

Add the Ethernet and wireless NICs to the OVS bridge. If you are accessing the device via a SSH connection, see section Accessing the device.

```
ovs-vsctl add-port br0 eth0
ovs-vsctl add-port br0 wlan0
```

Restart the dhcpand hostapddaemons for the above instructions to take effect

```
sudosystemctl restart dhpcd.service
sudosystemctl restart hostapd.service
```

6.3.5 External SDN Controller

Set an external SDN-controller for the OVS bridge br0 to connect to

```
ovs-vsctl set-controller br0 tcp:<address-of-controller>:6634
```

6.3.6 Hostname Configuration for Registration Server

For the onboard registration server an alias was set by using the device IP and a host name. Open the hosts file:

```
vim /etc/hosts
```

and add the following lines at the end of the file:

```
<IP-of-raspberry>leaseconnection
```

With this addition the android application can connect to the registration server without knowing the IP of the server beforehand.

6.3.7 Hostapd patch

For the solution to be functional the hostapd needs to be patched so it can be used an AP can be created in bridged mode and added in the Open vSwitch logical interface. For that, download the patch from Git

```
git clone https://github.com/Unomic/hostapd_ovs.git
```

Go into the newly created folder and access the hostapd directory. Once there, build the patch. Then, reboot for the changes to be established.

```
cd hostapd_ovs/hostpad
make
sudo reboot
```

6.4 RYU SDN controller

Installation of RYU Framework

The RYU framework depends on some programs to be pre-installed to the system before it is installed

```
sudoapt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev python-pip
```

Upon completion, install the RYU controller either via the pip3 python installer or from a git repository

Using the pip3 installer

```
sudo pip3 install ryu
```

Using the project’s git repository, clone the repository, access it and run the installation script via python3 with super user privilidges.

```
git clone git://github.com/osrg/ryu.git
cd ryu
sudo python3 setup.py install
```

Installing and Updating Dependencies

Before any application is executed, install via pip3 some additional python module dependencies. For ease of use, create a text file

```
vim module_dependencies.txt
```

and add the following packets as they are shown

```
packaging
termcolor
netaddr
eventlet==0.31.1
oslo_config
routes
tinyrpc==1.0.4
msgpack
ovs
webob
```

Save and exit the text editor and execute the following command to install and or update already installed modules

```
pip3 install -r module-dependencies.txt
```

6.5 MP4 video meta info

For a video to be played to a mobile user as soon as it is received, the meta information needs to be placed at the front of the video file so the client’s media player can access them quicker.

Execute the following command to achieve it

```
ffmpeg -i input.mp4 -movflagsfaststart -acodec copy -vcodec copy output.mp4
```

6.6 Generate videos

FFmpeg can resize a video and generate a new file. For this solution, I generated a 1080p, 720p and 480p videos from the big buck bunny video in 2160p resolution.

The command to resize a video to a set of height and width is the following

```
ffmpeg -i input.mp4 -vf scale=<width>:<height> -preset slow -crf 18fps=30

 output.mp4
```

The width and height values for the aforementioned resolutions are 1920:1080, 1280:720 and 640:480 respectively. It should be noted that due to hardware limitations of the tested devices, the frame rate is set to 30.

The input.mp4 is the 2160p video resolution and the output.mp4 video is every desired resolution.

7 CONCLUSION

In this paper, we have proposed a proof-of-concept solution for the user-centric RE-CENT method over the Wi-Fi protocol. In the solution was included an application for Android devices, and an SDN-Controller application for two different access point configurations, them being the bridged and routed access point configurations. The performance of the Android application was evaluated for the responsiveness of different APIs related to retrieving information about Wi-Fi APs in a device’s vicinity as well as the connection to a selected Wi-Fi network. Results revealed the prowess of APIs able to initiate on-demand functions for the aforementioned cases. For the SDN-Controller applications, the bridged access point configuration proved to be more superior due to its simplicity on monitoring data usage of different mobile users at the same time. Future work includes the expansion of this solution to cellular systems as well as the development of the Android application for the latest Android OS.

ABBREVIATIONS - ACRONYMS

IoT	Internet-of-Things
NR	New Radio
MNO	Mobile Network Operator
QoS	Quality of Service
QoE	Quality of Experience
OTT	OTher Third-party
AAA	Authentication, Authorization, Accounting
RAN	Radio Access Network
WLAN	Wireless Local Area Network
ISP	Internet Service Provider
SLA	Service License Agreement
RE-CENT	REsource sharing model for user-CENTric
RAT	Radio Access Technology
IMSI	International Mobile Subscriber Identity
KPI	Key Performance Indicator
ASIC	Application Specific Integrated Circuit
SDN	Software-Defined Networking
CDPI	Control to Data-Plane Interface
ACL	Access Control List
ONOS	Open Network Operating System
VSC	Virtualized Services Controller
ForCES	Forwarding and Control Element Separation
POF	Protocol Oblivious Forwarding
NSA	National Security Agency
U.S.	United States
VLAN	Virtual Local Area Network
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
TLS	Transport Layer Security
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

ARP	Address Resolution Protocol
ICMP	Internet Control Message Protocol
MPLS	MultiProtocol Label Switching
TTL	Time-To-Live
VM	Virtual Machine
KVM	Kernel-based Virtual Machine
NIC	Network Interface Controller
LACP	Link Aggregation Control Protocol
GRE	Generic Routing Encapsulation
VXLAN	Virtual Extensible LAN protocol
STT	Stateless Transport Tunneling
LISP	Locator/Identifier Separation Protocol
DPDK	Data Plane Development Kit
MEC	Multi-access edge computing
ETSI	European Telecommunications Standards Institute
MEC	Mobile Edge Computing
ISG	Industry Specification Group
UE	User Equipment
OPEX	OPeratingEXpenses
O-RAN	Open Radio Access Network
CU/DU	Control and Distributed Unit
AI/ML	Artificial Intelligence/Machine Learning
OA	Orchestration and Automation
RIC near-RT	RAN Intelligent Controller near-Real Time
RIC	RAN Intelligent Controller
LAN	Local Area Network
PXE	PrebooteXecution Environment
MAC	Media Access Control
PHY	Physical Layer
OHA	Open Handset Alliance
SMS	Short Message Service
API	Application Programming Interface
UI	User Interface

DVM	Dalvik Virtual Machine
ART	Android RunTime
DEX	Dalvik Executable
AOT	Ahead-Of-Time
JIT	Just-In-Time
GC	Garbage Collection
HAL	Hardware Abstraction Layer
OpenGL	Open Graphics Library
BTC	Bitcoin
ETH	Ethereum
SC	Smart Contract
PoW	Proof-of-Work
PoS	Proof-of-Stake
PoA	Proof-of-Authority
UCI	Unified Configuration Interface
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
NAT	Network Address Translation
ISA	Instruction Set Architecture
HOSTAPD	HOST Access Point Daemon
RADIUS	Remote Authentication Dial-In User Service
EAP	Extensive Authentication Protocol
SSID	Service Set Identifier
DNAT	Destination NAT
AP	Access Point
Wi-Fi	Wireless-Fidelity
IEEE	Institute of Electrical and Electronics Engineers
URL	Uniform Resource Locator
ID	Identifier
CGI	Cellular Global Identifier
PCI	Physical Cell ID
OS	Operating System
SDK	Software Development Kit

RSSI	Received Signal Strength Identifier
HTTP	Hyper Text Transport Protocol
JSON	JavaScript Object Notation
REST	REpresentational State Transfer
RSD	Relative Standard Deviation
SMPT	Society of Motion Picture and Television Engineers
ITU	International Telecommunication Union
SNI	Server Name Indication
SSI	Server Side Includes
XSLT	eXtensible Stylesheet Language
ISAM	Indexed Sequential Access Method
VSAM	Virtual Storage Access Method
SQL	Structured Query Language
RDBMS	Relational Database Management System
CPU	Central Processing Unit

8 REFERENCES

[1] P. R. Newswire, *The Public Safety LTE & Mobile Broadband Market: 2016 - 2030 - Opportunities, Challenges, Strategies & Forecasts*., 2016.

[2] 3GPP, "NR; NR and NG-RAN Overall Description; Stage 2," 2018.

[3] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 3, 2018.

[4] Dionysis Xenakis, Anastasia Tsiota, Christos Thrasyvoulos Koulis, Christos Xenakis, and Nikos Passas, "Contract-Less Mobile Data Access beyond 5G: Fully-Decentralized, High-Throughput and Anonymous Asset Trading over the Blockchain," *IEEE Access*, vol. 9, pp. 73963-74016, 2021.

[5] Open Networking Foundation, "Open Network Operating System (ONOS) | SDN Controller for SDN/NFV Solutions,". [Online]. https://opennetworking.org/onos/

[6] OpenDaylight. OpenDaylight. [Online]. https://www.opendaylight.org/

[7] Faucet. Faucet SDN Controller. [Online]. https://faucet.nz/

[8] RYU. Ryu SDN Framework. [Online]. https://ryu-sdn.org/

[9] Confluence. Floodlight Controller. [Online]. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

[10] Nuage Networks. Virtualized Services Platform. [Online]. https://www.nuagenetworks.net/platform/virtualized-services-platform/

[11] Software Development Kit. The Ultimate SDN Framework. [Online]. https://lighty.io/

[12] B Laurie, A Langley, and E Kasper, "RFC 6962 | Certificate Transparency," 2070-1721, 2013.

[13] Shengru Li et al., "Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability," *IEEE Network*, vol. 31, no. 2, pp. 58-66, Mar. 2017.

[14] Cisco. Cisco ACI and OpFlex Connectivity for Orchestrators. [Online]. https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/kb/b_Cisco_ACI_and_OpFlex_Connectivity_for_Orchestrators.html

[15] Carmelo Cascone, Luca Pollini, Davide Sanvito, Antonio Capone, and Brunilde Sansò, "SPIDER: Fault Resilient SDN Pipeline with Recovery Delay Guarantees," Nov. 2015. [Online]. http://arxiv.org/abs/1511.05490

[16] Open Networking Foundation. OpenFlow. [Online]. https://opennetworking.org/sdn-resources/customer-case-studies/openflow/

[17] Open Networking Foundation. Open Networking Foundation. [Online]. https://opennetworking.org/

[18] Linux Foundation Collaborative Projects. Open vSwitch. [Online]. https://www.openvswitch.org/

[19] IEEE, "IEEE Std 802.11™-2016," 2016.

[20] Statcounter Global Stats. (2023, Mar.) Mobile Operating System Market Share Worldwide. [Online]. https://gs.statcounter.com/os-market-share/mobile/worldwide

[21] Open Handset Alliance. (2012) Open Handset Alliance. [Online]. www.openhandsetalliance.com

[22] Y A Min, "A study on the performance evaluation items of the private blockchain consensus algorithm considering consensus stability," *Journal of the Korea Society of Computer and …*, 2020.

[23] Vitalik Buterin, "A next-generation smart contract and decentralized application platform," *Etherum*, no. January, 2014.

[24] Sunny King, "Primecoin: Cryptocurrency with Prime Number Proof-of-Work," *King, Sunny*, vol. 4, no. 2, 2013.

[25] Sunoo Park, Krzysztof Pietrzak, Joel Alwen, Georg Fuchsbauer, and Peter Gazi, "Spacecoin : A Cryptocurrency Based on Proofs of Space," *IACR Cryptology ePrint Archive*, 2015.

[26] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz, "Permacoin: Repurposing bitcoin work for data preservation," *Proceedings - IEEE Symposium on Security and Privacy*, 2014.

[27] The Hyperledger White Paper Working Group, "An Introduction to Hyperledger," *Advances in Computers*, vol. 121, 2018.

[28] B. Vitalik and G. Virgil, "Casper the Friendly Finality Gadget," Oct. 2017. [Online]. http://arxiv.org/abs/1710.09437

[29] OpenEthereum. Aura - Authority Round. [Online]. https://openethereum.github.io/Aura

[30] Debian Manpages. hostapd. [Online]. https://manpages.debian.org/testing/hostapd/hostapd.8.en.html

[31] Debian Wiki. dnsmasq. [Online]. https://wiki.debian.org/dnsmasq

[32] Ralph Droms, "RFC 2131 | Dynamic Host Configuration Protocol,". [Online]. https://datatracker.ietf.org/doc/html/rfc2131

[33] "RFC 951 | Bootstrap Protocol,". [Online]. https://datatracker.ietf.org/doc/html/rfc951

[34] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman, "RFC 4861 | Neighbor Discovery for IP version 6 (IPv6),". [Online]. https://datatracker.ietf.org/doc/html/rfc4861

[35] Syam Madanapalli, Jaehoon Paul Jeong, Soohong Daniel Park, and Luc Beloeil, "RFC 6106 | IPv6 Router Advertisement Options for DNS Configuration,". [Online]. https://datatracker.ietf.org/doc/rfc6106/

[36] Dr. Thomas Narten, Richard P. Draves, and Suresh Krishnan, "RFC 4941 | Privacy Extensions for Stateless Address Autoconfiguration in IPv6,". [Online]. https://datatracker.ietf.org/doc/html/rfc4941

[37] Michael Carney, Charles E. Perkins, Bernie Volz, Ted Lemon, and Jim Bound, "RFC 3315 | Dynamic

Host Configuration Protocol for IPv6 (DHCPv6),". [Online]. https://datatracker.ietf.org/doc/html/rfc3315

[38] FFmpeg License and Legal Considerations. [Online]. https://www.ffmpeg.org/legal.html

[39] Flask Documentation. [Online]. https://flask.palletsprojects.com/en/2.2.x/

[40] Android Developers and Google. WifiManager | startScan. [Online]. https://developer.android.com/reference/android/net/wifi/WifiManager#startScan()

[41] Android Developers and Google. WifiManager | enableNetwork. [Online]. https://developer.android.com/reference/android/net/wifi/WifiManager#enableNetwork(int,%20boolean)

[42] Android Developers and Google. WifiManager | disconnect. [Online]. https://developer.android.com/reference/android/net/wifi/WifiManager#disconnect()

[43] Android Developers and Google. WifiManager | reconnect. [Online]. https://developer.android.com/reference/android/net/wifi/WifiManager#reconnect()

[44] Android Developers and Google. WifiManager | addNetworkSuggestions. [Online]. https://developer.android.com/reference/android/net/wifi/WifiManager#addNetworkSuggestions(java.util.List%3Candroid.net.wifi.WifiNetworkSuggestion%3E)

[45] Progressive - ExoPlayer. [Online]. https://exoplayer.dev/progressive.html

[46] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks," *IEEE Communications Surveys and Tutorials*, vol. 22, no. 2, 2020.

[47] GitHub - bilibili/ijkplayer: Android/iOS video player based on FFmpeg n3.4, with MediaCodec, VideoToolbox support. [Online]. https://github.com/bilibili/ijkplayer

[48] Unomic. GitHub - HostApd version 2.6 with patch from Helmut Jacob to make it compatible with OpenVswitch. [Online]. https://github.com/Unomic/hostapd_ovs

[49] Linux man page. iptables(8). [Online]. https://linux.die.net/man/8/iptables

[50] Raspberry Pi. Raspberry Pi OS. [Online]. https://www.raspberrypi.com/software/

[51] Site-specific configuration hook. [Online]. https://docs.python.org/3/library/site.html#module-site

[52] NXP Semiconductors. VortiQa® Software for Networking. [Online]. https://www.nxp.com/design/software/embedded-software/vortiqa-software-for-networking:VORTIQA

[53] aiohttp 3.8.4 documentation. [Online]. https://docs.aiohttp.org/en/stable/