Adaptive QoS in SDN-enabled heterogeneous mobile tactical networks

Master Thesis

Sharath Maligera Eswarappa

Matriculation Number

3068883

This work was submitted to the fulfillment
of the requirements for the degree of Master of Science
from

Institute of Computer Science

University of Bonn, Germany

Advisers:

Dr. Paulo Henrique Rettore Lopes
Dr. Roberto Rigolin F Lopes

Examiners:

Prof. Dr. Peter Martini
Dr. Matthias Wübbeling

Registration date: 10-11-2020
Submission date: 23-05-2021

In collaboration with the Fraunhofer Institute for Communication,
Information Processing and Ergonomics (FKIE), Bonn, Germany

UNIVERSITÄT BONN

Rheinische Friedrich-Wilhelms-Universität Bonn

Prüfungsausschuss Informatik

universität bonn · Institut für Informatik · 53012 Bonn

Vorsitzender des Prüfungsausschusses

Prof. Dr. Thomas Kesselheim

Prüfungsamt:
Judith König
Tel.: 0228/73-4418
Fax : 0228/73-4788
pa@informatik.uni-bonn.de
Postanschrift: 53115 Bonn
Endenicher Allee 19a

www.informatik.uni-bonn.de

Erklärung über das selbständige Verfassen einer Abschlussarbeit/ Declaration of Authorship

Titel der Arbeit/Title:

....

Adaptive QoS in SDN-enabled heterogeneous

mobile tactical networks

Hiermit versichere ich, Maligera Eswarappa, Sharath,
Name / name, Vorname / first name

dass ich die Arbeit – bei einer Gruppenarbeit meinen entsprechend gekennzeichneten Anteil der Arbeit – selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

[signature]
....
Unterschrift/signature

Bonn, den 21/05/2021
date

Abstract

Tactical Networks (TNs) support exchange of information among it's users such as dismounted soldiers, convoys, command and control, medical team etc., to accomplish mission oriented task. These networks are limited by low bandwidth, high latency and constraints of network devices operating in these networks like battery, storage and computational power. Moreover, these networks are composed of heterogeneous radio link characteristics varying in range and frequencies such as High Frequency (HF), Very High Frequency (VHF), Ultra High Frequency (UHF) and Satellite Communications (SatCom). During a certain operational context, these networks experience a dynamic change in network conditions as well as data exchange. Ensuring End-to-End (E2E) Quality-of-Service (QoS) of data in these networks become a daunting task considering the *ever-changing* communication scenario. The traditional networking devices used in these networks offers very little flexibility and complexity in ensuring QoS of data due to coupled control and data functions within these devices. With the emergence of new computer networking concepts such as Software-Defined Networking (SDN) in which the control functions are de-coupled from the forwarding functions of the network devices, we have a new opportunity to re-design the QoS control in TNs using SDN paradigm. Motivated by the features offered by SDN, in this thesis we introduce a mechanism to adaptively ensure QoS for user data flow in heterogeneous TNs by leveraging SDN paradigm. We start with a hypothesis that an application running on the northbound interface of a SDN controller can support the management of unreliable radio links at the edge of tactical networks. Thus, we developed applications to support adaptive shaping of user data flows over data rates supported by VHF, UHF and SatCom radios, and to ensure the dropping of expired messages. We also introduce a *hybrid* scheduling, composed of *priority* and *fairness* based scheduling mechanisms for these data flows using Linux Queuing Disciplines (Qdiscs). The goal is to differentiate IP packets from different command and control services. Our hypothesis was verified with experiments using four classes of messages with different QoS requirements, such as priority, reliability, and time of expire. Experimental results in an emulated network suggest that our solution can differentiate data-flows in a heterogeneous tactical network while ensuring QoS requirements.

Acknowledgments

I would like to express my sincere gratitude to my supervisors: Prof. Dr. Peter Martini and Dr. Matthias Wübbeling for their time, involvement and support, academically and administratively, in this thesis journey making it as smooth as possible. I would like to thank my advisors at Fraunhofer FKIE: Dr. Paulo Henrique Rettore Lopes and Dr. Roberto Rigolin F Lopes for providing an opportunity to do my Master Thesis at the institute. Their continuous support, feedback, ideas, motivation and patience throughout the entire thesis is invaluable.

I would like to extend my thanks to student researchers at Fraunhofer FKIE: Johannes Loevenich and Pooja Hanavadi Balaraju for their support, ideas and insightful comments during this thesis. Last but not the least, I would like to thank my family and friends whose continued encouragement and support never failed to uplift me and my effort during this Master Thesis, especially at the time of this pandemic.

Contents

1

Introduction

Tactical Networks (TNs) are heterogeneous, mobile ad hoc networks ensuring End-to-End (E2E) connectivity between nodes in the battlefield [24]. TNs consist of radio communication links with different characteristics in terms of network capacity, range of coverage and high latency. Current heterogeneous TNs support a wide range of radio communication systems such as High Frequency (HF), Very High Frequency (VHF), Ultra High Frequency (UHF) and Satellite Communications (SatCom) [40]. As the nodes move, the characteristics of these communication systems change, increasing the complexity of data delivery among them. To maximize the data delivery between nodes in a highly dynamic environment, resource efficient routing techniques has to be employed, maximizing the utilization of network resources while ensuring Quality-of-Service (QoS) of the data being delivered [48].

Supporting E2E QoS of data in TNs is crucial because of the low capacity of the links and frequently the users would like to send and receive more data than the network can handle. If QoS is not applied, it can influence network congestion resulting in packet delay and loss. When the network capacity is low, it is of utmost importance that the mission-critical traffic is prioritized by the network at the expense of less important ones [40]. Therefore, the system needs to support different QoS requirements for traffic classes over a tactical heterogeneous network by means of classification and prioritization of flows, and the ability to drop low priority flows when the network approaches congestion and thus maintain fair QoS for high priority traffic flows.

A modern Tactical Network (TN) might consist of multiple mobile tactical units forming a network of unpredictable mobility resulting in rapid topology changes with the nodes joining and leaving the network on a random basis, where link capacities vary; both in time and per-link characteristics. Due to this ever-changing network scenario, network control and management becomes a complex problem, requiring rapid reconfiguration through a resilient control plane since TNs have strong focus on robustness and network utilization. Recent literature [30] suggests that Software-Defined Networking (SDN) is a promising solution since it provides the potential for highly configurable, automated networks by separating the control and forwarding functions of network devices. Because of this separation, network control is directly programmable enabling centralized (logically) network intelligence in software-based controllers, and network hardware devices becoming simple forwarding devices. Since TNs are migrating more towards software and virtualization based solutions, QoS mechanisms in a military-context are being re-designed using SDN paradigm.

In recent years, researchers have explored the potentials of leveraging SDN to address service differentiation, traffic management, and network dynamics at the edge of tactical networks [42, 46, 47, 50]. The features of SDN concept is appealing since it offers significant advantages over conventional networks in tactical networks such as: (i) simplifying the administrative tasks in monitoring and controlling network traffic; (ii) flexible routing mechanism which applies to per traffic flow, based on packet header attributes rather than depending only on destination IP and MAC address or in combination with TCP/UDP port numbers as in conventional networks; (iii) lower complexity and flexibility in network adaption mechanisms since the SDN controllers have visibility of the whole network unlike conventional networking.

1.1 Problem Description

Dismounted soldiers, convoys, medical team along with Command and Control (C2) communicate among themselves with a set of C2 services while representing as nodes in a tactical network. The magnitude of user behaviour within this network depends on unpredictable operational context like warfare, the conduct of combat, evacuation and disaster relief. They generate data without knowing the underlying network conditions. The users forming this network are often limited by infrastructure-less communication environment with connectivity limitation and self configuration characteristics due to technical and physical reasons. While operating within a certain operational context, they experience dynamic network conditions across asymmetric link characteristics resulting in variable packet loss and changes in link data rate.

Considering the challenges in ensuring optimal data delivery among the nodes in these ever-changing network conditions, several studies [20, 29, 32, 36–39, 49] have explored issues in provisioning QoS in TNs. With limited bandwidth and high latency for traffic flows, service differentiation and QoS of these services become important attributes to address, in any tactical scenario. There must be a mechanism to ensure mission-critical data to get priority in network resources when the resources are scare, thereby reducing the packet loss incurred by higher-priority flows when compared to low-priority flows. Also using this mechanism, traffic flows requiring higher bandwidth should be routed through radio link offering sufficient bandwidth towards the same destination. And this mechanism should be implemented with minimal human intervention, in other words the QoS mechanism should be adaptive to the changes in network conditions.

Network devices in traditional TNs couples together control and data plane functions. The control plane within these devices, in the nodes, operate autonomously. This provides very little flexibility and complexity to configure the heterogeneity in network devices, since control and data planes are vertically integrated. Reconfiguring or updating these network devices could be a time consuming and tedious process for network administrators considering different types of network devices requiring them to take offline, leading to the loss of connectivity in network segments. Moreover, these network devices would consist of vendor specific operating system, applications and protocols which hinders the development of new network services.

However, new paradigm in computer networking such as SDN separates network control from forwarding functions of the network devices, where the network control is moved to a central entity called SDN controller. The controller communicates over a vendor-neutral protocol with networking devices, thereby network devices can remain simple forwarding devices and offer interfaces to the higher layers. Thus, network control becomes directly programmable in a more convenient and vendor independent way, accelerating the innovation and development of new network services. SDN in Tactical Networks (TNs) opens up for moving the enforcement of policy management and QoS control from a service application oriented control plane to network oriented control plane [27] providing granular control over data flows and flexibility throughout the network.

1.2 Research question

Based on the previous discussions on challenges of ensuring Quality-of-Service (QoS) requirements of user data flow in resource constrained heterogeneous tactical networks, and inspired by the features supported by Software-Defined Networking (SDN) paradigm, the purpose of this thesis is to study and validate:

- How SDN could be used to enforce QoS requirements in tactical networks?
  - By leveraging SDN, how to shape the traffic to suit the present network capacities over heterogeneous networks considering constant link changes?
  - Moreover, how to drop expired IP packets and prioritize data flow?

1.3 Hypothesis

The thesis began with a research on SDN controller capabilities in supporting efficient management of data over unreliable radio communication links at the edge of TNs to meet stringent and varying QoS requirements. Therefore, we start with a hypothesis that the SDN controller could be used as a tool for monitoring and controlling traffic in a dynamic tactical environment. Network applications running on top of this SDN controller could be utilized in order to differentiate distinct Command and Control (C2) services, ensure QoS requirements for these services, and dynamically shape the data flow across heterogeneous link depending on the network conditions.

1.4 Objectives

The main objective of this thesis is to explore how SDN could be leveraged in addressing QoS requirements for user data flow in ever-changing network conditions experienced in heterogeneous tactical networks. As a proof-of-concept, a QoS framework based on SDN concept was developed to ensure QoS requirements of these data flows in a tactical communication scenario. Depending on the network capacities, adaptively serve these QoS requirements within the provisioned framework. SDN capabilities and constraints towards development of this framework is explored and validated within a SDN emulation platform. Therefore, the objective of this thesis work is formulated with the following specific goals:

- To study related works in provisioning QoS of user data flow in Tactical Networks (TNs), both in traditional as well as SDN based TNs. Study their shortcomings and potential enhancements. Find works related to contributions of SDN towards QoS management.
- To emulate generation of messages from a set of Command and Control (C2) services within a SDN emulation platform, depicting communication between two radios over heterogeneous links.
- To study and implement traffic scheduling mechanisms available within Linux environment to enforce IP differentiation for QoS-constrained data flows from C2 services.
- To shape the data flow with respect to the nominal data rate supported by the heterogeneous radio communication links using Linux Queuing Disciplines (Qdiscs).
- Develop a network application to adaptively shape the data flow depending on the network bandwidth using features supported by OpenFlow switch.
- The proposed QoS framework should be validated by a network topology depicting a minimal communication scenario in a tactical network, deployed within a SDN emulation platform.

1.5 Contributions

During this master thesis, it was investigated how QoS requirements for user data flow could be enforced using the features supported by OpenFlow switch and SDN controller, together with Linux Queuing Disciplines (Qdiscs). MGEN traffic generator was used to emulate five exemplary Command and Control (C2) services with different 'Priority' in form of Type-of-Service (ToS) bits set in the header of an IPV4 packet. Using Qdiscs, 'Priority' QoS requirement was ensured for these distinct services. A *hybrid* traffic scheduling mechanism was introduced using Enhanced Transmission Selection (ETS) Qdisc, merging the functionalities of *priority* and *fairness* based scheduling mechanisms. Hierarchy Token Bucket (HTB) Qdisc was used to emulate the data rates supported by different radio communication technologies and to shape the data flow according to the emulated data rate. Using custom Representational State Transfer (REST) applications running on the north-bound interface of the SDN controller, *flow rules* within the flow tables of OpenFlow switch were configured to differentiate the traffic flows from distinct C2 services and 'Time-of-Expiry (ToE)' QoS requirement was provisioned for these services. Similarly, using a REST application, *meter table* of the OpenFlow switch was configured with entries of data rates supported by a VHF, UHF and SatCom radio modulations. And by assuming that a network application running on the SDN controller should have access to the information on the current data rate of the radio interface in use, a proof-of-concept was developed by serving the information about the data rate at these interfaces to a REST application by Remote Procedure Calls (RPC). Based on the data rate in use at these interfaces, the REST application adaptively shaped the data flow at OpenFlow switch by directing the flow towards appropriate *meter entries* in it. Drawback of this adaptive shaping mechanism was investigated along with the suggestion for improvements. The proposed QoS framework was implemented and validated within the Mininet [28] emulation platform by a minimal network topology depicting the heterogeneous tactical network scenario.

1.6 Thesis structure

The structure of this thesis text is as follows. Chapter 2 briefly introduces heterogeneity in tactical networks followed by the description of SDN architecture and concluded by the discussion on related studies in ensuring QoS in tactical networks by both traditional as well as SDN paradigm. Chapter 3 describes the design and implementation of our adaptive QoS framework on a network topology within a SDN emulation platform. Chapter 4 discusses the experimental results of provisioning this framework in ensuring 'Priority' and 'Time-of-Expiry (ToE)' QoS requirement for user data flows in a tactical network. Chapter 5 finalizes this thesis by presenting the conclusion and future work while listing our publications related to this thesis.

2

Background

This chapter introduces the fundamental concept of Software-Defined Networking (SDN) and Quality-of-Service (QoS), required to understand the work presented in this thesis. The chapter starts with a brief introduction to tactical networks followed by the description of SDN architecture and components of OpenFlow switch, serves the foundation for understanding the SDN based QoS framework presented in chapter 3. This chapter is concluded by discussing some of the related studies in ensuring QoS in tactical networks using traditional as well as SDN paradigm.

2.1 Tactical Networks

Tactical Networks (TNs) are heterogeneous, mobile ad hoc networks (MANETs) hosting critical information systems for communication purposes in the battlefield. It facilitate information sharing among the users to accomplish mission-oriented tasks. It also enables the C2 system's capabilities for network-centric warfare [18] using robust radio links across a contested region. The nodes forming TENs are agile, highly mobile, and inevitably heterogeneous with varying channel characteristics, including range and frequency such as HF, VHF, UHF, SatCom, sensor networks and Unmanned Aerial Vehicles (UAVs) [40]. The nodes which form TENs usually operate in harsh environments with a possibility of frequent link outages. Link outage leads to the loss of connectivity among the nodes forming TENs, resulting in data transmission timeouts and routing failures. Due to these reasons, TENs require rapid reconfiguration and recovery of network topology through a resilient control plane. Recent literature [30] suggests that Software Defined Networking (SDN) is a promising solution since it provides the possibility of configurable, automated networks by separating the control and forwarding functions of network devices. Furthermore, by leveraging the concept of SDN, previous investigations [17, 19] have shown how to handle link failures using one of the two main approaches, proactive and reactive.

2.2 Software-Defined Networking (SDN)

Open Networking Foundation (ONF) defined SDN [15] as an architecture that separates the control and forwarding functions of network devices, such as routers and switches through a process of abstraction. Network control becomes fully programmable because of this de-coupling and moved to a logically centralized software based SDN controller, at a layer above the data plane. The controller could be used to consistently monitor the data flow through the network devices, gain statistical information using which the behaviour of individual network devices can be altered in real-time. As a result of this, network devices become simple forwarding elements through which the packets will be forwarded to other network devices as per the rules defined by the SDN controller. This reduces the complexity of adding, replacing and upgrading the network devices, thereby adding flexibility and scalability in network control.

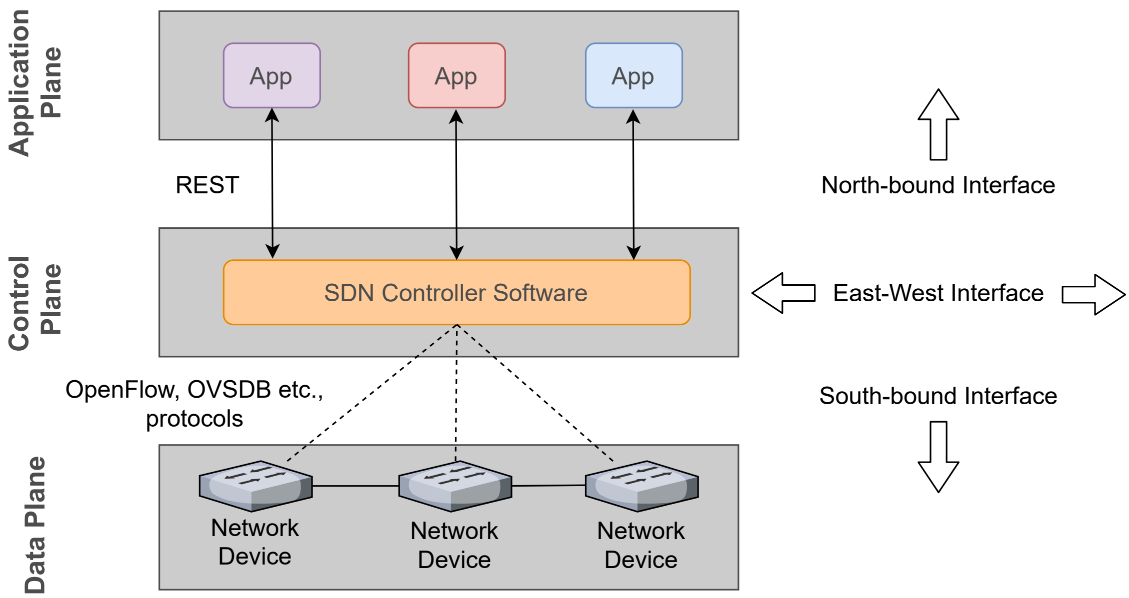

Figure 2.1 Software-Defined Networking (SDN) Architecture proposed by ONF [15]

Figure 2.1 shows the basic SDN architecture proposed by ONF. The lowest level of abstraction consists of hardware network forwarding devices (switches, routers, load balancers etc.,) in the data plane. As per the architecture, network control functions has been removed from these devices and abstracted within a software based controller running on any server. All the functionalities of a control plane in a network device will be implemented in this controller. A layer above this controller lies application plane consisting of network applications using which the end users will manage the underlying forwarding devices. The SDN controller exposes it's north-bound interface to develop business and network applications. Different SDN controllers can have their own north-bound interfaces since there is no *de-facto* interface because of the diverse nature of applications built using this interface. These applications communicate to the hardware devices using south-bound interface exposed by the SDN controller. Unlike north-bound interface, OpenFlow protocol [41] is widely accepted as *de-facto* standard south-bound interface. In recent years, there is surge in number of vendors manufacturing network devices supporting OpenFlow protocol. In addition to this, software switches (or virtual switches) such as Open vSwitch (OVS) supports OpenFlow configuration protocols such as Open vSwitch Database (OVSDB) [44] protocol to manage it's network accessible database system. In addition to these two interfaces, East-West interface provides an interface for logically distributed control plane allowing SDN controllers to exchange notifications as well as services. This interface is still in exploratory phase in the scientific community.

2.2.1 OpenFlow Software Switch Architecture

Most commonly used OpenFlow switches such as Open vSwitch (OVS) [8] consists of one or more secure OpenFlow channels to communicate with one or more SDN controllers and a datapath. As shown in Figure 2.2, an OVS architecture consists of *flow table*, *meter table*, *group table* and *queues* on the egress port as part of the data path in the switch, where packet processing pipeline takes place. A typical packet processing pipeline in the data path of an OVS consists of (1) receiving packets on the input port (2) filtering them based on packet header fields in a *flow table* (3) executing a list of actions for matched packets including additional methods of forwarding (using *group table*), rate limiting (using *meter table*) and forwarding packets to relevant output port(s) or *dropping* the packet. Using applications running on the north-bound interface of a SDN controller, *flow*, *group*, and *meter* entries can be inserted into *flow*, *group* and *meter tables* respectively using OpenFlow protocol through the south-bound interface of the SDN controller. Using the same interface, *queues* on the egress port of an OVS can be configured using OVSDB [44] protocol.

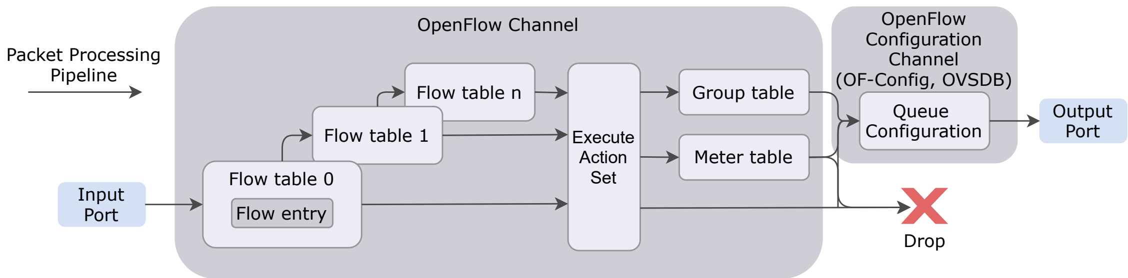

Figure 2.2 Open vSwitch (OVS) Architecture

An OVS can consist of one or more *flow tables* depending on the data pipeline design [45]. Each of these flow tables consist of one or more flow rules defining the forwarding policy for a particular flow. Table 2.1 lists the components of a flow entry. Each flow entry can be given a priority using 'Priority' field. The values for 'Priority' can range from *0* to *65535*. Within the flow table, packets will be first tried to match with the flow rule having the highest priority. If the packets do not match, then lower priority flow rules will be used to match the packet header fields until all the rules exhaust. For example, later in Chapter 3, Table 3.2 lists the *flow entries* used by the mechanism introduced in the present thesis. 'Match Fields' of the flow entry provides the filters to match the packet header attributes such as IPV4 and Ethernet addresses, input port, IP DSCP values etc., Once all the filters within this field are satisfied, a set of actions will be performed on the matched packets, as specified in the 'Instructions' field. The set of actions could

Priority	Match Fields	Cookie	Duration	Idle Timeout	Hard Timeout	Instructions	Packet Count	Byte Count

Table 2.1 Components of a *flow entry* in a flow table of an OVS

consist of forwarding the packet to other *flow table* or *group table* having further instructions of forwarding, and shaping the flow by directing the packets towards *meter table* through relevant output port or to drop the packet by specifying empty set of actions. 'Cookie' field consist of a *64* bit number used by SDN controller to add, modify and delete the flow entries. Two types of timeouts are specified for the flow entries i.e 'Idle Timeout' and 'Hard Timeout'. Timeout value (in seconds) of 'Idle Timeout' specifies how long the flow rule can exist within the switch without matching any packets. Value in 'Hard Timeout' specifies the maximum time, the flow rule can reside within the switch regardless of the number of packets matching to it. When either of the timeout expires, the switch evicts the flow rule. 'Duration' field specifies the time since the flow rule was inserted, in seconds. 'Packet Count' and 'Byte Count' field specifies the number of packets matched and the amount of bytes through the flow rule, respectively.

Group Table was introduced in OpenFlow 1.1 version to perform complex operations on packets that cannot be defined using a flow table alone. Operations such as flooding, load-balancing, sniffing and port mirroring could be performed using group table. Table 2.2 lists the components of a *group entry* in a group table of an OVS. Each entry in the group table is identified by a unique 'Group Identifier', while 'Group Type' specifies the type of group the entry belongs to. One of the four types of group i.e ALL, SELECT, INDIRECT and FAST-FAILOVER can be specified for an entry. 'Action Buckets' consist of one or more buckets specifying the set of actions to be performed on the matched packets, depending on the group type. By defining the group type to ALL, the matched packets could be duplicated and forwarded to all the buckets within the group entry to perform distinct actions on the packets. By defining the group type to SELECT, the matched packets will be directed to a single bucket among a list of buckets, in a round-robin order. The buckets could be given certain weight to select particular bucket, often. When the group type is specified as INDIRECT, only one bucket will be selected each time for the matched packets. FAST-FAILOVER group type executes the first live bucket in the group which is associated with a live egress port. 'Counter' field maintains the number of packets going through the particular group entry.

Group Identifier	Group Type	Counters	Action Buckets

Table 2.2 Components of a *group entry* in a group table of an OVS

Meter Table feature was introduced in OpenFlow protocol version 1.3 . A meter table consists of meters, defining the rate at which the flow has to be shaped. Meters are associated with the flows rather than egress ports of the switch. Flow entries in the flow table are assigned specific meters through which the packets matching the *flow rule* has to go through before being forwarded to the output ports. A flow is not required to be attached to a meter entry, it is up to the developer to specify which flows, or type of flows that should be attached to a meter entry and passed through the meters. As listed in Table 2.3 a *meter entry* in the meter table consists of two main components:

- **Meter Identifier (Meter ID)**: The Meter ID field consists of a 32-bit unsigned integer uniquely identifying the *meter entry.* The Meter IDs are used to attach to the *flow entries* in the flow table as part of the action set for the flows.

- **Meter Bands**: Meter bands specify the rate at which packets have to be shaped. The rate could be specified in terms of either kilobits per second (kbps) or packets per second (pktps) using 'Flags' field. Within this *meter band*, we can specify what action to be performed when the packets arrive at a rate higher than the specified rate of the band, by specifying the 'Type' of the band. Two kinds of actions could be performed: (i) **DROP**: By specifying the *drop* action, switch drops the packets if the current flow rate exceeds the *meter band* rate. (ii) **DSCP_REMARK**: This action is used to implement DiffServ. The DSCP value of the packets which exceeds the defined *meter band* rate will be decreased by a magnitude of '1'. For example, if the *meter band* is specified to shape the data rate at 240 kbps, and the packets from a message having DSCP value '30' arrive at 300 kbps to this band, then the DSCP value of all the packets exceeding 240 kbps will be remarked to next decreasing value i.e '28', according to the values described in RFC 2475 [13]. Even though OpenFlow protocol let us specify 'DSCP_REMARK' action for the meter bands, Open vSwitch has not yet implemented this functionality in it's latest release versioned 2.15.90 .

Meter Identifier	Bands		Flags
	Type	Rate	

Table 2.3 Components of a *meter entry* in a meter table of an OVS

2.2.2 SDN Controller

In this section we discuss and compare some of the open source SDN controllers being used in the research and industrial community. SDN controller acts as an intermediary between applications connected through it's north-bound interface and network devices connected through it's south-bound interface. The controller software is dubbed as the network operating system within the SDN concept. Table 2.4 lists some of the widely used open source SDN controllers. Controllers differ in the programming language of their implementation, they differ in support for OpenFlow protocol versions and OpenFlow configuration protocols such as OF-CONFIG and OVSDB. Some of them are in active development while others are hardly maintained. In our quest to find suitable controller which is actively maintained while supporting recent OpenFlow version and configuration protocols, we shortlisted Ryu [14] and OpenDaylight SDN controllers to be used in our QoS framework. Since we chose open source software switch i.e Open vSwitch (OVS) within our framework, supporting it's OVSDB configuration protocol became the deciding factor to choose between

Controller	Description	Language of Implementation	Active	OpenFlow version	Developers
Ryu [14]	Limited support to NETCONF and OF-CONFIG protocols but extensively supports OVSDB through OVSDB manager library	Python	Yes	1.5	NTT
POX [10]	Gained popularity in the initial days of OpenFlow research but hardly maintained nowadays. Does not support REST north-bound interface	Python	No	1.1	Independent Contributors
OpenDaylight [9]	Supports both OF-CONFIG and OVSDB protocols OpenFlow configuration protocols but does not support recent OpenFlow versions	Java	Yes	1.3	Linux Foundation Networking
Floodlight [11]	Does not support neither OVSDB nor OF-CONFIG protocols	Java	Yes	1.4	Big Switch Networks

Table 2.4 Comparison of popular open source SDN controllers

the two controllers. Even though OpenDaylight supports OVSDB protocol, it is not frequently updated with support for recent OpenFlow protocol versions. Also it does not contain separate module for QoS as in case of Ryu controller. Thus Ryu SDN controller was chosen for implementation of our QoS framework. Ryu extensively supports OVSDB protocol through a library using which REST applications could be built to configure the database of OVS. Since it was implemented in Python programming language, it facilitated quick prototyping to design our QoS framework. Although OpenFlow 1.3 version was used for communication between Ryu and OVS, which was sufficient for our use-cases, we intend to use OpenFlow 1.4 version or greater for our future work which differs the way the controller communicates with the networking devices.

2.3 Quality-of-Service (QoS)

QoS is the ability of a set of network technologies to assure the guarantee in ensuring network services to the users. The vendors of the network devices, develop QoS control mechanisms to manage these devices in a resource efficient way and provide QoS to the traffic flows through them. In traditional computer networking protocols such as IP and TCP/IP, the traffic is served on first-come-first-serve basis which is termed as *best-effort* service. Network resources are equally shared among all the traffic classes in this *best-effort* mechanism. As long as the availability of network resources is unlimited, this kind of service works. But the network devices used in Tactical Networks (TNs) are limited by capabilities in power usage, computing and storage. Serving traffic flows through *best-effort* mechanism in TNs results in unsatisfactory experience for all the users in TNs, since different traffic flows have distinct QoS requirements in terms of delay, jitter and packet loss. On one hand, critical data such as sensor data and medical help request messages are sensitive to packet delay, jitter and loss. Whereas on other hand, data from voice or video streaming application are less sensitive to packet loss. Failure to meet these standards, result in low Quality of Experience among the users.

In this regard, The Internet Engineering Task Force (IETF) defines two QoS control architecture commonly presented as Integrated Service (IntServ) [5] and Differentiated Service (DiffServ) [1]. IntServ is based on resource allocation using Resource ReSerVation Protocol (RSVP) [12]. Based on the QoS requirements of the application, IntServ computes the required bandwidth between source and destination, and allocates the network resources between them to serve the QoS on per-hop basis for this application. IntServ requires each network device along the path from source to destination, to maintain the state information for each traffic flows. Alternative to this, DiffServ control architecture is based on traffic classification where QoS mechanisms differentiate traffic flows into different classes of traffic, and assign network resources to certain class of traffic rather than allocating resources for traffic flows, each time. Based on the QoS requirements for distinct traffic classes, network resources are allocated. For example, traffic classes having low-latency requirements are served on a priority basis by the resources. DiffServ is easier to implement and network devices are not required to maintain a state of information on traffic flows. Since the devices in Tactical Networks (TNs) are constrained by the battery and memory, in IntServ model, the required memory to maintain the state information grows along with the signalling messages when the number of traffic flow increases. Also, since the links in TNs changes continuously, guaranteeing the bandwidth to serve these traffic flows becomes difficult. Therefore, DiffServ QoS control architecture is the most commonly used mechanism in addressing QoS requirements in TNs. DiffServ uses Differentiated Services Code Point (DSCP) values in the header of an IPV4 packet to distinguish packets coming from different traffic classes. In the next section, we describe how QoS requirements can be specified for packets of certain classes by encoding it's DSCP values.

2.3.1 Differentiated Services Code Point (DSCP)

Packets can be classified based on the DSCP bits in their IP header to serve them at different priority levels. DSCP is the most significant 6-bits in the 8-bit Type-of-Service (ToS) field in the IPv4 packet header. Packets with the same DSCP bits are treated equally irrespective of the traffic flow for which the packet belongs to. The ToS byte in the packet header can further be categorized into three components as shown in Table 2.5. The values in the 'Precedence' field is composed of 3-bits through which the priority level for the packets can be specified. Total eight different values (or priority levels) could be encoded using this 'Precedence' field as listed in Table 2.6. Packets with binary value '000' (Routine) have the lowest priority while packets with binary value '111' (Network) have the highest priority in DiffServ mechanism.

Precedence			**Type-of-Service**				**CU**
0	1	2	3	4	5	6	7

Table 2.5 Composition of Type-of-Service (ToS) byte in the Internet Protocol version 4 (IPv4) header [2]

Binary value	Description
000	Routine
001	Priority
010	Immediate
011	Flash
100	Flash Override
101	Critical
110	Internet
111	Network

Table 2.6 Definition for values in the Precedence field of ToS byte [2]

Binary value	Description
1000	Minimize Delay
0100	Maximize Throughput
0010	Maximize Reliability
0001	Minimize Cost
0000	Normal Service

Table 2.7 Definition for values in the ToS field of ToS byte in IPv4 header [2]

2.4 Related Works

In the following sections, we discuss some of the related studies for provisioning QoS in tactical networks. The literature review is further classified into investigations for QoS management in tactical networks and investigations describing QoS management mechanisms available in SDN. Further we summarize the contributions of these investigations in the form of a table while categorizing with respect to traffic classification, adaptivity of QoS framework, and single-hop versus multi-hop scenario.

2.4.1 QoS Management Mechanisms in SDN

The authors in [31] conducted a QoS-motivated literature review in OpenFlow enabled SDN networks characterized by network characteristics such as bandwidth, delay, jitter and loss. The authors begin with the discussion of QoS capabilities of OpenFlow protocol by looking at it's different versions. They highlight QoS related features and changes implemented from OpenFlow specification version 1.0 through version 1.5. Summarizing QoS capabilities all through these versions; OpenFlow protocol provides: (i) ability to forward packet through a queue attached to the port; (ii) ability to add, modify and remove VLAN tags along with support for multiple levels of VLAN tagging; (iii) ability for a SDN controller to query all the queues in the switch while collecting statistics about the network; (iv) rate monitoring and limiting functionality by means of meter tables consisting of meter entries; (v) flow monitoring framework that allows a controller to monitor the flow statistics in flow tables in real-time.

Furthermore, the authors elaborate relationship between SDN and QoS stressing benefits of using SDN concept in ensuring QoS. As mentioned by the article's authors, decoupling of control plane and data plane in SDN brings in many opportunities to ensure QoS such as: (i) set of flow policies and classes are unrestricted while it's limited in conventional networks because of many vendor-specific firmwares at use; (ii) through the use of SDN controller, network statistics can be monitored on different levels with respect to per-flow, per-port, per-device while overcoming conventional network's limited global view and QoS possibilities, and per-hop decision making. Due to above mentioned prominent and easier ways of ensuring QoS for applications, SDN is a better candidate when compared to conventional networks which lacks control over QoS.

SDN-capable switches, like Open vSwitch (OVS) [8], support QoS through OpenFlow [41] and Open vSwitch Database (OVSDB) [44] protocols to control the flow table, meter table and queues on the egress port. In [22], the authors proposed an algorithm to actively monitor flow statistics and dynamically re-assign flow bandwidth using meter table to achieve optimal throughput for all QoS flows. This dynamic configuration of the meter table is achieved as an extension to the controller functionalities rather than being an independent application using Representational State Transfer (REST) northbound interface of the controller. As a result, it increases the complexity of the controller and limits the enhancement of the QoS framework by independent developers. Complementing this work, we developed an external REST application to monitor traffic flows through flow table and meter table to adaptively achieve QoS requirements.

The initial effort for providing RESTful interface to ease the process of queue creation and deletion on switch ports was carried out by [43] with the support of OVSDB protocol. But the paper does not specify how to configure these queues upon creation. Overcoming this limitation, [23] proposed an interface for flexible configuration of these queues according to QoS policies. The proposed interface allowed external applications to control the configuration of these priority queues. The configuration included rate limiting and DiffServ capabilities on these queues. Our application uses this interface to configure queues on switch port with respect to traffic classes providing bandwidth guarantees.

In [26], the authors included an authentication scheme to control access for privileged users to implement QoS functions. They chose to aggregate QoS behavior based on VLAN-IDs instead of traffic types by installing flow rules to map one VPLS to a prioritized queue on OVS. They developed a QoS module utilizing both, the queues (on OVS via OVSDB) and the OpenFlow's meter table features. Since meter entries alone cannot suffice bandwidth sharing among different priority flows, the authors combined the metering with queuing to provide bandwidth reservation for flows from different VLAN-IDs. In our work, we employed similar combination of metering with queuing to ensure bandwidth guarantee for different C2 services during low data rates.

2.4.2 QoS Management in Tactical Networks using traditional networking

Prior to SDN, [32] introduced a tactical QoS framework based on Differentiated Service (DiffServ) and Simple Network Management Protocol (SNMP). The framework consists of tactical QoS information exchange, QoS module configuration and Differentiated Service Code Point (DSCP) marking taking into account hierarchical tactical network architecture. The hierarchical multi-layer architecture was designed based on a tactical structure (platoon, company, battalion, and brigade) where the leader controls and establishes QoS framework within the members of the same layer. Within the layer, members exchange QoS information with the leaders through request/response method. Once the member node receives the QoS information, it configures QoS module in it's kernel through Linux Traffic Control (tc) commands. The QoS module configuration consists of two phases i.e DSCP marking and hierarchical scheduling. In DSCP marking phase, DiffServ field of the packet will be marked to a specific DSCP value using DSCP Marking Information Base (DMB). The DSCP values are mapped to a combination of traffic types and precedence. Traffic precedence is distinguished into flash, immediate, priority and routine along with real-time and non-real-time traffic types. The DSCP marking information consists of destination IP/port, DSCP value, and egress interface. The marking takes place at the traffic source (traffic generator) and cannot be changed by intermediate nodes until the packet reaches the destination. Intermediate nodes only perform packet forwarding according to the DSCP value of the packets. In hierarchical scheduling phase, the packets will be filtered with respect to DSCP value mapping traffic type and precedence. The authors showed that the packets with higher priority, experience a shorter delay and lower jitter when compared to lower priority packets. Similarly, in our work, we use some of these ideas together with SDN capabilities to distinguish the traffic flows using DSCP values marked by a traffic generator enforcing QoS.

In [49] the author examined the effect of employing two different priority queuing disciplines, namely Fixed Priority Queue (FPQ) and Weighted Fair Queuing (WFQ) in the MAC protocol. The performance of these two queuing disciplines were tested on 40 nodes placed randomly within a quadratic area. The author concluded that by employing FPQ, the higher-priority traffic class exceeding the network capacity will dominate the links such that lower-priority traffic classes cannot pass through the network. However, in military networks, this strict priority among traffic classes makes sense because *flash messages* should always take precedence over other traffic classes. On the other hand in WFQ, no traffic class could dominate the network. Instead, the author suggests that a combination of both queues could be used by employing WFQ under FPQ. But there is no quantitative evidence supporting such suggestion in the paper. In this thesis, we introduce a hybrid priority scheduling mechanism to enforce strict priority for high priority messages and fair scheduling for low priority messages in order to address the limitation noticed in the literature review.

The investigation in [29] describe the importance of providing End-to-End (E2E) QoS over a combination of radio systems in an operation by an efficient and robust network. The authors describe QoS-aware mechanisms for inter-domain and intra-domain heterogeneous networks, where mobility can lead to reduction and/or renegotiation of QoS parameters. They mention that the QoS architecture should consist of admission control, resource monitoring and management, and the ability to preempt flows when the network is congested. While describing a possible QoS architecture for providing connectivity and differentiated QoS support in heterogeneous mobile tactical networks, the authors present a multi-topology proactive routing protocol which maintains multiple paths from source to destination to support the required QoS for a specific traffic class. The routing protocol maintains three different QoS topologies: (i) Low data-rate (ii) High data-rate and (iii) Low delay topologies depicting a group of radio transmission (VHF, UHF and SatCom) technologies. The protocol finds a route by traversing the group of radio transmission technologies that best suits the required QoS for a specific traffic class. The authors also showed that the traffic streams that cannot be supported by the path can be preempted at the source of the traffic flow. In our work, we maintained a similar multi-topology routing by using the SDN features with different flow tables in OpenFlow switches for low and high data rate QoS topologies.

A queuing mechanism for delivering QoS-constrained user data flow in tactical networks given the network conditions was developed in Fraunhofer FKIE, and described in [37, 38]. Here the authors assume that a set of QoS-constrained Command and Control (C2) services are available to the users in tactical networks where each service comes with a predefined QoS-requirement. The authors proposed a stochastic model to combine different types of messages in different ways to emulate wide range of military operations. They were particularly interested in message's *priority* and *time of expire* to shape the user data flow for a given network condition. Messages were sorted according to their priority in the queue of messages. The queuing mechanism was designed for one-hop in a VHF network. If a message's period of stay in the queue is higher than it's *time of expire* QoS-requirement, it is dropped to ensure the QoS. The results of the queuing mechanism were discussed at the receiver side showing that high priority messages had higher delivery rate and lower delay when compared with the low priority messages. In our work, we extended the shaping mechanism from one-hop to multi-hop radio networks, and we used similar stochastic models to emulate randomness in network conditions and message combinations.

2.4.3 QoS Management in Tactical Networks using SDN

Post SDN, the authors in [42] proposed the usage of SDN controller in provisioning E2E QoS in Tactical Edge Networks (TENs). They propose to categorize data flows among the nodes forming TENs into interactive (eg. audio/video streaming) and non-interactive (eg. file transfer) traffic. They suggest interactive traffic having strict requirements in terms of delay could be prioritized over non-interactive traffic using DSCP marking. And, a flow optimization application running in SDN controller could be used to filter the pre-defined packet headers corresponding to high priority traffic and select proper radio communication technology to disseminate these traffic with low delay. Despite the proposal, the article lacks practical emulation/simulation of the described traffic classification and prioritization. On the contrary, in this thesis, we demonstrate the quantitative results by emulated the usage of SDN controller in monitoring the traffic flows while adaptively ensuring QoS for these flows.

Article	Contribution	QoS Classification	Hops	Adaptive QoS	Using SDN
[32]	Introduction of a tactical QoS framework based on DiffServ and SNMP	Using DSCP marking	multi-hop	No	No
[49]	Priority scheduling of different traffic classes using FPQ and WFQ	Using DSCP marking	single-hop	No	No
[29]	Multi-topology proactive routing protocol for ensuring QoS in heterogeneous tactical networks	No Traffic classification	multi-hop	Partially adaptive	No
[37]	Emulation of randomness in network conditions and message generation using stochastic processes	By middleware	single-hop	Yes	No
[42]	Proposal to use SDN controller to categorize traffic flows and select proper radio network to disseminate these flows	DSCP marking	NA	Yes	Yes
[34]	Provisioning of QoS for real-time data NSS by computing optimal bandwidth between nodes in NSS	No Traffic classification	multi-hop	No	Yes

Table 2.8 QoS in tactical networks

In [34], authors proposed an algorithm called "Real-time Transmission via Flow-rate-Control" by leveraging SDN to provision QoS for real-time data over a tactical scenario in Naval Ship System (NSS). An optimization problem in terms of delay and priority constraints were formulated; to improve network utilization by controlling the bandwidth of the links between different nodes in NSS. Although the bandwidth is controlled by the proposed algorithm, it can't control the traffic rate generated at the source. If the traffic rate generated at the source node is higher than the allocated bandwidth by the algorithm for that flow, then the packets are dropped at the source node. The performance of the proposed algorithm was verified using mininet [33] with customized Floodlight SDN controller. Since the topology of nodes in NSS is fixed, the work does not depict the dynamic and heterogeneous nature of TENs. The authors does not elaborate on the types of traffic encountered in NSS. Also, instead of shaping the traffic at source node when it exceeds the bandwidth capacity, the packets are dropped. Our work does not focus on computing optimal bandwidth between nodes but instead of dropping we queue the packets at source node during traffic burst avoiding packet loss.

Table 2.8 summarizes the studies for provisioning QoS in tactical networks. The contribution of each article is summarized along with the method used for classification of traffic flows. Further, the contributions were categorized based on single-hop and multi-hop implementations while provisioning QoS adaptively or not, and whether using SDN paradigm or not.

3

Design and Implementation

In this chapter, we discuss the design of our SDN experimental framework using which we emulated a mechanism to adaptively ensure Quality-of-Service (QoS) of user data flow in Software Defined Heterogeneous Tactical Networks. Figure 3.1 depicts the topology of our experimental network setup within the Mininet [28] emulating platform, used throughout our experiments, to evaluate our adaptive QoS mechanism. Using Linux Network Namespaces [6], we deploy containers to emulate heterogeneous nature in TNs.

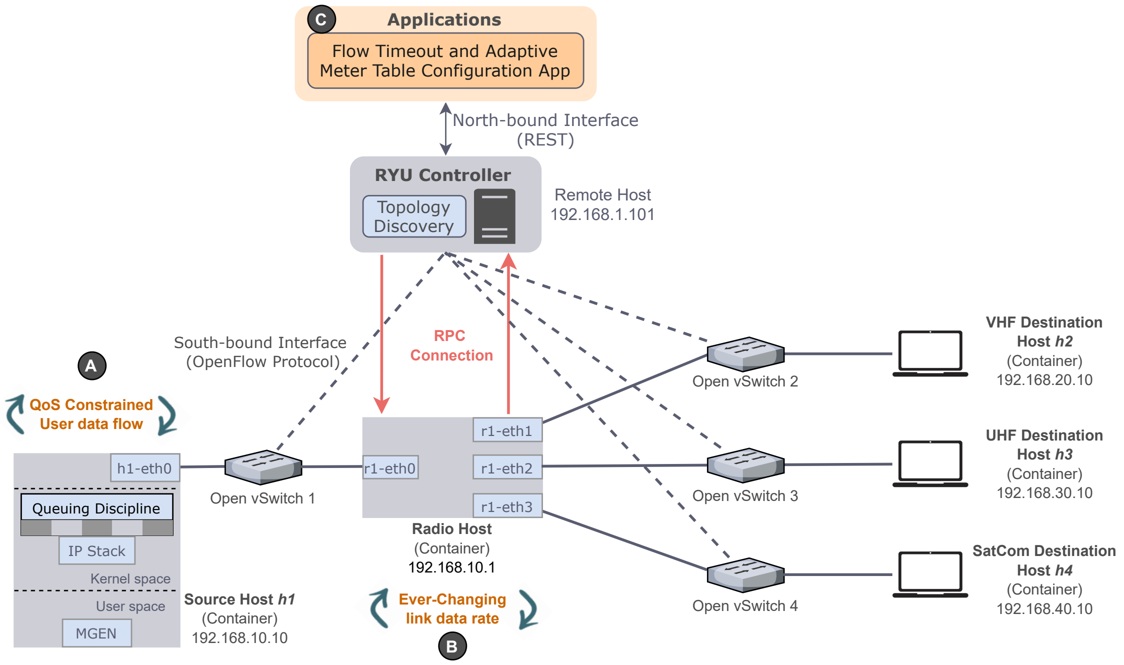

Figure 3.1 Experimental Network Setup

As shown in Figure 3.1, we deploy two containers i.e Source Host (h_1) and Radio Host (r_1) with IP addresses 192.168.10.10 and 192.168.10.1 respectively, connected by a Open vSwitch (OVS)-1 (OVS_1) and a remote host with IP address 192.168.1.101

to depict the software defined radio infrastructure at a message *Sender* radio. To emulate the message sent by this *Sender* radio (r_1) over a VHF, UHF and a SatCom link, we deploy three $OVS_{1,2,3}$ connected to three containers representing VHF, UHF and SatCom destination radio hosts i.e h_2, h_3 and h_4 with IP addresses 192.168.20.10, 192.168.30.10 and 192.168.40.10 respectively. The destination hosts are connected to their respective OVS through virtual Ethernet (veth) link. The *Sender* radio r_1, has three veth interfaces: r_1-eth_1, r_1-eth_2 and r_1-eth_3 connected to OVS_1, OVS_2 and OVS_3 bridges. Using Linux Queuing Disciplines (Qdiscs) deployed on these three virtual interfaces, we emulate VHF, UHF and SatCom radio link characteristics between *Sender* radio (r_1) and *Receiver* radios (h_2, h_3 and h_4).

Ryu [14] SDN controller instance running on the remote host 192.168.1.101, connects to OVS_1 via south-bound interface using OpenFlow protocol [41] to install *flow* and *meter* table entries to adaptively ensure QoS requirements of user data flow. We have Representational State Transfer (REST) applications running on the north-bound interface of the SDN controller to support adaptive installation of these entries. Ryu was chosen as part of our network set up due to the reasons as described in section 2.2.2. The *Topology Discovery* module running on the Ryu controller, discovers the deployed topology in Mininet and inserts the MAC address table entries in the corresponding *flow tables* of OVS_1.

A *Communication Scenario* in TN can be described as a combination of user behaviour Ⓐ and network conditions Ⓑ (refer Figure 3.1), which are changing independently over time. To describe and emulate the challenges of addressing QoS requirements of user data flow over ever-changing communication scenarios in heterogeneous tactical networks, we further divide this chapter into five sections as follows. In Section 3.1, we begin with describing the importance of ensuring QoS requirements of user data flow while listing different types of command and control services available to users in TNs. We also describe a mechanism to emulate message generation from these distinct services using a traffic generator. In Section 3.2, we describe the heterogeneous nature of radio communication in TNs. We considered to emulate data rates supported by three kinds of radio communication techniques i.e VHF, UHF and SatCom in our emulation platform. In Section 3.3, we describe some of the traffic shaping and scheduling mechanisms available within Linux environment. We list some of the Queuing Disciplines (Qdiscs) available within Linux kernel to implement traffic management capabilities. In Section 3.4, we describe the features available within OVS to implement QoS requirements of user data flow. We conclude this chapter by describing our adaptive QoS mechanism Ⓒ (refer Figure 3.1) using REST applications running on Ryu controller together with OVS to adaptively ensure QoS requirements of user data flow within our emulating platform.

3.1 QoS constrained user data flow

Ensuring Quality-of-Service (QoS) of data in heterogeneous TNs is important so that the exchange of user generated data can be utilized to a maximum extent. Often, users of TN generate more data than the network can handle resulting in network congestion. So packets with higher priority should be served first than the lower priority ones. This may result in bandwidth starvation for low priority traffic classes

Type of Service	Priority	Reliability	Time-of-Expiry ToE (sec)
m_0 Medical evacuation (0x78)	0 Flash	Yes	300
m_1 Obstacle alert (0x50)	1 Immediate	Yes	150
m_2 Picture (0x28)	2 Priority	Yes	3600
m_3 Chat (0x04)	3 Routine	No	120
m_4 FFT (0x00)	3 Routine	No	120

Table 3.1 Message priority and time of expiry [37]

```
1) Medical Evacuation:  <start-time> ON <flowd-id> UDP <SRC-PORT> <DST-IP/PORT> <PATTERN> TOS 0x78
2) Obstacle Alert:      <start-time> ON <flowd-id> UDP <SRC-PORT> <DST-IP/PORT> <PATTERN> TOS 0x50
3) Picture:             <start-time> ON <flowd-id> UDP <SRC-PORT> <DST-IP/PORT> <PATTERN> TOS 0x28
4) Chat:                <start-time> ON <flowd-id> UDP <SRC-PORT> <DST-IP/PORT> <PATTERN> TOS 0x04
5) FFT:                 <start-time> ON <flowd-id> UDP <SRC-PORT> <DST-IP/PORT> <PATTERN> TOS 0x00
```

Figure 3.2 MGEN script to generate messages listed in Table 3.1

or even they can be preempted. Due to the delay experienced by these low priority traffic flows, they may not be relevant to the recipient by the time they were received. In these cases, the expired data should be discarded or dropped. Different priorities can be assigned to different traffic classes based on the type of military operation. The network should be adaptable in supporting differential treatment to these traffic classes. Table 3.1 lists five exemplary command and control services available to the users in tactical networks. Each of these services has unique QoS-requirements in terms of message priority, reliability and Time-of-Expiry (ToE).

To simulate message generation from these distinct services within Mininet, we use Multi-Generator (MGEN) [7] traffic generator developed by US Naval Research Laboratory since it is based on generating traffic patterns for a tactical scenario. The MGEN tool provides the ability to generate messages with different priority in form of Type-of-Service (ToS) bits set in the header of an IPv4 packet. MGEN is set to use UDP as transport protocol using the commands as shown in Figure 3.2. Line_1 in Figure 3.2 represents the command to generate *Message Evacuation* message with ToS bits set to '0x78' in hexadecimal format. Similarly Line_2 to Line_5 represents generation of *Obstacle Alert* (0x50), *Picture* (0x28), *Chat* (0x04) and *FFT* (0x00) messages with respect to their 'Priority' QoS requirement (refer Table. 3.1). Each of these messages can be generated at a specific time since the execution of the script through distinct source port, to a destination IP address and port combination. The "Pattern" of message generation can be specified in "ON" events by specifying the message size and frequency of transmission.

In our experimental setup as shown in Figure 3.1, using an MGEN instance running on source host h_1, we generate a burst of these five messages with their corresponding "Priority". These messages were sent through VHF, UHF and SatCom link to their corresponding destination hosts h_2, h_3 and h_4, simultaneously. An MGEN instance was running on these destination hosts to receive and log the messages in terms of time-stamp (sent and received), packet sequence number and it's size. Using these features we computed packet loss and delay to quantify the system's ability to differentiate messages with distinct QoS requirements.

3.2 Ever-changing link data rates

It is a challenging task to meet all the requirements of a radio communication among the nodes in a Tactical Network (TN) considering the node mobility, constrained resources and unreliable communication links between them. The movement of these nodes in the battlefield scenarios or disaster relief is extremely dynamic. The communication environment in these scenarios is usually infrastructure-less and thus these nodes have to be connected through wireless communication network. Further, based on the degree of mobility, these nodes can be categorized into static (like sensors), dynamic (like convoys, platoons etc.,) and into extremely dynamic (like UAVs). Different radio communication systems are used such as HF, VHF, UHF and SatCom to connect mobile units and command posts together to meet all the requirements of a tactical radio communication. These radio communication systems have different characteristics in terms capacity, range, anti-jamming capabilities and robustness [21]. Figure 3.3 illustrates three types of radio communication techniques emulated in our experimental setup using Queuing Disciplines (Qdiscs). Very High Frequency (VHF) radios such as PR4G by Thales supports a maximum data rate of 9.6 kilobits per second (kbps) over a range of 20 kilometers (kms). Depending on the distance between these radios, they support a data rate of (0.6, 1.2, 2.4, 4.8 and 9.6) kbps. Similarly depending on the vendor of the radio, Ultra High Frequency (UHF) radios support a maximum data rate of 240 kbps over a range of 2 kms. And we assume data rates supported by these UHF radios to be (15, 30, 60, 120 and 240) kbps. Also we assume Satellite Communications (SatCom) to support a data rate of (32, 64, 128, 256 and 512) kbps in our experiments in the emulating platform.

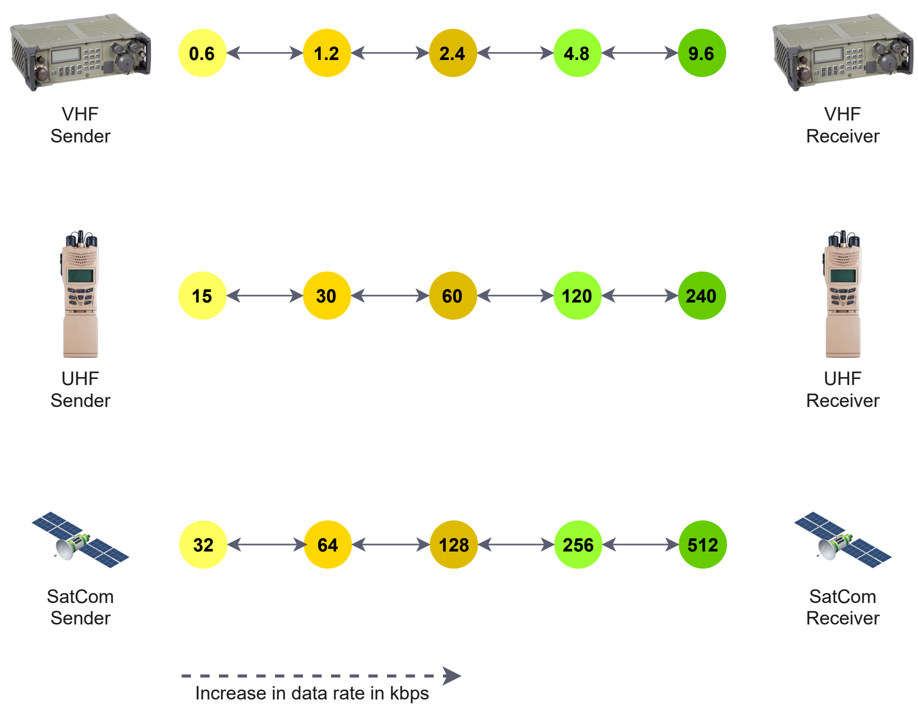

Figure 3.3 Exemplary data rates supported by VHF, UHF and SatCom waveforms

We considered to simulate these different data rates of different radio communication systems in a heterogeneous node to emulate dynamic network conditions experienced by nodes in a tactical scenario. The sources of varying network conditions could be due to (i) frequent topology changes because of node mobility. (ii) intermittent communication links due to terrain masking and (iii) due to unpredictable operational conditions. In our experimental network setup as shown in Figure 3.1, we attach Qdiscs on interfaces: r_1-eth_1, r_1-eth_2 and r_1-eth_3, representing a radio with heterogeneous communication setup to shape the data rate with respect to the data rates supported by VHF, UHF and SatCom, as mentioned before.

3.3 The QoS Mechanisms in Linux Kernel

This Section describes a range of Quality-of-Service (QoS) mechanisms available within Linux environment. Queuing Disciplines (Qdiscs) implement the traffic management capabilities within Linux Kernel. The Qdisc is located between the IP Stack and the network interface, which can be configured using user space utility program called Traffic Control (TC). Based on the Qdisc being used, TC lets us configure the rate at which packets are to be transmitted through the network interface along with preferential treatment to certain packets to move ahead than the rest, while being transmitted. Preferential treatment is achieved by classifying packets coming from different traffic classes and scheduling them based on their priorities assuming that the packets have already been marked with their respective ToS bits in the packet header. Rate limiting and scheduling of packets are always carried out at the outbound interface while having no control over how packets can be treated at the inbound interface, except policing them. In the following sub-sections, we describe various Qdiscs supported by the Linux Kernel to shape the traffic with respect to the data rates supported by different radio communication systems as described before and to schedule the traffic from distinct command and control services (refer Table 3.1) according to their "Priority" QoS requirement.

3.3.1 Traffic shaping

Traffic shaping is a mechanism in which the amount of data sent over a network could be controlled, typically implemented as an algorithm in the kernel networking stack. In our experimental network setup, we refer *traffic shaping* as a combination of *pacing* and *rate limiting*. *Pacing* refers to injection of inter-packet gaps to smooth the traffic considering the network bandwidth, to avoid packet loss when the amount of data generated exceeds the network capacity. *Rate limiting* refers to enforcement of data rate on flow-aggregate basis. In our experiments, we perform *pacing* and *rate limiting* on the source host h_1 and Radio Host r_1 (refer Figure 3.1), considering the individual and aggregate of bandwidth available over VHF, UHF and SatCom links. In the following sub-sections we discuss the two most widely used algorithms for traffic shaping and their implementation within our network setup.

3.3.1.1 Token Bucket Filter (TBF)

Traffic shaping algorithm such as Token Bucket Filter (TBF) Qdisc [4], buffers the packets in a queue while waiting for tokens to pass through the queue of the network interface. These tokens can be generated at a specific rate, defining the rate at which

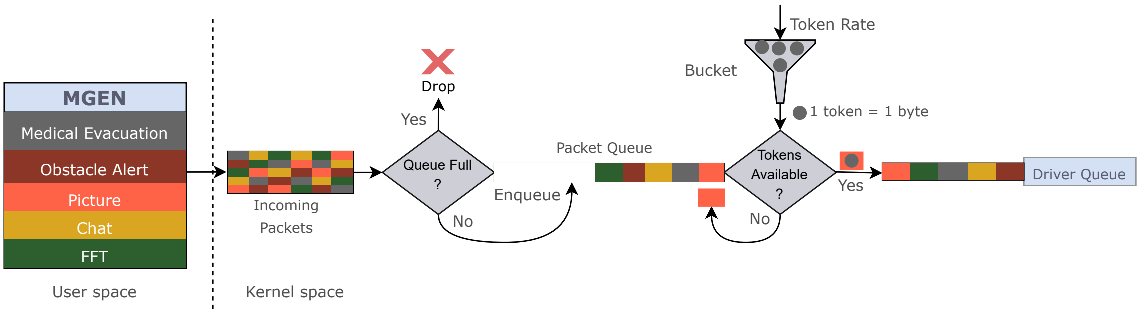

Figure 3.4 Structure of Token Bucket Filter (TBF) Qdisc

packets have to be transmitted. Figure 3.4 describes the structure of a TBF Qdisc. As the name suggest, the Qdisc consist of a bucket, constantly filled with some virtual pieces of information called *'tokens'* at a specific rate called *token rate*.

The messages generated by MGEN in the *user space* are broken down into packets in the *kernel space* within the Linux environment. The incoming packets will be enqueued to the packet queue if it has sufficient space to admit the packet else the packet will be dropped. While dequeuing the packets from packet queue, the token bucket is inspected to check if it contains sufficient tokens with respect to the size of the packet in byte, where 1 token equals to 1 byte. If sufficient tokens are available, the packet is allowed to pass through, to driver queue at the outbound interface, else the packet has to wait until appropriate amount of tokens are available. Here we can specify the *token rate* to match the data rate at which traffic has to be shaped corresponding to the radio modulation in use.

3.3.1.2 Hierarchy Token Bucket (HTB)

HTB Qdisc uses a tree organization (refer Figure 3.5) of class-based hierarchical system and filters to shape the data flow. HTB uses filters based on packet header attributes to classify incoming packets into several classes, each associated with a TBF Qdisc at leaf level in the tree. These filters can be assigned to a class or a Qdisc or both based on the design of the tree. Figure 3.5 describe the structure of HTB Qdisc, used throughout our experiments which was attached to the outbound interface h_1-eth_0 of source host h_1 (refer Figure 3.1) and was used to shape the data flow destined towards a VHF (h_2), UHF (h_3) and a SatCom (h_4) node. Figure 3.6 lists a set of commands to setup this HTB Qdisc structure at the egress *interface* of host, h_1.

The IP packets coming from the user space MGEN traffic generator with appropriate ToS byte as listed in Figure 3.2 are enqueued to the root HTB Qdisc **(1:)** deployed in the kernel space of a Linux environment running on host, h_1. Line_1 in Figure 3.6 describe the command to setup this root Qdisc on egress interface h_1-eth_0. Under this root, lies another HTB Qdisc **(1:1)** which classifies the packets based on the destination IP address. Line_2 describe the command to setup this inner Qdisc under the root. All the packets passing through this root Qdisc are enqueued to the inner Qdisc with the help of a filter (refer Line_6 in Figure 3.6). The rate at which packets have to be dequeued from this inner Qdisc is configured according to the traffic aggregation of leaf Qdiscs. Under this inner Qdisc, three leaf HTB Qdiscs i.e

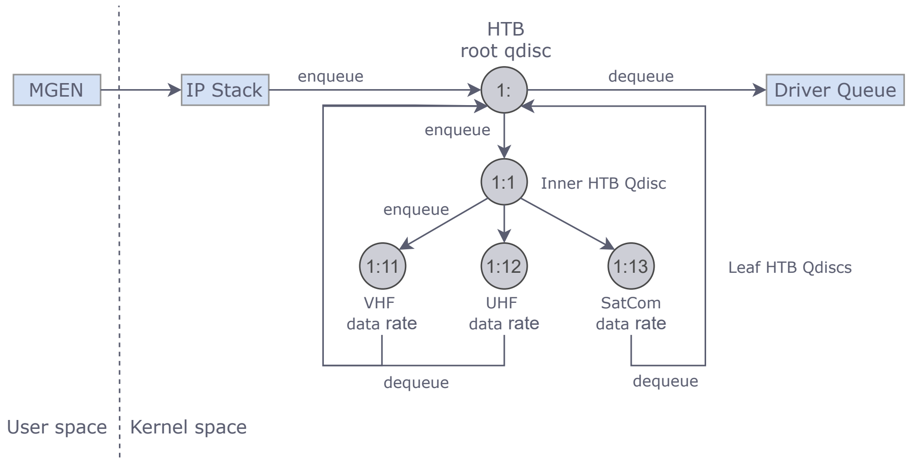

Figure 3.5 Structure of Hierarchy Token Bucket (HTB) Qdisc deployed on the egress interface of Source Host (192.168.10.10)

```
1) tc qdisc add dev <interface> root handle 1: htb
2) tc class add dev <interface> parent 1: classid 1:1 htb rate 762kbit ceil 762kbit burst 762kb
3) tc class add dev <interface> parent 1:1 classid 1:11 htb rate 9.6kbit ceil 9.6kbit burst 10kb
4) tc class add dev <interface> parent 1:1 classid 1:12 htb rate 240kbit ceil 240kbit burst 240kb
5) tc class add dev <interface> parent 1:1 classid 1:13 htb rate 512kbit ceil 512kbit burst 512kb
6) tc filter add dev <interface> parent 1:0 protocol ip prio 1 u32 match ip src <address>
6) match ip protocol 17 0xff flowid 1:1
7) tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <VHF-DST-address>
7) match ip protocol 17 0xff flowid 1:11
8) tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <UHF-DST-address>
8) match ip protocol 17 0xff flowid 1:12
9) tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <SatCom-DST-address>
9) match ip protocol 17 0xff flowid 1:13
```

Figure 3.6 Linux Traffic Control (TC) commands to add HTB Qdisc on the egress interface of Source Host (192.168.10.10)

(1:11), **(1:12)** and **(1:13)** are attached to shape the data rate according to the data rates (refer Figure 3.3) supported by a VHF, UHF and SatCom link respectively. Inner Qdisc **(1:1)** uses a filter to classify the packets destined towards a VHF (h_2), UHF (h_3) and SatCom (h_4) host and enqueues corresponding packets to leaf Qdiscs **(1:11)**, **(1:12)** and **(1:13)** respectively. In $\text{Line}_{3,4,5}$ we specify *ceil* rate and *burst size* to define the maximum data rate supported by a VHF, UHF and SatCom radio links respectively.

To emulate ever-changing link data rates at a heterogeneous node, we attached HTB Qdiscs on r_1-eth_1, r_1-eth_2 and r_1-eth_3 interfaces of Radio Host (192.168.10.1) connected to VHF, UHF and SatCom destination hosts, respectively. $\text{Line}_{1,2,3}$ of Figure 3.7 lists the Linux TC commands to setup a HTB Qdisc on r_1-eth_1 interface to emulate the data rates supported by a VHF radio as described in Section 3.2. Similarly, $\text{Line}_{4,5,6}$ and $\text{Line}_{7,8,9}$ describe the commands to setup HTB Qdiscs on r_1-eth_2 and r_1-eth_3 interfaces to emulate data rates supported by UHF radio and SatCom respectively. During initial deployment of our network setup within Mininet, all these three interfaces were configured to shape the data rate with respect to the maximum rate supported by the corresponding radio link to the destination host.

$\text{Line}_{1,2,3}$ of Figure 3.8 describe the commands to change the link data rate on interfaces r_1-eth_1, r_1-eth_2 and r_1-eth_3 respectively. Using these commands, we emulate

```
1) tc qdisc add dev r1-eth1 root handle 1: htb default 1
2) tc class add dev r1-eth1 parent 1: classid 1:1 htb rate 9.6kbit ceil 9.6kbit burst 10kb
3) tc filter add dev r1-eth1 parent 1: protocol ip prio 1 u32 match ip dst 192.168.20.10
   match ip protocol 17 0xff flowid 1:1

4) tc qdisc add dev r1-eth2 root handle 1: htb default 1
5) tc class add dev r1-eth2 parent 1: classid 1:1 htb rate 240kbit ceil 240kbit burst 240kb
6) tc filter add dev r1-eth2 parent 1: protocol ip prio 1 u32 match ip dst 192.168.30.10
   match ip protocol 17 0xff flowid 1:1

7) tc qdisc add dev r1-eth3 root handle 1: htb default 1
8) tc class add dev r1-eth3 parent 1: classid 1:1 htb rate 512kbit ceil 512kbit burst 512kb
9) tc filter add dev r1-eth3 parent 1: protocol ip prio 1 u32 match ip dst 192.168.40.10
   match ip protocol 17 0xff flowid 1:1
```

Figure 3.7 Linux TC commands to add HTB Qdiscs on the interfaces of Radio Host (192.168.10.1)

```
1) tc class change dev r1-eth1 parent 1: classid 1:1 htb rate <VHF-data-rate>kbit
2) tc class change dev r1-eth2 parent 1: classid 1:1 htb rate <UHF-data-rate>kbit
3) tc class change dev r1-eth3 parent 1: classid 1:1 htb rate <SatCom-data-rate>kbit
```

Figure 3.8 Linux TC commands to change the link data rates on the interfaces of Radio Host (192.168.10.1)

dynamic network conditions experienced by a heterogeneous node in a tactical scenario as described in Section 3.2 and test our adaptive QoS mechanism to ensure QoS requirements of user data flow, depending on the network condition.

3.3.2 Traffic scheduling

Traffic scheduling refers to the preferential treatment of certain packets to move ahead than the rest while being transmitted though the egress interface of the host. The Linux Kernel supports various Qdiscs implementing Differentiated Service (DiffServ) for traffic classes. These Qdiscs prioritize certain packets by decreasing the time that these packets have to wait in the queue during transmission. Qdisc implementations for scheduling the traffic can further be classified into *priority* and *fairness* based scheduling mechanisms for traffic classes. A recent Qdisc implementation, merges the strict *priority* and *fairness* based scheduling mechanism into a hybrid scheduling mechanism. In the following sub-sections, we describe a Qdisc implementation available for all these three scheduling mechanisms. In our experimental network setup (refer Figure 3.1), we deploy these Qdiscs at the egress interface: h_1-eth_0 of host, h_1 to schedule the packets once it is shaped with respect to the available network bandwidth, thereby ensuring "Priority" QoS requirements of the user data flow.

3.3.2.1 Priority First-In-First-Out (PFIFO)

The PFIFO Qdisc is a class of *priority* based scheduling mechanism. This Qdisc is the basis for providing a relatively simple method of supporting DiffServ for traffic classes. Figure 3.9 describes the structure of PFIFO Qdisc, where a band (or queue)

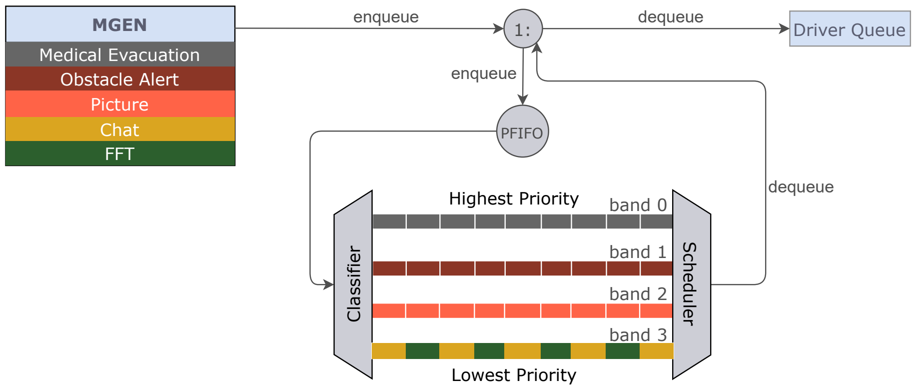

Figure 3.9 Structure of Priority First-In-First-Out (PFIFO) Qdisc

is created for each *priority* based traffic classes. It consists of a *classifier* which classifies the incoming packets based on the ToS bits set in the packet header and places them onto corresponding *priority* bands. It also consists of a *scheduler* which dequeues the packets from these *priority* bands in FIFO order. As described in the Figure 3.9, the PFIFO Qdisc structure in our experimental setup, consists of four $bands_{0,1,2,3}$ inorder to enforce "Priority" QoS requirements listed in Table 3.1 for the distinct command and control services. The packets generated from these services were classified and enqueued to corresponding bands using *classifier*. Packets from *Medical Evacuation* message with ToS bit '0x78' were enqueued to $band_0$. Similarly, packets from *Obstacle Alert* (0x50) were enqueued to $band_1$, packets from *Picture* (0x28) to $band_2$ and packets from *Chat* (0x04) and *FFT* (0x00) to $band_3$. Within these strict *priority* bands, packets in $band_0$ has the highest priority and packets in $band_3$ has the lowest priority while scheduling the traffic. Packets from lower priority *bands* will only be dequeued once the higher priority *bands* become empty.

In our network setup, we attached three PFIFO Qdiscs referred by handles **(11:)**, **(12:)** and **(13:)** under three Qdiscs **(1:11)**, **(1:12)** and **(1:13)** respectively to schedule the traffic, once it is shaped with respect to VHF, UHF and SatCom link capacity. $Line_6$ to $Line_{20}$ in Figure 3.10 describe the commands to setup these three PFIFO Qdiscs. $Line_6$ describe the command to setup a PFIFO Qdisc by handle **(11:)** under HTB Qdisc **(1:11)** with four $bands_{0,1,2,3}$ while mapping the packets to the corresponding *bands*. Mapping occurs based on the ToS octet of the packet with 16 different possible octet values. By defining the mapping sequence to '3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0', we specify the classifier to map lowest priority packets from *Chat* and *FFT* message to $band_3$, packets from *Picture* to $band_2$, *Obstacle Alert* to $band_1$, and packets from *Medical Evacuation* message including packets from other higher priority messages to $band_0$. In each of these four *bands*, a netem Qdisc was added to include 5 milliseconds (ms) delay to accommodate the classification and scheduling phases. Without this injection of minimum delay, we experienced packet loss in either of the two phases. $Line_9$ to $Line_{12}$ describe the command to setup four netem Qdiscs **(111:)**, **(112:)**, **(113:)** and **(114:)** with $bands_{0,1,2,3}$ respectively. Similarly, $Line_7$ and $Line_8$ describe the commands to include PFIFO Qdiscs **(12:)** and **(13:)** under HTB Qdiscs **(1:12)** and **(1:13)** respectively, along with four netem Qdiscs ($Line_{13}$ to $Line_{20}$) for each of these PFIFO Qdiscs.

```
tc qdisc add dev <interface> root handle 1: htb
tc class add dev <interface> parent 1: classid 1:1 htb rate 762kbit ceil 762kbit burst 762kb
tc class add dev <interface> parent 1:1 classid 1:11 htb rate 9.6kbit ceil 9.6kbit burst 10kb
tc class add dev <interface> parent 1:1 classid 1:12 htb rate 240kbit ceil 240kbit burst 240kb
tc class add dev <interface> parent 1:1 classid 1:13 htb rate 512kbit ceil 512kbit burst 512kb
tc qdisc add dev <interface> parent 1:11 handle 11: prio bands 4 priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0
tc qdisc add dev <interface> parent 1:12 handle 12: prio bands 4 priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0
tc qdisc add dev <interface> parent 1:13 handle 13: prio bands 4 priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0
tc qdisc add dev <interface> parent 11:1 handle 111: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 11:2 handle 112: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 11:3 handle 113: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 11:4 handle 114: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 12:1 handle 121: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 12:2 handle 122: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 12:3 handle 123: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 12:4 handle 124: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 13:1 handle 131: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 13:2 handle 132: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 13:3 handle 133: netem limit 1000 delay 5ms
tc qdisc add dev <interface> parent 13:4 handle 134: netem limit 1000 delay 5ms
tc filter add dev <interface> parent 1:0 protocol ip prio 1 u32 match ip src <address>
match ip protocol 17 0xff flowid 1:1
tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <VHF-DST-address>
match ip protocol 17 0xff flowid 1:11
tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <UHF-DST-address>
match ip protocol 17 0xff flowid 1:12
tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <SatCom-DST-address>
match ip protocol 17 0xff flowid 1:13
```

Figure 3.10 Linux Traffic Control (TC) PFIFO Qdisc configuration commands

3.3.2.2 Enhanced Transmission Selection (ETS)

To ensure each traffic class has fair access to the network resources, *fairness* based Qdiscs such as Stochastic Fair Queuing, Weighted Fair Queuing and Deficit Round Robin (DRR) exist. These Qdiscs serve flows from traffic classes by assigning certain percentage of bandwidth in round-robin order. The primary benefit of employing these *fairness* based Qdiscs is that the bursty higher-priority traffic flows do not degrade the QoS of lower-priority traffic flows. However, employing these *fairness* based Qdiscs makes sense when none of the traffic classes should be allowed to dominate the low-bandwidth network link. But in tactical networks, high-priority traffic flows such as *Medical Evacuation* and *Obstacle Alert* should always take precedence over other traffic flows. So in our quest to find the right balance between priority and fair scheduling of traffic classes, we found Enhanced Transmission Selection Scheduler (ETS) Qdisc that merges the functionality of PFIFO and DRR Qdiscs in one Qdisc. ETS is a new Qdisc added to the recent Linux kernel 5.8 version [3]. ETS allows us to create bands (or queues) for strict *priority* as well as *fairness* based scheduling. The number of bands for both the category of scheduling can be configured. Packets from fair-sharing bands are only dequeued if all the bands reserved for strict priority is empty. Further, among fair-sharing bands the amount of bytes a band is allowed to dequeue in one round-robin can be configured.

We conducted experiments by replacing PFIFO Qdiscs under the leaf HTB Qdiscs by three ETS Qdiscs **(11:)**, **(12:)** and **(13:)** to schedule *Medical Evacuation* and *Obstacle Alert* messages in strict *priority* order. Remaining messages were scheduled through *fairness* based mechanism. In our work, we term this mixture of scheduling mechanisms as a *hybrid scheduling* mechanism. Figure 3.11 describes the structure of ETS Qdisc placed under three HTB Qdiscs **(1:11)**, **(1:12)** and **(1:13)** to schedule

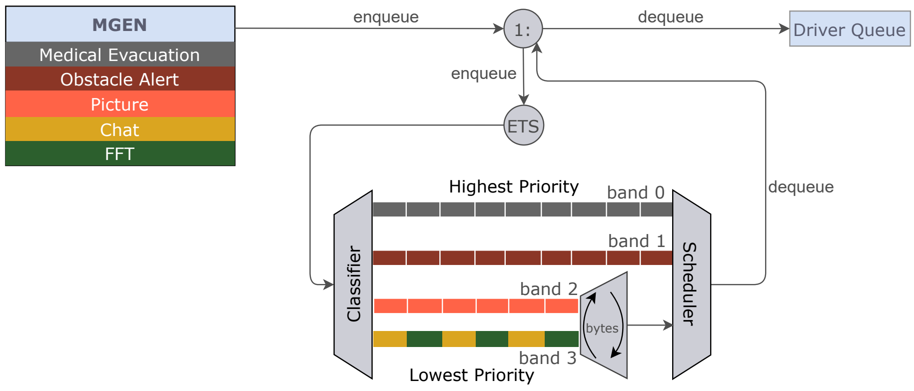

Figure 3.11 Structure of Enhanced Transmission Selection (ETS) Qdisc

```
1)  tc qdisc add dev <interface> root handle 1: htb
2)  tc class add dev <interface> parent 1: classid 1:1 htb rate 762kbit ceil 762kbit burst 762kb
3)  tc class add dev <interface> parent 1:1 classid 1:11 htb rate 9.6kbit ceil 9.6kbit burst 10kb
4)  tc class add dev <interface> parent 1:1 classid 1:12 htb rate 240kbit ceil 240kbit burst 240kb
5)  tc class add dev <interface> parent 1:1 classid 1:13 htb rate 512kbit ceil 512kbit burst 512kb
6)  tc qdisc add dev <interface> parent 1:11 handle 11: ets strict 2 quanta 900 600 priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0
7)  tc qdisc add dev <interface> parent 1:12 handle 12: ets strict 2 quanta 900 600 priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0
8)  tc qdisc add dev <interface> parent 1:13 handle 13: ets strict 2 quanta 900 600 priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0
9)  tc qdisc add dev <interface> parent 11:1 handle 111: netem limit 1000 delay 5ms
10) tc qdisc add dev <interface> parent 11:2 handle 112: netem limit 1000 delay 5ms
11) tc qdisc add dev <interface> parent 11:3 handle 113: netem limit 1000 delay 5ms
12) tc qdisc add dev <interface> parent 11:4 handle 114: netem limit 1000 delay 5ms
13) tc qdisc add dev <interface> parent 12:1 handle 121: netem limit 1000 delay 5ms
14) tc qdisc add dev <interface> parent 12:2 handle 122: netem limit 1000 delay 5ms
15) tc qdisc add dev <interface> parent 12:3 handle 123: netem limit 1000 delay 5ms
16) tc qdisc add dev <interface> parent 12:4 handle 124: netem limit 1000 delay 5ms
17) tc qdisc add dev <interface> parent 13:1 handle 131: netem limit 1000 delay 5ms
18) tc qdisc add dev <interface> parent 13:2 handle 132: netem limit 1000 delay 5ms
19) tc qdisc add dev <interface> parent 13:3 handle 133: netem limit 1000 delay 5ms
20) tc qdisc add dev <interface> parent 13:4 handle 134: netem limit 1000 delay 5ms
21) tc filter add dev <interface> parent 1:0 protocol ip prio 1 u32 match ip src <address>
22) match ip protocol 17 0xff flowid 1:1
23) tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <VHF-DST-address>
24) match ip protocol 17 0xff flowid 1:11
25) tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <UHF-DST-address>
26) match ip protocol 17 0xff flowid 1:12
27) tc filter add dev <interface> parent 1:1 protocol ip prio 1 u32 match ip dst <SatCom-DST-address>
28) match ip protocol 17 0xff flowid 1:13
```

Figure 3.12 Linux Traffic Control (TC) PFIFO Qdisc configuration commands

the traffic flow, once it is shaped according to the radio communication technology (refer Figure 3.5). Line_6 to Line_{20} in Figure 3.12 describe the commands to setup three ETS Qdiscs under these three HTB Qdiscs. Two bands ($band_0$ and $band_1$) were configured for strict *priority* scheduling while $band_2$ and $band_3$ were configured for *fairness* based scheduling. 900 bytes were dequeued from $band_2$ in one-round robin while 600 bytes were dequeued from $band_3$. Configuration commands 'ets strict 2 quanta 900 600' in $\text{Line}_{6,7,8}$ in Figure 3.12 describe this *hybrid* scheduling setup. Similar to the mapping sequence in case of PFIFO Qdisc, in ETS Qdisc we map packets coming from *Medical Evacuation* and *Obstacle Alert* to strict *priority* bands $band_0$ and $band_1$ respectively. Packets from *Picture* to *fairness* based $band_2$ and packets from *Chat* and *FFT* to $band_3$. Command 'priomap 3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0' in $\text{Line}_{6,7,8}$ describes this mapping setup. In each of these four *bands*, a netem Qdisc was added to include 5 milliseconds (ms) inter-packet delay to avoid packet loss during classification and scheduling phases, similar to the setup in PFIFO Qdisc. $Line_9$ to $Line_{20}$ in Figure 3.12 describe commands to inject this delay.

3.4 QoS Implementation using Open vSwitch (OVS)

Considering the architecture of OVS, as described in the Section 2.2.1 of chapter 2, we make use of *flow table* and *meter table* features for our adaptive QoS mechanism. Using north-bound REST interface of Ryu SDN controller, flow and meter entries can be inserted into *flow* and *meter table* respectively using OpenFlow protocol. Table 3.2 lists the *flow entries* inserted into *flow tables* of OVS_1 in our network setup (refer Figure 3.1). In the *flow table*$_0$ of OVS_1 in our network setup as listed in Table 3.2, we have defined two kinds of flow rules: one with the highest priority (65535) to match Link Layer Discovery Protocol (LLDP) packets and the other with *Priority* '1' to match IP packets. The matched LLDP packets were sent to the Ryu controller running on the remote host '192.168.1.101' (refer Figure 3.1). A *Topology Discovery* module running on the Ryu controller, on receiving these LLDP packets from OVS_1, discovers the network topology and inserts MAC address table entries in *flow table*$_1$. We defined *15* flow entries with *Priority* '1' in *flow table*$_0$ of OVS_1 to match IP packets coming from the source host '192.168.10.10' destined towards VHF (192.168.20.10), UHF (192.168.30.10) and SatCom (192.168.40.10) destination hosts as shown in Figure 3.1. Within these *15* flow rules, *3* sets consisting of *5* rules each were used to match IP packets with distinct DSCP values for a combination of source and destination IP addresses. Packets coming from *Medical Evacuation* message were matched using DSCP value *30*. Similarly, packets from *Obstacle Alert*, *Picture*, *Chat* and *FFT* were matched using DSCP values *20*, *10*, *1* and *0* respectively. DSCP values were used in the *Match Fields* to ensure Time-of-Expiry (ToE) QoS requirement of these messages using REST applications running on the Ryu controller. Packets that were matched with respect to source and destination addresses, were assigned a *meter id* from the *meter table* of OVS_1 to shape the data flow corresponding to VHF, UHF and SatCom link bandwidth. All the packets that were matched with flow rule *Priority* '1' was forwarded to *flow table*$_1$ on OVS_1, having further instructions in the form of MAC address table entries to send it to the destination host.

Flow table	Priority	Match Fields	Instructions	Timeouts
0	65535	<eth_type: LLDP>	<OUTPUT: CONTROLLER>	0
	1	<eth_type: IP> <ip_src>, <ip_dst> <ip_dscp>	<GOTO_TABLE: 1> <METER: id> <SET_QUEUE: id>	0
1	1	<in_port> <eth_dst: MAC addr>	<OUTPUT: port>	0

Table 3.2 Flow entries in Flow tables

In Table 3.3, we list a set of rate limiting entries inserted into the *meter table* of OVS_1 corresponding to the data rates supported by a SatCom link (512, 256, 128, 64 and 32 kbps), a UHF radio (240, 120, 60, 30 and 15 kbps) and a VHF radio (9.6, 4.8, 2.4, 1.2 and 0.6 kbps) as described in the Section 3.2. Entries for the latter were rounded off due to *meter table* not supporting neither decimal nor bits per second entries. In our adaptive QoS mechanism we use these entries in the *meter table* to adaptively attach these entries with unique meter IDs to the *flow entries* of *flow table*$_0$ as defined in Table 3.2. We specified the *action* for each of these meter bands to 'DROP', since OVS do not support 'DSCP_REMARK' *action* for meter bands as mentioned in the Section 2.2.1 of chapter 2 while describing the components of a meter table.

Meter ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Rate in kbps	512	256	128	64	32	240	120	60	30	15	10	5	3	2	1

Table 3.3 Meter entries in Meter table

3.5 The Adaptive QoS Mechanism

In this Section, we describe the mechanism to adaptively ensure QoS requirements of the user data flow through REST applications running on the north-bound interface of the remote Ryu controller (C) as described in Figure 3.1. First, we describe the commands to setup *meter table* and *flow table* of OVS_1 with initial configurations. Later, we describe the algorithms to manipulate these configurations of *meter table* and *flow table*, thus ensuring QoS requirements of user data flow, adaptively.

Ryu SDN controller supports retrieving and updating configurations of OpenFlow switches through the use of applications by exposing REST APIs. 'ryu.app.ofctl_rest' is one such application providing REST APIs to configure OpenFlow switches such as Open vSwitch (OVS). Once Ryu controller connects to the OVS through OpenFlow channel, we can run custom REST applications using APIs provided by 'ofctl_rest' application. Similarly, Ryu provides 'rest_qos' application specifically designed to provide APIs to install and manipulate meter entries in the *meter table*. Figure 3.13 describe the REST python script to insert entries into the *meter table* of an OVS. In $Line_1$, we import the python 'requests' library to send HTTP requests to the OVS. Hypertext Transfer Protocol (HTTP) defines the underlying format to how *client* formulates a request and how *server* responds to it with a response message. In our network setup, the remote host with IP address 192.168.1.101 on which the Ryu controller instance is running acts as a HTTP client and the OVS_1 of which the meter table has to be configured, acts as a *server* responding with insertion of entries into it's *meter table*. $Line_2$ specifies the IP address and the port number on Open vSwitch Database (OVSDB) of OVS_1 is listening to get connected to the Ryu controller through OpenFlow channel. $Line_3$ specifies the URL of the OVS_1 with distinct 'datapath-id'. Once the connection to OVS_1 is established within this REST application, using $Line_4$ and $Line_5$ we insert meter entries as listed in Table 3.3 into the *meter table*. *15* meter entries with distinct 'meter_id' and 'rate' were added using the datapath identifier (dpid) of OVS_1 listing the data rates supported by VHF, UHF and SatCom radios.

Similarly, *flow table*$_0$ of OVS_1 is configured using REST application script as described in the Figure 3.14. Using $Line_{1,2,3}$ we establish HTTP connection to the

```
1) import requests

2) ovsdb_address = "tcp:127.0.0.1:6632"
3) requests.put('http://192.168.1.101:8080/v1.0/conf/switches/<datapath-id>/ovsdb_addr',
                data = ovsdb_address)

4) meter_entry = {"dpid": <id>, "flags": "KBPS", "meter_id": <id>,
                  "bands": [{"type": "DROP", "rate": <rate>, "burst_size": <burst_rate>}]}
5) requests.post('http://192.168.1.101:8080/qos/meter/<datapath-id>',
                data = meter_entry)
```

Figure 3.13 REST application to configure meter table on OVS_1

Open vSwitch Database (OVSDB) on OVS_1. $Line_4$ describes the format of flow rule to be inserted into *flow* $table_0$ as a key-value pair. The key-value pair 'table_id: 0' specifies the *flow* $table_{id}$, to which the flow rule has to be inserted along with dpid of the switch (i.e dpid of OVS_1). We specify the 'idle' and 'hard' timeout of this flow rule to '0' defining the existence of this rule inside *flow* $table_0$ to an infinite amount of time. As part of the *Match Fields*, we insert IP source and destination addresses along with DSCP values of IP packets (eth_type: 2048) through UDP protocol (ip_proto: 17). A total of *15* flow rules were added to *flow* $table_0$ to uniquely distinguish packets coming from *5* distinct command and control services (refer Table 3.1) destined towards VHF (192.168.20.10), UHF (192.168.30.10) and SatCom (192.168.40.10) destination hosts.

```
1) import requests

2) ovsdb_address = "tcp:127.0.0.1:6632"
3) requests.put('http://192.168.1.101:8080/v1.0/conf/switches/<datapath-id>/ovsdb_addr',
                data = ovsdb_address)

4) flow_entry = {"dpid": <id>, "table_id": 0, "idle_timeout": 0, "hard_timeout": 0, "priority": 1,
                 "match":{"ipv4_src": "192.168.10.10", "ipv4_dst": <dst_addr>,
                 "ip_dscp": <dscp_value>, "ip_proto": 17, "eth_type": 2048},
                 "actions":[{"type":"GOTO_TABLE", "table_id": 1},
                            {"type":"METER", "meter_id": <id>}]}
5) requests.post('http://192.168.1.101:8080/stats/flowentry/add', data = flow_entry)
```

Figure 3.14 REST application for initial configuration of *flow* $table_0$ on OVS_1

3.5.0.1 Adaptive Shaping Mechanism using meter table of Open vSwitch

We assume that in Software Defined Radios, an application running on SDN controller should have access to the information about the current radio modulation (or waveform) being used. Depending on the data rate supported by the current radio modulation, the application can modify flow entries in the *flow table* of OpenFlow switch to adaptively serve QoS requirements of the user data flow. Considering this assumption, we developed a REST application running on the Ryu controller in our network setup, to gain access to the current data rate on r_1-eth_1, r_1-eth_2 and r_1-eth_3 interfaces of the Radio Host (192.168.10.1) emulating ever-changing link data rates at a heterogeneous node as described in Section 3.3.1.2.

To serve the information about the data rates on these egress interfaces, we made use of tinyrpc framework [16] written in python for making and handling Remote

```
1) from tinyrpc.dispatch import RPCDispatcher
2) from tinyrpc.transports.wsgi import WsgiServerTransport
3) from tinyrpc.server.gevent import RPCServerGreenlets
4) from tinyrpc.dispatch import RPCDispatcher
5) import subprocess

6) dispatcher = RPCDispatcher()
7) transport = WsgiServerTransport(queue_class=gevent.queue.Queue)
8) rpc_server = RPCServerGreenlets(transport, JSONRPCProtocol(), dispatcher)

9) @dispatcher.public
10) def get_vhf_data_rate():
11)         output = subprocess.check_output(['tc', '-s', '-j', 'class', 'show', 'dev', 'r1-eth1'])
12)         vhf_data_rate = parse 'output' variable to get the data rate on r1-eth1 interface
13)         return vhf_data_rate

14) @dispatcher.public
15) def get_uhf_data_rate():
16)         output = subprocess.check_output(['tc', '-s', '-j', 'class', 'show', 'dev', 'r1-eth2'])
17)         uhf_data_rate = parse 'output' variable to get the data rate on r1-eth2 interface
18)         return uhf_data_rate

19) @dispatcher.public
20) def get_satcom_data_rate():
21)         output = subprocess.check_output(['tc', '-s', '-j', 'class', 'show', 'dev', 'r1-eth3'])
22)         satcom_data_rate = parse 'output' variable to get the data rate on r1-eth3 interface
23)         return satcom_data_rate

24) rpc_server.serve_forever()
```

Figure 3.15 Pseudo-code for running RPC server on Radio Host (192.168.10.1) to expose current link data rate information of it's interfaces

Procedure Call (RPC). We implemented a RPC server running on the Radio Host while exposing the information on data rates to RPC clients. Figure 3.15 describe the pseudo-code to implement this server. Relevant methods were imported from tinyrpc framework for the server implementation using Line$_{1,2,3,4}$ in the figure. We used python 'subprocess' module to fork a process on these three interfaces to get the information on data rate being shaped at. We used RPC Dispatcher (Line$_6$) to serialize this information, before being sent over to the RPC clients. The request made by RPC clients was forwarded to RPC server through the Web Server Gateway Interface (WSGI) as described in Line$_7$. Using Line$_8$, we start RPC server as a background greenlet events. Member functions 'get_vhf_data_rate' (Line$_{10}$), 'get_uhf_data_rate' (Line$_{15}$) and 'get_satcom_data_rate' (Line$_{20}$) are used to get the information about the data rates on corresponding interfaces. The data rate value returned by these member functions will be serialized by RPC Dispatcher and served forever to the RPC clients using Line$_{24}$.

To receive the information on current radio link data rate and adaptively shape the data rate according to it, we wrote a REST application running on Ryu controller acting as a RPC client forwarding HTTP requests to the RPC server as described before. Figure 3.16 describes the pseudo-code of this REST application wherein, for every 2 seconds we receive the information on the current VHF, UHF and SatCom link data rate (Line$_{10,11,12,13}$). Once we receive this information, we check whether the data rate has changed on these three interfaces. If Yes, we modify the flow entry on OVS$_1$ as described in Line$_4$ of Figure 3.14 to corresponding meter IDs defined in

```
from tinyrpc.protocols.jsonrpc import JSONRPCProtocol
from tinyrpc.transports.http import HttpPostClientTransport
from tinyrpc import RPCClient
import time
import sys

rpc_client = RPCClient(JSONRPCProtocol(), HttpPostClientTransport('http://localhost:8080/'))
remote_server = rpc_client.get_proxy()

try:
    while True:
        time.sleep(2)
        vhf_data_rate = remote_server.get_vhf_data_rate()
        uhf_data_rate = remote_server.get_uhf_data_rate()
        satcom_data_rate = remote_server.get_satcom_data_rate()
        if 'there is a change in data rate':
            # Procedure to manipulate flow entries of flow table0 on OVS1
            # to specify corresponding 'meter_id' on meter table
except KeyboardInterrupt:
    sys.exit(1)
```

Figure 3.16 Pseudo-code of REST application to determine the radio link quality and adaptively ensure QoS requirement

the meter table of OVS_1 (refer Table 3.3). To POST this modification request to OVS_1, *requests.post('http://192.168.1.101:8080/stats/flowentry/modify_strict', data = flow_entry)* was used.

3.5.0.2 Adaptive Mechanism to ensure Time-of-Expiry (ToE) QoS requirement

To enforce Time-of-Expiry (ToE) QoS requirement for messages, we used the *flow timeout* feature of OVS. The initial flow entries installed in the *flow table*$_0$ on OVS_1 had *timeout* entries specified to '0' as described in Table 3.2, meaning these entries would remain infinitely. But we can modify these timeout value of the initial flow entries with respect to the different DSCP value entries. We wrote a REST application to consistently monitor the number of packets matching the *Match Fields* in each flow entry. When the number of packets matched exceed '0', the timeout value of the corresponding flow entry was modified to Time-of-Expiry (ToE) value of the message as mentioned in Table 3.1. After exceeding this timeout value, the flow entry was automatically deleted, resulting in the absence of *Match Fields* for the packets from a particular message. Due to this absence, the packets from messages exceeding ToE were dropped at OVS_1. $Line_4$ to $Line_{12}$ in Figure 3.17 describe the pseudo-code of this REST application, used to ensure ToE QoS requirement of user data flow.

```
import requests

ovsdb_address = "tcp:127.0.0.1:6632"
requests.put('http://192.168.1.101:8080/v1.0/conf/switches/<datapath-id>/ovsdb_addr',
            data = ovsdb_address)

try:
    def set_timeout_for_flow_entry():
        while True:
            time.sleep(1)
            # Get the packet count matching the flow entry
            # If the number of packets matching the flow entry is > 0:
                # Modify the 'hard_timeout' value of the flow entry to the corresponding ToE value
except KeyboardInterrupt:
    sys.exit(1)
```

Figure 3.17 Pseudo-code of the REST application to ensure Time-of-Expiry (ToE) QoS requirement of user data flow

4

Evaluation

In this chapter, we discuss the experimental results of our adaptive QoS framework, designed based on Software-Defined Networking (SDN) paradigm. Network set up as shown in Figure 3.1 and described in the previous chapter was used to perform experiments to quantify the performance of the proposed framework. Messages from different Command and Control (C2) services having distinct QoS requirements were generated and sent across the nodes in the network set up to evaluate the provisioning of QoS requirements by this framework. We begin with experiments to prove the need for a traffic shaping mechanism to avoid packet loss across heterogeneous communication links supporting different data rates as described in section 3.2. Then we conduct experiments to evaluate 'Priority' QoS requirement for the messages by a combination of traffic scheduling mechanisms that are available within Linux Kernel. Using custom applications running on top of the SDN controller, experiments were conducted to prove the adaptability of the proposed QoS framework in ensuring 'Time-of-Expiry (ToE)' QoS requirement and shaping the traffic based on the current available bandwidth in a tactical scenario. Drawbacks of this adaptive mechanism were discussed along with a scope for future enhancements.

4.1 Message Generation and Logging

Messages with respect to five distinct C2 services were generated using MGEN traffic generator with appropriate 'Priority' as discussed in section 3.1. These messages were generated and sent from source host h_1 (192.168.10.10) to destination hosts h_2 (192.168.20.10), h_3 (192.168.30.10) and h_4 (192.168.40.10) using the commands as shown in Figure 4.1. Line_1 to Line_5 in this figure describe the commands used to send the messages to host h_2. Similarly Line_6 to Line_{10} and Line_{11} to Line_{15} describe the commands to send messages to hosts h_3 and h_4 respectively. All these messages were sent as a burst to these destination hosts to quantify the QoS framework's ability to differentiate the packets belonging to distinct services. These messages were generated on 0.0^{th} second and turned 'OFF' on the next (i.e 1.0^{st}) second to

```
0.0 ON 1 UDP SRC 5000 DST 192.168.20.10/5000 PERIODIC [100 1024] TOS 0x78
0.0 ON 2 UDP SRC 5001 DST 192.168.20.10/5000 PERIODIC [100 1024] TOS 0x50
0.0 ON 3 UDP SRC 5002 DST 192.168.20.10/5000 PERIODIC [100 1024] TOS 0x28
0.0 ON 4 UDP SRC 5003 DST 192.168.20.10/5000 PERIODIC [100 1024] TOS 0x04
0.0 ON 5 UDP SRC 5004 DST 192.168.20.10/5000 PERIODIC [100 1024] TOS 0x00
0.0 ON 6 UDP SRC 5005 DST 192.168.30.10/5000 PERIODIC [100 1024] TOS 0x78
0.0 ON 7 UDP SRC 5006 DST 192.168.30.10/5000 PERIODIC [100 1024] TOS 0x50
0.0 ON 8 UDP SRC 5007 DST 192.168.30.10/5000 PERIODIC [100 1024] TOS 0x28
0.0 ON 9 UDP SRC 5008 DST 192.168.30.10/5000 PERIODIC [100 1024] TOS 0x04
0.0 ON 10 UDP SRC 5009 DST 192.168.30.10/5000 PERIODIC [100 1024] TOS 0x00
0.0 ON 11 UDP SRC 5010 DST 192.168.40.10/5000 PERIODIC [100 1024] TOS 0x78
0.0 ON 12 UDP SRC 5011 DST 192.168.40.10/5000 PERIODIC [100 1024] TOS 0x50
0.0 ON 13 UDP SRC 5012 DST 192.168.40.10/5000 PERIODIC [100 1024] TOS 0x28
0.0 ON 14 UDP SRC 5013 DST 192.168.40.10/5000 PERIODIC [100 1024] TOS 0x04
0.0 ON 15 UDP SRC 5014 DST 192.168.40.10/5000 PERIODIC [100 1024] TOS 0x00
1.0 OFF 1
 . . .
1.0 OFF 15
```

Figure 4.1 MGEN script at the source host h_1

```
1) mgen event "listen UDP 5000"
2) <Packet Received timestamp> <flowd-id> <Packet Sequence Number>
2) <SRC-IP> <DST-IP> <Packet Sent timestamp>
```

Figure 4.2 MGEN log at the destination hosts h_2, h_3 and h_4

define the burst. Each of these flow of messages were set to use UDP as transport protocol consisting of 100 packets each with a payload size of 1024 bytes to evaluate the loss of packets while ensuring QoS requirements. ToS byte of these packets were encoded with appropriate value as defined in section 3.1. Since the ToS byte is a per-socket attribute, meaning the same ToS bits are applied to all the packets going through the same port, distinct ports ranging from 5000 to 5014 were used in the source host h_1 to send fifteen distinct flows (i.e 1 to 15) across three destination hosts. $Line_1$ in Figure 4.2 represents the command to receive these messages at the hosts h_2, h_3 and h_4. $Line_2$ represents the pattern of log messages received, using which we compute packet delay ($Packet\ Received\ timestamp - Packet\ Sent\ timestamp$) and packet loss (using $Packet\ Sequence\ Number$), and we use *flow-id* to quantify the system's ability to differentiate the five types of messages.

4.2 Experimental results

In this section, we discuss the experimental results of our adaptive QoS mechanism over the network setup as described in Figure 3.1 and explained in the previous chapter. We conducted experiments over five scenarios. First, four classes of user traffic were sent through the network without any traffic shaping and scheduling mechanism at the source host, h_1; this is the baseline for our comparative study. Then for the second scenario, we shaped the traffic using HTB Qdisc according to the data rates supported by VHF, UHF and SatCom link, and scheduled these shaped data flows with strict priority based PFIFO Qdisc. Both these Qdiscs were implemented on the egress port h_1-eth_0 of host h_1. In the third scenario, we implemented the hybrid priority scheduling using ETS Qdisc at h_1. In the fourth, we conducted experiments to adaptively shape the data flow through a combination of *meter table* at OVS_1 and Remote Procedure Calls (RPC) between the interfaces emulating the radio link data rate and a REST application running on the controller to modify the meter entries corresponding to the current link data rate. As part of the final scenario, we present the results of ensuring ToE QoS for the messages using REST application running on the north-bound interface of the Ryu SDN controller.

4.2.1 No traffic scheduling mechanisms

As part of the first experimental scenario, traffic shaping and scheduling mechanisms were not implemented at the source host, h_1. This experiment is the baseline for our comparative study. We emulated the maximum data rate supported by a VHF, UHF and SatCom link over interfaces r_1-eth_1, r_1-eth_2 and r_1-eth_3 respectively of Radio Host (192.168.10.1) as shown in our network set up (refer Figure 3.1). As mentioned in the previous section, a burst of messages were generated at host h_1 and sent to VHF (h_2), UHF (h_3) and SatCom (h_4) destinations hosts over corresponding egress interfaces of Radio Host, r_1. Figure. 4.3 shows the End-to-End (E2E) delay experienced by packets over VHF (left) and UHF (right) links. Packets over VHF link experienced a delay of up-to 420 seconds while packets over UHF link experienced up-to 20 seconds delay. Without any scheduling mechanism, all the five messages were treated with equal priority. In addition, packet loss were computed over both the links. Since traffic was not shaped and scheduled at the source host h_1, equal proportion of packet loss among messages was reported. While messages over VHF link reported packet loss of 6($\pm$1)%, over UHF link packet loss was just 1%, reported over repeated experiments. Since there was no packet loss over SatCom link and E2E delay for packets was not descriptive enough to compare with the delay over UHF link, the plot for SatCom link was not shown in the figure. Because of no Differentiated Service (DiffServ) for messages, *Priority* QoS requirement was not satisfied along with packet loss. Hence there was a need for employing a traffic shaping and scheduling mechanism, the results of which will be discussed in the following sections.

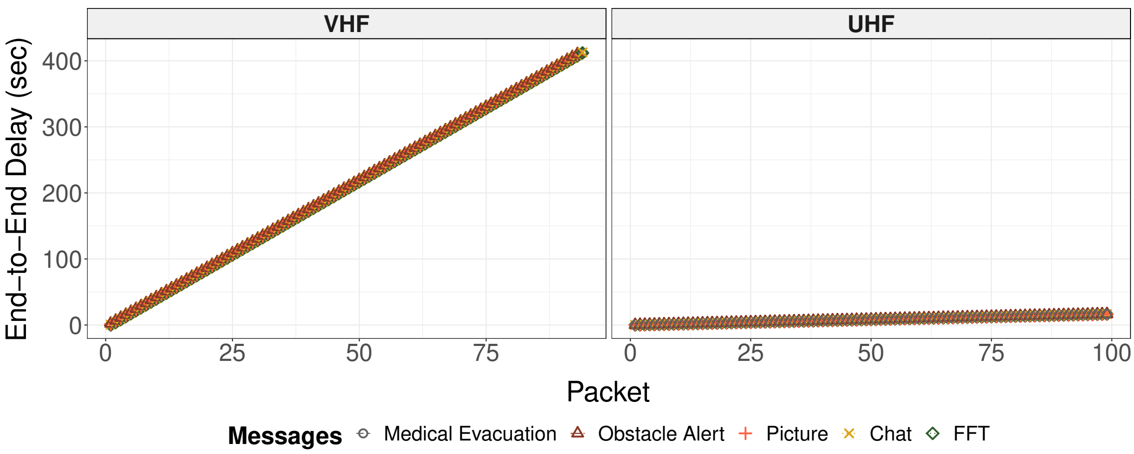

Figure 4.3 E2E delay for IP packets sent over VHF and UHF links without any scheduling mechanism

4.2.2 Traffic shaping and strict priority scheduling mechanism

As part of our initial effort to provide Differentiated Service (DiffServ) to messages, we considered using Priority First-In-First-Out (PFIFO) Qdisc. To avoid packet loss during a burst of messages as we had seen in the previous experimental scenario, we began with an attempt to include HTB Qdisc resulting in a tree structure as shown in the Figure 3.5. Using this tree setup, traffic was shaped at the outbound interface h_1-eth_0 of source host h_1 with respect to the data rates supported by VHF, UHF and SatCom links. PFIFO Qdiscs were placed under Hierarchy Token Bucket (HTB) Qdiscs to introduce strict priority among the messages, by creating four bands (or queues) within it, as described in the section 3.3.2.1.

Figure 4.4 shows the End-to-End (E2E) delay for packets from different messages when traffic was shaped from 0.6 kbps (leftmost) to 9.6 kbps (rightmost) with respect to the data rates supported by a VHF link. Since packets from *Chat* and *FFT* messages were enqueued to the lowest-priority bands, packets from them were not received until all the packets from *Picture*, *Obstacle Alert* and *Medical Evacuation* are received. Packets from lower-priority messages were received only after receiving all the packets from higher-priority messages. Packets from *Chat* and *FFT* messages experienced the longest delay within this *strict* priority mechanism. While shaping the traffic at 0.6 kbps, the first packets from these low-priority messages were not received on the destination host h_2 until 4000 seconds since they were sent, which is a delay larger than the Time-of-Expiry (ToE) for these packets (refer Table 3.1). Moreover, we have not considered excessive amount of high-priority traffic scenario in this experiment, which will further delay the reception of these low-priority traffic. But using this kind of scheduling mechanism makes sense when higher-priority traffic should always take precedence over lower-priority ones. Similarly, Figure 4.5 and Figure 4.6 depicts E2E delay experienced by packets sent over the data rates supported by UHF and SatCom links, respectively. One thing to notice by comparing both the figures is that the difference in scale of E2E delay for packets through their lowest data rate, i.e through 15 kbps and 32 kbps for UHF and SatCom respectively. It clearly depicts that the packet delay reduces by half in case of SatCom

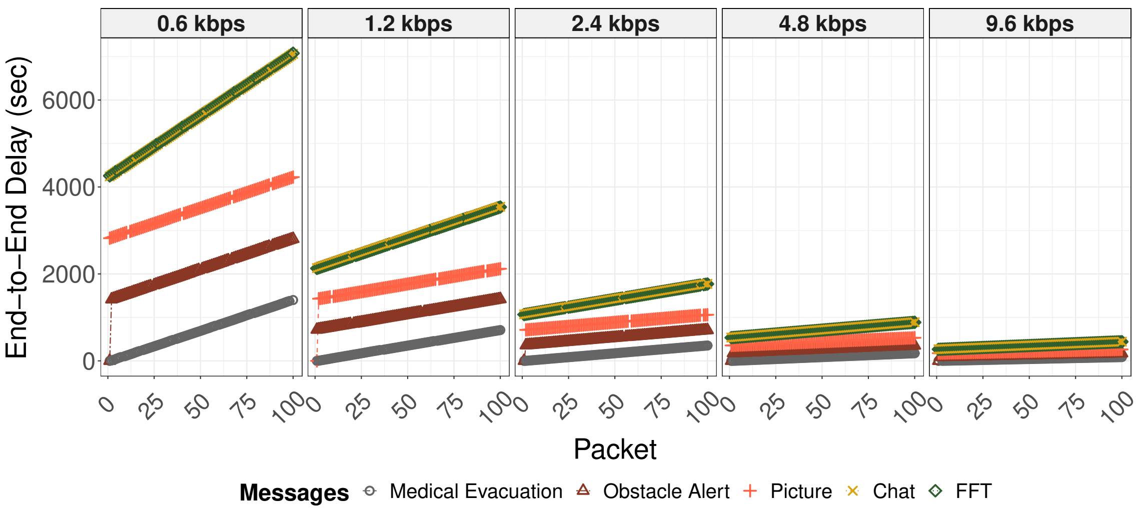

Figure 4.4 E2E delay for IP packets with strict priority scheduling over different data rates of a VHF link

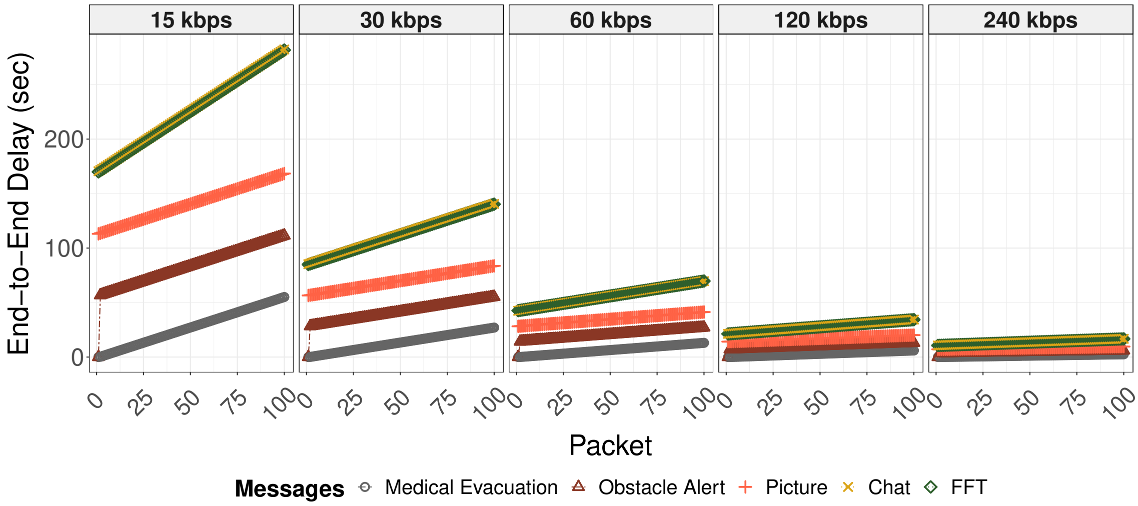

Figure 4.5 E2E delay for IP packets with strict priority scheduling over different data rates of a UHF link

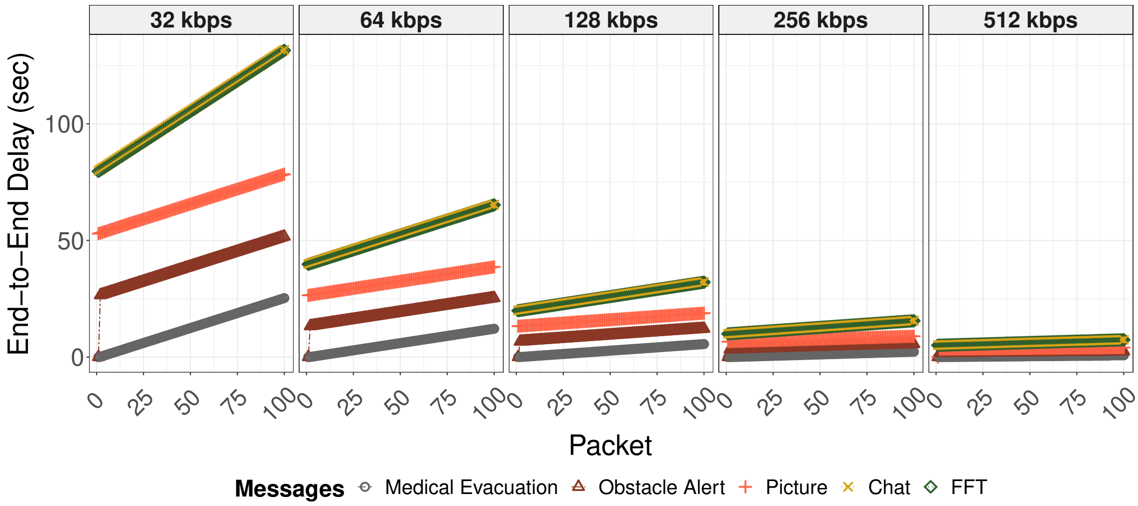

Figure 4.6 E2E delay for IP packets with strict priority scheduling over different data rates of a SatCom link

link when the data rate was shaped at almost the double than that of UHF link data rate. Further, none of the packets from these messages were lost because of the shaping mechanism implemented at the outbound interface of host h_1. As explained in the Section 3.3.2.2, we can combine the *priority* scheduling with *fairness* based scheduling mechanism to avoid bandwidth starvation for lower-priority traffic using ETS Qdisc. The result of which will be discussed in the next section.

4.2.3 Traffic shaping and hybrid priority scheduling mechanism

In this section, we discuss the effect of including hybrid traffic scheduling mechanism using Enhanced Transmission Selection (ETS) Qdisc with it's configuration as described in section 3.3.2.2. Figure 4.7 shows the End-to-End (E2E) delay for the packets from different messages when traffic was shaped with respect to the data rates supported by a VHF link (i.e 0.6, 1.2, 2.4, 4.8 and 9.6 kbps). ETS Qdisc was configured to schedule the packets from *Medical Evacuation* and *Obstacle Alert*

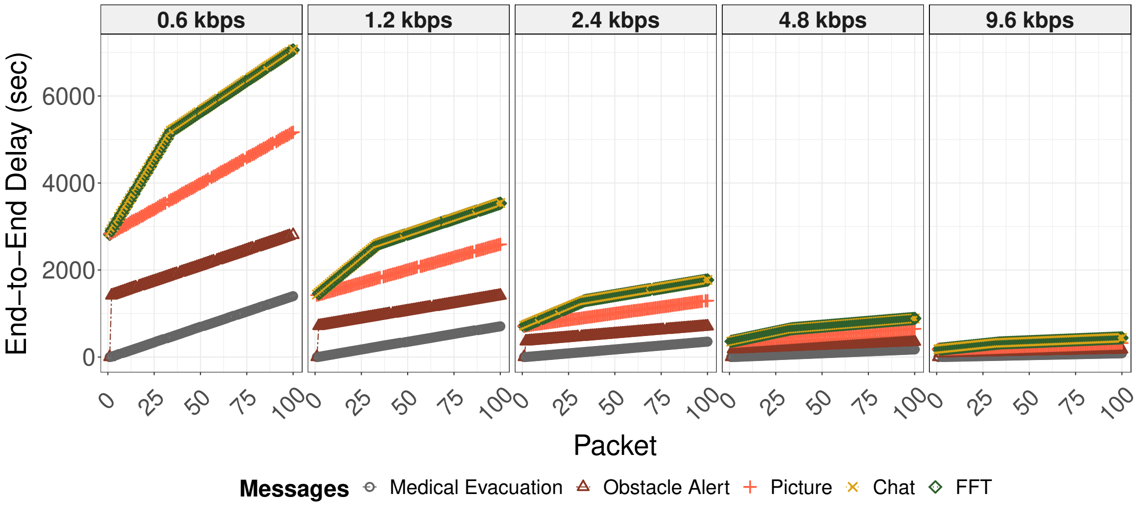

Figure 4.7 E2E delay for IP packets with hybrid priority scheduling over different data rates of a VHF link

messages in strict *priority* order, and packets from *Picture*, *Chat* and *FFT* messages in a *fairness* based scheduling mechanism. Because of this configuration, packets from *fairness* based scheduling mechanism were only received after receiving all the packets from strict *priority* mechanism. We term this combination of *priority* and *fairness* as *hybrid* scheduling mechanism. We considered to depict the delay for packets over VHF links since the effect of introducing this *hybrid* scheduling mechanism has better visibility over low data rates when compared with that of higher data rates.

Bandwidth was equally shared among the two queues (or bands) that were reserved for *fairness* based scheduling. One band was configured to queue the packets from *Picture* message, while the other was configured to queue the packets from both *Chat* and *FFT* messages. Because of this configuration, packets from *Picture* message were received sooner than the packets from the other two messages even though the first packets from all these three messages were received simultaneously. Further, to prove that all the packets from these five messages were categorized and placed onto respective bands as described in section 3.3.2.2, we continuously monitored the occupancy of these bands in packets, over time. Figure 4.8 shows the occupancy of all the four bands in the two ETS Qdiscs while shaping the data rate with respect to the rates supported by VHF and UHF links, simultaneously from lowest to highest. 100 packets from each *Medical Evacuation*, *Obstacle Alert* and *Picture* messages were enqueued and dequeued from bands: *band-0*, *band-1* and *band-2* respectively. Total 200 packets with 100 each from *Chat* and *FFT* messages were enqueued and dequeued from *band-3*.

4.2.4 Adaptive Traffic shaping mechanism using meter table of Open vSwitch

In this section we discuss the experimental results of adaptively shaping the traffic using *meter table* feature of Open vSwitch (OVS) based on the current link data rate on the interfaces of Radio Host as described in section 3.5.0.1. We also show the

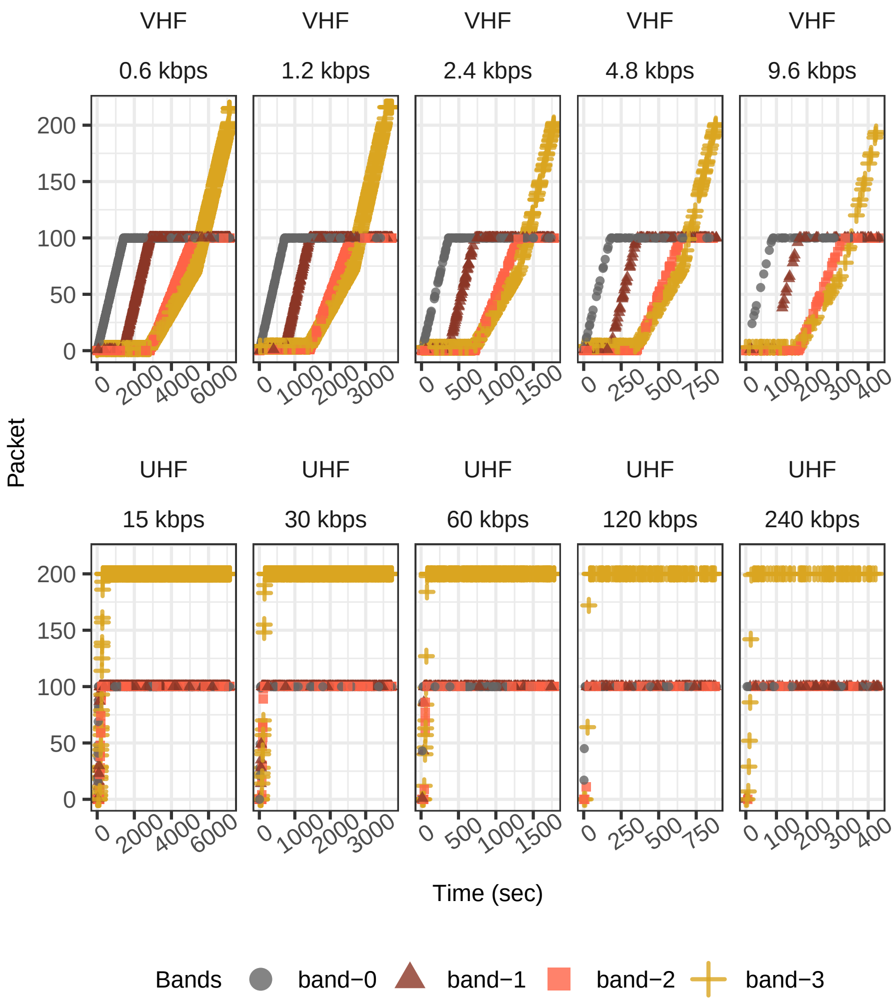

Figure 4.8 Occupancy of packets on ETS Qdisc bands over time through VHF and UHF links

experimental result of addressing the drawback of this adaptive shaping using *meter table*, by excluding shaping by *meter table* and replacing it with HTB Qdiscs at the OVS egress port. The experiments in this section was conducted on the assumption that, the applications running on the north-bound interface of the SDN controller should have access to the information about the current modulation (or waveform) being used in the radio.

To study the effect of adaptive shaping capabilities of *meter table*, we conducted five experiments with respect to the data rates supported by VHF, UHF and SatCom links. Messages that were generated and sent from source host h_1 were shaped and scheduled using ETS Qdisc at interface h_1-eth_0, with respect to the maximum data rates supported by these links. Then, in each of these five different experiments, we changed the data rates on interfaces: r_1-eth_1, r_1-eth_2 and r_1-eth_3 to emulate the five distinct data rates (from highest to lowest) supported by VHF, UHF and SatCom links respectively (refer Figure 3.1). Considering these changes on the interfaces, using already installed rate limiting entries in the *meter table* of OVS_1 as described in Table 3.3, we direct the flow towards corresponding meter IDs to shape the data rate with respect to the data rate emulated by these interfaces.

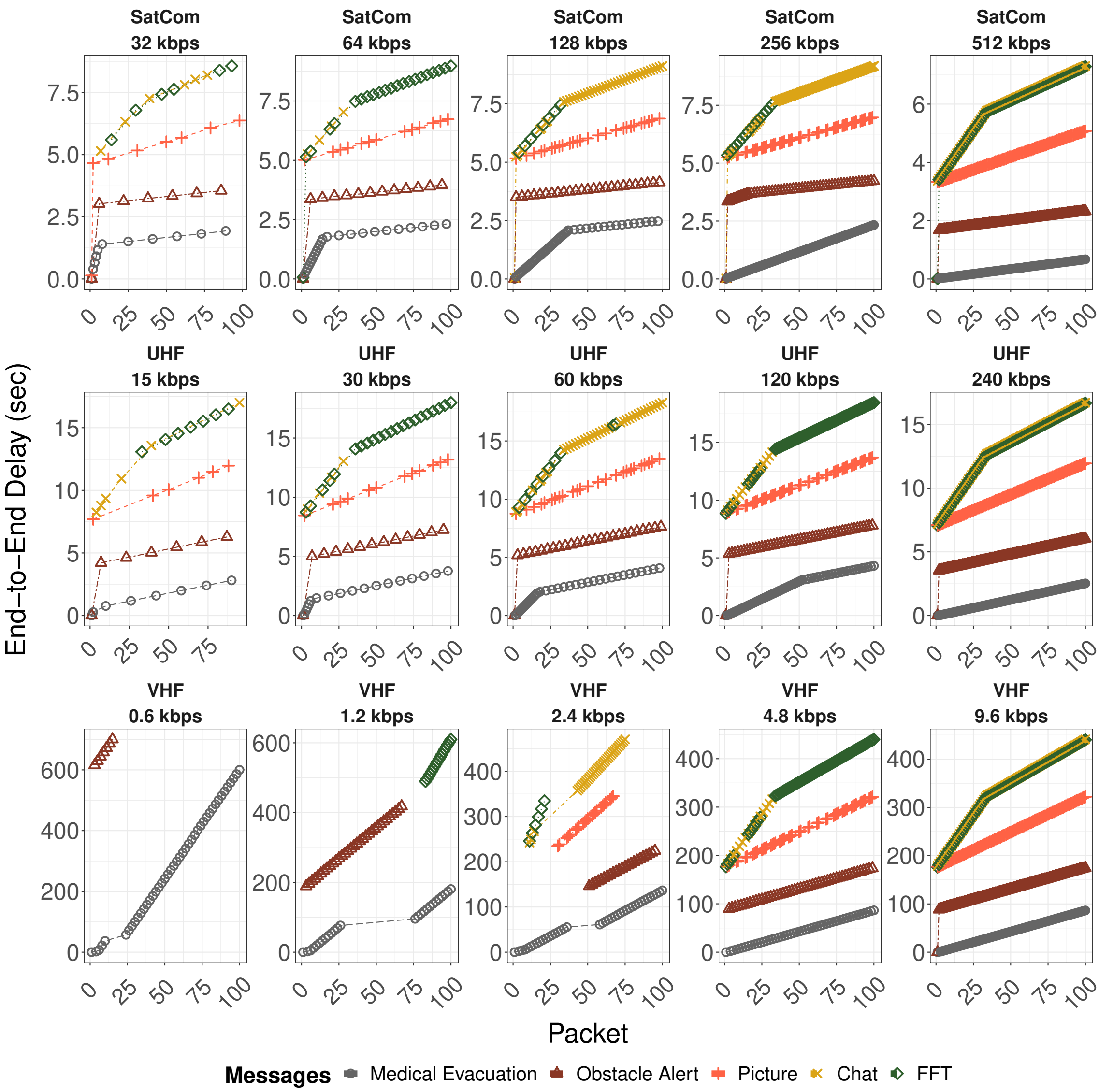

Figure 4.9 Packets received when traffic was shaped by utilizing meter table of OVS by manual re-directing to corresponding meter entries

As part of the first experiment, we shaped the data rate on these interfaces to emulate the highest data rate supported by corresponding links. Then we modified the flow entries in flow table to direct the flow destined towards SatCom (h_4), UHF (h_3) and VHF (h_2) hosts to meter IDs 1, 6 and 11 respectively. In the subsequent four experiments, we reduced the data rates on these interfaces and similarly as before we modified flow entries to re-direct them towards (2, 7, 12), (3, 8, 13), (4, 9, 14) and (5, 10, 15) destined to h_4, h_3 and h_2 hosts respectively. Figure 4.9 shows the packets received over these five different experiments while shaping the traffic using *meter table*. As we can notice that there were significant packet loss over lower data rate meter IDs. Table 4.1 list the number of packets received on host h_4 when the flow was directed towards meter IDs: 1, 2, 3, 4 and 5 representing the data rates supported by a SatCom link. Similarly, Table 4.2 list the number of packets received on host h_3 while directing the flow towards meter IDs: 6, 7, 8, 9 and 10 representing the data rates supported by a UHF link. To conclude, Table 4.3 list the number of

Service	Packets sent	Packets received over SatCom link data rates (kbps)				
		32	64	128	256	512
Medical Evacuation	100	11	25	52	100	100
Obstacle Alert	100	7	13	26	58	100
Picture	100	8	13	26	52	100
Chat	100	6	4	42	82	100
FFT	100	6	22	8	16	100
Total Packets Received/500		38/500	77/500	154/500	308/500	500/500

Table 4.1 Loss of packets when traffic was shaped by meter table over SatCom link

Service	Packets sent	Packets received over UHF link data rates (kbps)				
		15	30	60	120	240
Medical Evacuation	100	8	18	37	75	100
Obstacle Alert	100	7	13	26	50	100
Picture	100	6	12	25	52	100
Chat	100	6	4	40	16	100
FFT	100	7	22	10	82	100
Total Packets Received/500		34/500	69/500	266/500	276/500	500/500

Table 4.2 Loss of packets when traffic was shaped by meter table over UHF link

Service	Packets sent	Packets received over VHF link data rates (kbps)				
		0.6	1.2	2.4	4.8	9.6
Medical Evacuation	100	44	26	40	50	100
Obstacle Alert	100	7	33	23	49	100
Picture	100	0	0	23	52	100
Chat	100	0	0	36	16	100
FFT	100	0	18	6	82	100
Total Packets Received/500		51/500	77/500	128/500	249/500	500/500

Table 4.3 Loss of packets when traffic was shaped by meter table over VHF link

packets received on host h_2 by directing the flow towards meter IDs: 11, 12, 13, 14 and 15 corresponding to the data rates supported by a VHF link.

We wrote a REST application to consistently monitor the number of packets flowing through all the meter IDs at a time, throughout the experiment. Figure 4.10 shows the number of packets that went through the meter IDs in a given experiment (from 1 to 5). Notice that all the 100 packets from five messages (summing upto 500) pass through the specified meter IDs and none of them through the rest of the IDs. The result was consistent throughout five different experiments having directed the flow towards different meter IDs. Even though all the packets pass through the meter IDs, there was significant packet loss proving the fact that these packets were dropped by the *meter table* while shaping. The reason for the drop was because of

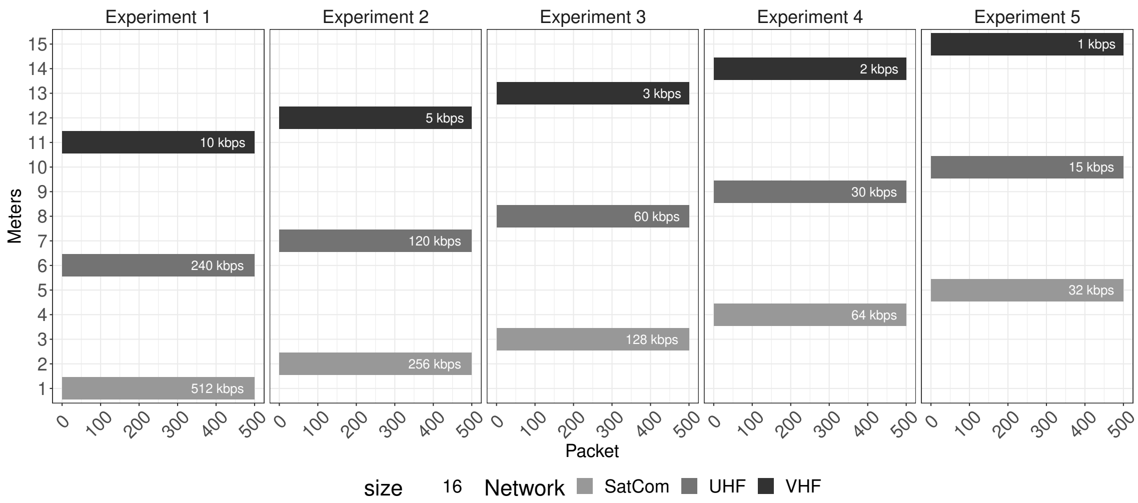

Figure 4.10 Packets directed through different meter IDs of a meter table throughout different experiments

the lack of queuing mechanism by HTB Qdiscs that are implemented within meter IDs. They have no mechanism to queue the packets when packets arrive at a rate higher than the specified rate at which these packets have to be shaped. Thus, we conclude that the traffic shaping using *meter table* of Open vSwitch is not suitable in it's current form of implementation for low-bandwidth networks such as tactical networks.

Instead of reducing the data rates over five different experiments, we reduced the data rates (from highest to lowest) on interfaces of Radio Host r_1 within a single experiment for an interval of 40 seconds for each data rate. A REST application running on the SDN controller was able to recognize the data rate change on interfaces of r_1 and adaptively modified the flow entry to direct the flow towards corresponding meter IDs representing the change in data rate, to be shaped at. The design of this adaptive mechanism is described in section 3.5.0.1. Figure 4.11 illustrates this adaptive shaping mechanism over VHF link while depicting the number of packets received over the duration of the experiment. Bottom part of this figure shows the adaptive shaping capability of the *meter table.* The experiment started with shaping the data rate with respect to 9.6 kbps, and after an interval of 40 seconds, shaping was reduced to 4.8 kbps followed by reducing the data rate until 0.6 kbps over a span of 40 seconds for each data rate. As we can notice in the bottom part of the figure, packet loss increased at each interval of data rate change depicting the inefficacy of *meter table* in shaping the data rate.

To prove that by implementing a queuing mechanism at the HTB Qdisc within these meter entries to queue the packets arriving at a higher rate, we modified this adaptive shaping mechanism by replacing *shaping by meter table* to *shaping by HTB Qdisc* at the egress port of OVS_1. The HTB Qdisc at this egress port, queue the packets when they arrive at a higher speed, using the queue attached to this outbound interface. In the upper part of the same Figure 4.11, we can visualize this queuing of packets from *Medical Evacuation* message by HTB Qdisc at the switch egress port, when link change happens.

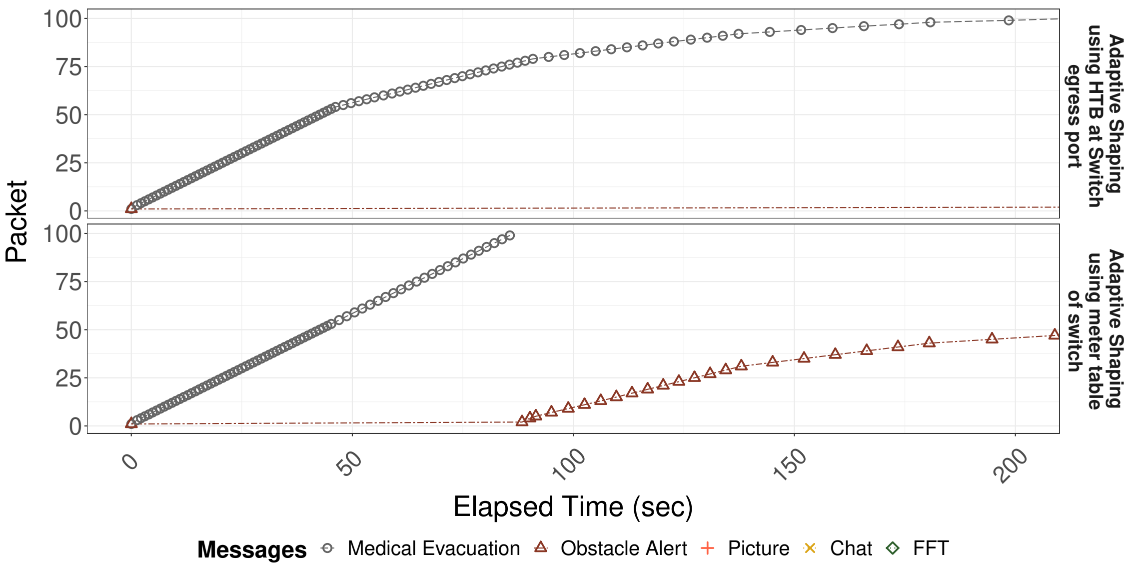

Figure 4.11 Comparison of adaptive shaping mechanism by meter table versus HTB Qdisc at Open vSwitch egress port over VHF link

4.2.5 Adaptive Mechanism to ensure Time-of-Expiry (ToE) QoS requirement

In this section we discuss the results of ensuring ToE QoS requirement for the messages from distinct C2 services. Using this mechanism as described in section 3.5.0.2, we conducted five experiments with respect to the data rates (maximum to minimum) supported by VHF and UHF radios. Messages that were generated and sent from source host h_1, were shaped using HTB Qdiscs and scheduled using ETS Qdisc. These messages were sent across VHF and UHF links to destination hosts h_2 and h_3 respectively. Adaptive traffic shaping was not performed on the OVS, instead the traffic was shaped at the source host to avoid any packet loss as described in section 4.2.3.

To enforce ToE QoS requirement for messages, we used *flow timeout* feature of OVS. This adaptive mechanism to ensure ToE for messages was designed and implemented as described in section 3.5.0.2. To visualize the effect of implementing this mechanism as shown in Figure 4.12, we considered to depict the comparison in number of packets received when ToE was applied (bottom) versus not being applied (top) over different rates of a VHF link. While shaping at the maximum data rate (9.6 kbps) at host h_1 and by employing ETS Qdisc, the first packets from *Chat* and *FFT* messages were received only after 150 seconds when ToE was not applied. Since this duration was more than the ToE value (refer Table 3.1) for both these messages, not a single packet was received from either of the two messages when ToE was applied. Similarly, packets from *Obstacle Alert* message was dropped after 150 seconds (it's ToE value). While shaping at 4.8 kbps, none of the packets from *Obstacle Alert* message was received since the first packet from this message arrives after 150 seconds when ToE was not applied. Accordingly, ToE QoS requirement was ensured for packets from *Medical Evacuation* (ToE - 300 seconds) and *Picture* (ToE - 3600 seconds) messages while shaping over low data rates. The effect of ensuring ToE QoS requirement for messages over UHF links could be visualized in Figure 4.13. Further in Table 4.4 we list the number of packets received on host h_3 (15, 30, 60,

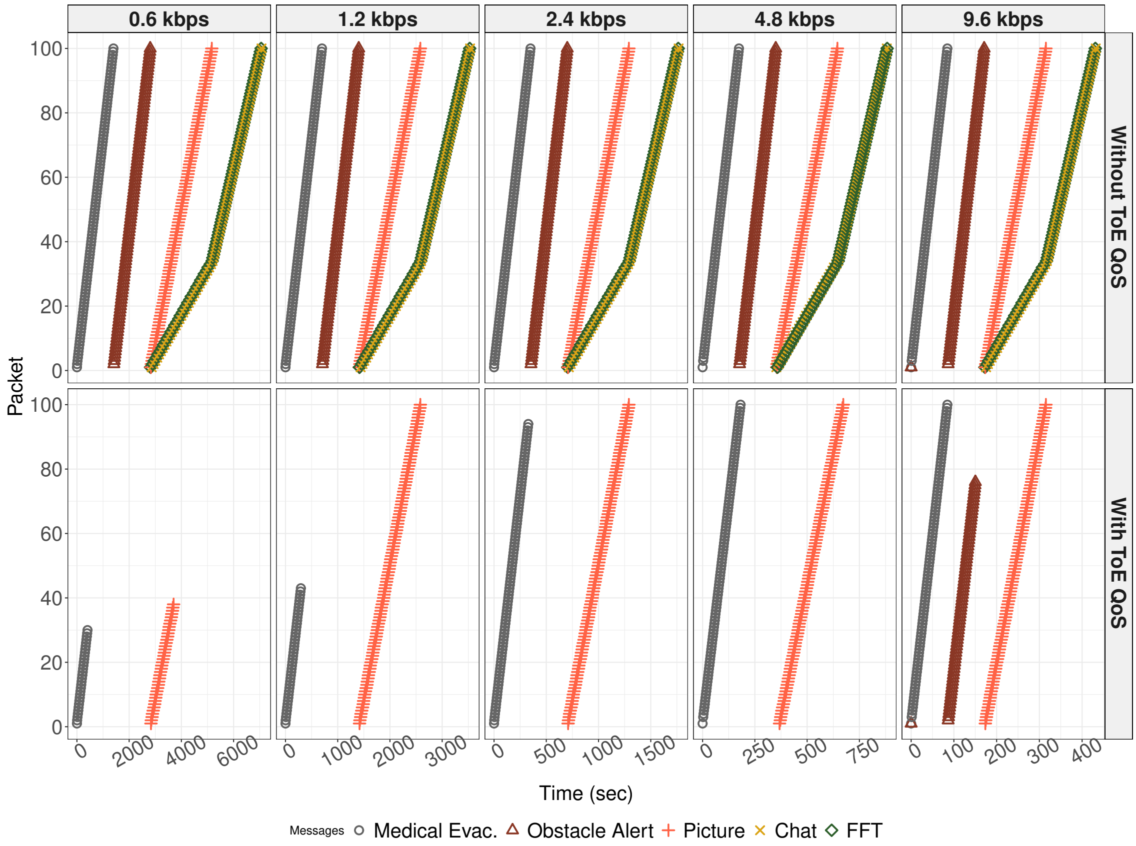

Figure 4.12 Packets received when compared with and without ToE QoS requirement applied over VHF links

120 and 240 kbps) and h_2 (0.6, 1.2, 2.4, 4.8 and 9.6 kbps) when the data rate was shaped with respect to UHF and VHF links along with the corresponding ToE for the messages.

One thing to note in this mechanism of ensuring ToE for messages is that, in our experiments we consider a burst of all the five messages to which ToE has to be ensured. But in real tactical scenarios these messages could be generated in different patterns. Since the timer for ensuring this ToE begins when the switch sniffs the first packets from these messages, the timer value does not depict the exact time since this message was generated. Also if packets spend more time while being shaped at the source host, the timer value could not be compared with the message generation timestamp. So this mechanism should be improved to consider message generation timestamp and offset the timer value according to it.

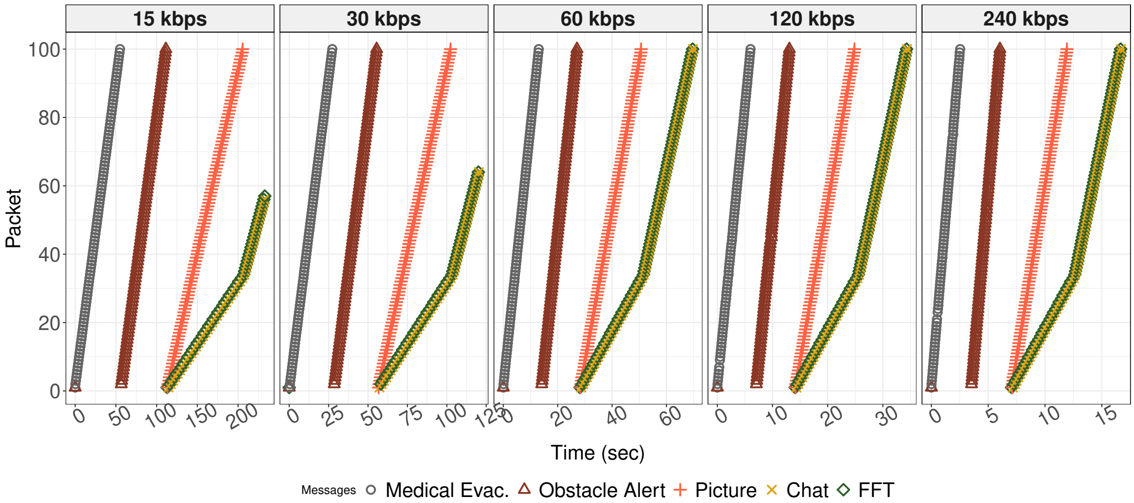

Figure 4.13 Packets received when ToE QoS requirement applied over UHF links

Service	Packets sent	Packets received over data rate (kbps)										ToE (sec)
		0.6	**1.2**	**2.4**	**4.8**	**9.6**	**15**	**30**	**60**	**120**	**240**	
Medical Evacuation	100	30	43	94	100	100	100	100	100	100	100	300
Obstacle alert	100	0	0	0	0	76	100	100	100	100	100	150
Picture	100	38	100	100	100	100	100	100	100	100	100	3600
Chat	100	0	0	0	0	0	56	64	100	100	100	120
FFT	100	0	0	0	0	0	57	64	100	100	100	120

Table 4.4 Loss of packets when ToE QoS requirement is applied for the messages

5

Conclusion

Supporting End-to-End (E2E) Quality-of-Service (QoS) requirements of user data flow in heterogeneous Tactical Networks (TNs) is a challenging task since the networks are characterized by low bandwidth, high delay, and constant change in the link conditions due to high degree of mobility among tactical nodes. The QoS mechanisms that are implemented in the network devices should be adaptive enough to serve the QoS requirements depending on the link conditions with minimal human intervention. In this sense, the network flexibility features provided by Software-Defined Networking (SDN) paradigm was explored in ensuring QoS requirements of the user data flow depending on the heterogeneous link conditions. Therefore the work done in this thesis focuses on the QoS mechanisms for TNs based on SDN technology. An adaptive mechanism was introduced in this thesis, by leveraging SDN concept to ensure/enforce QoS requirements from user data flows in tactical networks. Using a traffic generator, messages emulating distinct Command and Control (C2) services were generated to test the fulfilment of the QoS requirements for these messages by the adaptive mechanism developed in our QoS framework within a SDN network emulation platform. Features within the OpenFlow switch were used to distinguish the flows coming from distinct messages based on their Differentiated Services Code Point (DSCP) value of the packet header. A network application running on the SDN controller was used to continuously sniff the packets on the flow tables of the OpenFlow switch, and flow rules were modified to ensure Time-of-Expiry (ToE) QoS requirements for these distinct flows. Different data rates supported by radio modulations of VHF, UHF and SatCom were emulated on the interfaces of a container host depicting the heterogeneous network set up in a tactical scenario. Assuming that the SDN controller should have access to the information on the current data rate in use on these interfaces, a network application was developed to continuously request this information from the network interfaces and adaptively shape the data flow with respect to that, using *meter table* of the OpenFlow switch. With experimental results, we prove that the adaptive traffic shaping using *meter table*, in it's current form of implementation is not suitable for low-bandwidth and highly dynamic networks such as tactical networks. Also we explored the traffic shaping capabilities of HTB Qdisc, implemented within Linux Kernel. Further, we studied the effect of ensuring ‘Priority’ QoS requirement for four different classes of messages by using strict *priority* based PFIFO Qdisc. And we introduced a notion of *hybrid* scheduling mechanism combining strict *priority* and *fairness* based scheduling using recent ETS Qdisc, implemented within the Linux Kernel. The proposed SDN based QoS framework introduced in this thesis was validated with experimental results by a network topology depicting a minimal ever-changing network scenario, developed within a SDN emulation platform.

5.1 Future Work

The experiments that were conducted in this thesis considered to evaluate the QoS framework based on a burst of distinct messages. To extend the validation of the framework, messages could be generated in different patterns as in case of our previous investigation in [39]. The framework could also be tested with the different message sizes from distinct services. The framework was tested by periodic changes in the link conditions which could be extended by testing with random patterns of link changes as experimented in our previous investigation [39]. The network application ensuring ToE for messages could be improved to consider and offset the time with respect to the time at which these messages were generated. Further, the hybrid scheduling mechanism could be tested with various configurations. Finally, a multi-topology routing similar to the one introduced in [29] could be maintained by using different flow tables for low and high data rate QoS topologies.

5.2 Publications

The publications related to the this thesis are as follows:

- Eswarappa, S. M., Rettore, P. H., Loevenich, J., Sevenich, P., and Lopes, R. R. F. Towards adaptive QoS in SDN-enabled Heterogeneous Tactical Networks. In *International Conference on Military Communications and Information Systems (ICMCIS)* (Oeiras, Portugal, May 2021)
- Loevenich, J., Lopes, R. R. F., Rettore, P. H., Eswarappa, S. M., and Sevenich, P. Maximizing the probability of message delivery over ever-changing communication scenarios in tactical networks. *IEEE Networking Letter* (March 2021), 1–5. early access, doi:10.1109/LNET.2021.3066536
- Lopes, R. R. F., Loevenich, J., Rettore, P. H., Eswarappa, S. M., and Sevenich, P. Quantizing radio link data rates to create ever-changing network conditions in tactical networks. *IEEE Access* (September 2020), 1–20
- Balaraju, P. H., Rettore, P. H., Lopes, R. R. F., Eswarappa, S. M., and Loevenich, J. Dynamic adaptation of the user data flow to the changing data rates in VHF networks: An exploratory study. In *11th IEEE International Conference on Network of the Future (NoF)* (Bordeaux, France, Oct 2020), pp. 1–9

Bibliography

[1] An Architecture for Differentiated Services. Accessed: 26-04-2021. https://www.ietf.org/rfc/rfc2475.txt.

[2] Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers. Accessed: 26-04-2021. https://datatracker.ietf.org/doc/html/rfc2474.

[3] Enhanced Transmission Selection scheduler. Accessed: 22-01-2021. http://manpages.ubuntu.com/manpages/groovy/man8/tc-ets.8.html.

[4] Hierarchy Token Bucket. Accessed: 26-04-2021. https://man7.org/linux/man-pages/man8/tc-htb.8.html.

[5] Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational). Accessed: 26-04-2021. https://www.ietf.org/rfc/rfc1633.txt.

[6] Linux Network Namespaces. Accessed: 2-05-2021. https://man7.org/linux/man-pages/man7/network_namespaces.7.html.

[7] Multi-Generator (MGEN). Accessed: 22-01-2021. https://github.com/USNavalResearchLaboratory/mgen.git.

[8] Open vSwitch. Accessed: 22-01-2021. http://openvswitch.org/.

[9] OpenDaylight Controller. https://www.opendaylight.org/. Accessed: 22-01-2021.

[10] POX Controller. https://noxrepo.github.io/pox-doc/html/. Accessed: 22-01-2021.

[11] Project FloodLight. https://floodlight.atlassian.net/wiki/spaces/HOME/overview?mode=global. Accessed: 22-01-2021.

[12] Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard). Accessed: 26-04-2021. https://www.ietf.org/rfc/rfc2205.txt.

[13] RFC 2475 Architecture for Differentiated Services. Accessed: 26-04-2021. https://www.ietf.org/rfc/rfc2475.txt.

[14] Ryu SDN Framework. https://github.com/faucetsdn/ryu.git. Accessed: 22-01-2021.

[15] Software-Defined Networking (SDN) definition. https://www.opennetworking.org/sdn-definition/. Accessed Online: 24-09-2020.

[16] tinyrpc: A modular rpc library. Accessed: 26-04-2021. `https://tinyrpc.readthedocs.io/en/pre-version-1.0.0/`.

[17] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. A roadmap for traffic engineering in SDN-openflow networks. *Computer Networks 71* (2014), 1–30.

[18] Alberts, D. S., Garstka, J. J., and Stein, F. P. Network centric warfare: Developing and leveraging information superiority. Tech. rep., Assistant Secretary of Defense (C3I/Command Control Research Program), 2000.

[19] Ali, J., Lee, G.-m., Roh, B.-h., Ryu, D. K., and Park, G. Software-defined networking approaches for link failure recovery: A survey. *Sustainability 12*, 10 (2020), 4255.

[20] Balaraju, P. H., Rettore, P. H., Lopes, R. R. F., Eswarappa, S. M., and Loevenich, J. Dynamic adaptation of the user data flow to the changing data rates in VHF networks: An exploratory study. In *11th IEEE International Conference on Network of the Future (NoF)* (Bordeaux, France, Oct 2020), pp. 1–9.

[21] Barz, C., Fuchs, C., Kirchhoff, J., Niewiejska, J., and Rogge, H. Heterogeneous tactical radio networks with flexible IP-waveforms. In *2017 International Conference on Military Communications and Information Systems (ICMCIS)* (2017), IEEE, pp. 1–7.

[22] Boley, J. M., Jung, E.-S., and Kettimuthu, R. Adaptive qos for data transfers using software-defined networking. In *IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)* (2016), pp. 1–6.

[23] Caba, C., and Soler, J. Apis for qos configuration in software defined networks. In *IEEE Conference on Network Softwarization (NetSoft)* (2015), pp. 1–5.

[24] Elmasry, G. A comparative review of commercial vs. tactical wireless networks. *IEEE Communications Magazine 48*, 10 (October 2010), 54–59.

[25] Eswarappa, S. M., Rettore, P. H., Loevenich, J., Sevenich, P., and Lopes, R. R. F. Towards adaptive QoS in SDN-enabled Heterogeneous Tactical Networks. In *International Conference on Military Communications and Information Systems (ICMCIS)* (Oeiras, Portugal, May 2021).

[26] Flathagen, J., Mjelde, T. M., and Bentstuen, O. I. A combined network access control and qos scheme for software defined networks. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (2018), IEEE, pp. 1–6.

[27] Gkioulos, V., Gunleifsen, H., and Weldehawaryat, G. K. A systematic literature review on military software defined networks. *Future Internet 10*, 9 (2018), 88.

[28] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. Reproducible network experiments using container-based emulation. In *8th International Conference on Emerging Networking Experiments and Technologies* (2012), pp. 253–264.

[29] Hauge, M., Landmark, L., Lubkowski, P., and Amanowicz, M. Selected issues of QoS provision in heterogenous military networks.

[30] Jarschel, M., Zinner, T., Hossfeld, T., Tran-Gia, P., and Kellerer, W. Interfaces, attributes, and use cases: A compass for SDN. *IEEE Communications Magazine 52*, 6 (2014), 210–217.

[31] Karakus, M., and Durresi, A. Quality of service (qos) in software defined networking (sdn): A survey. *Journal of Network and Computer Applications 80* (2017), 200–218.

[32] Kim, B. C., Bang, Y., Kim, Y., Lee, J. Y., Kwak, D. G., Lee, H. S., and Ma, J. S. A QoS framework design based on DiffServ and SNMP for tactical networks. In *IEEE Military Communications Conference (MILCOM)* (2008), IEEE, pp. 1–7.

[33] Lantz, B., Heller, B., and McKeown, N. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (2010), pp. 1–6.

[34] Lee, K., Kwon, B., Kang, J., Heo, S., and Lee, S. Optimal flow rate control for SDN-based naval systems. *IEEE Transactions on Aerospace and Electronic Systems 53*, 6 (2017), 2690–2705.

[35] Loevenich, J., Lopes, R. R. F., Rettore, P. H., Eswarappa, S. M., and Sevenich, P. Maximizing the probability of message delivery over ever-changing communication scenarios in tactical networks. *IEEE Networking Letter* (March 2021), 1–5. early access, doi:10.1109/LNET.2021.3066536.

[36] Lopes, R. R. F., Balaraju, P. H., and P.Sevenich. Creating ever-changing QoS-constrained dataflows in tactical networks: An exploratory study. In *International Conference on Military Communications and Information Systems (ICMCIS)* (Budva, Montenegro, May 2019), pp. 1–8.

[37] Lopes, R. R. F., Balaraju, P. H., Rettore, P. H., and Sevenich, P. Queuing over ever-changing communication scenarios in tactical networks. *IEEE Transactions on Mobile Computing* (June 2020), 1–15.

[38] Lopes, R. R. F., Balaraju, P. H., and Sevenich, P. Creating and handling ever-changing communication scenarios in tactical networks. In *15th International Conference on the Design of Reliable Communication Networks (DRCN)* (Coimbra, Portugal, March 2019), pp. 67–74.

[39] Lopes, R. R. F., Loevenich, J., Rettore, P. H., Eswarappa, S. M., and Sevenich, P. Quantizing radio link data rates to create ever-changing network conditions in tactical networks. *IEEE Access* (September 2020), 1–20.

[40] Łubkowski, P., Hauge, M., Landmark, L., Barz, C., and Sevenich, P. On improving connectivity and network efficiency in a heterogeneous military environment. In *International Conference on Military Communications and Information Systems (ICMCIS)* (2015), IEEE, pp. 1–9.

[41] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[42] Nobre, J., Rosario, D., Both, C., Cerqueira, E., and Gerla, M. Toward software-defined battlefield networking. *IEEE Communications Magazine 54*, 10 (2016), 152–157.

[43] Palma, D., Goncalves, J., Sousa, B., Cordeiro, L., Simoes, P., Sharma, S., and Staessens, D. The queuepusher: Enabling queue management in openflow. In *Third European Workshop on Software Defined Networks* (2014), pp. 125–126.

[44] Pfaff, B., and Davie, B. RFC 7047 - The Open vSwitch Database Management Protocol, 2013.

[45] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2015), pp. 117–130.

[46] Phemius, K., Seddar, J., Bouet, M., Khalifé, H., and Conan, V. Bringing SDN to the edge of tactical networks. In *IEEE Military Communications Conference (MILCOM)* (2016), IEEE, pp. 1047–1052.

[47] Poularakis, K., Iosifidis, G., and Tassiulas, L. Sdn-enabled tactical ad hoc networks: Extending programmable control to the edge. *IEEE Communications Magazine 56*, 7 (2018), 132–138.

[48] Scott, K., Refaei, T., Trivedi, N., Trinh, J., and Macker, J. P. Robust communications for disconnected, intermittent, low-bandwidth (DIL) environments. In *IEEE Military Communications Conference (MILCOM)* (Nov 2011), pp. 1009–1014.

[49] Tronarp, O. Quality of service in tactical ad hoc networks by priority queuing. Master's thesis, Institutionen för systemteknik, Linköping University, Sweden, October 2004. Accessed: 22-01-2021.

[50] Zacarias, I., Gaspary, L. P., Kohl, A., Fernandes, R. Q., Stocchero, J. M., and de Freitas, E. P. Combining software-defined and delay-tolerant approaches in last-mile tactical edge networking. *IEEE Communications Magazine 55*, 10 (2017), 22–29.

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 31

C2 Command and Control. 2–5, 7, 17, 37, 47, 51

DiffServ Differentiated Service. 11–13, 15, 16, 18, 26, 39, 40

DMB DSCP Marking Information Base. 16

dpid datapath identifier. 31, 32

DRR Deficit Round Robin. 28

DSCP Differentiated Services Code Point. 9, 11, 13, 30, 32, 34, 51

DSCP Differentiated Service Code Point. 16–18

E2E End-to-End. iv, 1, 16, 17, 39–42, 51, 58

ETS Enhanced Transmission Selection. 4, 28, 29, 39, 41–43, 47, 52, 57, 58

FFT Friendly Force Tracking. 21, 27, 29, 30, 42, 47

FIFO First-In-First-Out. 27

FPQ Fixed Priority Queue. 16, 18

HF High Frequency. iv, 1, 7, 22

HTB Hierarchy Token Bucket. 4, 24–29, 38, 40, 43, 46, 47, 52, 57

HTTP Hypertext Transfer Protocol. 31, 33

i.e That is. 10, 11, 19, 20, 24, 32, 37, 38, 40, 41

IETF The Internet Engineering Task Force. 12

IntServ Integrated Service. 12, 13

IP Internet Protocol. iv, 2, 9, 12, 13, 19–21, 24, 30–32, 39–42, 58

IPv4 Internet Protocol version 4. 13, 14, 59

kbps kilobits per second. 11, 22, 31, 40, 46

kms kilometers. 22

LLDP Link Layer Discovery Protocol. 30

MAC Medium Access Control. 2, 20, 30

MANETs mobile ad hoc networks. 7

Meter ID Meter Identifier. 11

MGEN Multi-Generator. 4, 21, 24, 37

ms milliseconds. 27, 29

NA Not Applicable. 18

netem Network Emulation. 27, 29

NSS Naval Ship System. 18

ONF Open Networking Foundation. 8, 57

OVS Open vSwitch. 9–12, 15, 19, 20, 30–34, 42–44, 47, 57–59

OVSDB Open vSwitch Database. 9, 11, 12, 15, 31, 32

PFIFO Priority First-In-First-Out. 26–30, 38, 40, 52, 57

pktps packets per second. 11

Qdisc Queuing Discipline. 4, 23–30, 38–41, 43, 46, 47, 52, 57, 58

Qdiscs Queuing Disciplines. iv, 4, 20, 22–29, 38, 40, 42, 43, 46, 47, 57

QoS Quality-of-Service. iv, 1–5, 7, 11–21, 23, 26–28, 30–32, 34, 35, 37–39, 47–49, 51, 52, 57, 58

REST Representational State Transfer. 5, 12, 15, 20, 30–35, 39, 45, 46, 57

RPC Remote Procedure Calls. 5, 39

RPC Remote Procedure Call. 32, 33, 57

RSVP Resource ReSerVation Protocol. 13

SatCom Satellite Communications. iv, 1, 5, 7, 17, 20–25, 27, 30–33, 38–41, 43, 44, 51, 57, 58

SDN Software-Defined Networking. iv, 1–5, 7–12, 14–20, 30–32, 37, 39, 43, 46, 51, 52, 57, 59

SNMP Simple Network Management Protocol. 16, 18

TBF Token Bucket Filter. 23, 24, 57

TC Traffic Control. 23, 25, 26, 28, 29, 57

tc Linux Traffic Control. 16

TCP Transmission Control Protocol. 2, 12

TENs Tactical Edge Networks. 7, 17, 18

TN Tactical Network. 1, 20, 22

TNs Tactical Networks. iv, 1–4, 7, 12, 13, 19, 20, 51

ToE Time-of-Expiry. 5, 21, 30, 34, 35, 37, 39, 40, 47–49, 51, 52, 57, 58

ToS Type-of-Service. 4, 13, 14, 21, 23, 24, 27, 38, 59

UAVs Unmanned Aerial Vehicles. 7, 22

UDP User Datagram Protocol. 2, 32, 38

UHF Ultra High Frequency. iv, 1, 5, 7, 17, 20–25, 27, 30–33, 38–44, 47, 49, 51, 57, 58

URL Uniform Resource Locator. 31

veth virtual Ethernet. 20

VHF Very High Frequency. iv, 1, 5, 7, 17, 20–25, 27, 30–33, 38–48, 51, 57, 58

VLAN Virtual Local Area Network. 14

WFQ Weighted Fair Queuing. 16, 18

WSGI Web Server Gateway Interface. 33