NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

MSc THESIS

Adaptive UxV Routing Based on Network Performance

ATHANASIOS D. CHALVATZARAS

Supervisor: **Stathes P. Hadjiefthymiades,** Professor

ATHENS

August 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προσαρμοστική Δρομολόγηση μη Επανδρωμένων Οχημάτων με Βάση την Απόδοση του Δικτύου

ΑΘΑΝΑΣΙΟΣ Δ. ΧΑΛΒΑΤΖΑΡΑΣ

Επιβλέπων: **Ευστάθιος Π. Χατζηευθυμιάδης,** Καθηγητής

ΑΘΗΝΑ

Αύγουστος 2019

MSc THESIS

Adaptive UxV Routing Based on Network Performance

ATHANASIOS D. CHALVATZARAS
S.N.: M1611

SUPERVISOR: **Stathes P. Hadjiefthymiades,** Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προσαρμοστική Δρομολόγηση μη Επανδρωμένων Οχημάτων με Βάση την Απόδοση του Δικτύου

ΧΑΛΒΑΤΖΑΡΑΣ Δ. ΑΘΑΝΑΣΙΟΣ
Α.Μ.: Μ1611

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ευστάθιος Π. Χατζηευθυμιάδης,** Καθηγητής

ABSTRACT

Robotics and Internet of Things (IoT) have been experiencing rapid growth nowadays. IoT nodes are significantly enhanced with many different features. One of the most important is the mobility capabilities, given by the noticeably huge growth of UxV (UxVs-x stands for a different type of environment, i.e. ‘s’ stands for sea, ‘a’ for air and ‘g’ for ground) area. The idea is the assumption of a drone as a mobile sensor, that can be deployed wherever the experimenter wants. Some more characteristics that make the unmanned vehicles a very tempting decision as IoT nodes are the decision-making ability without human interaction, endurance, re-programmability and capability of multimedia streaming. These characteristics make drones an option for use cases of surveillance, security monitoring, and supporting crisis management activities. For instance, a UGV equipped with a high-definition camera and running an algorithm of object recognition can serve the purpose of border surveillance.

In this thesis, a framework that implements a network quality based decision-making process is developed. This framework adapts the information flow between the UxV and the Ground Control Station (GCS) based on network quality metrics (such as packet error rate etc.) and the principals Optimal Stopping Theory (OST). The goal of this framework is to ensure the optimal delivery of critical information from UxV to GCS and vice-versa. If the network behaves optimally then there is no limitation on the information flow, but if the network is saturated or overloaded restriction rules are applied. The proposed model introduces two optimal stopping time mechanisms based on change detection theory and a discounted reward process.

To support the implemented framework, an experimental environment has been set up and also a series of experiments with very promising results. As a mobile IoT node, a TurtleBot has been used, along with an XBOX Kinect sensor (RGB camera and depth sensor) and a Raspberry Pi running Robotic Operating System (ROS) and Apache Kafka pub-sub system with ultimate purpose the communication between the TurtleBot and the GCS.

SUBJECT AREA: Robotics, Decision Making, IoT

KEYWORDS: Robotics, Network reliability, Decision making, IoT, Optimal Stopping Theory, Unmanned Vehicles

ΠΕΡΙΛΗΨΗ

Μια μεγάλη και απότομη εξέλιξη παρατηρείται σήμερα στον τομέα της ρομποτικής και του διαδικτύου των πραγμάτων. Οι κόμβοι που αποτελούν την κύρια υποδομή του διαδικτύου των πραγμάτων έχουν εμπλουτιστεί με σημαντικές και πολυποίκιλες δυνατότητες. Η πιο σημαντική από αυτες τις δυνατότητες είναι η κινητικότητα, η οποία έχει προσφερθεί λόγω της επίσης σημαντικής εξέλιξης του τομέα που αφορά τα μη επανδρωμένα οχήματα. Ένα μη επανδρωμένο όχημα μπορεί να εξυπηρετήσει έναν ερευνητή ως κινητός αισθητήρας (θερμοκρασίας, πίεσης νερού) και να τοποθετηθεί σε οποιαδήποτε δυνατή τοποθεσία. Κάποια ακόμα χαρακτηριστικά που κάνουν δελεαστική την επιλογή μη επανδρωμένων οχημάτων ως κόμβους του διαδικτύου των πραγμάτων είναι η ικανότητα της λήψης αποφάσεων χωρίς την ανθρώπινη παρέμβαση, η αντοχή, η επαναπρογραμματισιμότητα καθώς και η δυνατότητα της ζωντανής ροής πολυμέσων. Με βάση αυτά τα χαρακτηριστικά τα μη επανδρωμένα οχήματα μπορούν να χρησιμοποιηθούν επίσης σε περιπτώσεις εποπτείας χώρων και συνόρων, παρακολούθηση καμερών ασφαλείας καθώς και για υποστήριξη σε περιπτώσεις διαχείρισης κρίσεων. Για παράδειγμα ένα μη επανδρωμένο όχημα ξηράς όπου φέρει μία υψηλής ευκρίνειας κάμερα, σε συνδυασμό με έναν αλγόριθμο αναγνώρισης αντικειμένων μπορεί να χρησιμοποιηθεί για επόπτεια συνόρων.

Σε αυτήν την διπλωματική εργασία προτείνεται ένα πλαίσιο, στο οποίο υλοποιείται μια διαδικασία λήψης αποφάσεων με βάση την ποιότητα του δικτύου. Το πλαίσιο αυτό προσαρμόζει την ροή της πληροφορίας μεταξύ του επανδρωμένου οχήματος και του σταθμού ελέγχου, βασισμένο σε μετρικές ποιότητας του δικτύου (όπως το ρυθμό απώλειας πακέτων) και στις αρχές της Θεωρίας Βέλτιστης Παύσης, με σκοπό να εξασφαλίσει το βέλτιστο ποσοστό παραλαβής πληροφοριών υψίστης σημασίας από το μη επανδρωμένο όχημα προς το σταθμό ελέγχου και το αντίστροφο. Όταν το δίκτυο συμπεριφέρεται άριστα δεν υπάρχει περιορισμός στην ροή πληροφοριών, αλλα έαν το δίκτυο είναι είτε υπερφορτωμένο, είτε κορεσμένο, τότε εφαρμόζονται περιοριστικοί κανόνες. Το προτεινόμενο μοντέλο, εισάγει δύο μηχανισμούς βέλτιστης παύσης βασισμένος στην Θεωρίας Βέλτιστης Παύσης, στη Θεωρία Ανίχνευσης Αλλαγής Κατεύθυνσης καθώς και σε μία διαδικασία εκπτωτικής ανταμοιβής.

Για την υποστήριξη του υλοποιημένου πλαισίου, έγινε μία σειρά πειραμάτων με πολύ υποσχόμενα αποτελέσματα. Σαν κινητός κόμβος χρησιμοποιήθηκε ένα ρομπότ TurtleBot, μαζί με ένα XBOX Kinect που έφερε μία έγχρωμη κάμερα και έναν αισθητήρα βάθους καθώς και με ένα Raspberry Pi, το οποίο εκτελούσε το Robotic Operating System (ROS) και το σύστημα Apache Kafka, με σκοπό να γεφυρώσει το χάσμα επικοινωνίας μεταξύ TurtleBot και σταθμού ελέγχου.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ρομποτική, Λήψη αποφάσεων, Δικτύωση των Πραγμάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ρομποτική, Αξιοπιστία Δικτύου, Λήψη Αποφάσεων, Δικτύωση των Πραγμάτων, Θεωρία Βέλτστης Παύσης, Μη Επανδρωμένα Οχήματα

Η εργασία αυτή αφιερώνεται στην οικογένεια μου και ιδιαίτερα στον παππού μου και τη γιαγιά μου, για την μόνιμη και ανιδιοτελή στήριξη τους κατά την διάρκεια των σπουδών μου.

ΕΥΧΑΡΙΣΤΙΕΣ

Για την διεκπαιρέωση της διπλωματικής αυτής εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Ευστάθιο Χατζηευθυμιάδη που μου έδωσε τα κίνητρα, την στήριξη και την ευκαιρία να ασχοληθώ με τον τομέα του pervasive computing. Επίσης θα ήθελα να ευχαριστήσω την υποψήφια διδάκτωρ Κυριακή Παναγίδη για την απεριόριστη βοήθεια και συμβολή της. Χωρίς αυτή η εκπόνηση αυτή της εργασίας θα ήταν αδύνατη.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In the last decade, we have been witnessing significant advancements and evolution of the Internet of Things (IoT). Going a step further to the IoT infrastructure nodes are enhanced with mobility capabilities forming the mobile IoT networks and especially huge growth can be noticed at unmanned vehicles area. One can assume a drone as a mobile sensor node deployed to different locations. Other characteristics that make unmanned vehicles (UxVs- x stands for different type of environment, i.e. ‘s’ for sea, ‘a’ for air and ‘g’ for ground) popular are the ability to make decisions without human intervention capable of carrying additional payloads, the endurance, re-programmable and capacity to stream multimedia content. As unmanned vehicles and especially drones become more advanced, they present greater value especially in use cases of surveillance, security monitoring, and supporting crisis management activities. For instance, consider the use case of UxVs equipped with a video camera and air-quality sensors to recognize objects in real-time including to cruise over forests and to spot fires early.

For example, let’s assume a ground unmanned device with video steam capabilities recognizing objects in real-time is sent to explore a disaster area from wildfire and to spot new outbreaks of fire. This device is equipped with a camera, a connection interface, a GPS, various sensors like thermostats and a back-end collection service. The question is how is this device going to operate in an unknown area while it ensures the successful execution of a mission? A mission is often described as a trajectory with specific way-points in which the vehicle is ordered to approach and gather various measurements from sensors or images from cameras. Can the dedicated round Control Station (GCS) control the device in real-time without risks? A GCS is the terrestrial system, which acts as a coordinator or master node at distance responsible for data acquisition and transmission. The communication between the unmanned vehicle and the GCS is established via wireless communications. A key feature of UxVs is the control of a possible mission. A mission is created by the users and, then, GCS is responsible for the successful execution of the mission autonomously. GCS control messages shall be delivered with a high assurance of low or minimal time delay to enable real-time management, monitoring, control, and feedback loops.

At IoT networks and especially after a disaster the successful delivery of messages cannot be taken for granted. However, telemetry can be divided into main categories: critical information transmission and sensor measurements. Critical information contains the commands sent by GCS and the responses to these commands by UxVs. Critical information requires real-time monitoring and control messages to be delivered with high accuracy and minimal delay, while the connection between GCS and UxV is always alive. In emergency cases, if UxV lost its connection to the base then it usually returns to its initial position. Hence this means that the mission is canceled, even if the device could be close to its end. This leads to waste of resources.

Unmanned vehicle is commanded to operate in an unknown area with no prior use of maps or localization techniques. The trajectories are also dynamically created by users. One possible metric to use of the mission is the quality of the network. Quality of the network has high importance for the mission because significant commands or sensor values can be lost. The quality of the network can be discriminated as proposed in. We can assume that even if control feedback is in high priority, telemetry can be paused for a short time in need of crisis. When the quality of network changes, UxV/GCS can decide on-line to pause the transmission of commands in order not to overload a saturated network or to risk to lose completely the messages. This loss can occur in

unwanted robot's behavior like skipping a trajectory point, or even more devastating results such as aborting the mission and returning to its initial point.

In this thesis, a framework that implements a network quality based decision-making process is developed. This framework adapts the information flow between the UxV and the Ground Control Station (GCS) based on network quality metrics (such as packet error rate etc.) and the principals Optimal Stopping Theory (OST). The goal of this framework is to ensure the optimal delivery of critical information from UxV to GCS and vice-versa. If the network behaves optimally then there is no limitation on the information flow, but if the network is saturated or overloaded restriction rules are applied. The proposed model introduces two optimal stopping time mechanisms based on change detection theory and a discounted reward process.

2. ROBOTIC OPERATING SYSTEM (ROS) AND HARDWARE COMPONENTS

2.1 Robotic Operating System (ROS)

ROS is an open source operating system for robots [5] [6], but not in the traditional sense of an operating system regarding process management and scheduling. ROS can be characterized as a meta-operating system. This means that ROS does not replace, but instead works alongside the traditional operating system. Its main goal is to provide communication between a host operating system (e.g. Linux) and a robot (e.g. TurtleBot) and its philosophy is the quick and effective reusability of software on any robot running ROS, with just little changes so the need of reinventing the wheel vanishes.

The growth of interest in robotics combined with the simplicity and robustness of ROS, led to its usage from a lot of research institutions, expanding its popularity. Nowadays, commercial companies already adapting their products, in order to achieve compatibility with ROS, such as sensors and actuators. Every day the number of devices compatible with ROS increases.

Some of the standard operating system facilities that are provided from ROS are hardware abstraction, low-level device control, implementation of commonly used functionalities, message passing between processes, and package management. Architecture of ROS can be described like a graph architecture with centralized topology. This graph is composed of vertices, called ROS nodes (see section 2.1.3.1) where processing takes place and edges, called topics (see section 2.1.3.2). A node can post or/and subscribe to a topic, in order to pass or receive messages respectively. A representation of that graph architecture is illustrated in Figure 1Figure 1. So, nodes exchange messages using a pub/sub communication system (see section 2.1.1).

The ROS architecture has been designed and divided into three sections or levels of concepts:

- The Filesystem level.
- The Computation Graph level.
- The Community level.

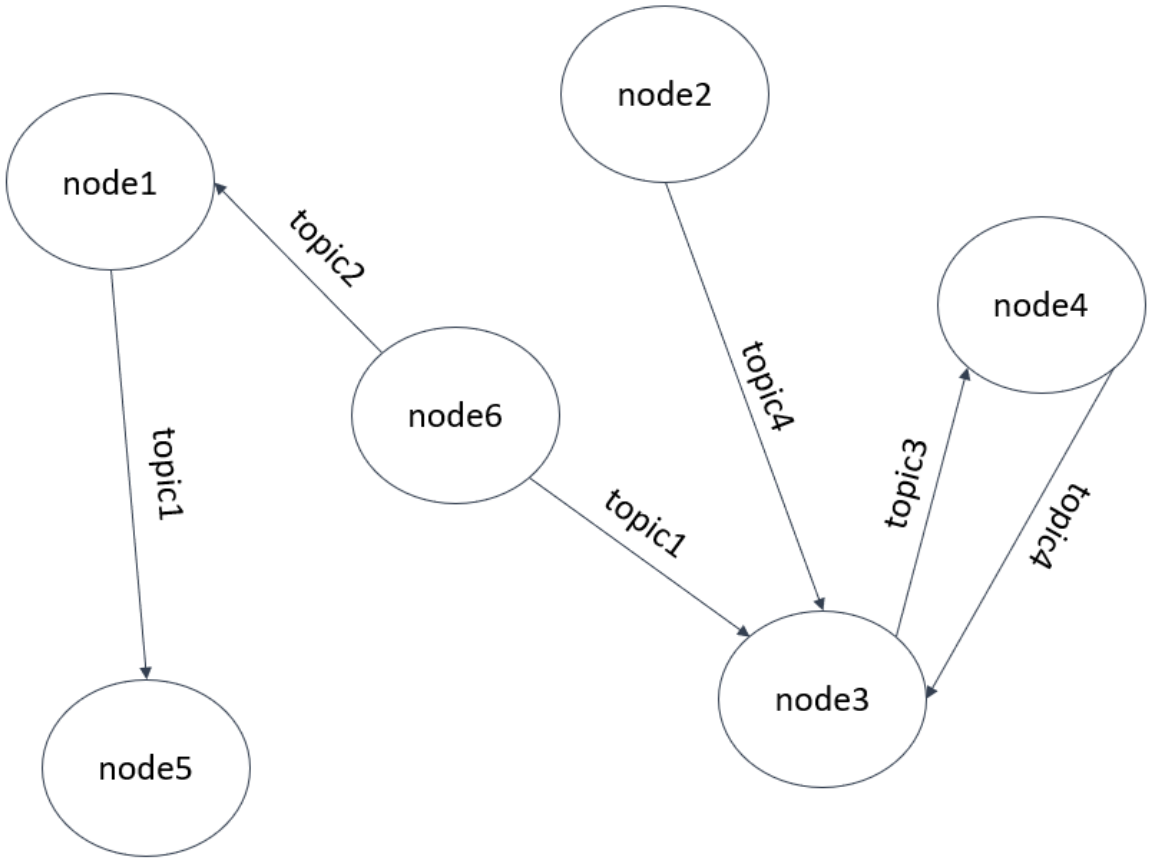

Figure 1: ROS graph architecture

2.1.1. Messaging Publish/Subscribe System Model

Publish-subscribe is a messaging pattern. In this messaging pattern there are producers and consumers. Producers, called publishers, send the messages and consumers, called subscribers, receive the messages. Publishers do not send the produced messages directly to the subscribers, on contrary they categorize these messages into classes being unaware of the existence of subscribers. Pub-sub messaging pattern is similar with message-queue pattern.

One specific subscriber does not need to consume all the messages produced and sent in the system. For this case there is the process of filtering. Two forms of filtering and the most common ones are topic-based filtering and content-based filtering. For the case of this thesis topic-based filtering was applied. In topic-base filtering messages are published on topics. Publishers choose the desired topic to produce their messages and consumers subscribe to any desired topic. A topic can have many producers and many consumers. All of the consumers will receive all messages that are published on this topic.

ROS pub-sub system consists of nodes and topics. Nodes are executables that can communicate with other processes using topics, services, or the Parameter Server. (see section 2.1.3.7) Topics provide this communication between the nodes by transmitting data. These data can be transmitted without a direct connection between nodes, meaning the production and consumption of data are decoupled. A node can subscribe and publish on any desired topic available. A topic can have multiple publishers and subscribers so there is no limitation.

The other publish-subscribe messaging system that was used is Apache Kafka, but more information about it can be found in Chapter 3.

2.1.2. Filesystem Level

Like any traditional operating system, ROS' programs are divided in folders and files. Every file has a type that performs a specific task or action. In the following Figure 2, an abstract representation of the filesystem level is illustrated.

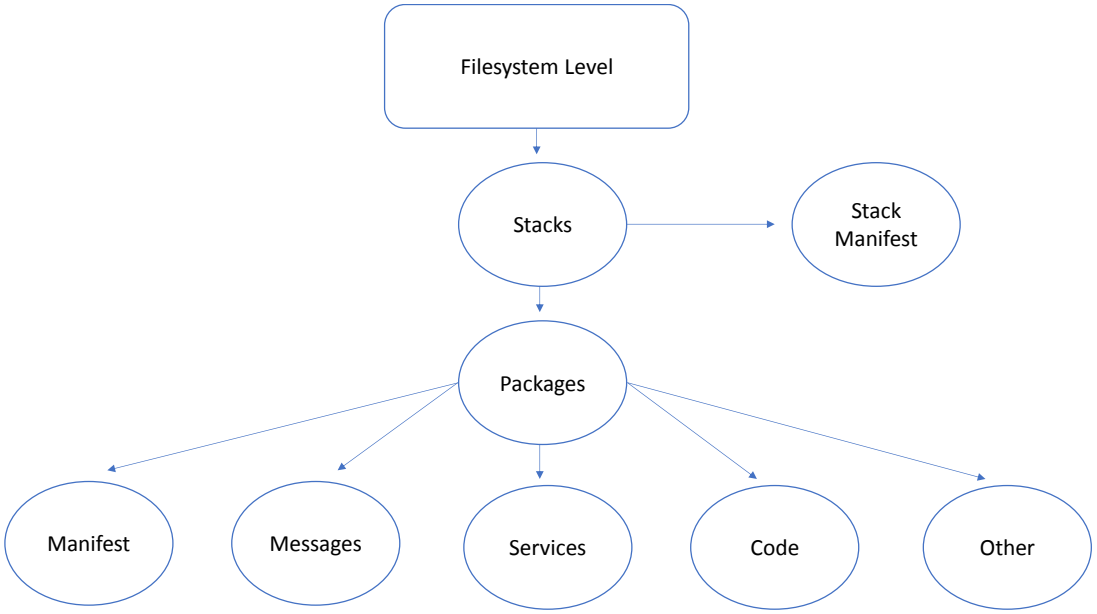

Figure 2: Abstract representation of ROS' filesystem level

2.1.2.1 Stack

A stack is a collection of packages, which co-operate in order to provide a specific functionality. ROS has numerous of default stacks. The most infamous of all being the navigation stack.

2.1.2.2 Stack Manifest

A stack manifest is a file that provides information about a specific stack, like its license information and its dependency on other stacks. Stack manifests usually, if not always follow the xml file format.

2.1.2.3 Packages

Packages are comprising the atomic level of ROS. They provide the minimum structure that a ROS program requires. A ROS program may contain a ROS runtime process (node), a configuration file etc. Packages also provides the benefit of code reusability within ROS. A package can be transferred from one project to another with just minor changes regarding the code of the runtime process and/or the configuration file. The most granular thing you can build and release is a package.

2.1.2.4 Manifests

Management of manifest is coordinated by a file called manifests.xml. A manifest provides a lot of information about a package including dependencies, compiler flags, license information etc.

2.1.2.5 Messages

Messages are used for data and information communication between processes. Every message type has its own structure and fields. ROS has numerous standard types of messages. More about messages on section 2.1.3.3

2.1.2.6 Services

Direct communication between nodes for request and response messages is provided from ROS services. There are no default services provided, and all of them should be created by the user. The source code files are stored in the srv folder.

2.1.2.7 Code

Source code for ROS nodes can be written in the following programming languages: C++, Python and Java. There is a good amount of documentation and support for these three languages, but there are also developers working in order to build support for other languages too.

2.1.2.8 Other

Other files can be any type of files. A video file in order to process it, a text file for the means of input or output, a config file to set the node parameters etc.

An example of the ROS filesystem level is illustrated in Figure 3.

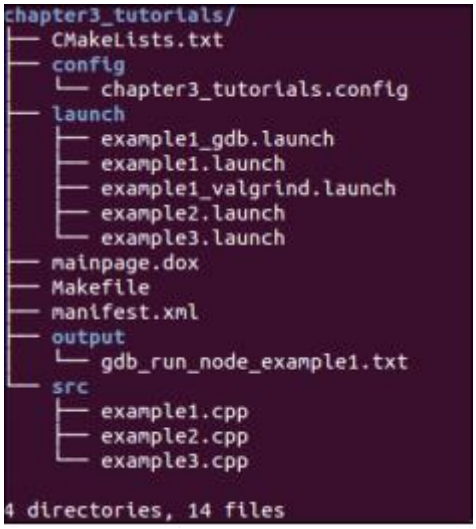

Figure 3: ROS Filesystem Level Example

2.1.3. Computation Graph Level

The second level is the Computation Graph level where communication between processes and systems happens. ROS creates a network where all the processes are

connected. Any node in the system can access this network, interact with other nodes, see the information they are sending, and transmit data to the network. The basic elements in this level are nodes, the Master, the Parameter Server, messages, services, topics, and bags, all of which provide data to the graph in different ways. A representation of this graph is illustrated in Figure 4.

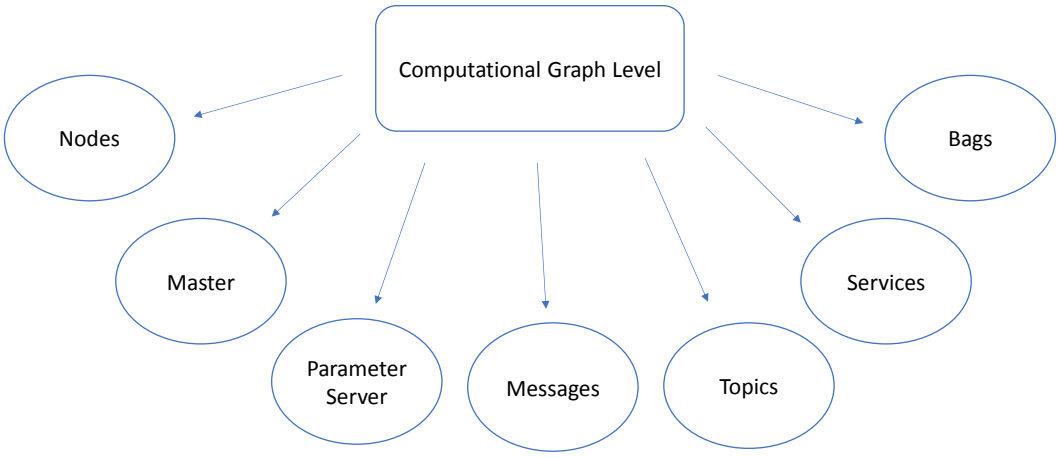

Figure 4: ROS Computational Graph Level

2.1.3.1 Nodes

Nodes are the most important component of ROS Computational Graph level. They are processes that interact with the ROS network and complete their given tasks. Every node has a unique name that differentiate it from the other nodes. This name is like its unique id, and it is used in order to achieve communication with other nodes via topics. A node can subscribe to a topic, so it can receive information, perform computation, control sensors and actuators, and publish data to topics for other nodes to use. Given that each node provides a specific functionality, it is wiser to have many nodes to control different functions rather that a big single node. This aspect is crucial for fault tolerance and code reusability. Nodes are written with a ROS client library. This library supports many programming languages, with more important ones being: python, java and C++.

2.1.3.2 Topics

Topics are unique named buses used for data transmission between nodes. Nodes can publish or subscribe to a topic, but data production and consumption are decoupled meaning that there is an indirect connection between the nodes. Every topic should have a unique name in order to avoid problems and confusion between same named topics. Finally, each topic is related to a message type, meaning that it only accepts specific message type. The connection between topics and nodes is depicted in Figure 1.

2.1.3.3 Messages

ROS messages are stored in .msg files and are used by nodes in order to publish information on topics. Each message file defines a message type that uses the standard ROS naming convention: the name of the package followed by the /, and the name of the .msg file. ROS has numerous predefined types of messages. An example of a message file is illustrated in Figure 5.

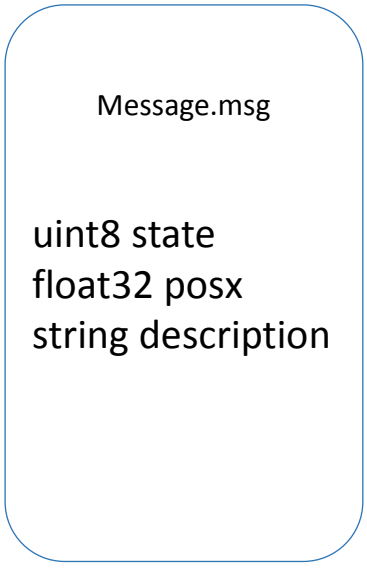

Figure 5: ROS message Example

As one can read from the figure Mesasge.msg type contains an 8-byte sized unsinged integer named state, a 32-byte sized float named posx, and a string named description.

In ROS, you can find a lot of standard types to use in messages as shown in the following Table 1:

Table 1: ROS standard message types

Primitive Type	Serialization	C++	Python2	Python3
bool	unsigned 8-bit int	uint8_t	bool	
int8	signed 8-bit int	int8_t	int	
uint8	unsigned 8-bit int	uint8_t	int	
int16	signed 16-bit int	int16_t	int	
uint16	unsigned 16-bit int	uint16_t	int	
int32	signed 32-bit int	int32_t	int	
uint32	unsigned 32-bit int	uint32_t	int	
int64	signed 64-bit int	int64_t	long	int
uint64	unsigned 64-bit int	uint64_t	long	int
float32	32-bit IEEE float	float	float	
float64	64-bit IEEE float	double	float	
string	ascii string	std::string	str	bytes
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time	
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration	

A special type in ROS is Header. This is used to add the timestamp, frame, and so on. This allows messages to be numbered so that we can know who is sending the message. Other functions can be added, which are transparent to the user but are being handled by ROS.

The header type contains the following fields:

- uint32 seq
- time stamp
- string frame_id

2.1.3.4 Services

ROS services are stored in .srv files. They are similar with topics in matter of format but they have a crucial difference regarding functionality. Topics can provide many-to-many communication whereas services provide one-to-one communication. One more difference is that all services should be created by the user and there are no default services in ROS. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call.

2.1.3.5 Bags

A primary feature of a well-designed ROS system that we will utilize is that the parts of the system that consume information do not care about the mechanism used to produce that information. A good subscriber node will work any time the messages it needs are

being published without any knowledge of which other node or nodes is publishing them. Bags are files created by ROS in the. bag format to save all the information of the messages, topics, services and others. Bags can be used later to process, analyze and visualize the flow of messages. Bags are created utilizing the rosbag tool, which subscribes to one or more ROS topics and stores message data as they are received. The bag file can be reproduced in ROS like a real session, sending the topics at the same time with the same data. This allows us to run the robot itself a few times, record the topics we need and then replay the messages on those topics many times in the simulation environment, experimenting with the software that processes those data.

2.1.3.6 Master

Master provides an application program interface (API), a set of routines and protocols, tracks publishers and subscribers and services. It also is the domain name system server, which stores topic's and services registration information for ROS nodes. But Master's main role is to guide individual ROS nodes in order to locate one another so they can establish a peer-to-peer communication between them.

2.1.3.7 Parameter Server

Parameter Server is a shared, multivariable dictionary that is accessible via a network. Nodes use this server to store and retrieve parameters at runtime. The ROS Parameter Server is implemented using XML-RPC (a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism), which means that its API is accessible via normal XML-RPC libraries. The Parameter Server uses XML-RPC data types for parameter values, including the following:

- 32-bit integers
- Booleans
- Strings
- Doubles
- ISO 8601 dates
- Lists
- Base 64-encoded binary data

A user can also set a parameter from the command line by using:

rosparam set <param_name> <param_value>

2.1.4. ROS Community Level

The third level is the Community level which consists of ROS resources that enable separate communities to exchange software and knowledge. These resources include ROS distributions, repositories, the ROS wiki and mailing lists.

- *Distributions:* Similar to the Linux distribution, ROS distributions are a collection of versioned meta packages that we can install. The ROS distribution enables easier installation and collection of the ROS software. The ROS distributions maintain consistent versions across a set of software.
- *Repositories:* ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.

- *The ROS Wiki:* The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.
- *Bug ticket system:* If we find a bug in the existing software or need to add a new feature, we can use this resource.
- *Mailing lists:* The ROS-users mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about the ROS software.

2.1.5. Sensors

Sensors are crucial in robotics nowadays. With the use of sensors, a robot can understand and map the environment around it and also it can report more types of data (temperature, GPS location, humidity, brightness, video etc.) back to the user. The more advanced the sensor is, the more accurate the data. ROS every day supports more and more sensors and actuators, not only by the official ROS packages, but also with the help from is community. A list of supported ROS sensors is provided in [7].These supported sensors are divided in categories. Some of them are the above:

1. 1D range finders
2. 2D range finders
3. 3D Sensors (range finders and RGB-D cameras)
4. Audio / Speech recognition
5. Cameras
6. Environmental (like measuring wind speed and direction)
7. Force / Torque / Touch Sensors
8. Motion Capture
9. Pose estimation (GPS / IMU)
10. Power Supply
11. RFID (Radio-frequency identification)

2.1.6. ROS Commands

There are some default commands and tools that helps the user interact with ROS, compile, execute and get the output of his programs etc. In the following Table 2, there is a list of basic ROS commands and also of some commands used in this thesis.

Table 2: ROS Commands

Command	**Description**
roscore	Starts ROS master
rosrun <pkg_name> <node_name>	Starts executable node
rosrun turtlesim turtlesim_node	Starts simple movement simulator
rosrun turtlesim turtle_teleop_key	Control the TurtleBot movement with arrow keys

rospack list	Lists all installed ROS packages
rospack find <pkg_name>	Prints file path to package
rosls <pkg_name>	Lists all files in package directory
roscd <pkg_name>	Allows you to change directories using a package name, stack name, or special location
rosnode info <node_name>	Display information about a node, including publications and subscriptions.
rosnode kill <node_name>	Kill one or more nodes by name. It is not guaranteed to succeed. If a node is hung or set to “respawn” in roslaunch, it may either fail to die or may quickly reappear.
Rosnode cleanup	Purge the registration of any node that cannot be contacted immediately. Prints list of unreachable nodes which has to be confirmed. It can potentially unregister functioning nodes.
rqt_graph	Provides a GUI plugin for visualizing the ROS computation graph.
rostopic echo </topic_name>	Prints topic messages to screen
Rostopic info </topic_name>	Provides data on topic such as type, subscribers and publishers.
rostopic list	Lists all active topics
rosbag record -O <filename></topic>	Starts rosbag tool to record data from a desired topic.
roslaunch turtlebot_bringup minimal.launch	Starts the communication between the TurtleBot and the ROS slave, in this case a Raspberry Pi.
roslaunch turtlebot_navigation amcl_demo.launch map_file:=/home/pi/Desktop/little_map.yaml	Loads a map, creating the TurtleBot’s space.
roslaunch turtlebot_rviz_launchers view_navigation.launch --screen	Opens a navigation simulation on the workstation.
roslaunch turtlebot_navigation gmapping_demo.launch	Starts the mapping process.
rosrun map_server map_saver -f /home/thanos/Desktop/thesis/maps/basement	Saves the created map.

roslaunch turtlebot_teleop ps3_teleop.launch	Starts the teleoperation using a PS3 controller.

2.2 Main Hardware and Software Components

For this thesis, it was critical to execute a series of experiments. A mobile network node was needed, with capabilities of posting sensor data and network data extracted from its link with the access point. Our choice was TurtleBot.

2.2.1. TurtleBot

TurtleBot [1] is a low-cost, personal robot kit with open source software. TurtleBot [1] was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. The model which was utilized in this thesis had a built in XBOX Kinect sensor. XBOX Kinect is equipped with a depth sensor, and an RGB camera. XBOX Kinect was required for implementing SLAM mapping (see section 2.2.7), in order to create a map (world) for the TurtleBot, so it can navigate autonomously.

Technical specifications of TurtleBot:

- Dimensions: 354 x 354 x 420 mm (14 x 14 x 16.5 in.)
- Weight: 6.3 kg (13.9 lbs.)
- Max Payload: 5 kg (11 lbs.)
- Speed and Performance
- Max Speed: 0.65 m/s (25.6 in./s)
- Obstacle Clearance: 15 mm (0.6 in.)
- Drivers and APIs: ROS

In the following Figure 6, Figure 7 and Figure 8 some more technical specifications are presented. The images were downloaded from the official TurtleBot web page [1] . Figure 9Figure 9 is a picture of the TurtleBot that was used for the experiments of this thesis.

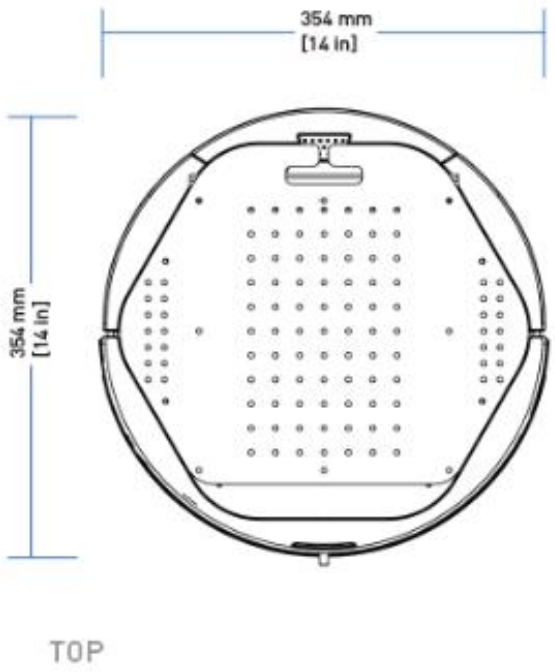

Figure 6: Upper View of TurtleBot

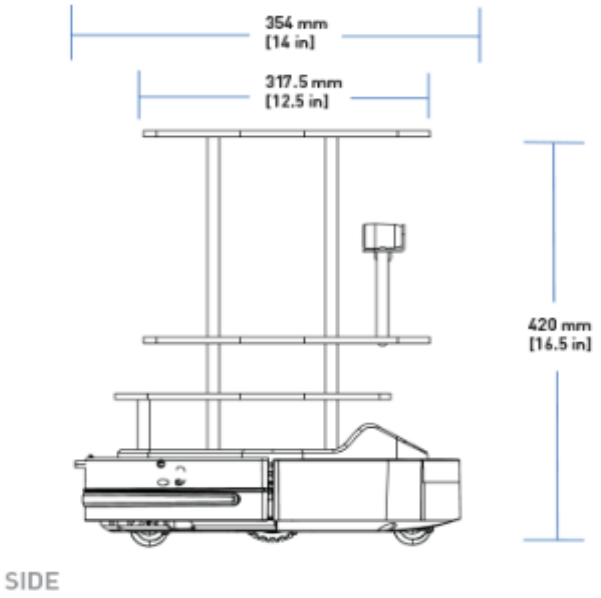

Figure 7: Side View of TurtleBot

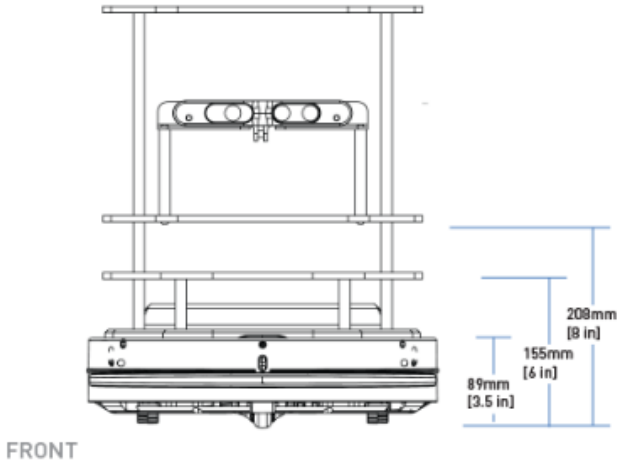

Figure 8: Front View of TurtleBot

Figure 9: Live TurtleBot used in this thesis

This figure presents the TurtleBot used. There are 3 shelves, the middle one has mounted the XBOX 360 Kinect used to map the area of the experiment and on the top shelf lies a Raspberry Pi model 3B.

2.2.2. Raspberry Pi - PlayStation 3 Controller

TurtleBot uses ROS, but ROS does not run on the TurtleBot, but usually on a laptop or netbook that lies on its top shelf. The laptop or netbook is connected with TurtleBot's base via USB cable to transfer the commands from ROS to the its hardware. Laptops with high performance are usually heavy and big in size for TurtleBot usage, so the option is almost always a netbook. Conventional low-cost netbooks do not have the performance required and cause performance problems. The solution is a high-cost netbook, but also a Raspberry Pi.

Figure 10: Raspberry Pi Model 3 B with case, wi-fi USB adapter and PlayStation3 controller.

The Raspberry Pi [2] is a series of small single-board computers. For the purpose of this thesis a Raspberry Pi Model 3 have been used, along with an SD Class 10 card and a wi-fi USB adapter compatible with Linux.

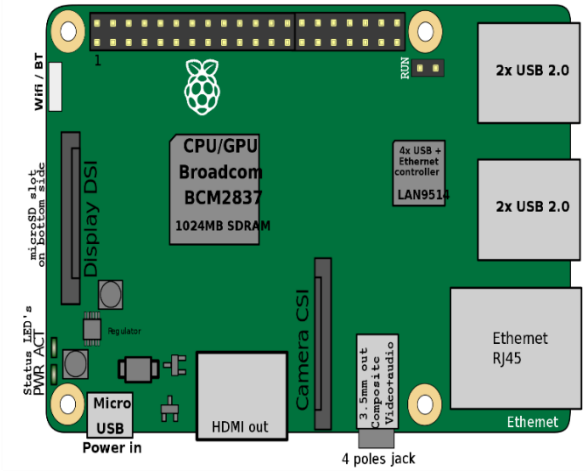

Figure 11: Raspberry Pi Model 3 B abstract architecture schema.

A Playstation3 controller also have been used for moving the TurtleBot around manually, because of its compatibility with ROS. The controller communicates with the Raspberry via Bluetooth. Figure 10 illustrates the aforementioned hardware and Figure 11 gives an abstract representation of Raspberry Pi's 3 architecture. As one can see in Figure 11, Raspberry Pi 3 already has a wi-fi interface, but it does not have the required capabilities to support the case of this thesis.

2.2.3. Ground Control Station (GCS)

A Ground Control Station (GCS) is usually a computer machine inside the field of the experiment, but it can also in this case be anywhere. For the purpose of this thesis, GCS communicates with the Raspberry Pi, passing commands and consuming measurements, feedback etc. The most important task of the GCS, is passing the movement commands to the TurtleBot. This procedure can have failures and delays due to saturation or overloading of the network, so the GCS should have the knowledge when to send a movement command, with a minimum possibility of being dropped as a result of poor network quality. This knowledge comes from monitoring the network and making decisions about pausing the transmission of movement commands with the use of OST. It is assumed that the GCS has strong connection to its base station, thus the network monitoring measurements come from the Raspberry Pi, that measures the link strength between the TurtleBot and its base station, quality of that link and packet loss.

2.2.4. Power Supply

Raspberry Pi is not a common choice for TurtleBot, because there are some critical issues regarding its compatibility with it. The most important of them is powering the Raspberry. One option is high-cost and high-capacity power banks, and the other more viable but harder option is to create a cable that draws power from TurtleBot's battery. Drawing energy from TurtleBot's battery in order to power the TurtleBot has the disadvantage of emptying the battery faster. On the other hand, the advantage is that energy consumption of the TurtleBot's hardware and of the algorithms that are implemented is measured as a whole. For this thesis a cable for this purpose has been created. The cable is connected to an already existent power output port of TurtleBot, that provides power of 5V and 1A with the required pin, and ends up into the micro-USB power supply port of the Raspberry Pi. Figure 12 illustrates this cable.

Figure 12: Cable that powers the Raspberry from TurtleBot's battery

2.2.5. ROS and Ubuntu Version

One more critical issue is that ROS is stable under specific Linux distributions. Two of them are Ubuntu 14.04 and Ubuntu 16.04. Neither of them exists for Raspberry Pi, but with some tweaks a version of Ubuntu Mate 16.04 can be installed on Raspberry Pi Model 3B and Model 3B+. Ubuntu Mate 16.04 supports specific version of ROS called ROS Kinetic Kame, so there is no other choice regarding ROS version.

2.2.6. Wicd

The Raspberry Pi that lies on the top of TurtleBot, should eventually shut down after the execution of specific number of missions, or because TurtleBot's battery is empty and needs recharging. When Ubuntu Mate boots, asks for root password in order to run the network-manager service and finally connect to a desired wireless network. For purposes of convenience and generality, every time that Raspberry turns on, should automatically connect to the testbed's base station or hotspot. This can be done with the use of Wicd.

Wicd, which stands for Wireless Interface Connection Daemon is an open source network manager for Linux and it provides a simple network connection interface.

The main advantage of Wicd is that bypasses the need to login as root, because it starts running as a service before any user logs in. So, it is sufficient just the one-time configuration of Wicd in order to solve the aforementioned problem. When the Raspberry boots up alongside TurtleBot, connects to the specified network automatically without having to login.

2.2.7. Simultaneous Localization and Mapping (SLAM)

TurtleBot should receive movement commands in order to approach the given trajectory's points and finally reach the goal point. Hence TurtleBot has to recognize the space around it. In other words, it needs a map of the experimental area. ROS provides the capability of mapping the TurtleBot's space using the XBOX 360 Kinect connected on it. Kinect has a depth sensor that allows it to sense how far an object is, and if it is combined with moving the TurtleBot manually around using the PS3 controller, then there is a complete map of the experimental area. This way ROS creates a map that lays inside the local storage of Raspberry Pi. The map is separated in two files, one yaml formatted file and one pgm file. Figure 13 and Figure 14 illustrate an example of a yaml file and an example of a pgm file respectively.

```
image: /home/pi/Desktop/basement.pgm
resolution: 0.050000
origin: [-12.200000, -12.200000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Figure 13: YAML file example

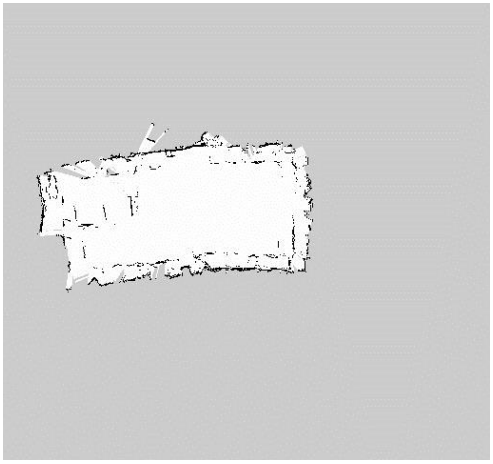

Figure 14: Pgm file example

After creating the map, the workstation can initiate rviz, open the map and take the coordinates of specific points in order to create a desired trajectory. This set of points (trajectory) is given as input in the experiment's source code. Figure 15 illustrates an example rviz instance. Commands can also be given directly from rviz instance but this is not the case of this thesis.

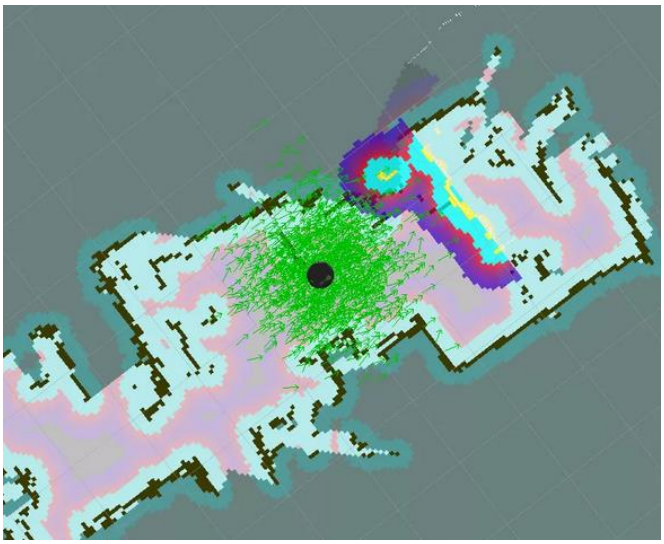

Figure 15: rviz map view example

The technique used for the map creation is called SLAM (Simultaneous localization and mapping). SLAM can be described as the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.

3. APACHE KAFKA

In this thesis two publish-subscribe (pub-sub) messaging systems were used. These two messaging systems are Apache Kafka [3] and ROS pub-sub communication system (see section 2.1.1). Apache Kafka is an open-source stream-processing software platform written in Scala and Java.

Streaming platforms has the following capabilities:

- Similarity to message queues, regarding publishing and subscribing to streams of records. In case of Kafka topics.
- Durable fault-tolerance.
- Immediate stream processing.

The main advantages of Kafka are that it provides high-throughput, and low latency, regarding real-time data feeds. Storage layer of Kafka is a pub-sub message queue. This means that there are application instances which act as consumers and application instances which act as producers. Producers publish messages to specific Kafka topics and consumers subscribe to these topics in order to consume the messages.

Kafka has a similar directional graph representation as ROS pub-sub system (see section 2.1.1). An overview graph is illustrated in Figure 16.

In this work Kafka was used in order to fill the communication gap between the UxV and the Ground Control Station (GCS) (see section 2.2.3). For a more analytical explanation about the communication abstract introduced please refer to section 6.1.1.

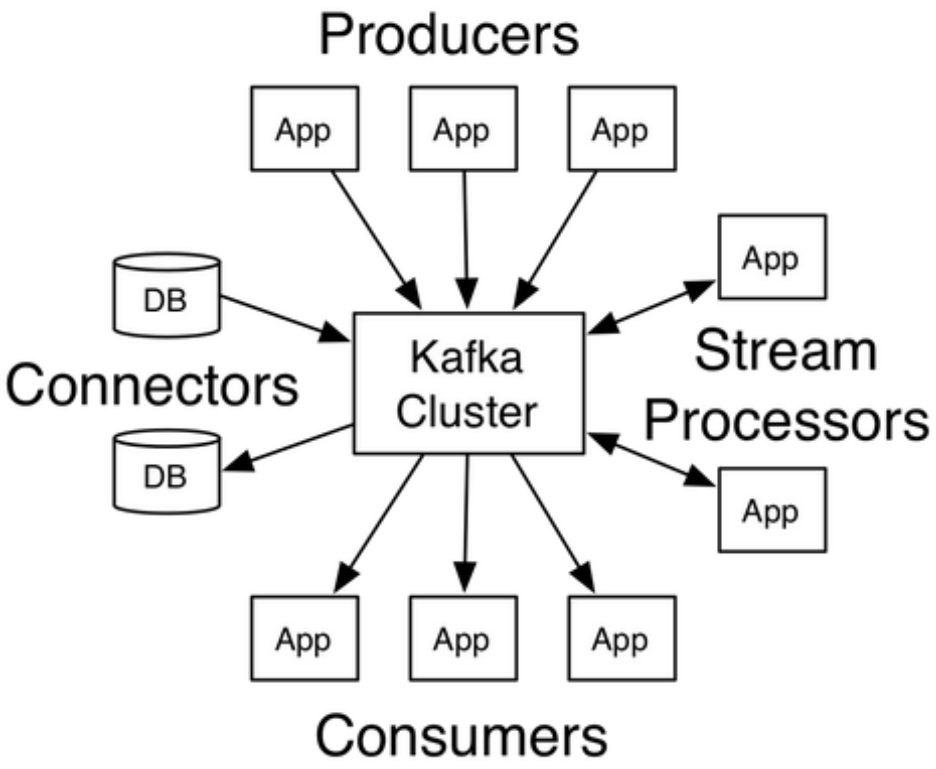

Figure 16: Kafka architecture as directional graph

Examples of application that Apache Kafka can be utilized perfectly:

- Internet of things: Swarms of robots, smart watches, smart TV's, or even personal health monitors can send telemetry data through Apache Kafka.
- Sensor Networks: Areas and complex can be designed with an array of sensors to track data or current status.
- Positional Data: Massive Multiplayer Online games, delivery tracks, robot swarms.
- Other Real-Time Data like satellite data or medical sensor data.

3.1 Ideal Publish-Subscribe System

The idea behind ideal publish-subscribe system is pretty simple, Publisher A's messages should be delivered to Subscriber A's, Publisher B's messages should be delivered to Subscriber B's etc. as it is illustrated in Figure 17. But as it is known in real-world architectures the existence of this system is impossible.

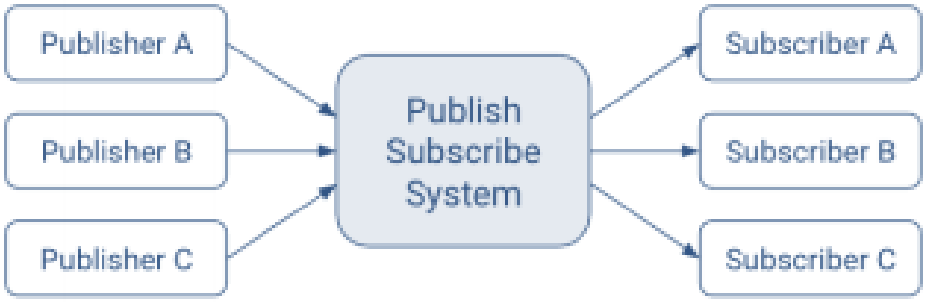

Figure 17: Publish-Subscribe system

The ideal Publish-Subscribe system has the benefit of the following features:

- Unlimited Lookback: A new Subscriber can read any subsequence of the sent messages.
- Message Retention: Message loss is zero.
- Unlimited Storage: An infinite number of messages can be stored.
- No Downtime: The system is never down.
- Unlimited Scaling: Delivery latency is constant, no matter the number of publishers and/or subscribers.

3.2 Apache Kafka Key Characteristics

The key differences of Apache Kafka in comparison with the ideal Publish-Subscribe system are:

- Messaging is implemented on top of a replicated, distributed commit log.
- The functionality of the client is increased.
- Batch optimization instead of individual messages optimization.
- Retention of messages even after consumption, so they can be consumed again.

The results of these design decisions are:

- Extreme horizontal scalability

- Very high throughput
- High availability
- Different semantics and message delivery guarantees

3.3 Topics

Topics are the buses that help messages to find their way from the Publisher to the Subscriber. A topic is a queue of messages written by one or multiple Producers and read by one or multiple consumers. The identification of a topic is provided by the knowledge of its name. In Apache Kafka, publishers are called producers and subscribers are called consumers. In the following Figure 18, one can see an abstract representation of the topics.

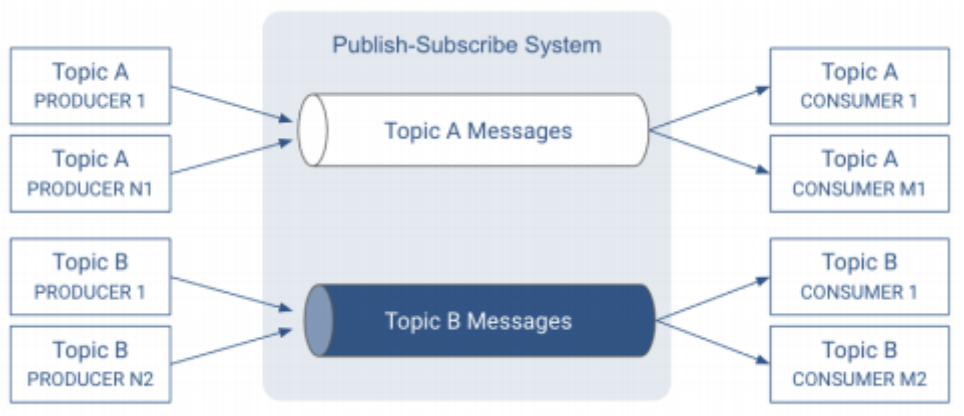

Figure 18: Abstract representation of pub-sub system topics

3.4 Brokers

The design of Apache Kafka offers distributed nature, meaning that Apache Kafka runs on multiple hosts, with one broker per host. There are guarantees that there are no downtime and unlimited scaling, because Apache Kafka ensures that always one host is up and running. Brokers are coordinated by Zookeeper, in order to achieve the goal of unlimited scaling. Furthermore, topics are replicated across brokers, contributing to no downtime, unlimited scaling and message retention goals. As one can extract from the above is that Apache Kafka behaves much like an ideal public-subscribe system. In the following Figure 19, the issues explained in this paragraph are presented.

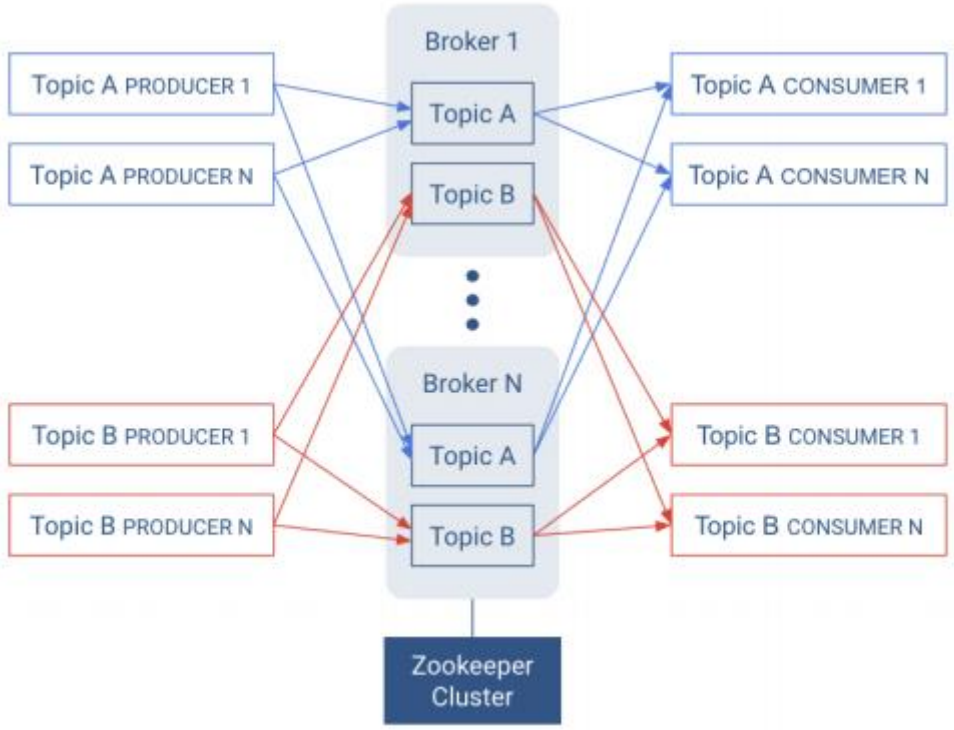

Figure 19: Abstract representation of Apache Kafka Brokers

3.5 Records

In Apache Kafka semiology messages send between the producers and the consumers are called records. Records consist of a key/value pair and metadata including a timestamp. Key is used in order to identify the source of the message, and it is not required. Keys and values are stored as arrays of bytes, and the format is not strict. The metadata of each record can include headers. Headers may store application-specific metadata as key-value pairs. In the context of the header, keys are strings and values are byte arrays.

3.6 Partitions

Apache Kafka divides records into partitions. Partitions can be thought of as a subset of all the records for a topic. In each partition the records are sorted by arrival time. Topics are created by setting the following two parameters:

- Partition count: The number of partition that records will be spread among.
- Replication factor: The number of copies of a partition that are maintained to ensure consumers always have access to the queue of records for a given topic.

Every topic has its leader partition. If the replication factor exceeds one, there will be additional follower partitions. Apache Kafka clients communicate only with leader partition for data. The rest of the partitions act like a fail-safe mechanism, providing redundancy and failover. They are responsible of copying new records from their leader partitions. With N brokers and topic replication factor M:

- If M < N, each broker will have a subset of all the partitions
- If M = N, each broker will have a complete copy of the partitions

In the following Figure 20, there are illustrated two brokers (N = 2) and a replication factor of two (M = 2).

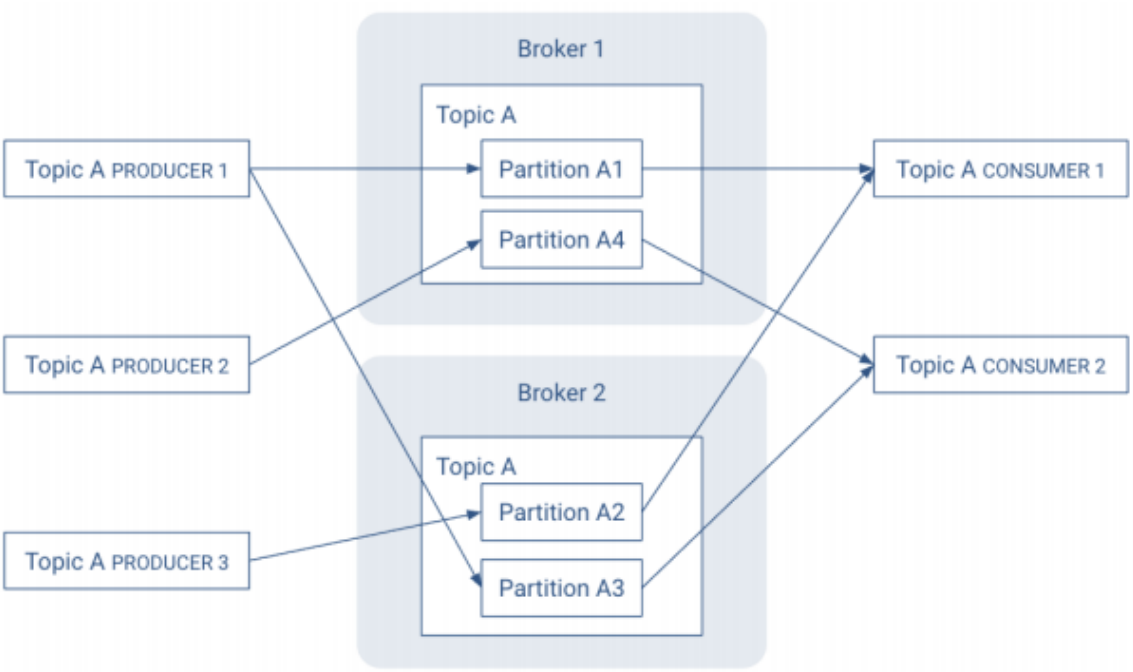

Figure 20: Abstract representation of Apache Kafka partitions

Partitions are the key to keeping good record throughput. Correct amount of partitions and partition replications for a topic has the following benefits:

- Spreads leader partitions evenly on brokers throughout the cluster.
- Makes partitions within the same topic are roughly the same size.
- Load balancing on the brokers.

3.7 Record Order and Assignment

Apache Kafka uses round-robin regarding the assignment of records to partitions. There are no guarantees that multi-partition records will retain their order by production time.

If the order of the records is of high importance, there can be guarantees from the producer that records are sent to the same partition. This can be done by including specific metadata in the record. These metadata are the following:

- The record can indicate a specific partition.
- The record can include an assignment key.

The hash of the key and the number of partitions in the topic determines which partition the record is assigned to. Including the same key in multiple records ensures all the records are appended to the same partition.

3.8 Logs and Log Segments

Inside topics and partitions Apache Kafka stores records in a log structured format. In the following Figure 21, this log structured format is illustrated.

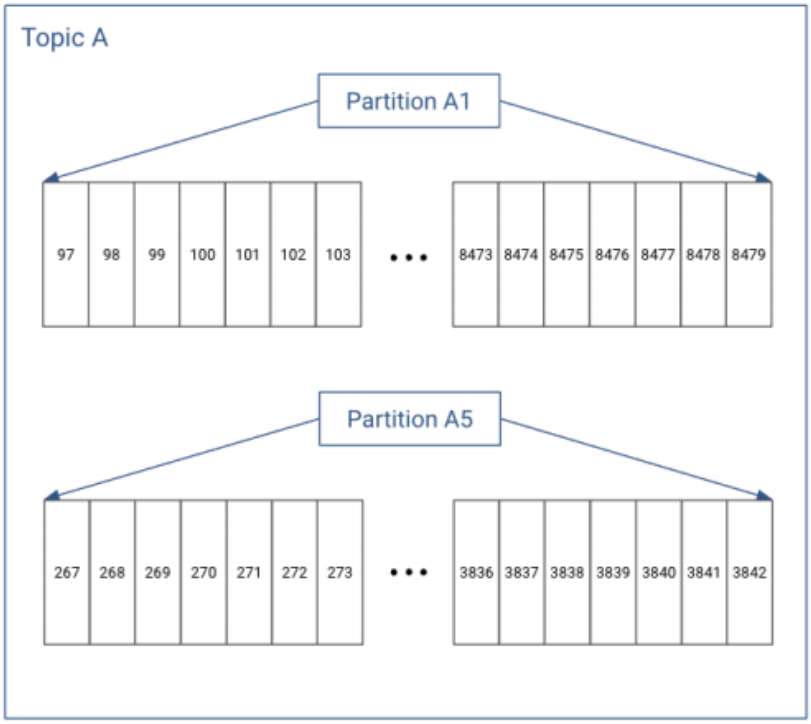

Figure 21: Log structure format of partitions

Actually, not all the records are kept sequentially into a large file, but instead each log is divided in segments, called log segments. Each log segment can be defined using a size limit, a time limit, or both. Each of the partitions is broken into segments, with Segment N containing the most recent records and Segment 1 containing the oldest retained records. This is configurable on a per-topic basis. The following Figure 22 illustrates the above statements.

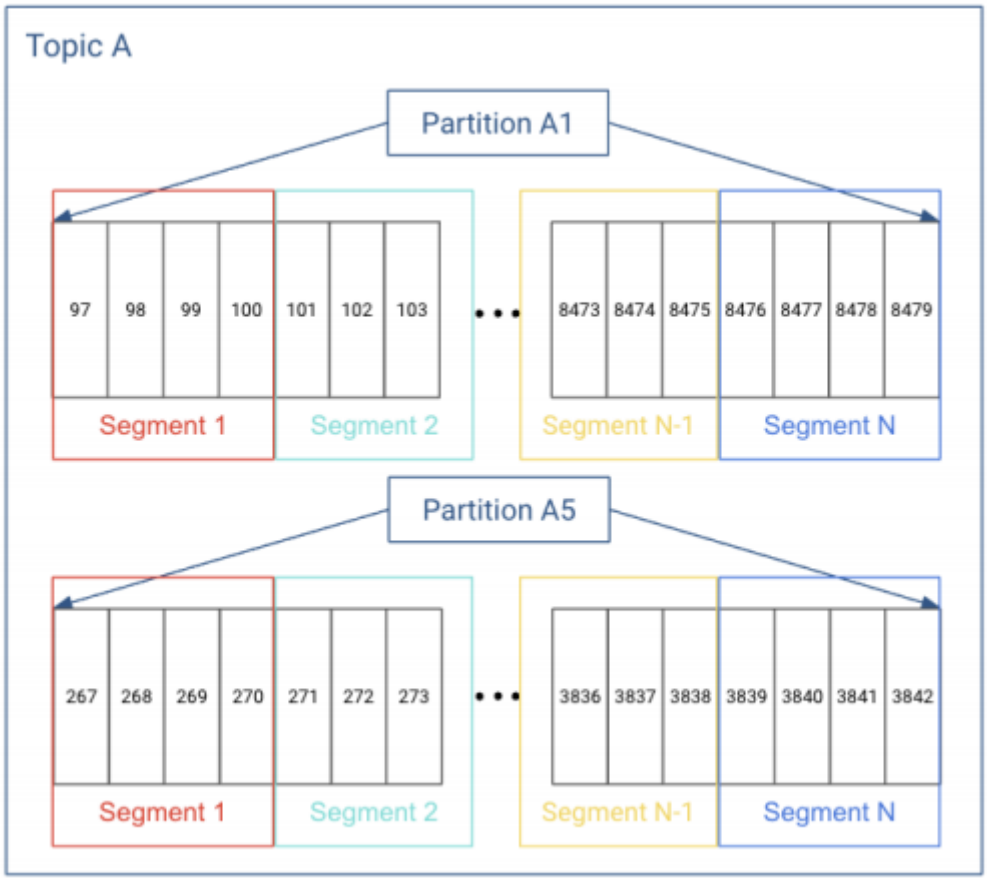

Figure 22: Apache Kafka Partition Log Segments

3.9 Kafka Brokers and ZooKeeper

Zookeeper maintains broker, topic and partition information. Partition information such as replica and partition locations, are updated regularly. Because of this frequent metadata refreshes, a reliable connection between the brokers and the Zookeeper is crucial. Some of the Zookeeper features are the following:

- Kafka Controller maintains leadership via Zookeeper
- Kafka Brokers also store other relevant metadata in Zookeeper
- Kafka Partitions maintain replica information in Zookeeper

The following Figure 23 illustrates the relationship between the brokers and the Zookeeper.

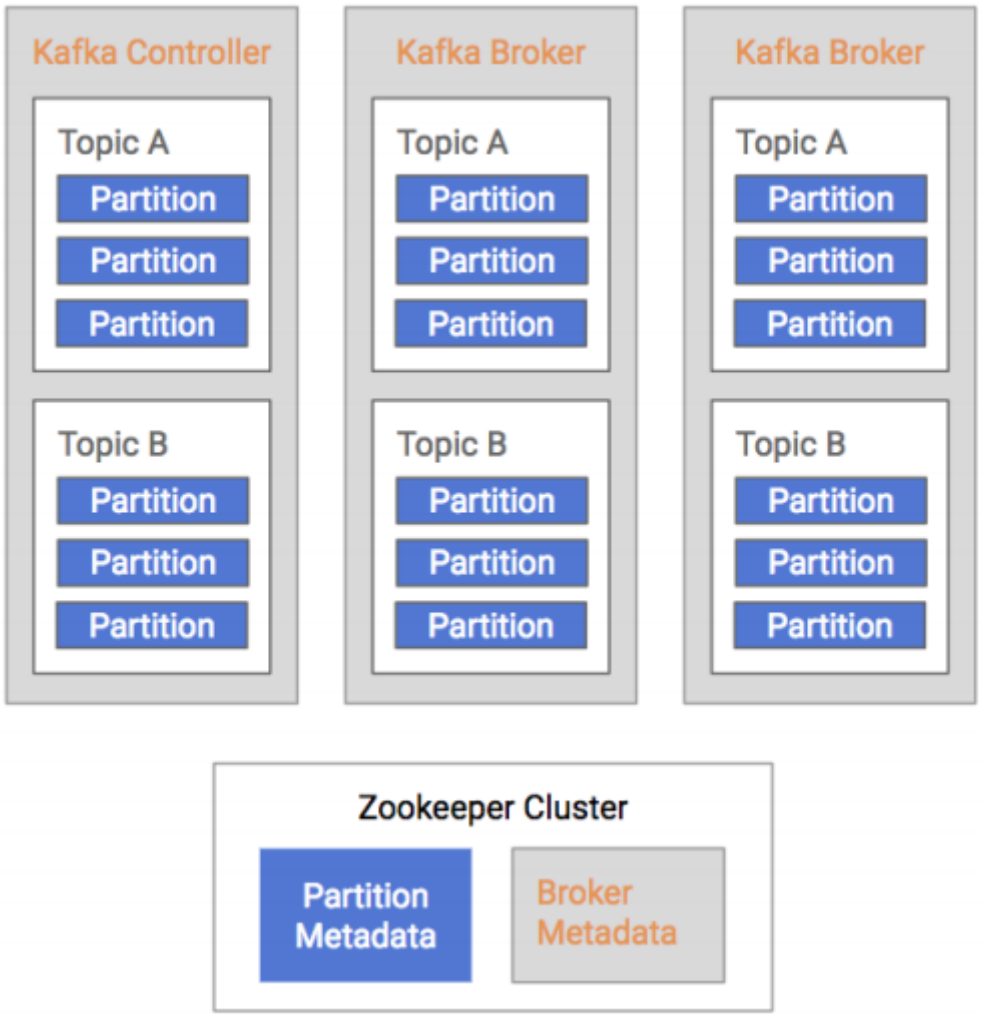

Figure 23: Brokers/Zookeeper relationship

4. OPTIMAL STOPPING THEORY (OST) AND CHANGE DETECTION

The theory of Optimal Stopping (OST) [8] is concerned with the problem of choosing an optimal time to take a given action based on sequentially observed random variables in order to maximize an expected reward or minimize an expected cost. Problems of this type can be found in areas of statistics and operations research.

The theory of Optimal Stopping was considerably stimulated by A. Wald (1947). He showed that – in contrast to the classical methods of the Mathematical Statistics, according to which the decision is taken in a fixed (and nonrandom) time – the methods of the sequential analysis take observations sequentially and the decision is taken, generally speaking, at a random time whose value is determined by the rule (strategy) of observation of a statistician. Wald discovered the remarkable advantage of the sequential methods in the problem of testing (from i.i.d. observations) two simple hypotheses. He proved that there is a sequential method (sequential probability-ratio test) which requires on average a smaller number of observations than any other method using fixed sample size (and the same probabilities of wrong decisions). It turned out that the problem of optimality of a sequential statistical decision can be reformulated as an "optimal stopping problem," and this was the essential step in constructing the General Optimal Stopping Theory.

The change to transmission of commands and sensor values is detected through an optimal stopping rule based on the principles of Optimal Stopping Theory (OST); this statistically secures the best time instance to maximize an expected pay off as will be introduced later in our objective function. Before elaborating on our rationale and time-optimized mechanisms, we provide the fundamentals and principles adopted from the OST.

4.1 Definition of the problem

Stopping rule problems [8] are defined by two objects,

- a sequence of random variables, $X_1, X_2, \ldots$,whose joint distribution is assumed known, and
- a sequence of real-valued reward functions

$$y_0, y_1(x_1), y_2(x_1, x_2), \ldots, y_\infty(x_1, x_2, \ldots).$$

Given these two objects, the associated stopping rule problem may be described as follows. You may observe the sequence $X_1, X_2, \ldots$ for as long as you wish. For each $n = 1,2, \ldots$,after observing $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$, you may stop and receive the known reward $y_n(x_1,\ldots, x_n)$ (possibly negative), or you may continue and observe X_{n+1}. If you choose not to take any observations, you receive the constant amount, y_0. If you never stop, you receive$y_\infty(x_1, x_2, \ldots)$. (We shall allow the rewards to take the value $-\infty$; but we shall assume the rewards are uniformly bounded above by a random variable with finite expectation so that all the expectations below make sense.)

Your problem is to choose a time to stop to maximize the expected reward. You are allowed to use randomized decisions. That is, given that you reach stage *n* having observed $X_1 = x_1 \ldots, X_n = x_n$, you are to choose a probability of stopping that may depend on these observations. We denote this probability by $\varphi_n(x_1, \ldots, x_n)$. A (randomized) stopping rule consists of the sequence of these functions,

$$\Phi = (\varphi_0, \varphi_1(x_1), \varphi_2(x_1, x_2), \ldots),$$

where for all n and d$x_1, \dots, x_n, 0 \leq \varphi_n(x_1, \dots, x_n) \leq 1$. The stopping rule is said to be non-randomized if each $\varphi_n(x_1, \dots, x_n)$is either 0 or 1.

Thus, φ_0 represents the probability that you take no observations at all. Given that you take the first observation and given that you observe $X_1 = x_1, \varphi_1(x_1)$ represents the probability you stop after the first observation, and so on. The stopping rule, φ, and the sequence of observations, $X = (X_1, X_2, \dots)$, determines the random time N at which stopping occurs, $0 \leq N \leq \infty$, where $N = \infty$if stopping never occurs. The probability mass function of N given $X = x = (x_1, x_2, \dots)$ is denoted by $\psi = (\psi_0, \psi_1, \psi_2, \dots, \psi_\infty)$, where

$$\begin{array}{rcl} \psi_n(x_1, \dots, x_n) & = & P(N = n | X = x) \text{for} n = 0,1,2, \dots, \\ \psi_\infty(x_1, x_2, \dots) & = & P(N = \infty | X = x). \end{array} \quad (1)$$

This may be related to the stopping rule φ as follows:

$$\begin{array}{rcl} \psi_0 & = & \varphi_0 \\ \psi_1(x_1) & = & (1 - \varphi_0)\varphi_1(x_1) \\ & \vdots & \\ y_n(x_1) & = & [\prod_1^{n-1}(1 - \varphi_j(x_1, \dots, x_j))]\varphi_n(x_1, \dots, x_n) \quad (2) \\ & \vdots & \\ \psi_\infty(x_1, x_2, \dots) & = & 1 - \sum_0^\infty \psi_j(x_1, \dots, x_j) \end{array}$$

$\psi_\infty(x_1, x_2, \dots)$ represents the probability of never stopping given all the observations.

Your problem, then, is to choose a stopping rule φ to maximize the expected return, V(φ), defined as

$$\begin{array}{rcl} V(\varphi) & = & E_{yN}(X_1, \dots, X_N) \\ & = & E_{yN} \sum_{j=0}^{=\infty} \psi_j(X_1, \dots, X_N) y_j(X_1, \dots, X_j) \end{array}$$

where the "$= \infty$" above the summation sign indicates that the summation is over values of j from 0 to ∞, including ∞. In terms of the random stopping time N, the stopping rule φ may be expressed as

$$\varphi_n(X_1, \dots, X_n) = P(N = n | N \geq n, X = x), \text{where } n = 0,1, \dots. \quad (4)$$

The notation used is that of Section 7.1 of Ferguson (1967).

4.1.1. Loss VS Reward

Often, the structure of the problem makes it more convenient to consider a loss or a cost rather than a reward. Although one may use the above structure by letting $y_n$denote the negative of the loss, clarity is gained in such cases by letting $y_n$denote the loss incurred by stopping at n, and considering the problem to be one of choosing a stopping rule to minimize V(φ).

4.1.2. Random Reward Sequences

For some applications, the reward sequence is more realistically described as a sequence of random variables $Y_0, Y_1, \dots, Y_\infty$ whose joint distribution with the observations $X_1, X_2, \dots$ is known. The actual value of Y_n may not be known at time n when the decision to stop or continue must be made. However, allowing returns to be random does not represent a gain in generality because, since the decision to stop at time n may depend on $X_1, \dots, X_n$, we may replace the sequence of random rewards Y_n by the sequence of reward functions $y_n(x_1, \dots, x_n)$ for $n = 0,1, \dots, \infty$ where

$$y_n(x_1, \dots, x_n) = E\{Y_n | X_1 = x_1, \dots, X_n = x_n\} \quad (5)$$

Any stopping rule φ for the payoff sequence $Y_0, Y_1, \dots, Y_\infty$ should give the same expected.

4.2 Stopping Rule Existence

It is crucial that a stopping rule exists for a given problem. Optimal Stopping cannot be applied if there is no stopping rule existence. For example, assume a problem that someone throws a dice for an infinite time of times and wins the sum of all the throws. It is clear that he should never stop, because the expected reward increases continuously. For this problem a stopping rule does not exist. In this chapter the requirements for the existence of stopping rule will be listed.

Assume the general stopping rule problem described in section 4.1 with observations $X_1, X_2, \dots$ and rewards $Y_o, Y_1, \dots, Y_\infty$ where $Y_n = y_n(X_1, \dots, X_n)$. The existence of the stopping rule relies on the two following requirements:

1. $E\{sup_n Y_n\} < \infty$
2. $lim\ sup_{n\to\infty} Y_n \leq Y_\infty$

The meaning of these two requirements is given bellow with no strict mathematical explanation:

- The maximum of Y_n should exist, or else someone who can predict the future will always receive a non-finite reward.
- Y_n should not go to ∞, because there will be cases that someone by chance will not get a finite reward.

For better understanding of the above requirements, refer to the following two examples:

- Example 1: Let $X_1, X_2, \dots$ be independent Bernoulli trials with probability ½ of success, and let Y_0 = 0,

$$Y_n = (2^n - 1)\prod_1^n X_i, \qquad (6)$$

and Y_∞ = 0. As long as only successes have occurred, you may stop at stage n and receive $2^n - 1$; after the first failure has occurred, you receive 0. Since $Y_n \to 0$ a.s., the second requirement is satisfied. On the other hand, $sup_n Y_n = 2^k - 1$ with probability $\frac{1}{2^{k+1}}$ for k= 0,1,2,… so that $E\{sup_n Y_n\} = \sum_0^\infty (1 - {}^1/_{2^k})/2 = \infty$ and the first requirement is not satisfied. If you reach stage n without any failures, your return for stopping is $2^n - 1$, while if you continue one stage you can get an expected value of at least $(2^{n+1} - 1)/2 = 2^n - 1/2$, which is better. Thus, it can never be optimal to stop before a failure has occurred. Yet continuing forever

gives you a zero payoff so there is no optimal stopping rule. In fact, $sup_N EY_N = 1$, but the supremum is not attained.

- Example 2: Let Y_0 = 0, Y_n = 1 – 1/n for n = 1, 2… and Y_∞ = 0. (The X_n are immaterial). Here requirement one is satisfied and requirement two is not. Yet, like the previous example, the longer you wait the better off you are, but if you wait forever you win nothing. There is no optimal rule.

4.3 The Secretary Problem

A commonly known application of the optimal stopping theory is known as the Secretary Problem [8]. The problem is usually described as the problem of a decision maker who is called to choose the best secretary among a finite number of applicants with the goal of picking the best applicant.

1. There is one position available.
2. There are n applicants for the position where n is a known finite number.
3. You can rank the applicants linearly from worst to best without ties.
4. The applicants are interviewed sequentially.
5. As each applicant is interviewed you must either reject the applicant and interview the next one or accept the applicant and end the decision problem.
6. You must make the decision to accept or reject each applicant using only the relative ranks of the applicants interviewed so far.
7. A rejected applicant cannot be recalled later.
8. The objective of the general solution is to maximize the probability of selecting the best applicant. This is the same as maximizing the expected payoff, with payoff defined to be 1 if you do select the best, 0 otherwise.

We place this problem into the guise of a stopping rule problem by identifying stopping with acceptance. We may take the observations to be the relative ranks, $X_1, X_2, \dots, X_n$, where $X_j$is the rank of the jth applicant among the first j applicants, rank 1 being best. By assumption 4, these random variables are independent and $X_j$has a uniform distribution over the integers from 1 to j. Thus, $X_1 \equiv 1, P(X_2 = 1) = P(X_2 = 2) = 1/2$, ect.

Note that an applicant should be accepted only if it is relatively best among those already observed. A relatively best applicant is called a candidate, so the jth applicant is a candidate if and only if $X_j = 1$. If we accept a candidate at stage j, the probability we win is the same as the probability that the best candidate overall appears among the first j applicants, namely j/n. Thus,

$$y_i(x_1, \dots, x_j)\{\begin{matrix} j/n & \text{if applicant j is a candidate,} \\ 0 & \text{otherwise.} \end{matrix} \quad (7)$$

Note that $y_0 = 0$and that for $j \geq 1, y_j$depends only on x_j.

This basic problem has a remarkably simple solution which we find directly without the use of the optimal rule for finite horizon problems:

$$V_j^{(T)}(x_1, \dots, x_j) = max\left\{y_j(x_1, \dots, x_j), E\left(V_{j+1}^{(T)}(x_1, \dots, x_j, X_{j+1})\middle| X_1 = x_1, \dots, X_j = x_j\right)\right\} \quad (8)$$

Let W_j denote the probability of win using an optimal rule among rules that pass up the first j applicants. Then $W_j \geq W_{j+1}$ since the rule best among those that pass up the first j + 1 applicants is available among the rules that pass up only the first j applicants. It is optimal to stop with a candidate at stage j if $j/n \geq W_j$. This means that if it is optimal to stop with a candidate j, then it is optimal to stop with a candidate at j + 1, since $(j+1)/n > j/n \geq W_j \geq W_{j+1}$. Therefore, an optimal rule may be found among the rules of the following form, $N_r$for some $r \geq 1$:

N_r: Reject the first r-1 applicants and then accept the next relatively best applicant, if any.

Such a rule is called a threshold rule with threshold r. The probability of win using $N_r$is

$$\begin{aligned} P_r &= \sum_{k=r}^{n} P(k^{th}\text{applicant is best and is selected}) \\ &= \sum_{k=r}^{n} P(k^{th}\text{applicant is best})P(k^{th}\text{applicant is selected}|\text{it is best}) \\ &= \sum_{k=r}^{n} \frac{1}{n} P(\text{best of first k - 1 appears before stage r}) \\ &= \sum_{k=r}^{n} \frac{1}{n}\frac{r-1}{k-1} = \frac{r-1}{n}\sum_{k=r}^{n}\frac{1}{k-1}\,. \end{aligned} \tag{9}$$

where $(r-1)/(r-1)$represents 1 if $r = 1$. The optimal r_1 is the value of r that maximizes P_r. Since

$$\begin{gathered} P_{r+1} \leq P_r\text{if and only if} \\ \frac{r}{n}\sum_{r+1}^{n}\frac{1}{k-1} \leq \frac{r-1}{n}\sum_{r}^{n}\frac{1}{k-1}\text{ if and only if} \\ \sum_{r+1}^{n}\frac{1}{k-1} \leq 1\,, \end{gathered} \tag{10}$$

we see that the optimal rule is to select the first candidate that appears among applicants from stage r_1 on, where

$$r_1 = min\left\{r \geq 1: \sum_{r+1}^{n}\frac{1}{k-1} \leq 1\right\}. \tag{11}$$

The following table is easily constructed.

n	=	1	2	3	4	5	6	7	8
r_1	=	1	1	2	2	3	3	3	4
P_{r1}	=	1.0	.500	.500	.458	.433	.428	.414	.410

Table 3: Secretary Problem Expected Probabilities

It is of interest to compute the approximate values of the optimal $r_1$and the optimal $P_{r1}$for large n. Since $\sum_{r+1}^{n}\frac{1}{k-1}$ ~ $\log(\frac{n}{r})$, we have approximately $\log(n/r_1)=1$, or $r_1/n=e^{-1}$. Hence, for large n it is approximately optimal to pass up a proportion e^{-1} of the applicants and then select the next candidate. The probability of obtaining the best applicant is then approximately e^{-1}.

4.3.1. The Parking Problem (Mac Queen and Miller (1960))

Assume that you are driving towards a destination on an infinite distance. When you are close enough, you have to park in order to disembark. If you pass an empty parking spot, you are not allowed to go back and take it, but it is not guaranteed that you will find a spot later on the road. So, the problem here is to pick the perfect spot, closer to your destination, without having to know if there are spots closer.

Here, we model this problem in a discrete setting. We assume that we start at the origin and that there are parking places at all integer points of the real line. Let $X_0, X_1, X_2, \ldots$ be independent Bernoulli random variables with common probability p of success, where $X_j=1$ means that parking place j is filled and $X_j=0$ means that it is available. Let if you do you lose |T-j|. You cannot see parking place j+1 when you are at j, and if you once pass up a parking place you cannot return to it. If you ever reach T, you should choose the next available parking place. If Y is filled when you reach it, your expected loss is then $(1-p)+2p(1-p)+3p^2(1-p)+\cdots=1/(1-p)$, so that we may consider this as a stopping rule problem with finite horizon T and with loss

$$y_T=0\ if\ X_T=0 \text{ and } y_T=1/(1-p)\ if\ X_T=1\,. \quad (12)$$

and for $j=0,\ldots,T-1$,

$$y_j=T-j \text{ if } X_j=0 \text{ and } y_j=\infty\ if\ X_j=1\,. \quad (13)$$

The value $y_j=\infty$ forces you to continue if you reach a parking place j and it is filled.

We seek a stopping rule, $N\le T$, to minimize EY_N.

First, we show that if it is optimal to stop at stage j when $X_j=0$, then it is optimal to stop at stage $j+1$ when $X_{j+1}=0$. As in Moser's problem, $V_j^{(T)}$depends only on X_j,and $X_j=1$ is a constant that depends only on $n-j$. It is optimal to stop at stage $n-j$ if $y_{n-j}\le A_j$. We are to show that if $n-j\le A_j$, then $n-j-1\le A_{j-1}$. This follows from the inequalities, $n-j-1<n-j\le A_j\le A_{j-1}$.

Thus, there is an optimal rule of the threshold form, $N_r$for some $r\ge 0$: continue until r places from the destination and park at the first available place from then on. Let $P_r$denote the expected cost using this rule. Then, $P_0=p/(1-p)$ and for $r\ge 1$, $P_r=(1-p)r+pP_{r-1}$. We will show by induction that

$$P_r=r+1+\frac{2p^{r+1}-1}{1-p}\,. \quad (14)$$

$P_0=p(1-p)=1+(2p-1)/(1-p)$, so, it is true for $r=0$. Suppose it is true for $r-1$; then $P_r=(1-p)r+pP_{r-1}=(1-p)r+pr+p\,(2p^r-1)/(1-p)=(r+1)+(2p^{r+1}-1)/(1-p)$, as was to be shown. To find the value of r that minimizes (7), look at the differences, $P_{r+1}-P_r=1+(2p^{r+2}-2p^{r+1})/(1-p)=1-2p^{r+1}$. Since this is increasing in r, the optimal value is the first r for which this difference is nonnegative, namely, $min\{r\ge 0: p^{r+1}\le 1/2\}$. For example, if $p\le 1/2$, you should start looking for a parking place $r=6$ places before the destination.

4.4 Change Point Detection

Let us assume that we monitor a sequence of i.i.d. random variable $X_1, X_2, \ldots$with a known distribution F_0. At some point T in time, unknown to you, the distribution will change to some other known distribution F_1, and we have to sound an alarm as soon as possible after the change occurs. It is assumed that you know the distribution of T. If the cost of stopping after the change has occurred is the time since the change, and if the cost of false alarm that is of stopping before the change has occurred is taken to be a constant c>0 then the total cost may be represented by

$$Y_n = cI\{n < T\} + (n - T)I\{n \geq T\} \text{ for n=0, 1..., and } Y_\infty = \infty . \quad (15)$$

In this display I(A) represents the indicator function of a set: so, for example $I\{n < T\}$ is equal to 1 if n<T and to zero otherwise. Since T is a random unobservable quantity, we may replace Y_n by conditional expected value given $X_{1,\ldots}, X_n$

$$y_n = cP(\,T > n \mid F_n\,) + E\big(\,(n - T)^{(+.)} \mid F_n\,\big) \text{ for n=0, 1..., and } Y_\infty = \infty . \quad (16)$$

Change-point detection has its origins almost sixty years ago in the work of Page [9] Shirayaev [10] and Lorden [11] who focused on sequential detection of a change-point in an observed stochastic process. The stochastic process was typically a model for the measured quality of a continuous production process, and the change-point indicated a deterioration in quality that must be detected and corrected. In our case, we are adopting this methodology for finding the best time instance for the controller strategy in order to maximize the possibility of successful message delivery by observing runtime network statistics.

The need of change point detection is based on the critical issue that the UxV should “know” when to stop transmitting sensitive and crucial information and data to a saturated network risking their loss. For that reason, a Time Optimized Change-Point Decision Making Process (TOCP) based on CUSUM algorithm was created. More about TOCP in section 5.3.2.

4.4.1. Change Point Detection Algorithms

Let a sequence of independent random variables $(y_k)_k$ with a probability density $p_\theta(y)$ depending upon only one scalar parameter. Before the unknown change time t_0, the parameter θ is equal to θ_0, and after the change it is equal to $\theta_1 \neq \theta_0$. The problems are then to detect and estimate this change in the parameter. The parameter θ_0 it is assumed that it is known before the change. Change point detection algorithms are based on a crucial concept of mathematical statistics, namely the logarithm of the likelihood ratio, defined by:

$$s(y) = \ln \frac{p_{\theta 1}(y)}{p_{\theta o}(y)} . \quad (17)$$

and referred for simplicity as log-likelihood ratio. Below is explained the key statistical property of this ratio. $E_{\theta 0}$ and $E_{\theta 1}$ are representing the expectations of the random variables under distribution $p_{\theta 0}$ and distribution $p_{\theta 1}$ respectively. Then,

$$E_{\theta 0}(\mathrm{s}) < 0 \text{ and } E_{\theta 1}(s) > 0. \quad (18)$$

This practically means that a change in parameter θ is reflected as a change in the sign of the mean value of the log-likelihood ratio. This property of the log-likelihood ratio can

be viewed as a measure of detectability of the change. Also, the Kullback information K is defined by $K(\theta_1, \theta_0) = E_{\theta 1}(s)$, there is the difference between the two mean values,

$$E_{\theta 1}(s) - E_{\theta 0}(s) = K(\theta_1, \theta_0) + K(\theta_0, \theta_1) > 0. \quad (19)$$

So, the detectability of a change can be found also with the aid of the Kullback information between the two distributions before and after the change.

4.4.1.1 CUmulative SUM (CUSUM) Algorithm

CUSUM algorithm, was first proposed in [23]. There are several different variations and approaches. The one described here is the Intuitive Derivation approach. Also, a simple pseudo-code approach is made.

4.4.1.1.1 Mathematical Approach of CUSUM Algorithm (Intuitive Derivation Approach)

Let S_k be the log-likelihood ratio. This ration shows a negative drift before the change and a positive drift after the change as it is illustrated in Figure 24.

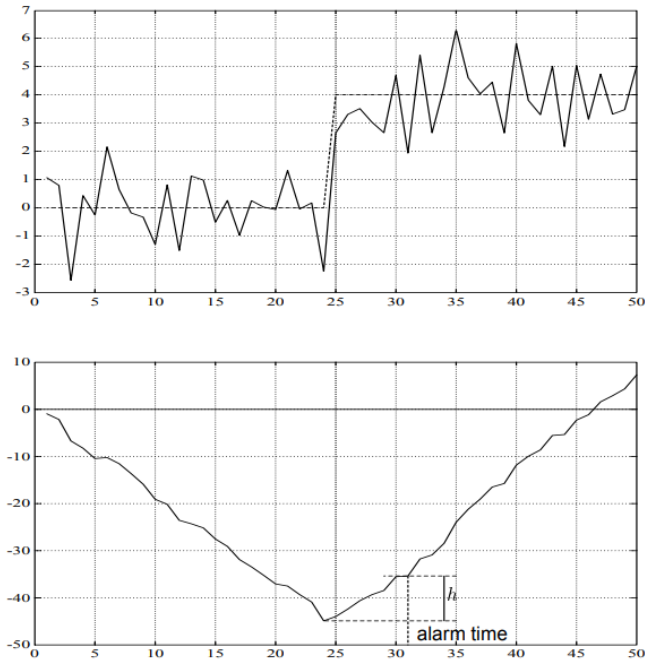

Figure 24: Typical behavior of the log-likelihood ratio S_k corresponding to a change in the mean of a Gaussian sequence with constant variance: negative drift before and positive drift after the change.

So, the relevant information lies in the deference g_k between the value of the log-likelihood ration and its current minimum value m_k; and the corresponding decision rule is then, at each time instant, to compare this difference to a threshold h as follows:

$$g_k = S_k - m_k \geq h\,, \quad (20)$$

Where,

$$S_k = \sum_{i=1}^{k} s_i$$

$$s_i = \ln \frac{p_{\theta 1}(y_i)}{p_{\theta o}(y_i)} \quad (21)$$

$$m_k = \min S_j \ where\ 1 \le j \le k$$

In the following Figure 25, the typical behavior of g_k is illustrated. The stopping time is:

$$t_a = \min\{k: g_k \ge h\}. (22)$$

which can be also written as:

$$t_a = \min\{k: S_k \ge m_k + h\}. (23)$$

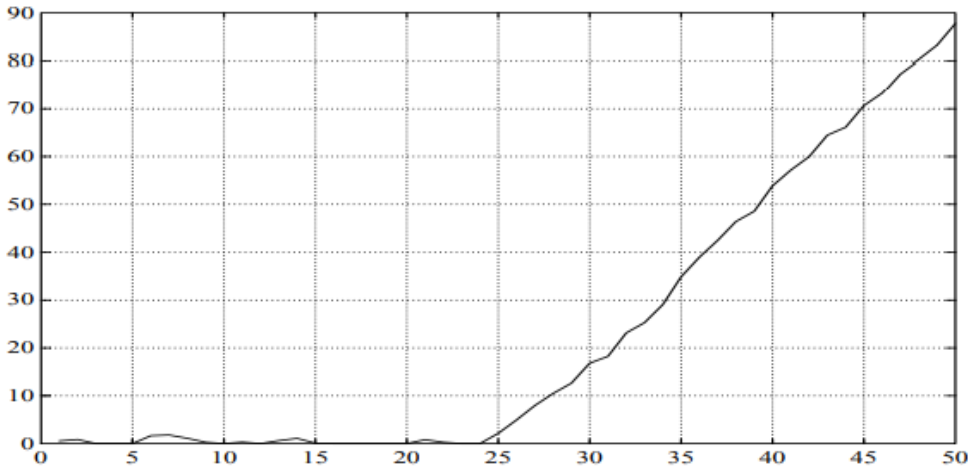

Figure 25: Typical behavior of the CUSUM decision function g_k

4.4.1.1.2 Algorithm

In this section, a general approach to the CUSUM algorithm is given. Its purpose is the better understanding of the algorithm. Assume a univariate time series $x_t \in \mathbb{R}$ consisting of data values collected over time and a target value μ for this data stream. CUSUM involves the calculation of positive and negative changes (P and N, respectively) in the time series x_t cumulatively over time and it compares these changes to a positive and a negative threshold ($thresh^+$ and $thresh^-$, respectively). Whenever these thresholds are exceeded, a change is reported through the above-detection and below-detection signals (s^+ and s^-, respectively) while the cumulative sums are set to zero. In order to avoid the detection of non-abrupt changes or slow drifts, the algorithm takes into consideration tolerance parameters for positive and negative changes (k^+ and k^-, respectively).

The input parameters for the CUSUM algorithm are the following:

- the target value $\mu \in \mathbb{R}$
- the above-tolerance value $k^+ \in \mathbb{R}$
- the below-tolerance value $k^- \in \mathbb{R}$
- the above-threshold value $thresh^+ \in \mathbb{R}$
- the below-threshold value $thresh^- \in \mathbb{R}$

The output parameters for the CUSUM algorithm are the following:

- the above-detection signal $s^+ \in \{0,1\}$

- the below-detection signal $s^- \in \{0,1\}$

CUSUM is presented in Algorithm 1 below.

ALGORITHM 1. Cumulative Sum (CUSUM)

Input: univariate time series x_t, target value μ, above-tolerance k^+, below-tolerancek^-, above-threshold $thres^+$, below-threshold $thres^-$

Output: above detection signal s^+, below detection signal s^-

$P \leftarrow 0$;
$N \leftarrow 0$;
$t \leftarrow 1$;
while ($true$)
    $s^+ \leftarrow 0$;
    $s^- \leftarrow 0$;
    $P \leftarrow max(0, x_t - (\mu + k^+) + P)$;
    $N \leftarrow min(0, x_t - (\mu - k^-) + N)$;
    if ($P > thres^+$) **then**
        $s^+ \leftarrow 1$;
        $P \leftarrow 0$;
        $N \leftarrow 0$;
    end
    if ($N < -thres^-$) **then**
        $s^+ \leftarrow 1$;
        $P \leftarrow 0$;
        $N \leftarrow 0$;
    end
    $t \leftarrow t + 1$;
End

The algorithm assumes that the arrived time series follow a normal distribution. In order the algorithm to work properly, the tolerance and threshold parameters should be tuned in a way that determines what an actual change is for a specific time-series.

This tuning can be performed by following these steps:

- Start with large $thresh^+, thresh^-$ values.
- Choose k^+, k^- parameters to half of the expected change, or adjust them such that P, N are zero more than half of the times.

- Then set the $thresh^+, thresh^-$ values so that the required number of false alarms or the required delay for detection is obtained.
- If faster detection is sought, try to decrease k^+, k^-values.

If fewer false alarms are desired or changes that do not make sense are detected, try to increase k^+, k^-values.

5. RATIONALE AND PROBLEM FORMULATION

5.1 Definition of the Problem

In practice, if a UxV loses contact with the GCS, it returns back to a given initial position. This is impractical for missions that require UxV to travel a good distance away from the GCS. It is also energy inefficient both for the UxV's battery and for the network, to start the mission from the beginning every time that communication is lost. The goal of this thesis is to prevent the aforementioned loss of communication by acting proactively based on the OST, in order to complete the mission without the need of restarting it. The algorithm implemented for the purpose, extracts network data from the UGV's and GCS's wireless interface and then uses the OST in order to take actions, for example starting and stopping the telemetry transmission from the sensors. In this thesis, there will be usage of live data produced by a UGV (TurtleBot), in order to prove the theoretical work and the simulations.

In this thesis a framework that implements a network quality based decision-making process is developed. This framework adapts the information flow between the UxV and the Ground Control Station (GCS) based on network quality metrics (such as packet error rate etc.) and the principals Optimal Stopping Theory (OST). The goal of this framework is to ensure the optimal delivery of critical information from UxV to GCS and vice-versa. If the network behaves optimally then there is no limitation on the information flow, but if the network is saturated or overloaded restriction rules are applied. The proposed model introduces two optimal stopping time mechanisms based on change detection theory and a discounted reward process.

5.2 Related Work

In the literature researchers have extensively studied message-routing protocol employed on the unmanned vehicles. Opportunistic networks are studied as long as they are capable of maintaining efficient operation in a wide range of network density and mobility conditions. These network environments are classified by their diversity of topological conditions. One of the classification categories are networks of almost static dense topologies. Regarding this category conventional topology-based protocols [14] are the best option; they just use labels/identities. When the density starts decreasing but the mobility status remains stable, then the best option is the position-based protocols [15] [16]. Position-based protocols rely on the spatial transposition of messages due to hops from one node to another. Another classification category are the networks of low nodal density. In these networks intense mobility is required, so nodal contact opportunities can be created. These topologies are based on the 'carry' action [17] [18], i.e. the spatial transposition of the message due to the physical movement of the carrier node, perform better; they employ information about the nodes motion characteristics.

Connectivity issues between mobile nodes is critical and there should be guarantees for the secure delivery of the messages. For this reason, a Decision-Making Process (DMP) can be created. The DMP relies on information that have an impact on message delivery rate, such as network ping, packet loss, delay deadlines, etc. In [4] ,a cross-layer optimization framework for single-user multimedia transmission over single hop wireless networks was created. In this framework the DMP takes into consideration the network conditions and the adaptation capabilities of the user at various layers of the protocol stack.

Methods that derive from the principals of Optimal Stopping Theory (OST), can also be studied. These methods can apply on information exchange regarding ad-hoc networks. However, the focus of the research in the literature is not based on the OST principals. In [19] - [21] contextual data mechanisms deal with the delivery of quality information to context-aware applications in static and mobile ad-hoc networks respectively assuming epidemic-based information dissemination schemes. The mechanism in [19] is based on the probabilistic nature of the “secretary problem” [8] and the optimal on-line problem. Authors in [20] study a dynamic video encoder that detects scene changes and tunes the synthesis of Groups-of-Pictures accordingly. Such dynamic encoding can be applied to infrastructures with restricted resources, like IoT facilities where multimedia streams are of use. They propose a time-optimized DMP that yields different sizes of groups-of-pictures (frames) to meet the previously discussed objectives i.e., transmit video sequences in acceptable quality with rational use of the wireless resources. In [22] authors propose optimal DMP decisions on the collection of contextual data from WSNs. The authors determine the best time to switch from decision to learning phase of Principal Component-based Context Compression (PC3) model, while data inaccuracy is taken into account. If data inaccuracy remains at low levels, then any deterministic switching from compression to learning phases of the observation.

Concluding, change-point is a legacy work, with his lineage going back to the work of Page [9], Shiryayev [10] and Lorden [11]. Lorden, focused his research on sequential detection of a change-point in an observed stochastic process. The stochastic process was typically a model for the measured quality of a continuous production process, and the change-point indicated a deterioration in quality that must be detected and corrected.

5.3 Time-Optimized Decision-Making Model for Unmanned Vehicles

5.3.1. Overview

The main goal of this thesis is the development of a framework that implements a network quality-based decision-making process. That decision-making process (DMP) adapts the information flow of UxV missions. UxV not only have to receive messages from a GCS, but also post data from their built-in sensors like position, temperature etc. Information flow is adjusted dynamically based on network metrics such as packet error rate (PER), by starting and stopping message production between the GCS and the UxV. So, the framework has two states of operation: active and passive. The duration and the transitions between the states, are defined by two on-line mechanisms. As mentioned in the introduction (see chapter **Error! Reference source not found.**), not a ll the messages have the same criticality, for example sensor telemetry (e.g. humidity) can be characterized as lower priority messages in comparison with the position reporting of UxV that is a perquisite for the safe execution of the mission. Higher priority messages should be sent constantly, but lower priority messages can be delayed if the network performance is not sufficient. With this behavior there is a better chance that the UxV will finish the mission without losing completely the link between its network interface and the mission’s access point.

The DMP runs locally on the UxV/GCS, and it is equipped with a Time-Optimized Change-Point Decision Making Process (TOCP), which is triggered every time there is a change to network performance (TOCP is fully described in section 5.3.2). When the DMP is in ‘passive’ state, stopping the transmission of various messages, a Discounted Secretary Problem (DSP) which is activated tries to recover the state back to ‘active’ (DSP is introduced in Section 5.3.3). DSP ranks the network quality measurements from

worst to best and optimally delays its pause interval. The pausing period has a maximum deadline of Th_{max}.

In the following Figure 26, everything of the above are explained with the help of a directed graph where QNi stands for Quality of Network Indicator.

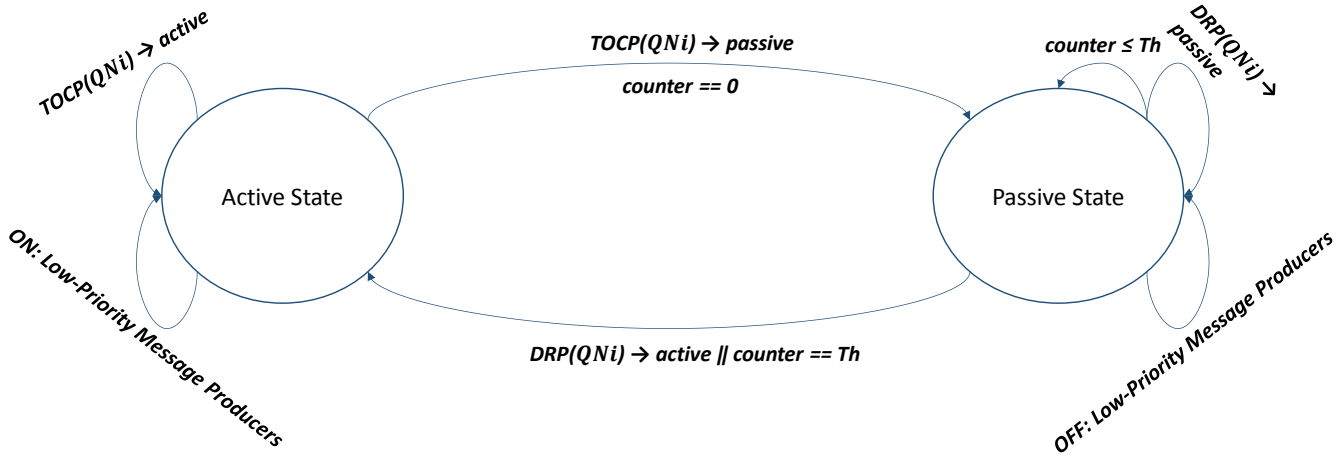

Figure 26: Graph representation of DMP

In the following

Table 4 there are the rules of state transitions.

Component	Optimal Network States	Poor Network States
UxV	Active: High-Priority sensors	Active: High-Priority sensors
UxV	Active: Low-Priority sensors	Passive: Low-Priority sensors
GCS	Active: High-Priority messages	Passive: High-Priority messages

Table 4: Rules of State Transition

5.3.2. Time-Optimized Change-Point Decision Making Process (TOCP)

When the UxV is in active state, then TOCP mechanism is triggered. Assume the network quality data follow a probability density function (x_n, f_i) where f_i expresses the normal distribution (with value μ_i and variance σ_i). These network data $x_1, x_2, \dots, x_n$ can be considered as a random signal with i.i.d. random variables observed in real-time. For the case of this thesis, a probability density function comparison method was adopted in

order to estimate $p(x_n, f_i)$. This method derives the closest distributions to the newly introduced Quality Network Indicator (QNI). QNI is calculated using live network data and more specifically: Packet Error Rate (PER), Signal-to-Noise Ratio (SNR), and the UNIX built-in interference quality indicator (Q). The description of these metrics can be found in Table 5 bellow. So, QNI is an affine combination of the above metrics normalized in [0,100]:

$$QNI_n = \widehat{PER_n} + \widehat{SNR_n} + \widehat{Q_n}. \quad (24)$$

Quality indicator	Description
Packet Error Rate (PER)	*Rate between the lost packets and the total packets sent through the network*
Signal-to-Noise Ratio (SNR)	*Ratio of signal power to the noise power*
Interference quality Indicator (Q)	*Exported by an access point in scale [0,100] and depends on the level of contention or interference, like the bit or frame error rate, or other hardware metric.*

Table 5: Description of Network Quality Indicators

Now assume two priors know distributions f_0 , f_1 where $f_0 \neq f_1$, where f_0 represents the data of QNI when the network quality is good, and f_1 represents the data of QNI when the network quality is bad. Probability Density Function (PDF) model fittings of these distributions are illustrated in Figure 27 and Figure 28.

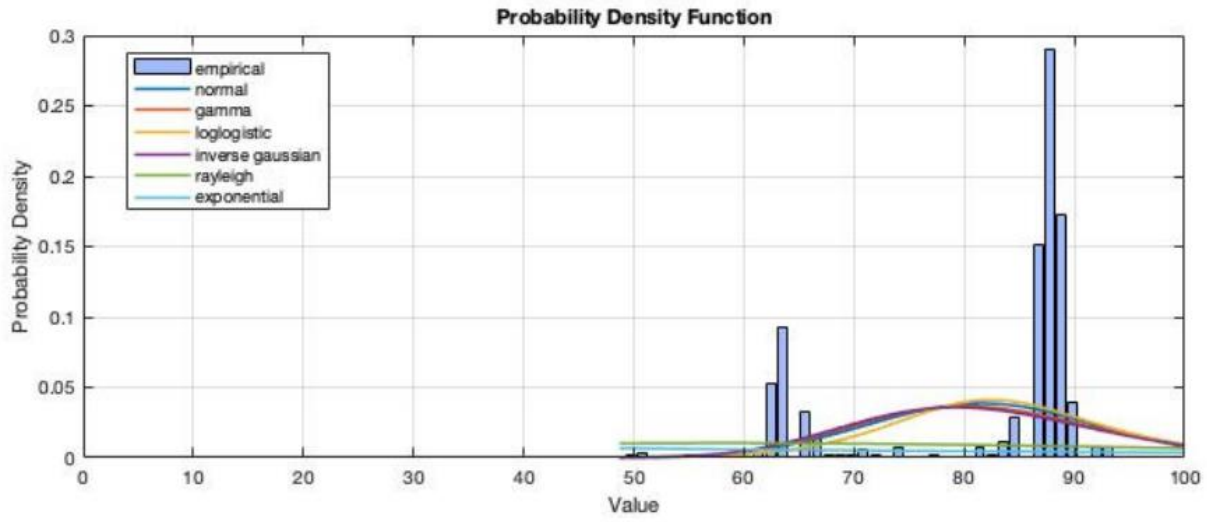

Figure 27: Probability Density Function of f_0 Model Fitting

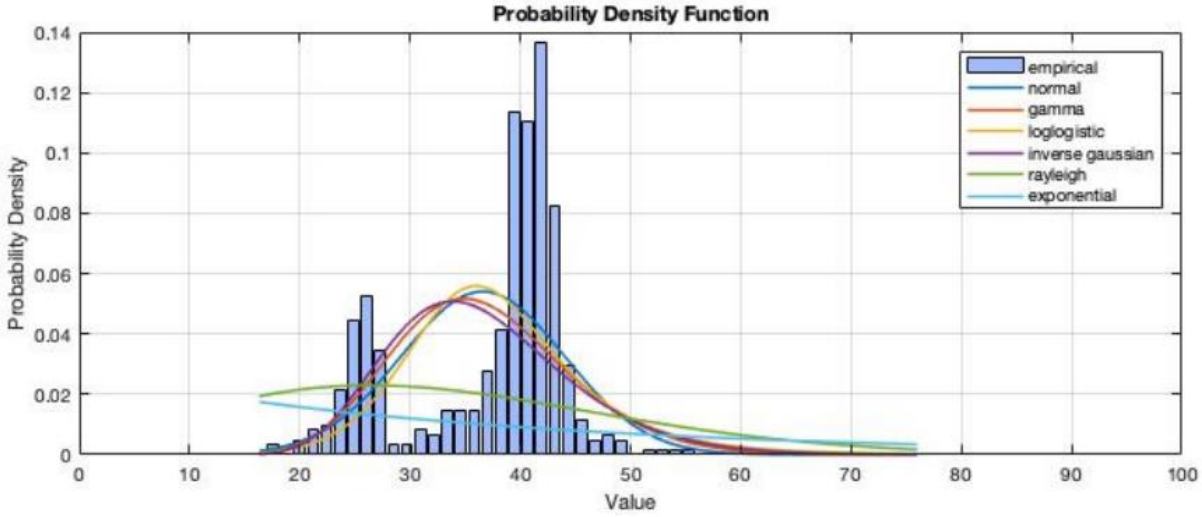

Figure 28: Probability Density Function of f_1 Model Fitting

The method for estimating $p(x_n, f_i)$, is based on model fitting of all the parametric probability distributions to the QNI, and then is observed if there is a concept drift of QNI from one distribution to another.

Now assume a time m. Before this time QNI follows the PDF of the distribution f_0 , and after this time shifted to the PDF of distribution f_1. Under the given observations, the following equation (25) can be extracted, where x_0 is the first sample, and x_k is the current sample.

$$p(x) = \begin{cases} \prod_{n=0}^{k} p(x_n, f_0)\,, no\ change;\ hypothesis\ H_0 \\ \prod_{n=0}^{m-1} p(x_n, f_0) \prod_{n=m}^{k} p(x_n, f_1)\,, change; hypothesis\ H_1 \end{cases} . (25)$$

The challenge is now to approximate the potential change point time m and decide between the hypothesis H_0 and H_1 based on the behavior of QNI. A good solution for this problem can be found in [9] which adopts the minimax approach. From [9] the conditional expected delay is defined:

$$E_{H1}[(N_d - m + 1)^+ | n = 0,1, \dots, m-1]. \ (26)$$

The minimax performance criterion is given by its supremum taken over. Specifically, the worst-case detection delay is estimated as:

$$D_n(\tau) = \sup_{n \geq 1} ess\ \sup E_k\,[(\tau - k + 1)^+ | F_{k-1}]. \ (27)$$

Where $x^+ = max\{x, 0\}$ and the False Alarm Rate (FAR) is defined in [12] as:

$$FAR(\tau) = \frac{1}{E_\infty[\tau]}\,, \quad (28)$$

$$where\ E_{\infty}[\tau]\ defines\ the\ expected\ time\ between\ false\ alarms$$

Log-likelihood ratio at time n is defined by:

$$L_x(n) = \ln\frac{p(x(n), f_0)}{p(x(n), f_1)} = \ln\frac{\sigma_1^2}{\sigma_0^2} + \frac{(x-\mu_1)^2}{2\sigma_1^2} + \frac{(x-\mu_0)^2}{2\sigma_0^2}\ . (29)$$

and $S(n)$ is defined as the cumulative summation of the log-likelihood ratios from 0 to n:

$$S(n) = \sum_{k=0}^{n} L_x(k).\ (30)$$

Under the Lorden criterion, the goal is to find the optimal solution to equation (27). This solution was determined in [13] and it is given by the CUSUM test proposed in [23]. The optimal stopping time of change point detection is given by:

$$\tau^* = min\ \left\{n \geq 1, \max_{1\leq k\leq n} \sum_{i=k}^{n} L_x(i) \geq \alpha\right\}, \quad (31)$$
$$where\ \alpha\ is\ the\ detection\ threshold$$

In the following Figure 29, is illustrated the behavior of log-likelihood ratio, when there is a change from good network state to bad network state.

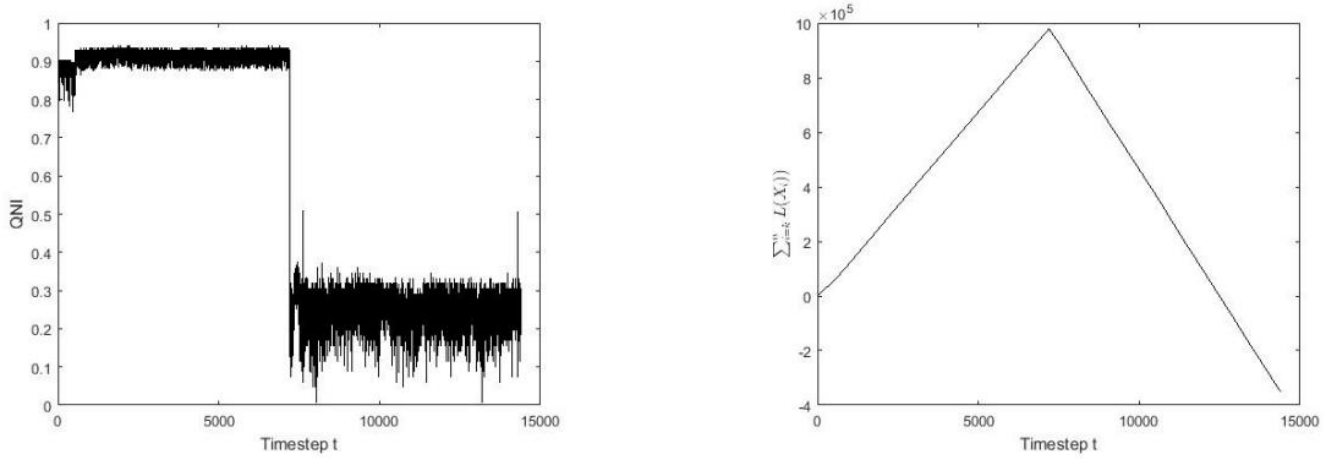

Figure 29: Log-Likelihood Ratio Behavior

5.3.3. Discounted Secretary problem (DSP)

When the UxV transits from the active state to the passive state, it cannot stay passive forever, because it has a hard limit for sending messages to GCS in order to report that its alive. This limitation does not only apply to the UxV, but also for the GCS; a GCS cannot leave a UxV with no control messages for a long time. TOCP introduced in section 5.3.2 optimally picks the moment, that the UxV's telemetry will transit from active to passive state. A finite horizon problem was applied in the pausing state. A discounted secretary problem was assumed in which the QNI measurements are treated as 'candidates' for the secretary position. DSP shall stop if the best candidate is selected prior to the deadline of the pausing period. In other words, it is used stopping rule that will maximize the probability of choosing the best/maximum QNI value x_k. Discounted factor works a penalty for every timestep that DSP do not conclude to the optimal stopping time.

6. PERFORMANCE EVALUATION

6.1 Experimental Setup

Approaching the problem effectively, there should be an experimental scenario, in order to have unbiased, accurate and representative data. The most common technique is to run a robotics simulator, in order to simulate the behavior and the sensors of a UxV, replay dummy network measurements, execute the experiment on the given environment and finally get the results as an output. In contrast to the usual practice of simulating the UxV and the network, in this thesis live UxV and network data were used.

In order to explain further the given approach, there is Figure 30. Figure 30**Error! Reference source not found.** is an abstract representation of the experimental setup.

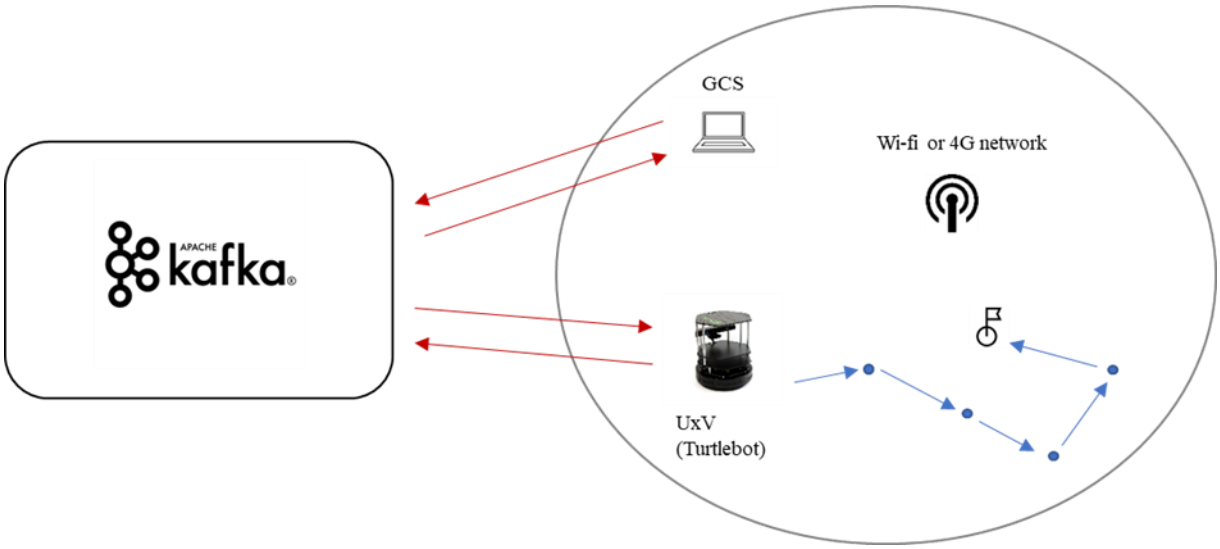

Figure 30: Abstract Representation of Experimental Approach

6.1.1. Communication Schema

Messages like movements commands should be passed from GCS to TurtleBot. Communication between GCS and TurtleBot is not direct. The messages first pass through Apache Kafka, then received by Raspberry Pi and finally are served to the corresponding TurtleBot nodes. The need for an intermediate messaging system between GSC and Raspberry exists because there is not a default ROS server which can accept messages directly from the GCS (see Figure 33). Apache Kafka fills this communication gap. Messages between GCS and Raspberry Pi are in JSON format. Messages between Raspberry and TurtleBot are in various JSON style formats. The messages that should be given to TurtleBot in order to move, has to be received from *cmd_vel* topic and be in *Twist* format. Figure 31 and Figure 3232 illustrate a JSON formatted movement command and a Twist formatted movement command respectively.

```
{
        'orientationz': 0,
        'orientationx': 0,
        'orientationy': 0,
        'orientationw': 0,
        'positionx': 0,
        'positiony': 0,
        'tst': '2019-02-08 11:31:47'
}
```

Figure 31: JSON message example for movement to specific point.

```
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0]
```

Figure 32: Twist message type example.

In the following Figure 3333 is illustrated an abstract representation of the communication schema.

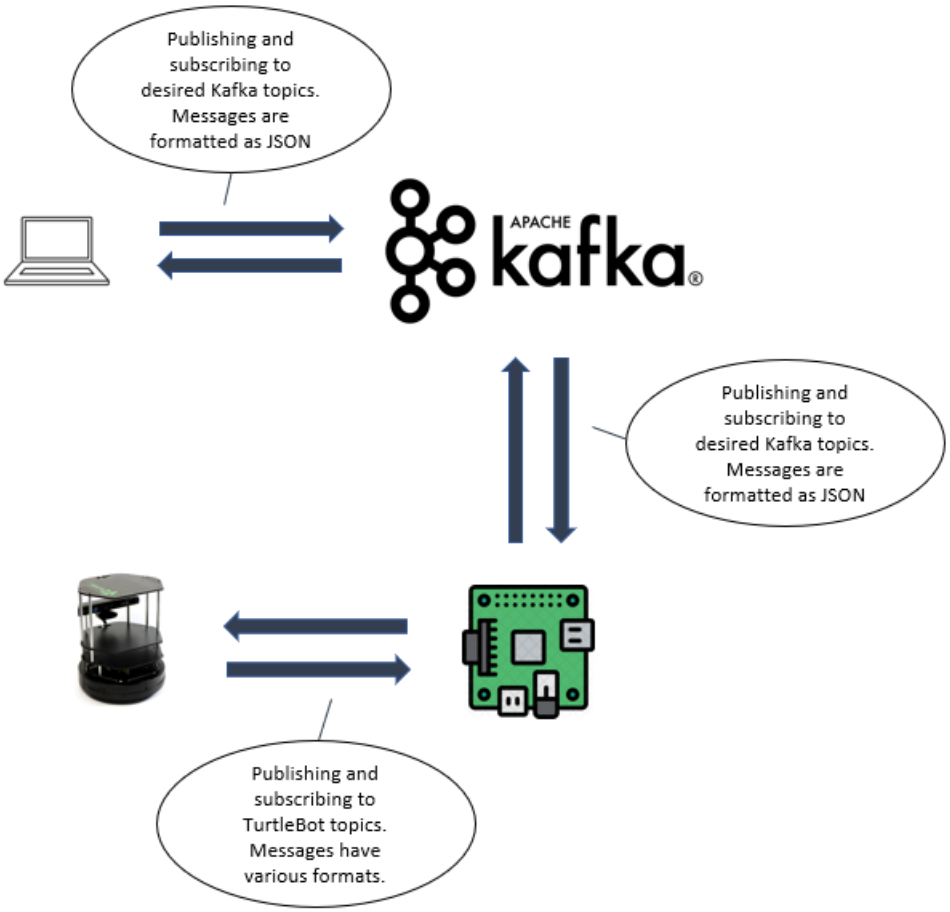

Figure 33: Abstract Communication Schema.

6.1.2. Measuring the network

As mentioned in section 5.3.2, it is critical to extract network information (PER, SNR, Q) in order to calculate QNI. For this reason, an algorithm for extracting these values was made. The source code can be found in **Error! Reference source not found.** by following the GitHub link listed. This algorithm executes the default ping command and iwconfig installed in UNIX systems. By parsing the output of these commands, the aforementioned network quality metrics can be extracted. The algorithm takes as input the interval between sending packets, address to ping (in this case the Access Point of the mission), size of the packet to send and the wireless interface of the UxV. Afterwards it transmits the extracted network quality data through Apache Kafka to the GCS, in a JSON formatted message. The messages carrying these data are considered as high priority messages and their transmission is never paused.

Snapshots of the code, can be found bellow.

```
#run ping command
	cmd1 = 'ping' + ' ' + str(host) + ' -c ' + str(count) + ' -s ' + str(size) + ' -i ' + str(interval)
+ ' -q'

	command1 = subprocess.Popen(cmd1.split(),stdout=subprocess. PIPE)
	rawoutput1 = command1.communicate()

	#check if network is ok
	if rawoutput1[0] == "":
	   print "Waiting to reconnect..."
	   time.sleep(3)
	   continue

	output1 = rawoutput1[0].split('\n')
	output1 = filter(None, output1)
	del output1[0]
	del output1[0]

	packets = output1[0].split(',')
	pnumbers = re.findall('\d+',output1[0])

iwconfout = os.popen('iwconfig %s' % interface).readlines()
```

Figure 34: Running ping, iwconfig commands and receiving their output

```
x = Record(sent, received, loss, quality, level, str(millis))

njson = json.dumps(x.__dict__)

print njson
```

Figure 35: Creating and sending to Apache Kafka a JSON message carrying the network quality data

6.1.3. Telemetry Data

While the UxV is in the passive state all transmissions regarding telemetry data except the network quality data and the UxV's location data are paused for a period of time, until it is decided to resume. But what exactly are these telemetry data?

The following Table 6 shows the types of telemetry data and their priority per type:

Telemetry Data Type	Priority (Importance)
Network Quality	High
TurtleBot's Battery Level	Low
Memory Consumption	Low
CPU Consumption	Low
CPU Temperature	Low
TurtleBot's Location	High

Table 6: Telemetry Data Types and their priorities

Network data are crucial because by reading these, TOCP and DSP can decide on-line about the UxV's state transition as mentioned in section 5.3.2. All the other data except TurtleBot's location data are considered as low priority data. TurtleBot's location is a critical piece of information, because the GCS should know anytime where the UGV is inside the field of the experiments, in order to modify its behavior, take photos etc.

JSON formatted messages of the aforementioned data types can be found bellow.

```
{
   "orz": 0.0,
   "orx": 0.0,
   "ory": 0.0,
   "orw": 0.0,
   "posx": 0.0,
   "posy": 0.0,
   "tst": "2019-02-08 11:31:47",
   "latency": "12312324124" //millis from 1970
}
```

Figure 36: TurtleBot's Position Data Message

```
{
   "battery": 150.0,
   "ts": "2019-02-08 11:43:54",
   "latency": "12312324124" //millis from 1970
}
```

Figure 37: TurtleBot's Battery Level Data Message

```
{
   "ts": "2019-02-08 11:43:54",
   "memory": 1.3,
   "latency”: "12312324124" //millis from 1970
}
```

Figure 38: Memory Consumption Data Message

```
{
   "cpu": 14.5,
   "ts": "2019-02-08 11:43:54",
   "latency”: "12312324124" //millis from 1970
}
```

Figure 39: CPU Consumption Data Message

```
{
   "loss": 0.0,
   "received": 0,
   "ts": "2019-02-08 11:43:54"
   "quality": 0,
   "level": 0,
   "sent": 0,
   "latency”: "12312324124" //millis from 1970
}
```

Figure 40: Network Quality Data Message

```
{
   "meantemp": 41.3,
   "ts": "2019-06-05 10:49:53",
   "latency": "1559720993088"
}
```

Figure 41: CPU Temperature Data Message

The code of ROS Nodes that implement this functionality can be found in **Error! R eference source not found.** by following the GitHub listed there.

6.2 Experiments and Results

An experimental evaluation is reported in order to compare the performance of the implemented framework. TurtleBot was used for the executed experiments for two applications:

- Following a trajectory reaching a goal point. (Mission A)
- Exhaustive scanning of a room. (Mission B)

For both mission categories, user gives the desired trajectory points, creating a path, that the UxV should follow in order to reach its final destination. In the following Figure 42 the UxV, the map, and the user-given trajectory for the two missions are illustrated. These screenshots were taken, while the TurtleBot was executing a live mission. The experiments were conducted in a classroom and a corridor of the University of Athens.

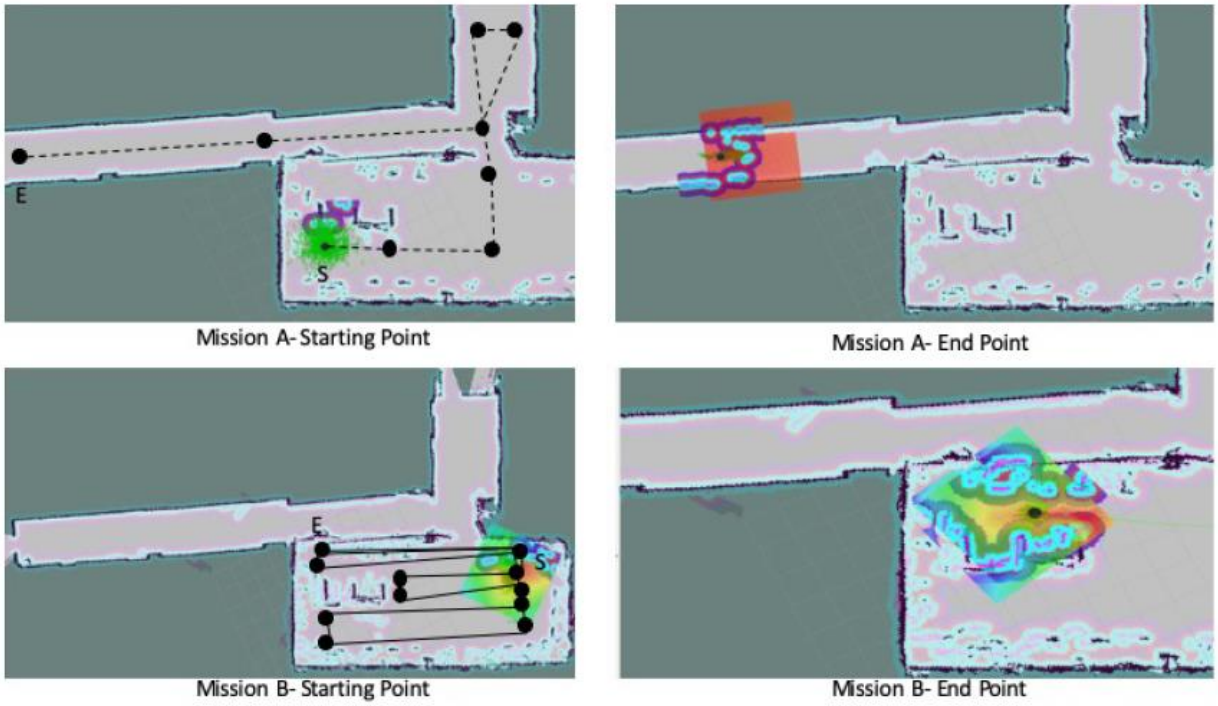

Figure 42: Mission A and Mission B Real-Time Illustration

100 runs of 10 minutes duration were performed and each run involved more than 100 observations for every integrated TurtleBot sensor. Four different policies of decision were adopted, in order to build the comparative assessment:

- No-policy, meaning that the UxV always sent and receive messages, with no concern of the network conditions.
- A heuristic threshold-based model. In this model the transmission of messages and telemetry stops when QNI falls under the interval of [0.4, 0.5].
- A TOCP model which applies a change detection policy.
- A TOCP-DSP model.

In the following Figure 43, Figure 44, Figure 45 and Figure 46 the QNI and PER performance of the four policies is plotted regarding Mission 1 – Path following. In Mission 1 there are two intervals that the network has very poor quality. These intervals are [35-45] and [75-90] and exist in all four different policies. There is a difference though regarding the TOCP-DSP policy. While for Threshold policy, No-policy and only

TOCP policy the QNI reach values less than 50%, for the TOCP-DSP policy QNI is close to 68%. PER maximum values per policy are:

- No-policy: 25%
- Threshold: 45%
- Only TOCP: 15%
- TOCP-DSP: 10%

These maximum values are reached in the aforementioned intervals of poor network quality. The gain of PER with the application of TOCP-DSP policy is up to 20%. TOCP-DSP policy is more efficient that the only TOCP policy because the “pausing” period is not only adaptively activated, but also adaptively deactivated. TOCP-DSP related to PER reaches from 20% up to 70% better results.

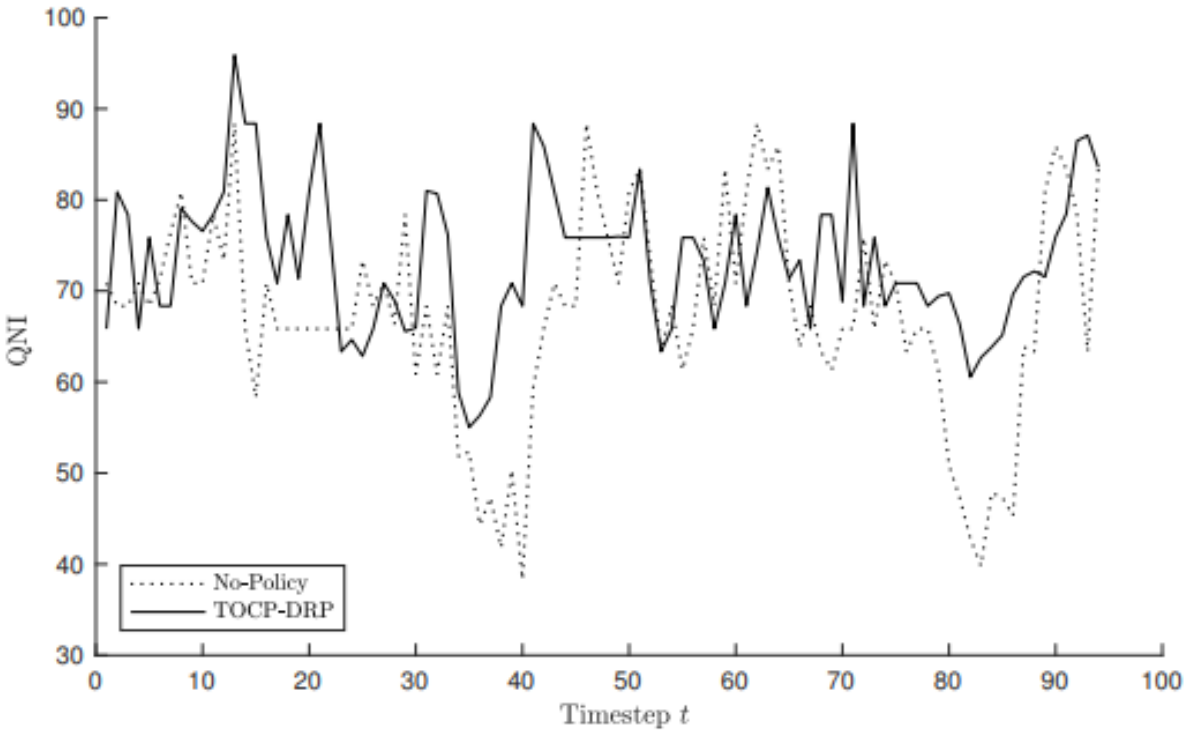

Figure 43: TOCP-DSP vs No-Policy regarding QNI in Mission 1- Path Exploration

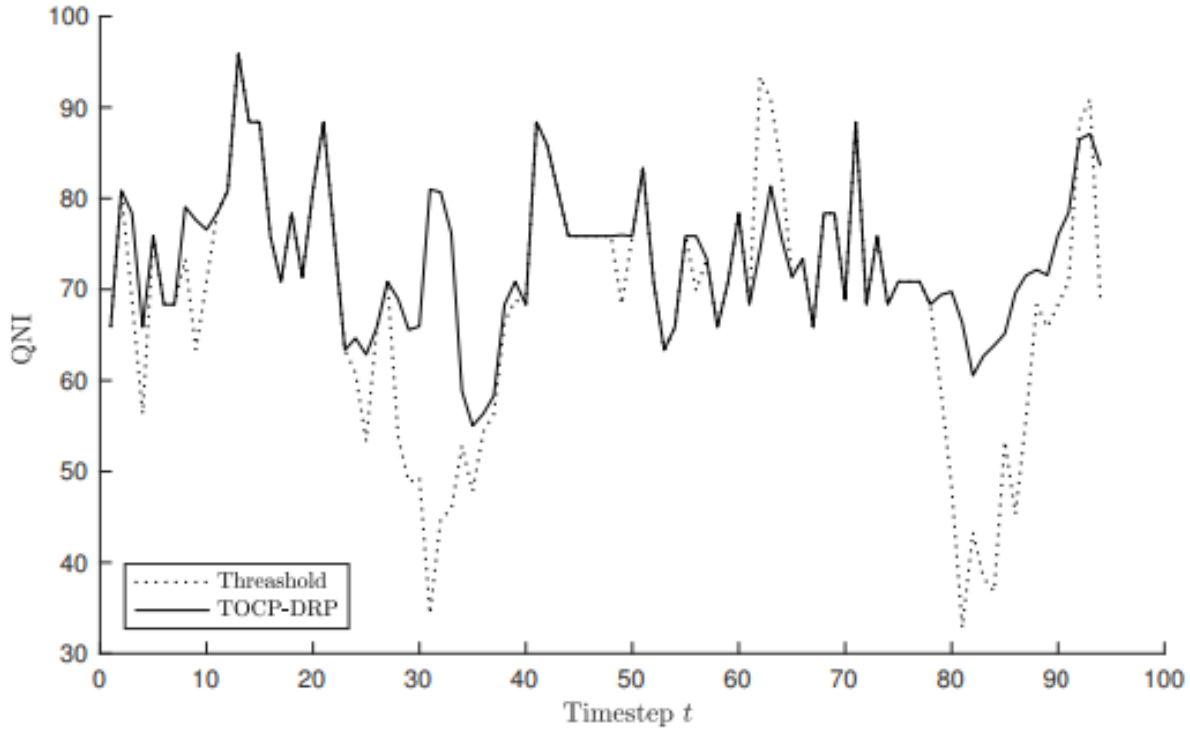

Figure 44:TOCP-DSP vs Threshold Policy regarding QNI in Mission 1- Path Exploration

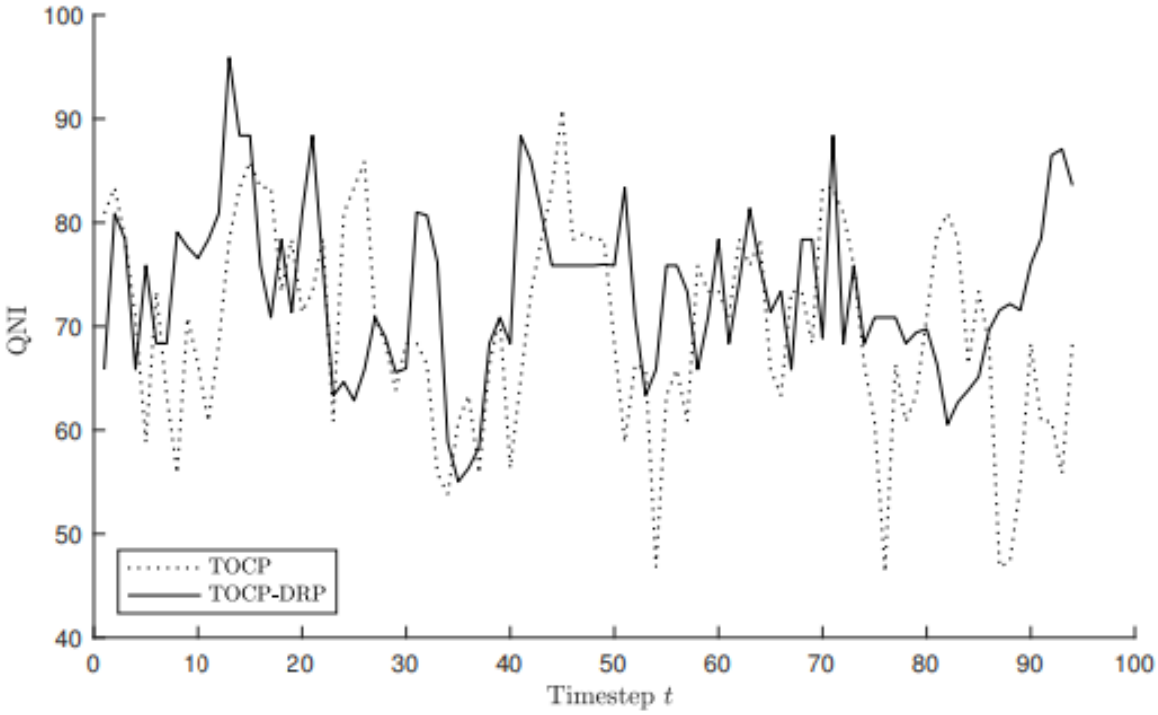

Figure 45:TOCP-DSP vs TOCP-Only Policy regarding QNI in Mission 1- Path Exploration

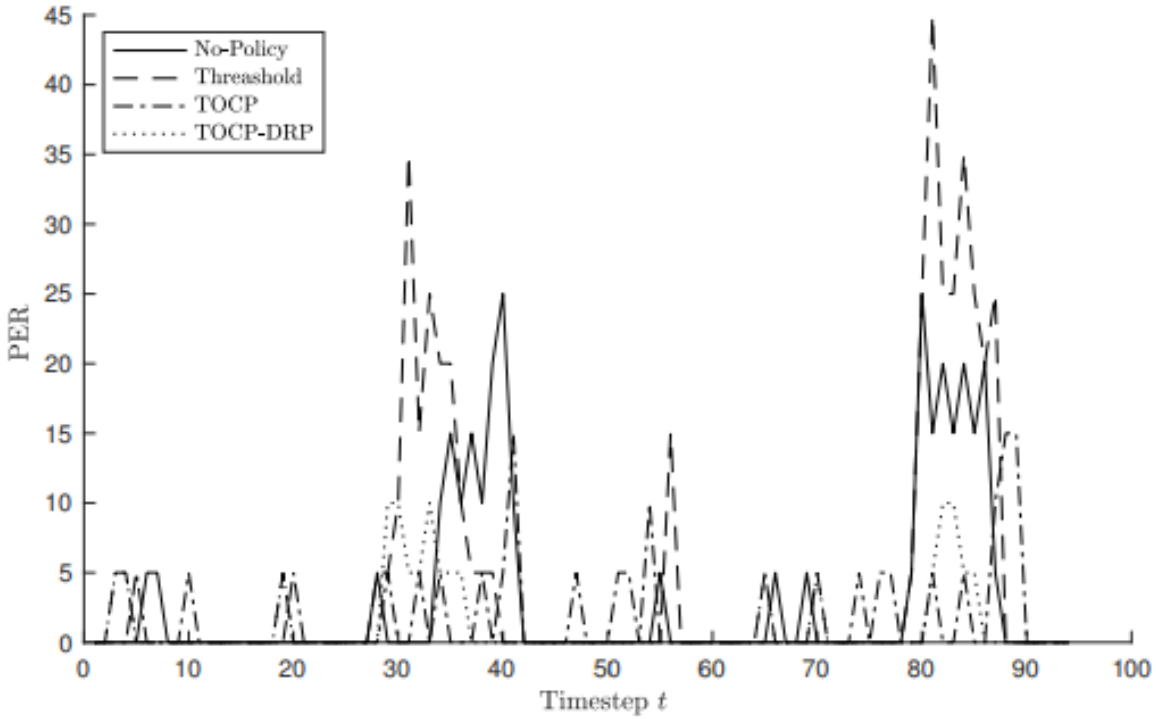

Figure 46: PER of all four policies in Mission 1- Path Exploration

In the following Figure 47, Figure 48, Figure 49 and Figure 50 the QNI and PER performance of the four policies is plotted regarding Mission 2 – Exhaustive Scanning. Mission 2 was executed inside a classroom of University of Athens. Because of the indoor space there are many obstacles and walls, that render the finalization of the mission harder than Mission 1.

QNI mean values for Mission 2 polices:

- No-policy: 68.4446
- Threshold: 70.8197
- Only TOCP: 65.8525
- TOCP-DSP: 76.3498

PER maximum values for every policy:

- No-policy: 30%
- Threshold: 20%
- Only TOCP: 20%
- TOCP-DSP: 10%

From the above data is shown that the TOCP-DSP policy, outperforms the rest of the policies, not only regarding PER, but also regarding network quality.

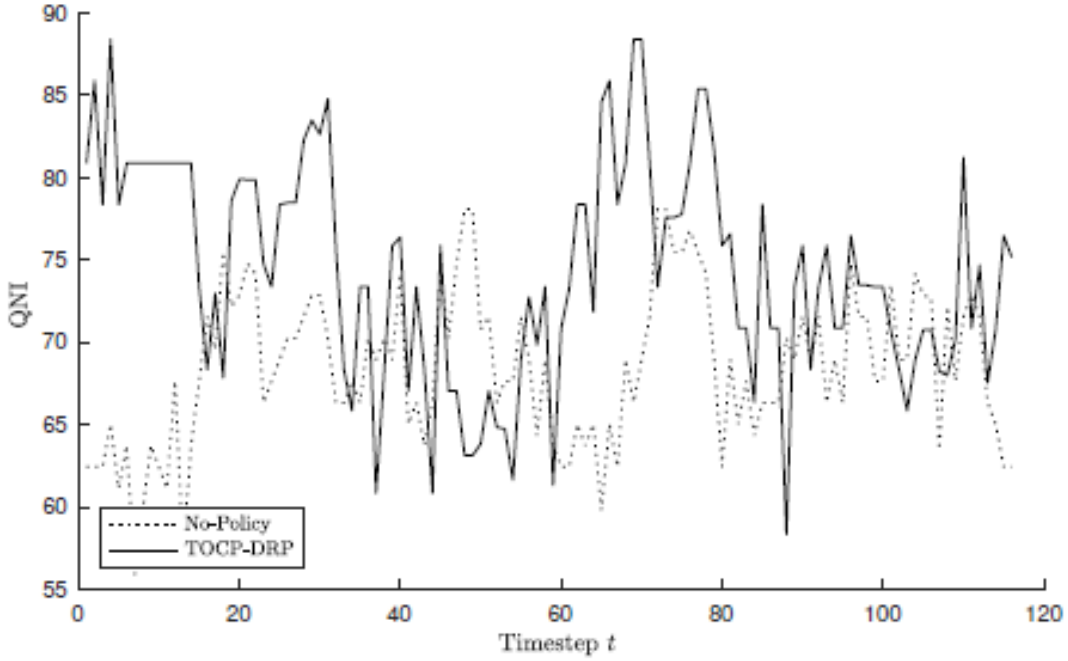

Figure 47:TOCP-DSP vs No-Policy Policy regarding QNI in Mission 2- Exhaustive Scanning

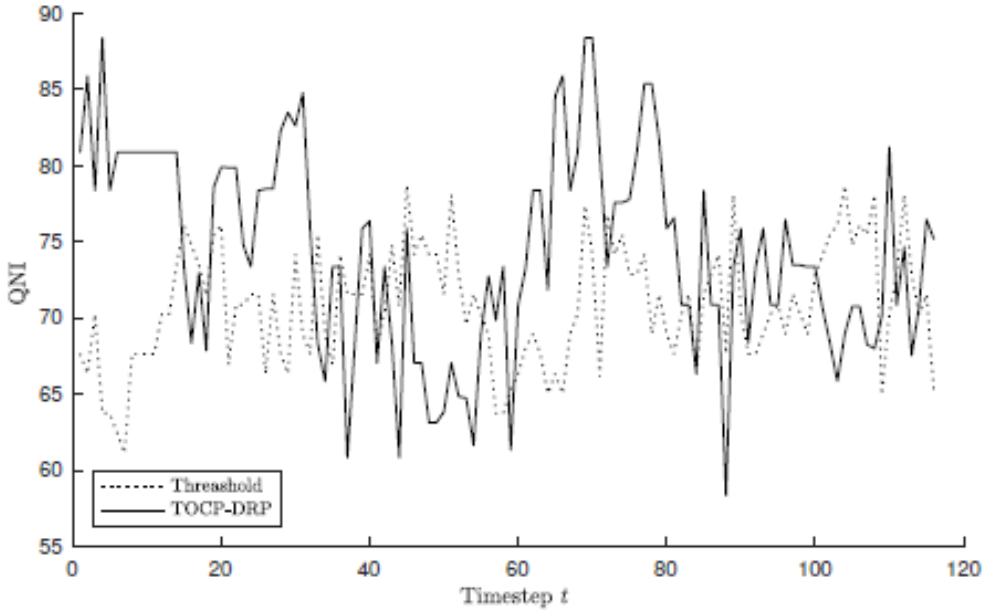

Figure 48:TOCP-DSP vs Threshold Policy regarding QNI in Mission 2- Exhaustive Scanning

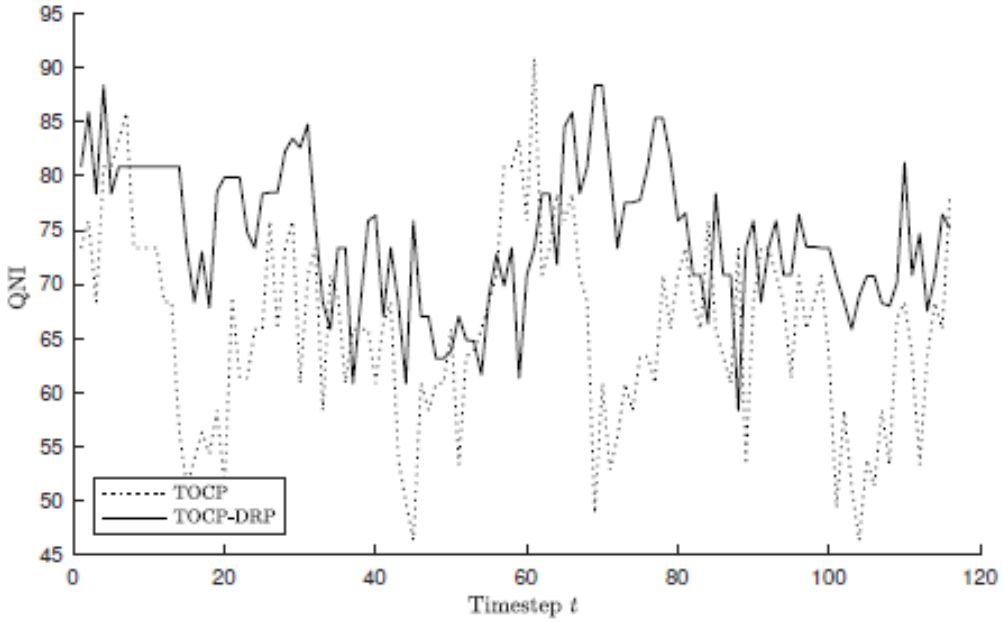

Figure 49:TOCP-DSP vs TOCP-Only Policy regarding QNI in Mission 2- Exhaustive Scanning

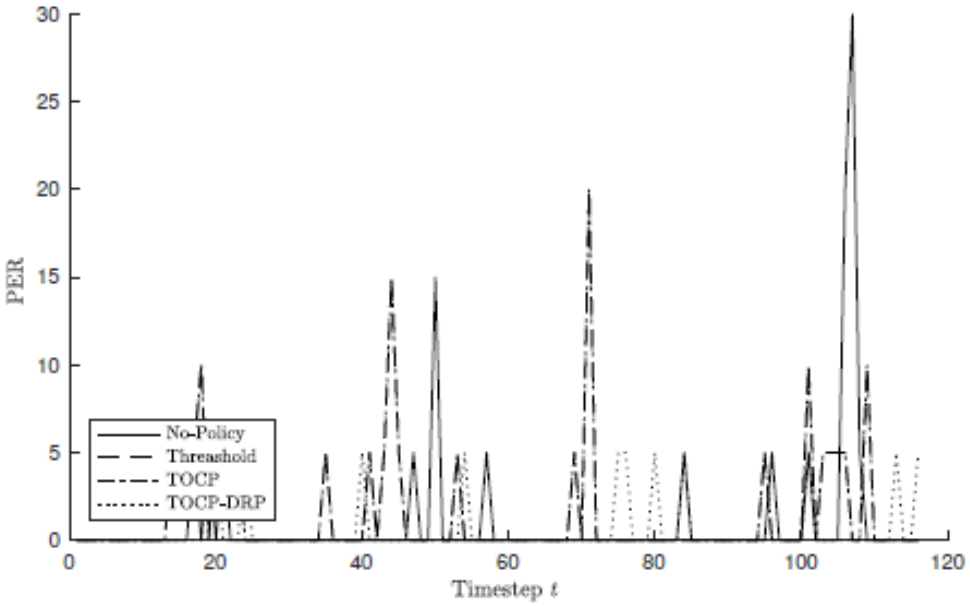

Figure 50: PER of all four policies in Mission 2- Exhaustive Scanning

In the following Figure 51 and Figure 52 latency of No-policy and TOCP-DSP is plotted regarding Mission 1 and Mission 2 respectively. As it is shown TOCP-DSP policy achieves better results in both missions. For Mission 1 TOCP-DSP outperforms No-policy by 24%. Furthermore TOCP-DSP achieves a constant maximum value less than 9% of original latency of the messages.

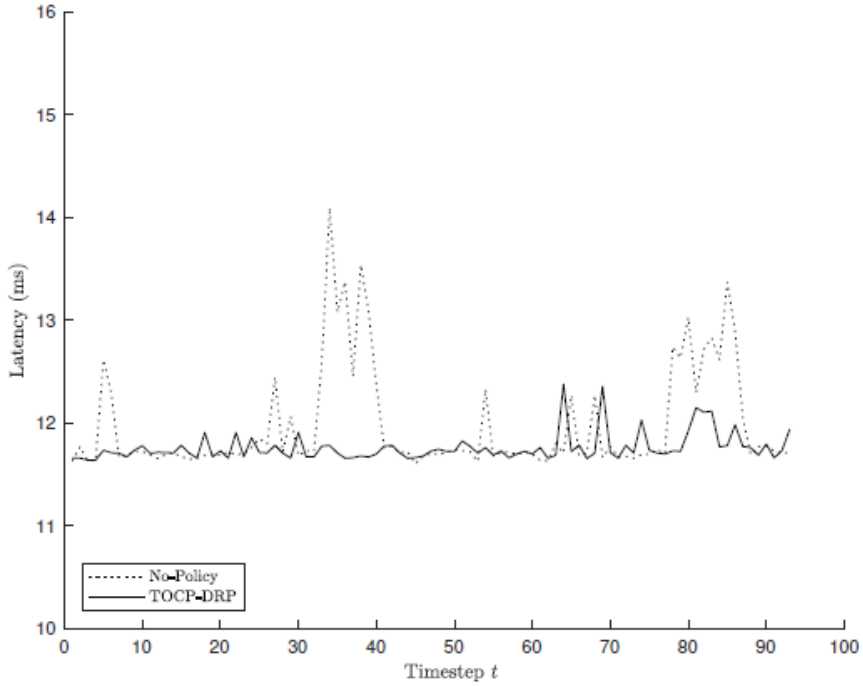

Figure 51: TOCP-DSP vs No-Policy policy regarding latency (ms) in Mission 1- Path Exploration

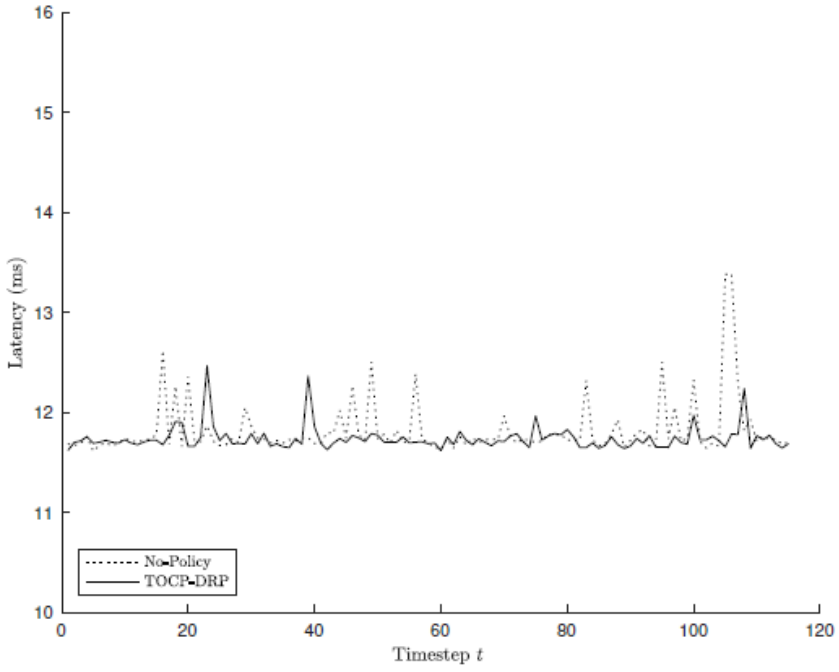

Figure 52:TOCP-DSP vs No-Policy policy regarding latency (ms) in Mission 2- Exhaustive Scanning

Having this data, as one can see, the double optimal stopping game (TOCP-DSP) based on network performance, outperforms regarding PER, QNI, and latency not only the default UxV policy (No-policy), but also the other tested policies (TOCP-Only, Threshold).

7. CONCLUSION

In this thesis a framework that implements a network quality based decision-making process is developed. This framework adapts the information flow between the UxV and the Ground Control Station (GCS) based on network quality metrics (such as packet error rate etc.) and the principals Optimal Stopping Theory (OST). The goal of this framework is to ensure the optimal delivery of critical information from UxV to GCS and vice-versa. If the network behaves optimally then there is no limitation on the information flow, but if the network is saturated or overloaded restriction rules are applied. The proposed model introduces two optimal stopping time mechanisms based on change detection theory and a discounted reward process. The performance evaluation showed the successful delivery of messages in poor network conditions and the moderate production of messages so as not to burden an already saturated network.

ABBRIVIATIONS - ACRONYMS

IoT	Internet of Things
UxV	Unmanned Vehicle, x can stand for aerial, ground or sea
UGV	Unmanned Ground Vehicle
GCS	Ground Control Station
OST	Optimal Stopping Theory
ROS	Robotic Operating System
SLAM	Simultaneous Localization and Mapping
CUSUM	CUmulative SUM
DMP	Decision-Making Process
PER	Packet Error Rate
TOCP	Time-Optimized Change-Point decision making process
QNI	Quality of Network Indicator
SNR	Signal-to-Noise Ratio
FAR	False Alarm Rate
GoP	Group of Pictures
WSN	Wireless Sensor Networks
DSP	Discounted Secretary Problem

APPENDIX I

The source code of this thesis can be found by following the link: https://github.com/Thanoschal/thesis

REFERENCES

[1] "TurtleBot2" [Online]. Available: https://www.turtlebot.com/. [Accessed: 09-Jul-2019]

[2] "Raspberry Pi hardware – Raspberry Pi Documentation" [Online] Available: https://www.raspberrypi.org/. [Accessed: 12-Jul-2019]

[3] "Apache Kafka" [Online] Available: https://kafka.apache.org/. [Accessed: 26-Jun-2019]

[4] F. Fu and M. van der Schaar, "Dependent optimal stopping framework for wireless multimedia transmission," *2010 IEEE Int. Conf. Acoust. Speech Signal Process.*, pp. 2354–2357, 2010.

[5] A. Martinez and E. Fernández, *Learning ROS for Robotics Programming*. Packt Publishing, 2013.

[6] J. M. O'Kane and J. M. O. Kane, *A gentle introduction to ROS*. 2013.

[7] "Sensors supported by ROS." [Online]. Available: http://wiki.ros.org/Sensors#Sensors_supported_by_ROS. [Accessed: 09-Jan-2018].

[8] T. Ferguson, *Optimal stopping and applications*. Mathematics Department, UCLA.

[9] E. S. Page, "Continuous Inspection Schemes," *Biometrika*, vol. 41, no. 1/2, p. 100, 1954.

[10] A. bert N. Shirayaev and A. B. Aries, *Optimal stopping rules*, no. 8. 2008.

[11] G. Lorden, "Procedures for Reacting to a Change in Distribution," *Ann. Math. Stat.*, vol. 42, no. 6, pp. 1897–1908, 1971.

[12] J. Unnikrishnan, V. V. Veeravalli, and S. Meyn, "Least favorable distributions for robust quickest change detection," *IEEE Int. Symp. Inf. Theory - Proc.*, pp. 649–653, 2009.

[13] G. V. Moustakides, "Optimal Stopping Times for Detecting Changes in Distributions," *Ann. Stat.*, vol. 14, no. 4, pp. 1379–1387, 1986.

[14] A. B. Mcdonald, "Survey of Adaptive Shortest-Path Routing in Dynamic Packet-Switched Networks 1 Introduction," pp. 1–29, 1997.

[15] Min Chen, V. C. M. Leung, Shiwen Mao, Yang Xiao, and I. Chlamtac, "Hybrid Geographic Routing for Flexible Energy—Delay Tradeoff," *IEEE Trans. Veh. Technol.*, vol. 58, no. 9, pp. 4976–4988, Nov. 2009.

[16] S. Giordano, I. Stojmenovic, and L. Blazevie, "Position Based Routing Algorithms for Ad Hoc Networks: a Taxonomy," *Ad Hoc Wirel. Netw.*, pp. 103–136, 2003.

[17] Y. Cao and Z. Sun, "Routing in delay/disruption tolerant networks: A taxonomy, survey and challenges," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 2, pp. 654–677, 2013.

[18] H. M. Lin, G. Yu, A. C. Pang, and J. S. Pathmasuntharam, "Performance study on delay tolerant networks in maritime communication environments," *Ocean. IEEE Sydney, Ocean. 2010*, 2010.

[19] C. Anagnostopoulos and S. Hadjiefthymiades, "Delay-tolerant delivery of quality information in ad hoc networks," *J. Parallel Distrib. Comput.*, vol. 71, no. 7, pp. 974–987, 2011.

[20] K. Panagidi, C. Anagnostopoulos, and S. Hadjiefthymiades, "Optimal grouping-of-pictures in iot video streams."

[21] C. Anagnostopoulos and S. Hadjiefthymiades, "Optimal, quality-aware scheduling of data consumption in mobile ad hoc networks," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1269–1279, 2012.

[22] C. Anagnostopoulos and S. Hadjiefthymiades, "Advanced Principal Component-Based Compression Schemes for Wireless Sensor Networks," *Acm Trans. Sens. Networks*, vol. 11, no. 1, pp. 1–34, 2014.

[23] E. S. Page. 1971. Procedures for reacting to a change in distribution. Ann. Math. Statist.42, 6 (1971), 1897–1908.