Software Defined MICRONet

A Report submitted

in partial fulfillment for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

AVIONICS

by

AMAL KRISHNA R

pursued in

DEPARTMENT OF AVIONICS

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

to

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

Thiruvananthapuram

May 2016

Software Defined MICRONet

A Report submitted

in partial fulfillment for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

AVIONICS

by

AMAL KRISHNA R

pursued in

DEPARTMENT OF AVIONICS

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

to

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
Thiruvananthapuram

May 2016

CERTIFICATE

This is to certify that the Project report entitled **Software Defined MICRONet** submitted by **Amal Krishna R**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the Degree of **Bachelor of Technology** in **Avionics**, is a bonafide record of the Final Year Project work carried out by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B. S. Manoj	**Dr. N. Selvaganesan**
Associate Professor	Head of the department
Department of Avionics	Department of Avionics
IIST	IIST

Place: Thiruvananthapuram

May 2016

DECLARATION

I declare that this Project report titled **Software Defined MICRONet** submitted in partial fulfillment for the award of the Degree of **Bachelor of Technology in Avionics** is a record of original work carried out by me under the supervision of **Dr. B. S. Manoj**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram

May 2016

Amal Krishna R

ACKNOWLEDGMENTS

I would like to thank my parents for their constant support and belief in me. I would like to thank **Dr. N. Selvaganesan**, Head of the Department, Avionics, IIST for allowing me to pursue this Project. I wish to express my sincere gratitude to **Dr. B. S. Manoj**, Associate Professor, Avionics, IIST for providing me an opportunity to do my Final Year Project under him and for his continuous guidance and support throughout the Internship. His suggestions and ideas made this Project possible. I would like to thank **Sarath Babu**, Research Scholar, and **Abhishek Chakraborty**, Research Scholar, IIST for helping me with concepts, programming and for revising this report. I would also like to thank lab assistants for allowing me to use the lab and providing me with equipments whenever I needed. Finally, I would like to thank all my fellow classmates who did their Final Year Project under networking for helping me in difficult situations.

Amal Krishna R

ABSTRACT

Software Defined Networking (SDN) is an emerging paradigm in the field of networking which provides the user with a software controlled network. Delay Tolerant Network (DTN) is a network which makes use of Store and forward mechanism for packet transfer due to the high probability of disconnection of links. In this report, we analyze the challenges faced by fishermen in a high delay environment or DTN environment with backhaul links. We address issues such as multi-dimensional optimization of the backhaul network for offshore communication with boats and distributed yet centralized intelligence based on SDN self-organization and resource planning for offshore communication.

Software Defined MICRONet achitecture provides intelligent communication among physical boat clusters in the sea. This will solve the technology challenges faced by the fishermen community in India today, specifically by providing software defined intelligent and adaptable communication and connectivity while they are out at sea. A scaled down model of Software Defined MICRONet enviroment was emulated in a testbed and meaningful results were obtained.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

MICRONet	Mobile Infrastructure for Costal Region Offshore Communications and Networks
WMN	Wireless Mesh Network
WMR	Wireless Mesh Router
SDN	Software Defined Network
DTN	Delay Tolerant Network
LAN	Local Area Network
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SFMS	Store and Forward Message Switching
wmSDN	Wireless Mesh Software Defined Networks
OLSR	Optimized Link State Routing
O2O	OLSR to OpenFlow
IPN	Inter Planetary Networks
ICN	Intermittently Connected Networks
ETX	Expected Transmission Count
ETT	Expected Transmission Time
FMR	Fisher Mesh Router
SN	Super Node
AN	Adaptive Node
APS	Average Packet Size

CHAPTER 1

INTRODUCTION

A computer network is a group of two or more computer systems connected together. The connections can be setup through either wired or wireless networks. In an ideal world, networking ’just works’. Your network connection is reliable, fast, and of low latency. Real world networks may not be completely reliable as links can break and, thereby, data rates may vary due to disruptions caused by broken links. An overloaded or broken network link can result in packet loss. If a link loses enough packets, it may be difficult to establish connections across that link, and performance may fall to a tiny fraction of what one expects. When a network link becomes saturated, routers on either side of that link buffer the traffic to avoid losing data. This adds additional latency.

A Wireless Mesh Network (WMN) is a communications network made up of nodes organized in a mesh topology. Nodes in a mesh network have relatively low power requirements, which could potentially be met by low-cost or renewable energy sources. For example, the US military forces use WMNs to connect their computers in the front line. The existence of multiple paths from one source to another destination increases reliability. An ad-hoc network is a local area network (LAN) that is built impulsively as the devices connect. Operating in ad-hoc mode allows all wireless devices within each other’s connection range to discover and communicate in peer-to-peer fashion without a central access point. A basic community network deployed using WMN is shown in Figure 1.1. Each of the houses in the community has an access point and they are connected to each other.

Delay Tolerant Networks(DTN) are designed to withstand long delays or outages, capable of storing packets in intermediate nodes until a stable link gets established. DTNs are designed to operate effectively over extreme distances such as those encountered in space communications or an interplanetary scale. In such an environment, long latency, sometimes measured in hours or days is inevitable. However, similar problems can also occur over more modest distances when interference is extreme or network resources are severely overburdened. SDN is a novel approach that is dynamic, manageable, cost-effective, and adaptable. In this approach, network intelligence is de-

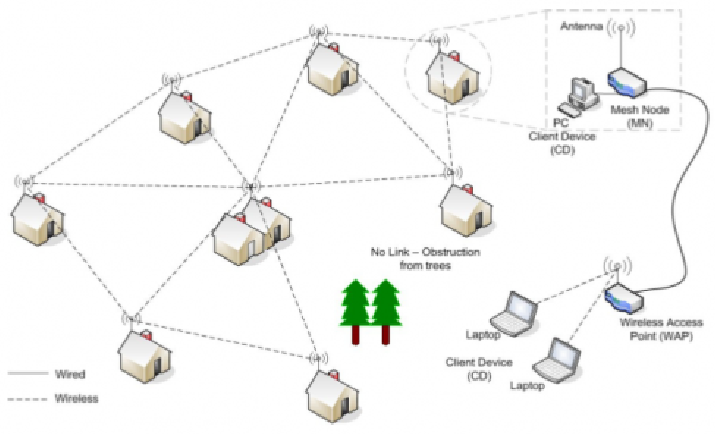


Figure 1.1: A Community network deployed WMN

coupled from the physical infrastructure. The control plane is the system that makes decisions about where the traffic is sent from the underlying systems that forward the traffic whereas data plane is the selected destinations to which the traffic is forwarded. The separation of the control plane and the data plane within the routers, allows network administrators to manage network services through abstraction of lower-level functionality. OpenFlow [1] is a communications protocol developed at Stanford University that gives access to the forwarding plane of a network switch or router over the network. A flow is an artificial logical equivalent to a call or connection present in the OpenFlow table in the network switch.

Fishing in India is a major industry in its coastal states, employing over 14 million people. Radio telecommunication at sea had undergone a sea of change in the last century. After the days of semaphores and flags, radio brought about a drastic change in marine communication at sea. Ship-to-ship communication was brought about by VHF radio. Design and implementation of a novel adaptive, demand aware, energy efficient algorithm and protocol for provisioning the backhaul links of a marine offshore communication network is important. Making such a network a Software Defined one can make the network more centralized and efficient. This is the task which needs to be addressed through this project. In order for that master-slave controller selection in an SDN environment and plausible usage of Expected Transmission Time(ETT) as metric for the same has to be analyzed and studied.

1.1 Motivation

Small-scale fishing is one of the major source of livelihood across thousands of villages in India, employing over 14 million people in India alone. Fishing sector has the highest index of relative risk [?] compared to any other occupation. The majority of their boats are poorly equipped in terms of security, communication, and navigation. There is a need for reliable and low-cost communication technology solutions which will enable fishermen to communicate between vessels in the sea. While developing an architecture for this purpose an additional overlay of Software Defined Networking will help the fishermen to maintain communication between the far away physical cluster of vessels and during disaster scenarios. The SDN controller has end-to-end visibility of a network. With this information, SDN is uniquely positioned to have a positive impact on the network. It can also perform existing applications better.

1.2 Major Contributions

- Development of Software defined MICRONet architecture
- Performance analysis and results for Expected Transmission Time (ETT) as a better metric for controller selection
- Performance analysis and results for Software defined MICRONet architecture

1.3 Overview of the Report

The remaining of the report is organized as follows. Chapter 2 discusses existing works related to MICRONet, Software Defined Networking, and other important concepts and metrics used for the Software-defined MICRONet architecture. Our architecture of Software Defined MICRONet is described in Chapter 3. Chapter 4 describes the implementation of Software Defined MICRONet architecture. Performance analysis, experiments, and results are discussed in Chapter 5. Finally, we conclude the report in Chapter 6 with a note on possible future work.

CHAPTER 2

LITERATURE SURVEY

Software defined netowkring approach allows us to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed firmware. Incorporating this concept to offshore fishermen communication, each of the physical boat clusters can be controlled by the network user. The existing MICRONet architecture gives a hierarchical model of offshore communication nodes which provides better performance. In this chapter, we will go through the literature survey papers that needs to be studied in order to give a brief overview of the existing MIRCONet architecture and other concepts that can be subsequently applied onto the Software Defined MICRONet architecture for it's improving performance and controllability.

2.1 MICRONet

In this section, the existing MICRONet architecture is studied in detail. In [4], the author proposes a resilient lightweight protocol for optimized dynamic provision of the backhaul network based on an adaptive and demand driven algorithm. The algorithm needs to be optimized from a local controller running at the network edge at each of the clusters.

2.1.1 Network Entities

The network entities as described by the authors in [4] is as follows:

1. Base Station (BS): The BS is the ground level entity which is connected to the outside world through Internet and relays this Internet through to the clusters at the sea.

2. Super Node (SN): The SN connects to multiple physical clusters at a time as is near to the shore with a backhaul link to the BS. The SN acts as an intermediate

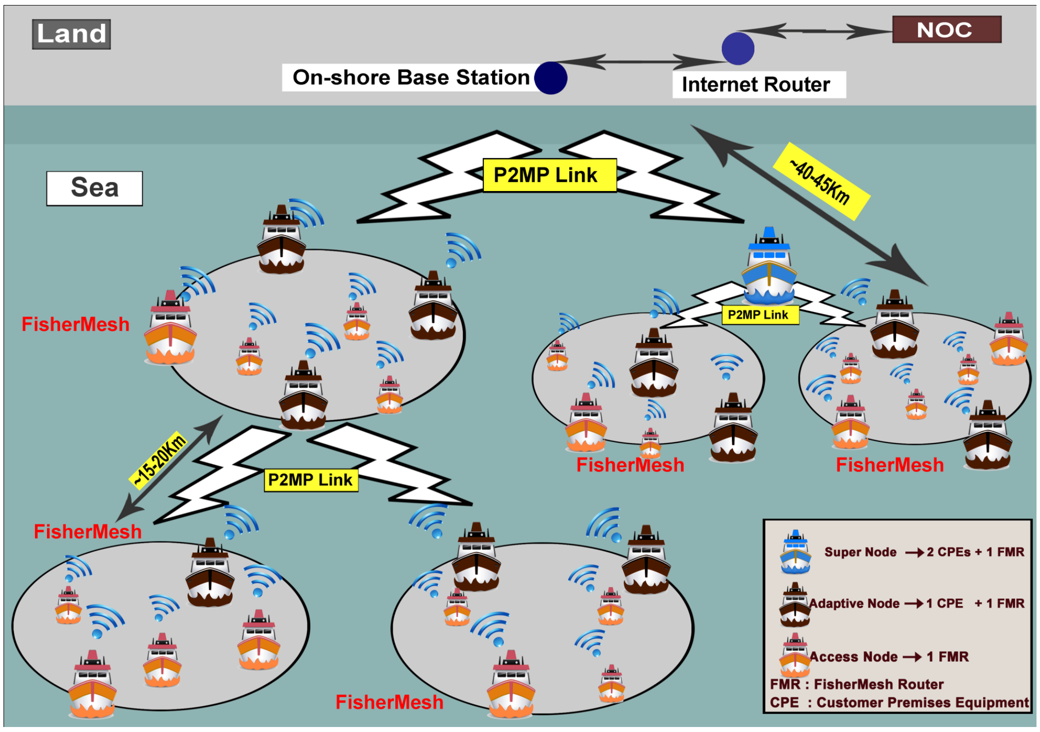


Figure 2.1: MICRONet Achitecture [4]

entity for communication between the BS and the routers at the physical clusters at sea. Each SN is a boat carrying one Router and 2 CPEs.

3. Local Controller (LC): The LC is responsible for broadcasting the periodic beacon message in specific intervals to advertise the network and the control information. A Standby LC is also introduced whose function is to take over the control as the next LC in case the router moves out of the network or becomes unresponsive. Adaptive Nodes (AN) have the capability to become LC in their specific clusters. Each AN is a boat carrying one router and one CPE.

4. Customer Premises Equipment (CPE): CPEs are Customer Premises Equipment's which do not directly take part in the message exchange process instead helps in creating long range back-haul links with AN. The AR connected to the CPE shares the CPE status with the LC. The CPE will also relay the cluster topology information from LC to the GC at BS.

5. Routers: Routers for the mesh network.

2.1.2 Hierarchy in MICRONet

In the MICRONet hierarchy as seen from fig 2.2, a cluster with N boats will have N routers, one on each boat and M CPEs. Routers will form a mesh network connecting to each other, and use CPEs as the gateways to connect to the BS over the backhaul network. As mentioned earlier, the SN is equipped with 2 CPEs, Adaptive Node (AN) with 1 CPE, and routers have no CPE according to the design proposed by [4]. A hierarchical model is proposed with a Base Station(BS) at the top level, which is on land connected to the Internet. Then a SN on the 2nd level which is a large boat nearer to the shore. Then comes the AN which consists of a LC which are medium sized boats in each of the clusters far away from the shore, and Local Controller sub-clusters. Next the boats with just the Fisher Mesh Routers (FMR) and so on. Running the cluster controller centrally at the BS would incur communication overheads as the topology of the mobile cluster keeps on changing dynamically. The SN may not be able to communicate with all boats in the cluster and may even end up losing connectivity to the cluster completely as a result of physical distance. There will be one LC per cluster, running at the network edge. A SN will run on the onshore, near base station to manage the local controllers in the clusters. Control information updates will flow predominantly from the edge to the core. Backhaul links connect each of the physical clusters. The Backhaul links are formed with the help of CPEs in the ANs. The selection of ANs to take part in the backhaul network is determined by the routers which have been elected as the controller by employing the proposed algorithm. Routing within the mesh network is done using OLSR routing protocol running on all routers. For a CPE to provide an effective backhaul link, the SNR of the backhaul link to the BS should be greater than a minimum threshold.

2.1.3 Prototype Implementation

The prototype implementation of the work in [4] was setup by creating a testbed of three ARs, two NanoStation M2 CPEs and a Rocket M2 BS from Ubiquiti Networks. The challenges in implementing this system were explored and lack of open source platform for simulation environment was one of them. The algorithm is implemented in a custom designed controller developed using Java Spring Framework. The communication

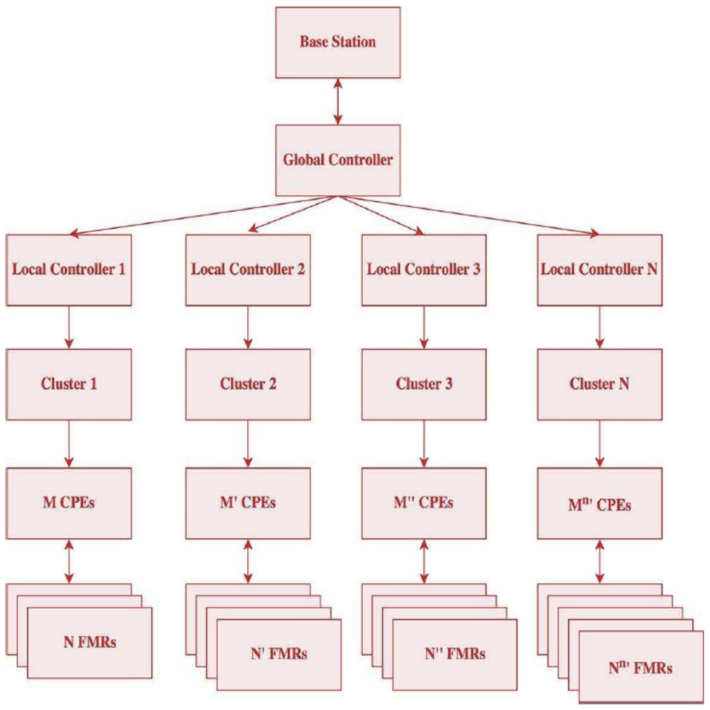


Figure 2.2: MICRONet Hierarchy [4]

network was modeled using real hardware. The small-scale prototype developed using a combination of physical and simulated devices in this work yielded encouraging positive results as per the author of [4].

2.2 Software Defined Networks

Software defined networks is a new paradigm in the field of networking where the network intelligence and state are logically centralized. This is done by decoupling the control plane and the data plane. Control plane is the system that makes decisions about where the traffic is sent from the underlying systems that forwards the traffic. Data plane is the selected destinations to which the traffic is forwarded. Control plane deals with routing, switching and decision making, while the data plane deals with the user traffic. In traditional networks both data and control plane resides together in the switches and are difficult to manage as they are to be dealt individually. Changing configuration of such switches became difficult and had to rely on frequent updates from the companies for any change in the system. The control plane of SDN provides an easy way for the network administrators to manage different parameters of the network. SDN offers numerous benifits including automation, increased uptime, less drain on

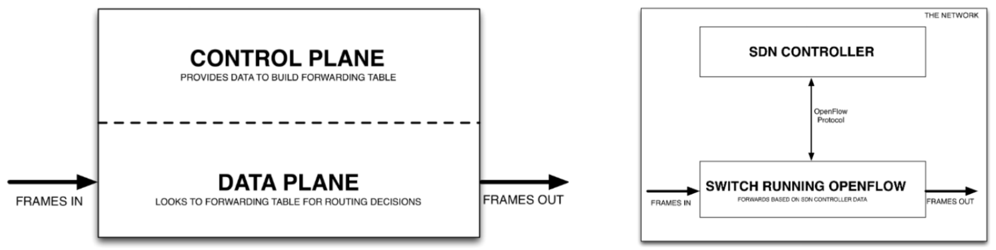


Figure 2.3: Traditional vs SDN Architecture [2]

resources, better visibility and the ability to scale the network resources with application and data needs. SDN paradigm shifts the networks more to a software based paradigm than the existing hardware based system.

2.2.1 SDN vs Traditional Model

In the traditional model, the network devices have a control plane that provides information used to build a forwarding table. The module also consists of a data plane that consults the forwarding table. The forwarding table is used by the network device to make a decision on where to send the packets. Both these planes exist together on the networking device such as switches or routers.

SDN abstracts this concept and places the control plane functions on an SDN controller. The SDN controller can be a server running SDN software such as RYU or POX. SDN provides the user with granular control over the way switches handle data, giving them the ability to automatically prioritize or block certain types of packets. The controller communicates with a physical or virtual switch data plane through OpenFlow protocol as flow rules. OpenFlow passes instructions to the data plane on how to forward data. This, in turn, allows for greater efficiency without the need to invest in expensive, application-specific network switches. The comparison of traditional and SDN architecture is shown in Figure 2.3.

2.2.2 OpenFlow

Modern Ethernet switches and routers contain flow-tables that perform forwarding functions based on packet headers. While each vendor has a different flow table, there is a common set of functions supported by a wide variety of switches and routers. This common set of functions is leveraged by OpenFlow. OpenFlow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture. It is an open standard that enables researchers to run experimental protocols in campus networks [5]. Due to the popularity of OpenFlow, many open source switches such as OpenvSwitch, Open Reference Switch and controllers such as RYU, POX etc are available. OpenFlow protocol is used to communicate between control and data planes in an SDN architecture. With OpenFlow, a single central controller can program all the physical and virtual switches in a network.

2.2.2.1 OpenFlow Switch

An OpenFlow switch is a software program or hardware device that forwards packets in an SDN environment. OpenFlow switches are based on OpenFlow protocol. The main components of an OpenFlow switch is shown in Figure 2.4. In an OpenFlow switch, all the functions of a conventional switch are run in the central OpenFlow controller. The OpenFlow enabled switch supports both OpenFlow flow forwarding and traditional ethernet switch routing. An OpenFlow Switch consists of one or more flow entries is shown in Table 2.2, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. In a conventional switch, the decision-making and packet forwarding are within the same device. The OpenFlow switch communicates with the controller and the controller manages the switch via the OpenFlow protocol. Using OpenFlow, the controller can add, update, and delete flow entries in flow tables.

2.2.2.2 Flow Table

OpenFlow switches support flow-based routing by making use of flow tables. Every flow table contains a set of flow entries and there can be multiple flow tables. Each flow entry consists of match fields, counters, priority, timeout, and a set of instructions. Table 2.1 shows the main components of a flow entry in a flow table and Table 2.2 shows a typical OpenFlow flow table with various match fields and actions. These flow entries

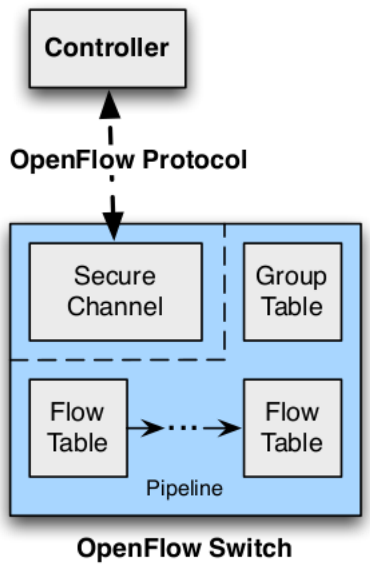


Figure 2.4: Main Components of an OpenFlow Switch [1]

Table 2.1: Main Components of a Flow Entry in a Flow Table [1]

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

are used for routing by matching with the fields of incoming packets. A flow table entry is identified by its match fields and priority. The match fields and priority are taken together to identify a unique flow entry in the flow table. The flow entry that wildcards all fields and has priority 0 is called the table-miss flow entry. OpenFlow tables undergo pipeline processing where each OpenFlow switch contains multiple flow tables, each flow table containing multiple flow entries. How packets interact with those flow tables is defined by the OpenFlow pipeline processing.

2.2.2.3 Operation of OpenFlow

When a packet arrives on a virtual port in the OpenvSwitch, the kernel module processes it by extracting its flow key and looking it up in the flow table. If there is a match, it

Table 2.2: OpenvSwitch Flowtable Example

MAC Src	MAC Dst	IP Src	IP Dst	TCP Dport	....	Action	Count
*	00:08:*	*	*	8.8.8.*	*	port 1	250
*	*	*	10.10.10.4	*	*	port 2	400
*	*	*	127.0.0.1	*	*	local	320
*	*	*	*	*	*	controller	140

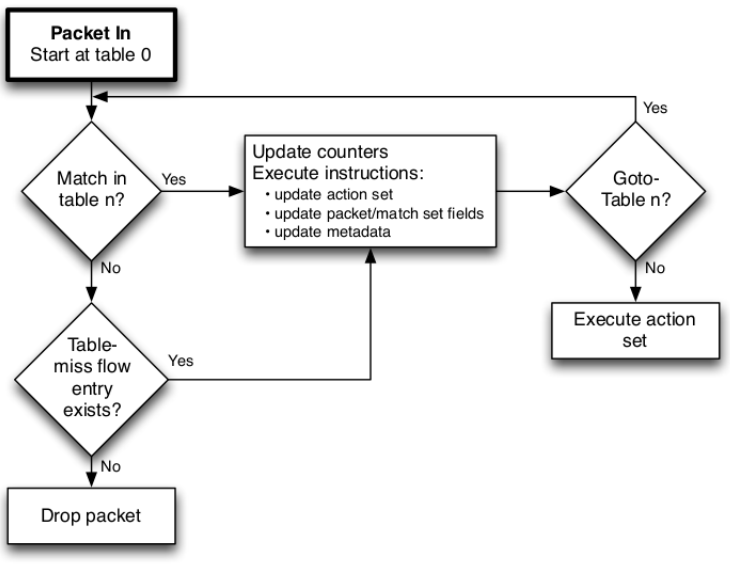


Figure 2.5: Packet flow through an OpenFlow switch [1]

executes the associated action. In the case of zero matching flows, OpenvSwitch checks for a table miss flow entry. If a table-miss flow entry exists the packet is dropped instantly or else the user action for table-miss flow entry is executed. Instructions may also direct the packet to any other flow tables for further processing. This pipeline processing continues till the corresponding flow entry does not direct the packet to another flow table. The basic flow of a packet once it reaches a switch is shown in Figure 2.5. In each flow table, the actions are added to the action set and are executed at the end of pipeline processing [1].

2.2.3 Related work

Since SDN is a recent paradigm for networking, immense research is being carried out in recent years. The work in [6] describes the definition and implementation of a solution for OpenFlow-based routing in WMNs and its application to the mobility management of mobile clients. The migration of client addresses in the case of highly dynamic environments is described in [6] and have proposed a new OpenFlow-based architecture which allows flexible control of packet routing in WMNs. The architecture combines the benefits of OpenFlow which enable flexible packet forwarding and WMNs with self-configuration and error-resilience in WMNs. The architecture is based on an out-of-band control network, as proposed by Dely et al in. [6], which shows an acceptable performance compared with the 802.11s standard. It is concluded in [7] that

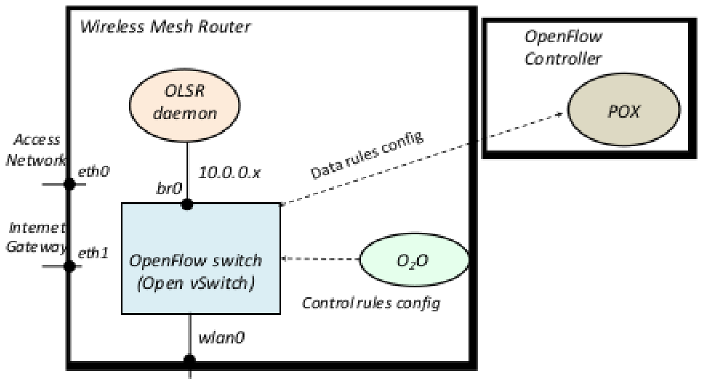


Figure 2.6: wmSDN router Architecture [3]

OpenFlow can be used for monitoring, mobility management, and network admission control.

In [3], the authors address the scenario of controller (POX) failures by creating an Optimised Link State Routing (OLSR) to OpenFlow (O2O) conversion table. The wmSDN architecture is shown in Figure 2.6. OLSR is a routing protocol developed for mobile ad-hoc networks. It exchanges the network topology information with other nodes using topology control packets. O2O module periodically checks for the liveliness of the controller. When a controller fails, the O2O module enters in an emergency status. During this interval, it removes all the existing rules inserted by the controller from the flow table. Then dumps all the OLSR routing table to make a dummy OLSR table in the OpenFlow. The dummy OLSR table includes the routes outside the control subnet and the default route advertised by the gateways. Routing of the mesh becomes completely controlled by the OLSR. When the controller becomes alive, the O2O leaves the emergency status and removes the existing entries in the flow table. This forces the ongoing data flows to send the packet in data units to the controller, which decides how to route the messages. Thus, the emergency situation of controller failure is handled using the information from the OLSR table. The authors use two different levels of controllers for setting up OpenFlow rules. 1) a local distributed controller taking care of control rules that couples OLSR to O2O, 2) and a remote centralized controller taking care of the rules for data traffic. An application of Wireless Mesh Software Defined Network (wmSDN) architecture has been mentioned in [3], in which OpenFlow switching is used to balance the traffic among gateways of a Wireless Mesh Network.

Work done so far concludes that OpenFlow can be used for monitoring systems, mobility management, and network admission. With the help of OpenFlow, the controller failure is handled by creating an O2O table conversion which is addressed by A. Detti in [3]. Wireless Mesh Networks may benefit from the flexibility and the simple management provided by the Software Defined Networking paradigm. The use of wireless resources can be optimized by a central server, which can perform processing actions on multiple levels of the protocol stack. A solution to integrate SDN functionality in a Wireless Mesh, trying to face the reliability concerns related to this environment has already been achieved.

2.3 Controller Selection in WMN SDN

In an SDN scenario with multiple concurrent controllers, the different controllers need to share a common view of the topology and of the network events that are relevant to take decisions in the controller layer. The controller also needs to synchronize about which controller is the master for each switch, performing the master election procedure.

In [8], the author consider the issue of controller selection in a scenario with intermittent connectivity. The author assumes that over time a single WMN can become split in two or more partitions and that separate partitions can merge into a larger one. Here, a set of SDN controllers can potentially take control of the Wireless Mesh Routers (WMR). At any given time, only one controller should be the master of a WMR based on a master election algorithm and similarly, the other controller will act as the slave.

Authors of [8] focus on the master assignment problem and propose a naive approach of master-selection rather than election. Each switch has a priority list of controllers and a switch autonomously selects the controller with the highest priority as the master. Here, in order to develop the architecture, the different controllers need to share a common view of the topology and of the network events that are relevant to take decisions and the controllers need to role information. The assumption here is that the OLSR topology information is available to all the nodes in the network. The controllers learn the topology and receive topology updates using OLSR. The master election procedure repeats every time the network gets partitioned.

The designed procedure has the WMR itself in charge of selecting the most appropriate controller given the topology. And the controller extracts the topology information from a nearby WMR. Therefore, from the topology discovery point of view, the WMR acquires topology information even before the controller. A WMR can also directly check the connectivity with potential controllers trying to establish TCP connections towards them as explained in [8]. The authors have experimented a sort of hard handover of the controller to the WMR. The authors of [8] also give more benefits on a WMR initiated controller selection. Performing the master selection on the WMR side has some advantages in our scenario:

- Each OpenFlow switch will be connected to a single controller at a time, and no conflicting rules can be injected.
- A coordination mechanism among controllers is not needed, each controller can operate on its own.

The authors of [8] have implemented the solution and deployed it in two environments: 1) a distributed experimental setup over physical wireless nodes in NITOS [8] interconnected with Ethernet over UDP tunnels across PlanetLab Europe 2) a single machine environment using the CORE emulator [8]. Network merging and network partitioning experiments were carried out. In the Network merging experiment they evaluated the time needed for the WMRs to connect to a higher priority controller after the merging of two network partitions. The time for this case is decomposed into two phases: 1) network connectivity, and 2) master selection. Network connectivity considers the time needed for the routing protocol to setup the IP routes in all WMRs taking into account the merged network topology. In the experiments carried out as shown in [8] it averages to 15 seconds. This is consistent with the OLSR routing protocol mechanisms, as three Hello messages need to be received in order to declare a link up and the default interval for sending OLSR Hello messages is 5 seconds. As for the master selection, the polling period is 3 seconds, in the experiment, it measured an average of more than 1.5 seconds for the latest connected WMR. In the network partitioning experiment, again the time required to switch from one controller to another was computed. In this case, the WMRs do not rely on OLSR to discover that a controller is not reachable, as it would require more than 15 seconds considering the default OLSR configuration. The polling procedure considers a 2 seconds timeout before declaring that a controller is down. With this procedure, an average master selection delay of 5.5 seconds was measured in the experimental setup in [8]. Performance experiment in the single-machine deployment showed results that were fully in line with the expectations of the work done.

2.4 ETT Metric for OLSR WMN

Authors of [9] presents the design of an Expected Transmission Time (ETT) plug-in for the OLSR protocol. By default, OLSR uses hop count and Expected Transmission Count(ETX) as the metric. For routing in simple terms, the goal of using ETX is to find the route with the highest probability of packet delivery,instead of the shortest path. ETX does not distinguish links with different capacities, and the loss probability of small probes differs from the loss probability of data packets. In order to cope up with these problems, the ETT metric was proposed in [9]. ETT estimates the time a data packet takes to successfully transmit over a link. The ETT metric can be calculated by multiplying the ETX value with the ratio of packet size and the transmission capacity of the link. Thus,

$$ETT = ETX * \frac{S}{B}$$

where S is the packet size and B is the link capacity. In this technique, a pair of back-to-back probe packets, one small followed by a large one, are sent to each neighbor. The neighbor then measures the inter-arrival time between the two packets and reports it back to the source. Upon receiving a pre-defined number of delay samples, the sender estimates the capacity of the link by dividing the size of the larger probe by the smallest delay sample obtained.

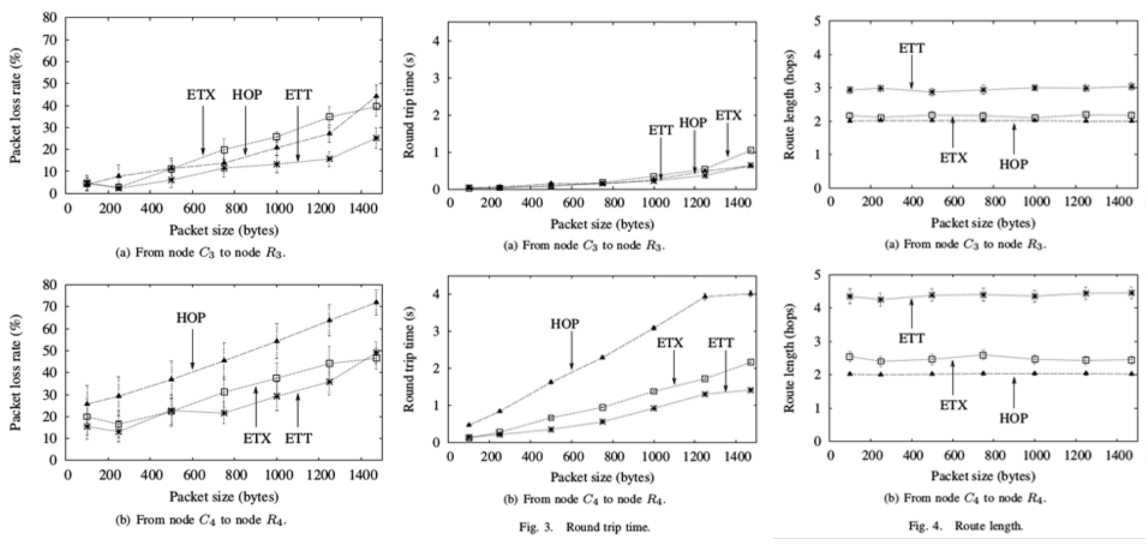


Figure 2.7: ETT results [9]

The experiments were carried out in an indoor testbed to validate our ETT plug-in. The performance of the OLSR protocol considering the ETT, the hop count, and the ETX metrics were evaluated. As the packet size increases, the packet loss rate also increases. The ETT metric is the only one that estimates the transmission time, which takes into account the parameters such as link throughput and bandwidth. Hence, ETT performs better than the ETX and the hop count metrics in terms of round trip time. ETX and ETT metrics choose routes according to link quality, whereas hop count chooses the shortest route. This leads to a trade-off between route length, which translates into the number of medium accesses, and link quality. ETX and ETT metrics produce longer routes than hop count, the route links have better quality and send packets using higher physical rates. This explains the lower packet loss rate and the lower round trip time for ETX and ETT metrics. The experiment showed that the ETT metric performs better than both hop count and ETX metrics.

2.5 Summary

The MICRONet architecture was studied in detail and acts as the base for the Software Defined MICRONet architecture. Further Controller selection in WMN SDN and ETT metric for OLSR was also studied which is implemented on the proposed Software Defined MICRONet achitecture.

CHAPTER 3

SOFTWARE DEFINED MICRONet

Software Defined MICRONet is designed to provide fishermen stable and reliable communication in the sea. It enables proper communication between fishing vessels that are at sea for days completely cut off from the mainland. It also optimizes the communication within the physical clusters of boats at fishing zones and also between the clusters and the Base station.

3.1 Software Defined MIRCRONet Architecture

We design the Software Defined MICRONet using different components as shown in figure 3.1. The key modules in this architecture are (i) Packet Pair Module, (ii) APS Module, (iii) ETT Module, (iv) Processing Module, (v) OLSR Module, (vi) Open VSwitch, (vii) Controller. Each of these modules are discussed in detail here.

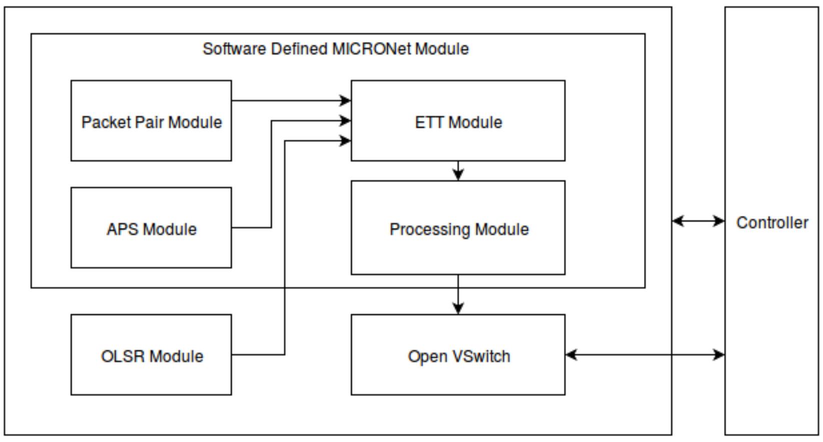


Figure 3.1: Software Defined MIRCRONet Architecture

3.1.1 Packet Pair Module

The packet pair module computes the link bandwidth to calculate ETT to use it as a metric for controller election.

3.1.1.1 Packet Pair Technique

The packet-pair technique uses two probe packets of different size sent back to back from the source to a destination. The link bandwidth is calculated by taking the minimum of 10 iterations of the difference in the arrival of the two probe packets at the destination.

3.1.1.2 Implementation of Packet pair technique

The probing is done by measuring the round trip time from each neighbor for two packets, namely: small unicast packets (e.g 137 bytes used in the experiment), and large unicast packets (e.g. 1137 bytes used in the experiment). By comparing the trip times of these packets, an indication of the link speed can be derived. First, a small unicast probe packet is sent to the neighbor. Secondly, the neighbor receives the probe packet and sends an equally small reply packet back to the sender. Thirdly, a large unicast probe packet is sent to the neighbor. Finally, the neighbor receives the probe packet and sends the same reply packet back to the sender.

Measure the round-trip time of the large and the small packets. On average, the single trip time of the packet is half that of the measured round-trip time. Round Trip Time (RTT) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received. The difference in the time of arrival of the small and the large packet at the receiver will be the sum of the single trip time of the large packet and the single trip time of the small packet. The available bandwidth is then estimated from the difference of the arrival of the small and large probe packets at the receiver. This process is repeated 10 times and the minimum value of the difference is taken. For example, say the arrival time difference of a packet of 1472 bytes is 1.472 milliseconds. Then the estimated bit rate is 1472 * 8 / 0.001472 = 8 Mbits/sec.

In this work, we implemented the aforementioned packet-pair technique into the default routing mechanism of OLSRd. Packet pair technique is incorporated into OLSR to calculate the throughput supported by each link, which is then used to compute the ETT metric. The packet-pair technique is performed, periodically every 10s. A per-packet approach is avoided as to minimize unnecessary network overhead.

3.1.2 Average Packet Size Module

The Average Packet Size (APS) Module calculates the average size of packets that are flowing through a link. We capture 10 packets from the link using TCP dump and computer the average packet size. This value is then passed onto the ETT module for calculation of ETT.

3.1.3 OLSR Module

The OLSR Module takes ETX data from the olsrd to the Software Defined MICRONet module. The real-time ETX values are required in order to compute the corresponding ETT values.

$$ETX = \frac{1}{Df * Dr}$$

where Df is the forward delivery ration and Dr is the reverse delivery ratio. ETX is the number of transmissions required to successfully deliver a packet over a wireless link.

3.1.4 ETT Module

ETT is the expected time to successfully transmit a packet over a link. The ETT module computes ETT for each link with the information received from the Packet Pair module and OLSR module.

$$ETT = ETX * \frac{S}{B}$$

where S denotes the average size of packet and B denotes current link bandwidth. The link bandwidth is calculated by the Packet pairing module and passed on to the ETT module. The Average Packet Size is calculated using the APS module and passed to the ETT module. Finally, this enables the router to calculate the ETT value for each of the connected links.

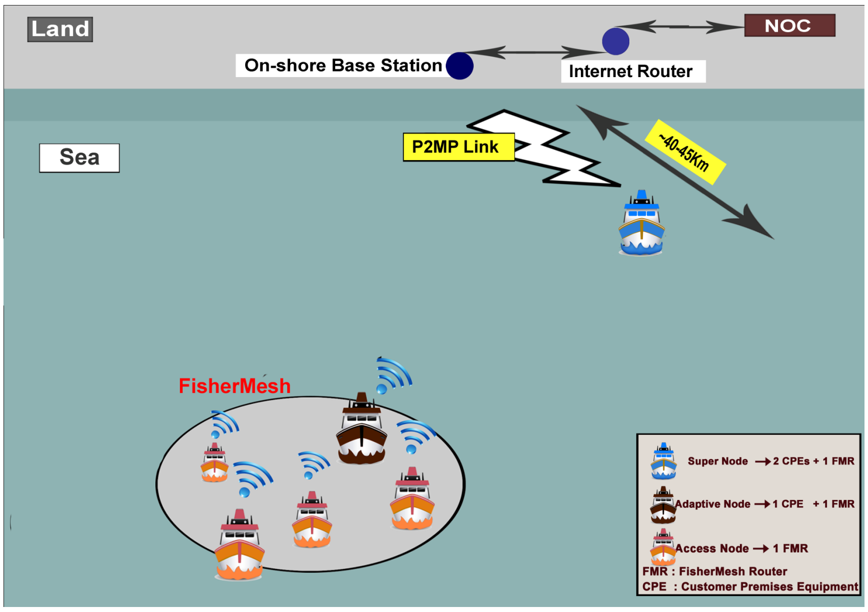


Figure 3.2: No back-haul links

3.1.5 Processing Module

The Processing Module decides the appropriate algorithm to be taken for various scenarios that can arise in a MICRONet environment. We analyze three different scenarios possible in the MICRONet environment.

3.2 MICRONet Scenarios

In this section we explain in detail the three different scenarios and solutions for each of them. The three scenarios we consider in our design are (i) No back-haul links, (ii) 2 back-haul links and (iii) hirearchical sub cluster with single back-haul link. Each of these scenarios are explained in details.

3.2.1 Scenario 1 - No Back-haul links

Here, the physical cluster formed has no back-haul links. The nodes in the cluster are completely cut-off from the base station and there is no means to communicate back to the shore as shown in Figure 3.2. We assume that the cluster of boats move far away from the shore such that they get completely cut-off from the bigger network with the Super Nodes. In this case, the isolated physical cluster is assumed to have a dynamic number of Adaptive Nodes and boats with routers alone. They may continue to form an isolated network away from the earlier network which they were a part earlier. Software Defined MICRONet Architecture has to enable efficient communication algorithm in such a cluster.

```
start olsrdm;
while back-haul links = 0 do
    determine the ANs;
    if number of AN != 1 then
        master-election();
        master = ANi slave = ANj start controller in ANi and
          ANj;
        invoke set-controller command in rest of the FMRs;
        start open-vswitch in rest of the FMRs;
    else
        master = AN;
        start controller in master;
        invoke set-controller command in rest of the FMRs;
        start open-vswitch in rest of the FMRs;
    end
end
```

Algorithm 1: No back-haul links

According to the MICRONet hierarchy shown in figure 2.1, Adaptive Nodes have a higher capability to work as the controller in the cluster. So after confirming the absence of any backhaul links in the network, the controller is activated in the Adaptive Node and the switches will run on the Access Routers and take control signals from the controller. In the case of multiple Adaptive Nodes in the same isolated physical cluster, each of the router will use the ETT metric for controller election procedure. The Software Defined MICRONet Processing module will decide which controller should act as the master and slave with the available ETT information.

3.2.2 Scenario 2 - 2 Back-haul links

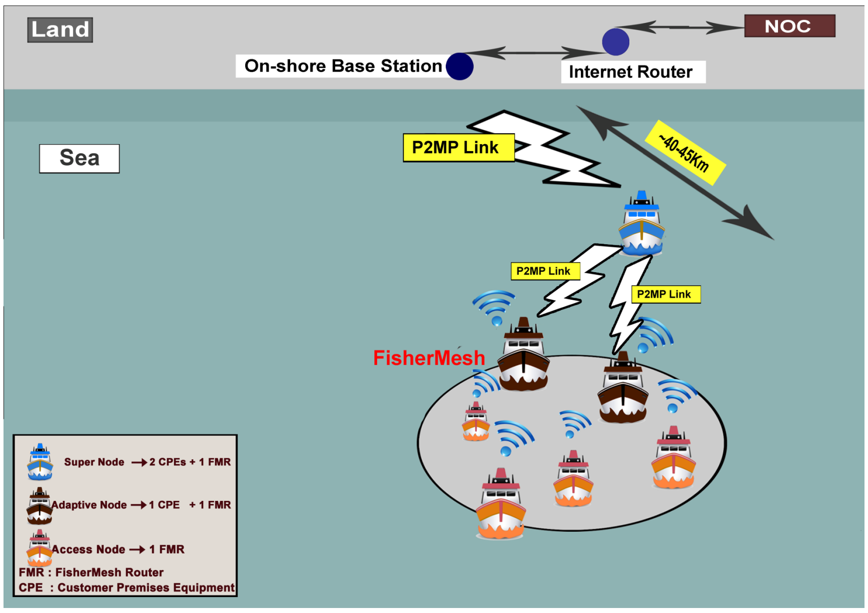


Figure 3.3: 2 back-haul links

```
start olsrdm;
while back-haul links = 2 do
    adaptive-node-list();
    compute ETT to both access node;
    if $ETT\text{-}AN1 < ETT\text{-}AN2$ then
        master = AN1;
        slave = AN2;
    else
        master = AN1;
        slave = AN2;
    end
    start controller in master;
    invoke set-controller command in rest of the FMRs;
    start open-vswitch in rest of the FMRs;
end
```

Algorithm 2: 2 back-haul links

The physical cluster formed here has two back-haul links. The nodes in the cluster can communicate to the base station through the backhaul links as shown Figure 3.3. In this scenario, we assume that the physical cluster formed is able to maintain

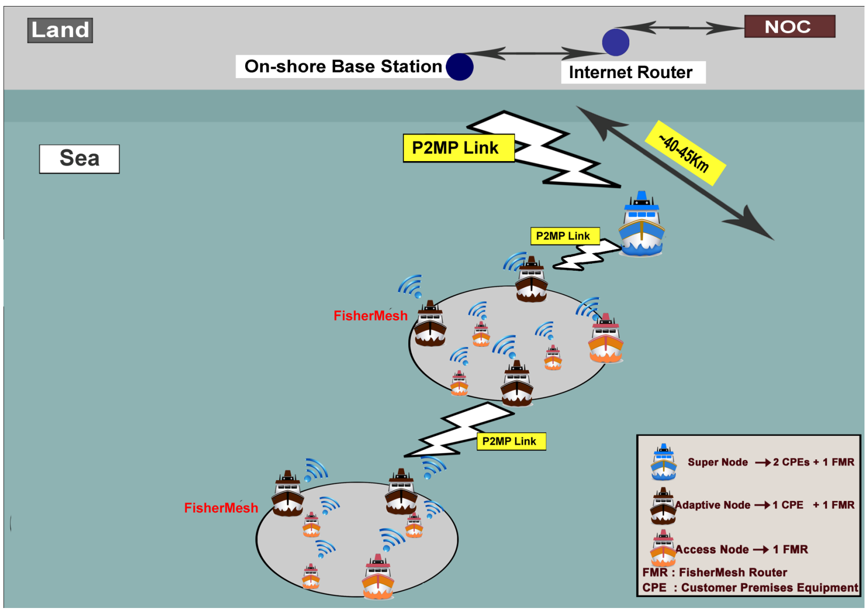


Figure 3.4: Hirearchical sub cluster with back-haul link

two back-hauls links to the nearest SN. In such a scenario how can we ensure a stable communication within the physical cluster such that the Internet access and easily be provided to each of the boats within the cluster from the base station.

Assuming a 2 back-haul link scenario, the controller has to be activated in the ANs due to it's higher capability compared to boats with only routers. So the controller will be activated in the ANs having backhaul links to the Super Node. The Software Defined MICRONet Processing module at each of the router in the boat will decide on the controller election mechanism. It will assign the master-slave role between the two ANs existing in the physical cluster at any point of time using the ETT metric calculation from the ETT module. This will enable the network to have a congestion free intelligent communication mechanism which will enable faster communication within the physical cluster and to outside world.

3.2.3 Scenario 3 - Hirearchical Sub Cluster with Back-haul link

The scenario in case 3 forms a hirearchical network with more than one sub clusters. An example hirearchical formation is shown in Figure 3.4.

```
start olsrdm;
while AN and back-haul links = 1 do
    adaptive-node-list();
    determine forward back-haul AN;
    determine backward back-haul AN;
    if backward back-haul link then
        master = AN;
        start controller in master;
    else
        pass;
    end
    invoke set-controller command in rest of the FMRs;
    start open-vswitch in rest of the FMRs;
end
```

Algorithm 3: Hirearchical sub cluster with back-haul link

The assumption is that, each of the sub-cluster consists of ANs and normal boats with routers. For establishing reliable and stable communication to the lowest level in the hirearchy, an algorithm has been proposed. Each of the physical clusters consists of a AN with a forward back-haul link and a AN with a backward back-haul link. The AN with the backward back-haul link takes up the role of the controller in the cluster and the AN with the forward back-haul link acts as a normal router. The assigned controller will manage the control traffic in there respective cluster and the communication to the outside world. Each sub cluster will have one AN acting as a controller in this scenario.

3.3 Summary

The proposed Software Defined MICRONet achitecture provides intelligent communication among physical boat clusters in the sea. We have analyzed three major realated scenarios and how the this architecture will solve these problems above. This will solve the technology challenges faced by the fishermen community in India today, specifically by providing software defined intelligent and adaptable communication and connectivity while they are out at sea.

CHAPTER 4

SOFTWARE DEFINED MICRONET IMPLEMENTATION

For implementing and testing the proposed Software Defined MICRONet, we build a testbed with 7 routers. The routers are running on a light weight linux operating system called Voyage linux 9.2 [10]. Open vSwitch 2.3.1 [11] kernel space implementation is installed on IIST MeshNet [12] testbed. Wireshark 1.12.6 [13] is used for analysing the OpenFlow packets. OpenFlow support is available with many software switches. However, only a few switches support OpenFlow 1.3 and above. OpenFlow 1.3 is the most stable version with maximum support. We choose Open vSwitch due to its stability and online support. Most of the linux distributions provide Open vSwitch by default. We use ALIX.3d3 [14] based computer for building routers. Voyage Linux 0.9.2 [10] is installed on top of the routers. ALIX.3d3 board comes with two mini PCI slots for wireless connectivity, flash drive for the meeting memory requirements. Router use OLSR [15] as the ad-hoc routing protocol for Layer-3 connectivity.

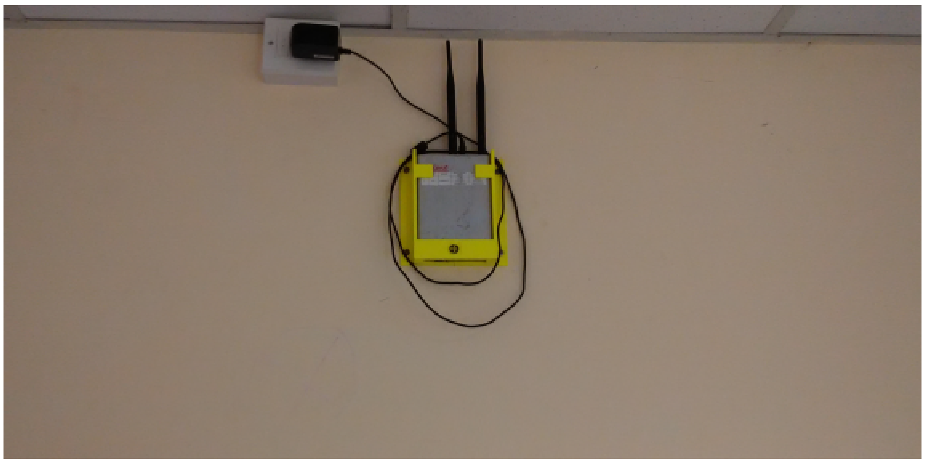


Figure 4.1: A view of WMN router in IIST MeshNet

4.1 Installing Voyage 0.9.2 Linux

Voyage Linux is derived from Debian to run on a x86 embedded platforms such as PC Engines ALIX [14]. Start the ALIX board with a bootable Voyage Linux USB drive.

1. Installation Steps

   ```
   $ mkdir /root/tmp
   $ mount -o loop /lib/live/mount/medium/live/filesystem.squashfs /root/tmp
   $ mkdir /root/cf
   $ /usr/local/sbin/format-cf.sh /dev/sda
   $ /usr/local/sbin/voyage.update
   ```

   set **/root/tmp** as the distribution directory, **7 - ALIX** as target profile, **/dev/sda** as which device accesses the target disk, **/root/cf** as the mount point, **2-console interface** and give default option for rest of the steps. Exit and reboot after installation. Finally, Login as user **root** with root password **voyage**.

2. Connecting to Wireless Access Point

   ```
   $ remountrw                               #Mounting file system in r&w mode
   $ ifconfig wlan0 down                     #Turning off wlan0 device
   $ ifconfig wlan0 essid IIST-BTECH-4Y      #Applying network access point
   $ ifconfig wlan0 up                       #Turning on wlan0 device
   $ dhclient wlan0                          #Querying DHCP server for IP address
   #If wired LAN is available it can be used as an alternative.
   ```

3. Updating the OS

   ```
   $ date -s "mm/dd/yyyy hh:mm"              #Configuring the Date
   $ apt-get update                          #Updating OS repository
   $ apt-get upgrade                         #Upgrading OS
   ```

4. Installing Dependencies and Build Tools

   ```
   $ apt-get install build-essential bison flex netcat vim git
                                             #Installing dependencies
   ```

4.2 Installing OLSR Daemon

The Installation and configuration of OLSR [15] can be done using the following steps.

1. Download and extract OLSR package

```
$ wget http://www.olsr.org/releases/0.6/olsrd-0.6.5.4.tar.bz2
                                        #Downloading the Installation file
$ tar -jxf olsrd-0.6.5.4.tar.bz2        #Extracting the file
$ cd olsrd-0.6.5.4
```

2. Building and Installation

```
$ make                                  #Building
$ make install                          #Installing
$ make install install_libs
```

3. Configuring OLSR Daemon Before the execution, the configuration file (/etc/olsrd.conf) needs to be modified. The changes that need to be done are provided in Appendix B.

4. Running the Application

```
$ ifconfig wlan0 down                   #Down wlan0
$ iwconfig wlan0 mode-adhoc essid iist_mesh freq 2.427G
                                        #Connect to iist_mesh
$ ifconfig wlan0 10.10.xx.xx netmask 255.255.255.0 up
                                        #Configure IP and Netmask
$ olsrd -d 1
```

4.3 Open vSwitch Installation

Open vSwitch [11] does not come by default in voyage linux. Therefore, it has to be installed manually by the user. For kernel space implementation of Open vSwitch, the kernel modules needs to be installed prior to the Open vSwitch installation.

1. Installing Dependencies

```
$ apt-get install python-all libc-dev perl \
  libtool kernel-package libssl-dev
                                        #Installing Dependencies
```

2. Downloading and Extracting Open vSwitch 2.3.1

```
$ wget http://openvswitch.org/releases/openvswitch-2.3.1.tar.gz
                                        #Downloading
$ tar -zxf openvswitch-2.3.1.tar.gz     #Extracting
$ cp -r openvswitch-2.3.1/* /usr/src/openvswitch-2.3.1
```

3. Building Kernel Modules

```
$ apt-get install openvswitch-datapath-source
$ apt-get install linux-headers-3.10.11-voyage
$ apt-get install linux-source-3.10.11-voyage
$ cd /usr/src
$ tar -jxf linux-source-3.10.11-voyage.tar.bz2
$ cd linux-source-3.10.11-voyage
$ make-kpkg --added-modules=openvswitch modules
$ apt-get install openvswitch-switch openvswitch-common
```

4. Installation

```
$ cd /root/openvswitch-2.3.1
$ ./configure --with-linux=/usr/src/linux-headers-3.10.11-voyage
$ make                                  #Building
$ make install                          #Installing
$ make modules_install
```

5. Running the Application

```
$ ps -A | grep ovs              #Kill the running ovsdb-tool & ovsdb-server
$ mkdir -p /usr/local/etc/openvswitch
$ rm /usr/local/etc/openvswitch/conf.db
$ cd openvswitch-2.3.1/
$ ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
  vswitchd/vswitch.ovsschema
                                        #Starting the ovsdb-tool
$ ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/ \
  db.sock --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
  --pidfile --detach
                                        #Starting the ovsdb-server
$ ovs-vsctl --no-wait init
$ ovs-vswitchd --pidfile --detach       #Running the switch in background
$ ovs-vsctl del-br br0
$ ovs-vsctl add-br br0 -- set Bridge br0 fail-mode=secure
                                        #Setting up a new Bridge
$ ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000003
$ ovs-vsctl add-port br0 wlan0
$ ifconfig wlan0 0
$ ifconfig br0 10.10.xx.xx netmask 255.255.255.0 up
$ route add default gw 10.10.1.1 br0
$ ovs-vsctl set-controller br0 tcp:10.10.xx.xx:6633
                                        #Configuring the Controller
```

4.4 Setting up the RYU Controller

RYU [16] is developed using Python which makes customization an easy process. Steps for building are given below. The RYU controller runs on a laptop machine running Ubuntu 14.04 LTS.

1. Installing Prerequisites

```
$ time sudo apt-get install python-eventlet python-routes \
  python-webob python-paramiko              #Installing prerequisites
$ cd ryu
```

2. Installing RYU

```
$ git clone http://osrg.github.io/ryu/      #Downloading
$ cd ryu
$ sudo python ./setup install               #Installing RYU
```

3. Running the Application

```
$ PYTHONPATH=. ./bin/ryu-manager ryu/app/<your-app-name>
```

4.5 Installing Wireshark 1.12.6

Wireshark is a free and open-source packet analyzer. The Wireshark 1.10.6 [13] that comes with standard linux distrubtuions does not support OpenFlow by default. Rather than installing OpenFlow dissectors, we use Wireshark 1.12.6 with OpenFlow 1.0 to 1.4 support.

1. Installing Dependencies

```
$ sudo apt-get install qt4-default
$ sudo apt-get build-dep wireshark
```

2. Downloading and Extracting Wireshark

```
$ wget https://1.as.dl.wireshark.org/src/all-versions \
  /wireshark-1.12.6.tar.bz2
                                   #Directly download if link is broken
$ tar -jxf wireshark-1.12.3.tar.bz2
```

3. Building and Installation

```
$ sudo ./configure
$ sudo make                        #Building
$ sudo make install                #Installing
$ sudo ldconfig
```

4. Running the Application

```
$ sudo wireshark
```

4.6 Summary

A total of 3 ALIX based routers were set up, 3 laptops running in Ubuntu 14.04 LTS and one laptop running in Ubuntu 14.04 to capture the network traffic. The testbed for Software defined MICRONet implementation was successfully set up.

CHAPTER 5

PERFORMANCE ANALYSIS AND RESULTS

Two sperate sets of experiments were carried out for performance analysis and results, (i) ETT Implementation, (ii) Software Defined MICRONet Implementation.

1. ETT Implementation
    - Two-node ETT Implementation
    - ETT based stationary controller hand-off
    - ETT based mobile controller hand-off
2. Software Defined MICRONet Scenario

5.1 ETT Implementation

ETT implementation is done for two-node, stationary controller hand-off and mobile controller hand-off scenarios. The test results have been obtained for (i) Two-node ETT calculation, (ii) Stationary ETT based Controller hand-off and (iii) Mobile ETT based Controller hand-off.

5.1.1 Two-node ETT Implementation

The test-bed for two-node ETT implementation consists of two routers running on Ubuntu 14.04 LTS systems. A load is generated in the link using 1024 bytes ICMP packets. Various parameters such as average packet size, bandwidth, ETX and ETT variation is studied and meaningful conclusions are made from those results.

Here a two-node single link is created and as observed from Figure 5.1 the link traffic is manually increased with 1024 bytes ICMP of packets for an interval of 300s.

The Bandwidth variation curve from figure 5.2 shows a slight decrease in the link bandwidth. This change is observed to very negligible from the existing bandwidth in the network.

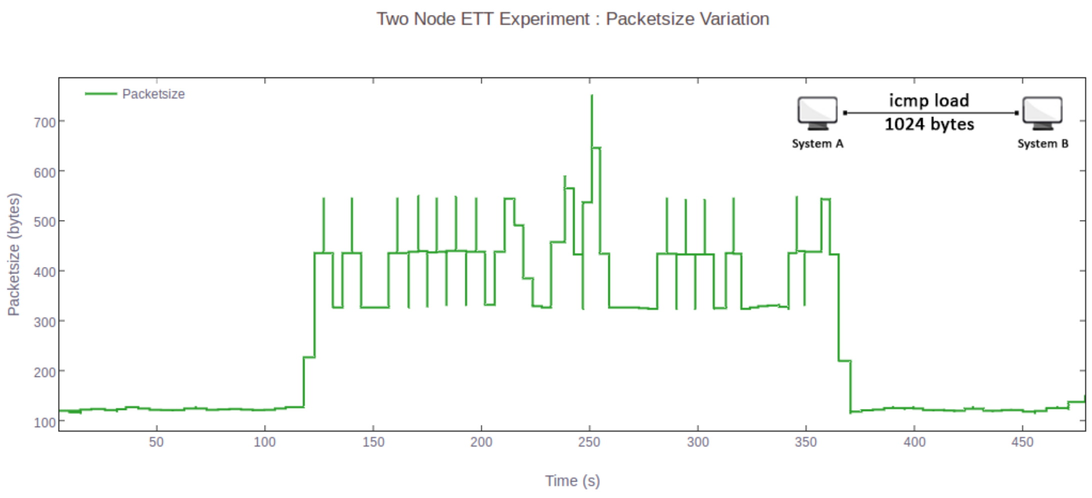


Figure 5.1: Two node experiment : Packet Size Variation

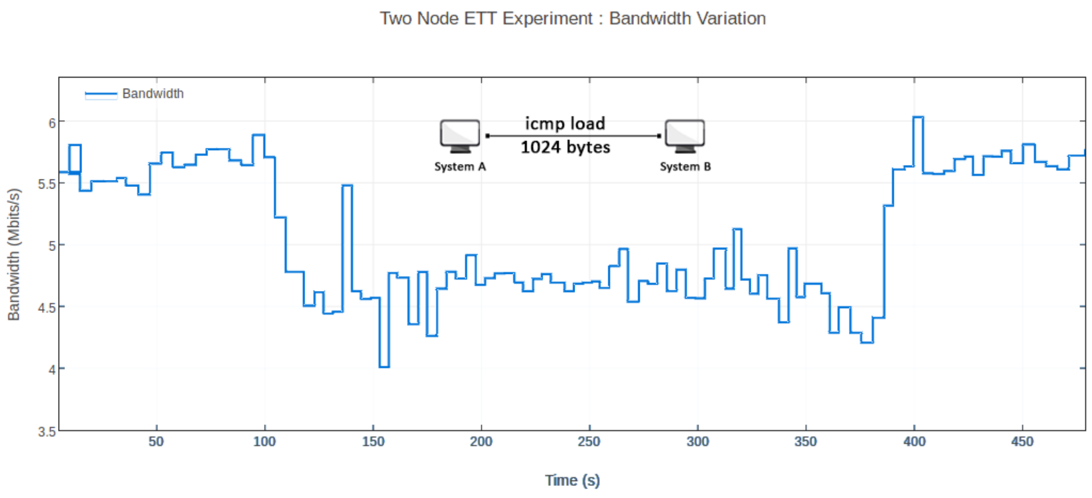


Figure 5.2: Two node experiment : Bandwidth Variation

The nodes are placed at one a hop distance, so the ETX value gives a 1 for majority of the experiment time as seen from Figure 5.3. We can also notice that there is no change in ETX value when the link traffic is increased. This is a negative aspect of ETX metric which does not take into account parameters such as throughput and bandwidth of the link. Any algorithm following ETX metric will solely be based on delivery ratios and will not take into account the link traffic and congestion. The ETT curve almost follows the packet size variation curve as the ETX value is 1 throughout and Bandwidth variation is minimal. Whatsoever, as observed from the Figure 5.3 we can conclude that ETT is a better metric in choosing links with ETX as a factor as well considering parameters such as bandwidth and link throughput so that ETT based metric algorithms can reduce network congestion and increase the overall throughput.

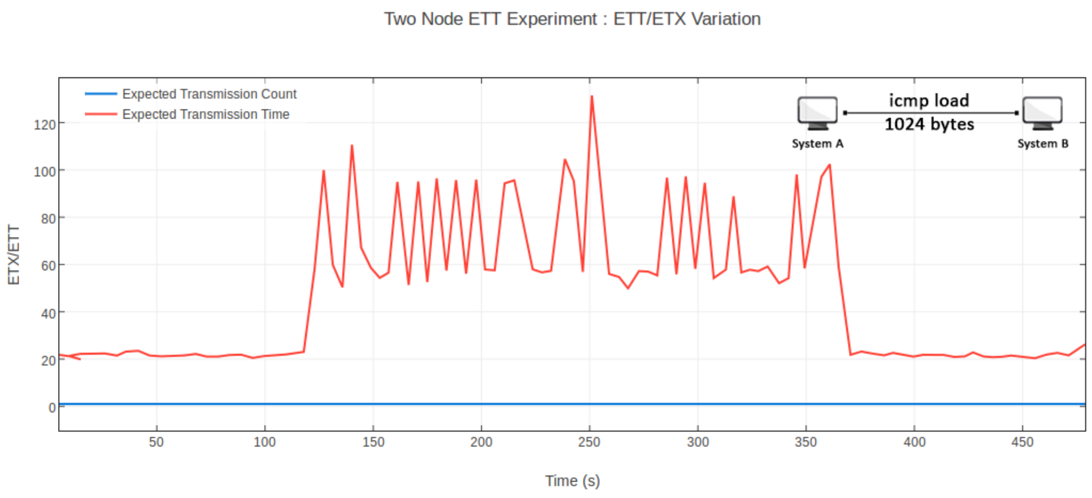


Figure 5.3: Two node experiment : ETX/ETT Variation

5.1.2 ETT-based Stationary Controller Hand-off

For the implementation of ETT based stationary controller hand-off, 2 controllers System 2 and System 4 and 1 switch Router 3 are taken for the testbed. The traffic between the switch Router 3 and controller System 2 is increased for a short span of time.

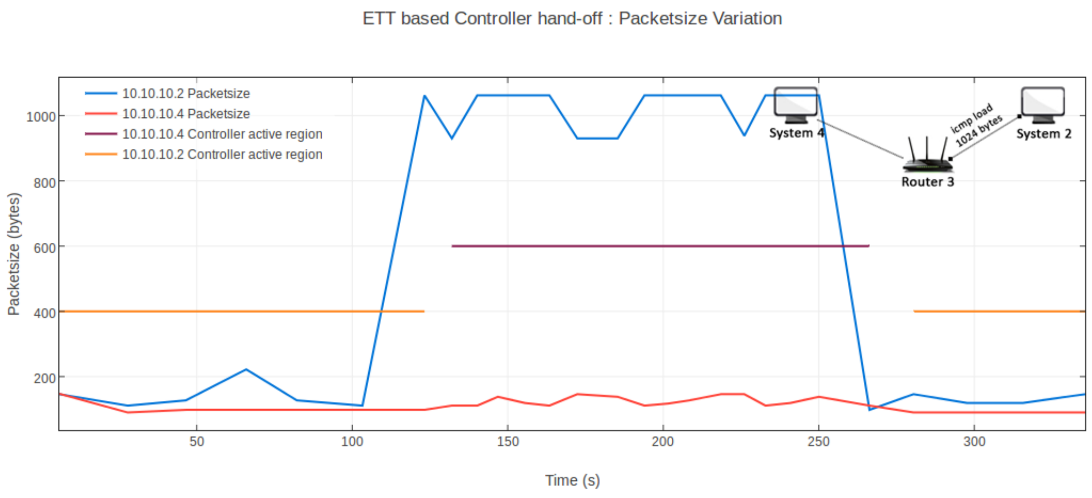


Figure 5.4: Stationary Controller Hand-off : Packet Size Variation

The generated traffic and bandwidth can be observed from the packetsize variation curve in Figure 5.4 and Figure 5.5. The ETX variation between the switch and the controller System 2 can be observed from figure 5.6. The controller System 2 is nearer to the switch Router 3 than the controller System 4. This can be observed from the higher ETX value in figure 5.6. Also, when the traffic between System 2 and Router 3 was increased there was no change in the corresponding ETX values as it is independent

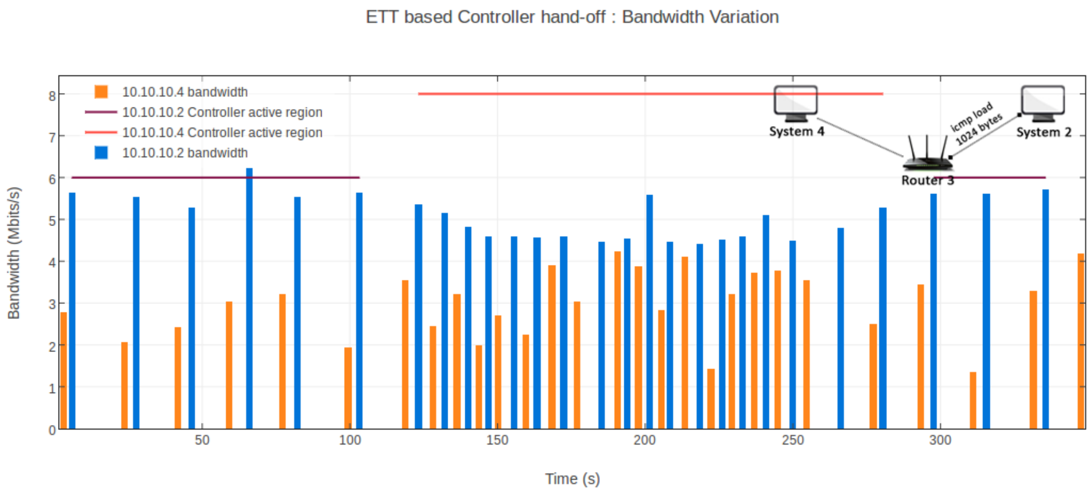


Figure 5.5: Stationary Controller Hand-off : Bandwidth Variation

of these factors.

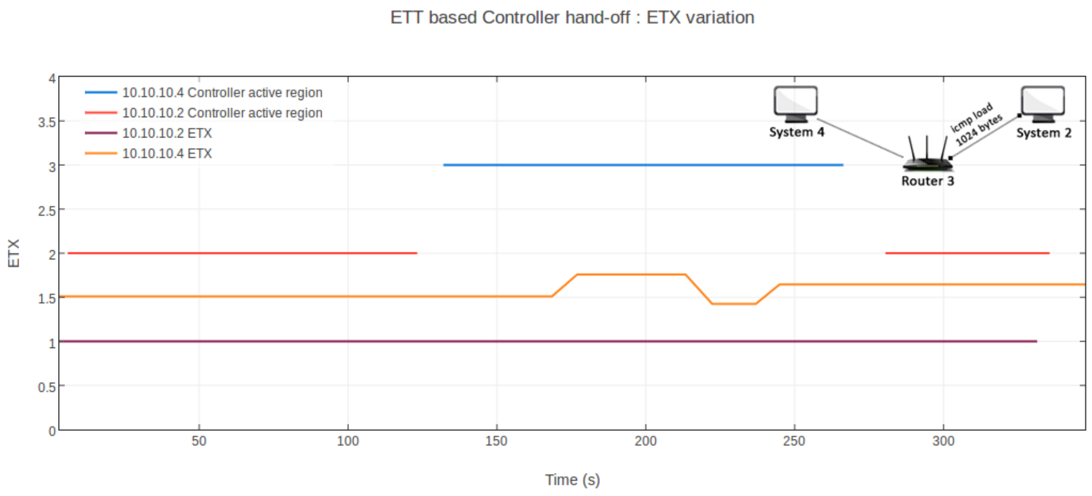


Figure 5.6: Stationary Controller Hand-off : ETX Variation

The ETT value for the link between System 2 and Router 3 increases during the interval at which the traffic was generated. This made the switch Router 3 to set System 4 as the master controller during this period due to it's lesser ETT value. After the traffic generation period was completed, it can be seen that the master controller role was shifted back to System 2 as it is nearer to the switch with a lower ETT value. This can be observed from Figure 5.7.

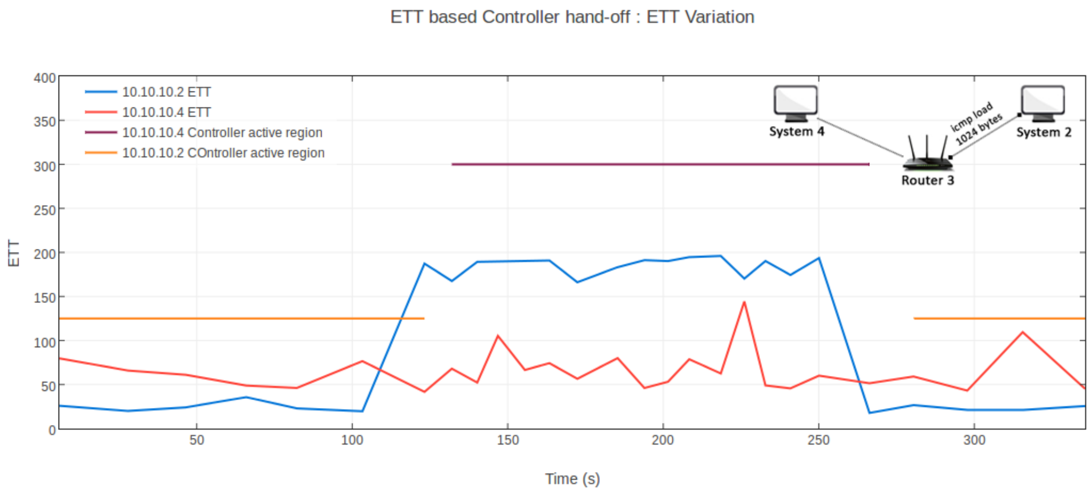


Figure 5.7: Stationary Controller Hand-off : ETT Variation

5.2 ETT-based Mobile Controller Hand-off

The implementation of ETT based mobile controller hand-off was performed in a testbed with 2 controllers System 2 and System 4 and 1 switch Router 3. The mobility experiment is analysed by moving Router 3 between the two controllers and generating a load of 1024 bytes ICMP packets while in motion. The testbed setup and the mobility experiment is shown in Figure 5.8.

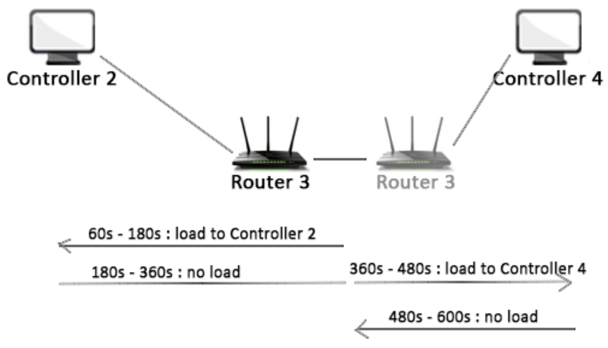


Figure 5.8: ETT-based Mobile Controller Hand-off test-bed

From Figure 5.9 we can observe that we have manually created an increase in the link traffic between router 3 and controller 3 in the first half of the experiment and to controller 4 in the second half of the experiment.

The ETT variation to the controllers from the switch router 3 has be shown in Figure 5.10. The switch shifts the master role from controller 2 to controller 4 as soon as the ETT to controller 2 has increased after 100s. This was due to the traffic generated during

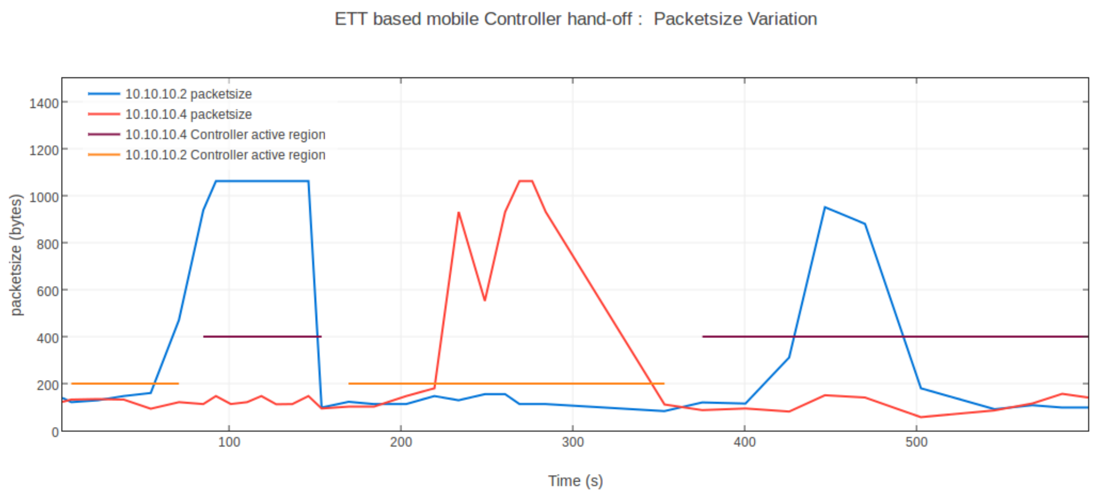


Figure 5.9: Mobile Controller Hand-off : Packetsize variation

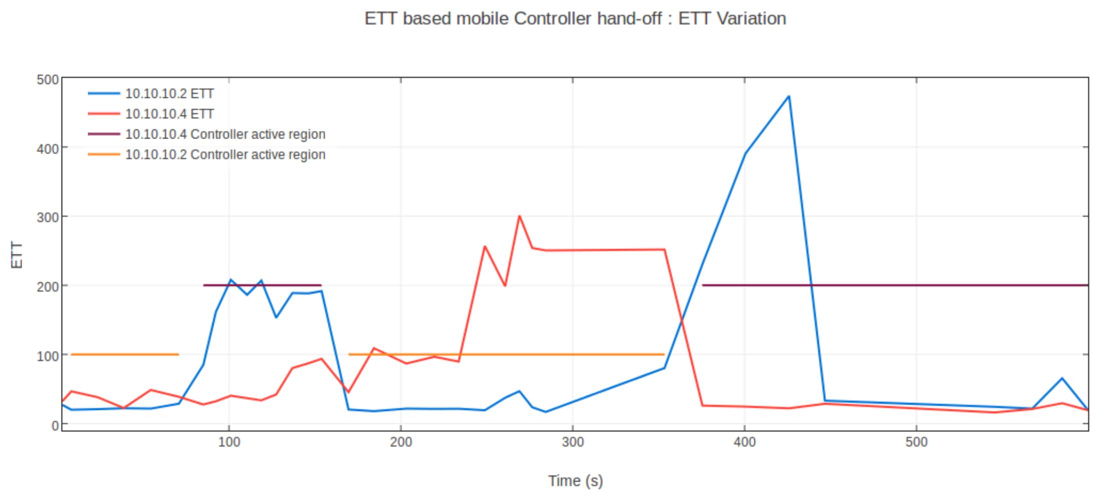


Figure 5.10: Mobile Controller Hand-off : ETT variation

that time interval between the router and the controller. Similary, we can observe the opposite when we switch the traffic to the other link and the switch router 3 sets the master back to controller 2.

5.3 Software Defined MICRONet Scenario

Software Defined MICRONet implementation is experimented on two testbed implementations in which the architecture has to adapt to the scenario and improve the performance of the network.

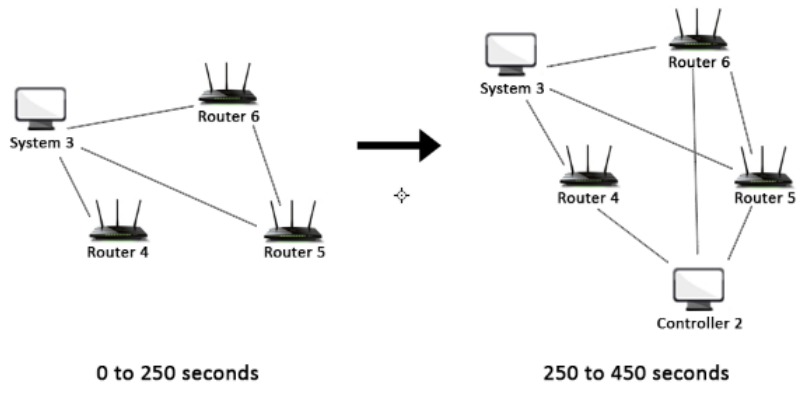


Figure 5.11: Software Defined MICRONet test-bed 1

In this case, we take a set of routers which work on OLSR and analyse the existing throughput in the network. Later after 250s into the experiment we introduce a Software defined micronet controller onto the network. The testbed implementation for the said experiment can be seen from Figure 5.11.

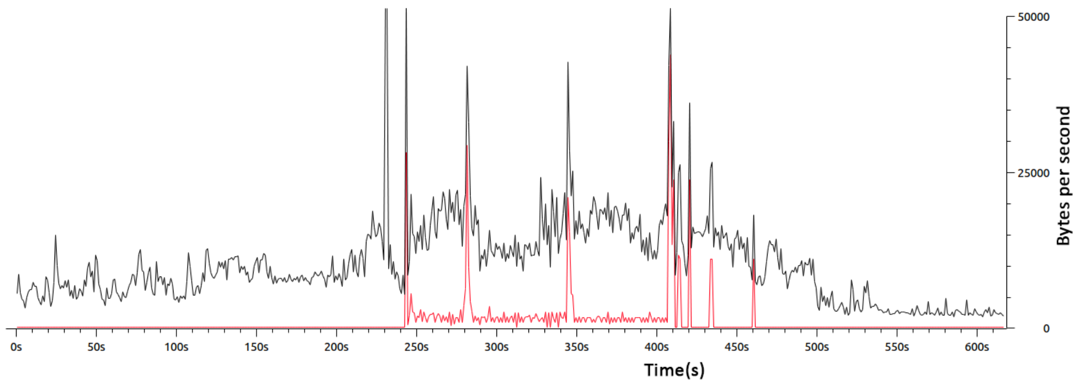


Figure 5.12: Software Defined MICRONet : Throughput Analysis

The Input/Output graph obtained from wireshark is shown in Figure 5.12, we observe that the throughput of the network has increased in the presence of a Software defined enviroment. With the help of the Software defined module as soon as the controller is down the network adapts to the new network without a controller and sets up a OLSR based network.

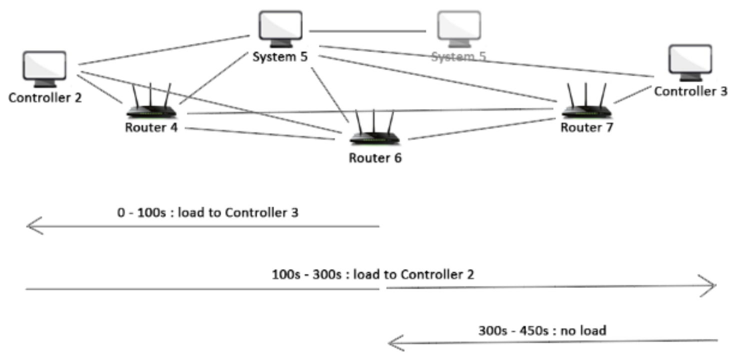


Figure 5.13: Software Defined MICRONet test-bed 2

In the next case, we develope a full prototype of the Software defined MICRONet enviromnent with 4 routers and 2 controllers in the physical cluster and observe the packetsize variation, ETX variation, ETT variation and throghput analysis using tcp-dump and wireshark. The testbed setup can be seen from Figure 5.13.

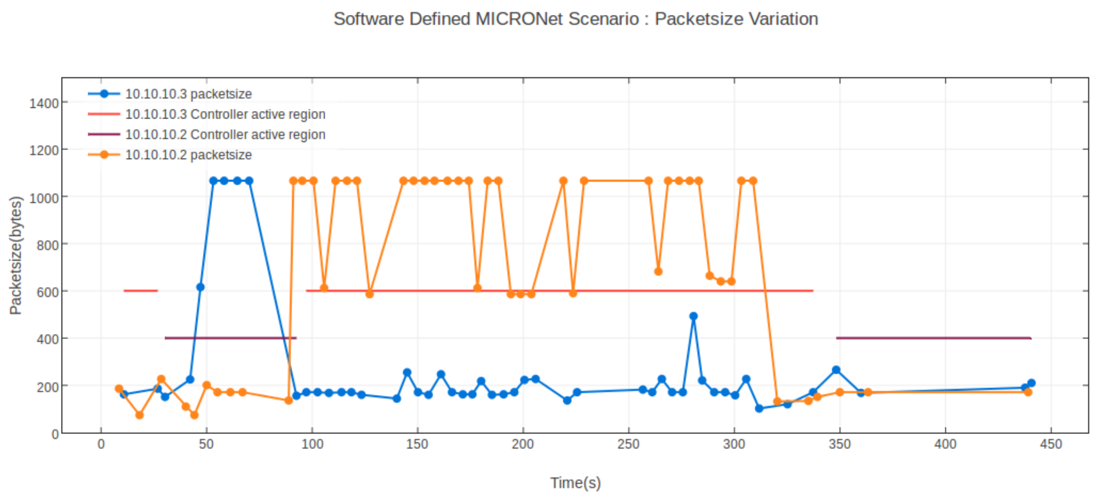


Figure 5.14: Software Defined MICRONet : Packetsize variation

From Figure 5.14, 5.15, 5.16 we can observe the packet size variation, ETX and ETT variation in the Software defined MICRONet physical cluster with the presence of two controller. Taking the observation from router 5, we observe that when the router is in mobility it adapts to the ETX value, link throughput, linkbandwith to generate the corresponding ETT values to the two controllers and based on the metric decide on which should be the master and slave. The controller hand-off using ETT can be seen clearly from the Figure 5.16 which adequately selects a controller with lesser ETT value

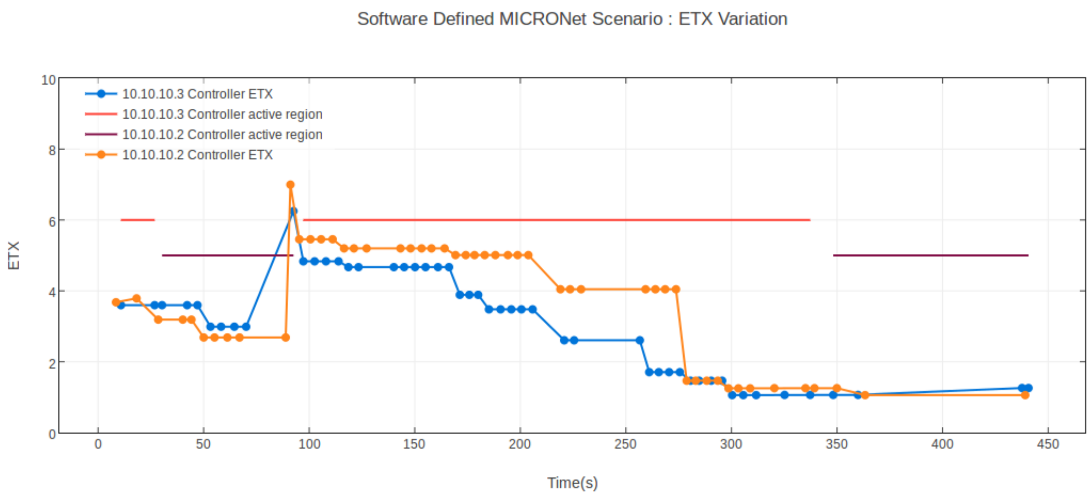


Figure 5.15: Software Defined MICRONet : ETX variation

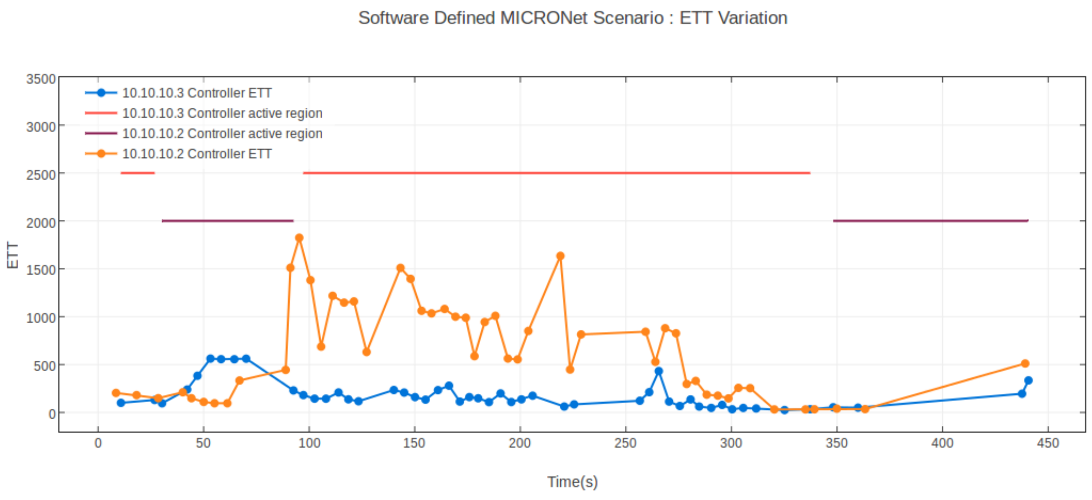


Figure 5.16: Software Defined MICRONet : ETT variation

as it's master.Each of the router in the network decides which controller should be it's master and slave.

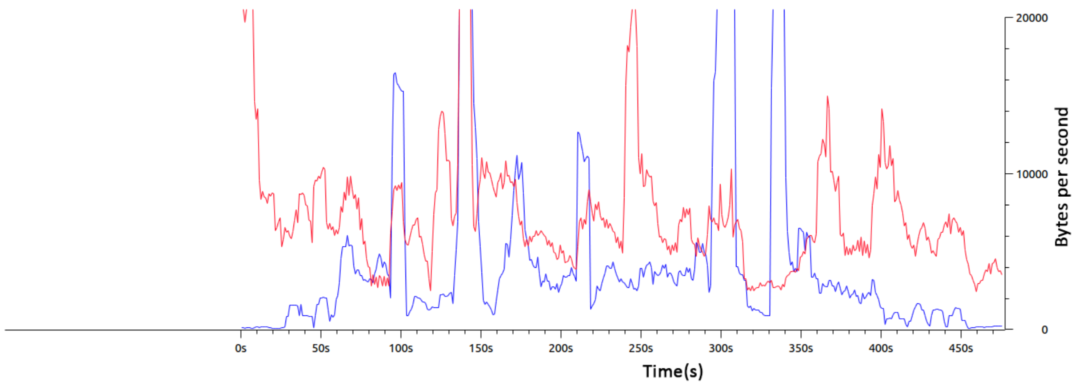


Figure 5.17: Software Defined MICRONet : Throughput Analysis

The Input/Output graph obtained from wireshark is shown in Figure 5.17, we observe the consistent OpenFlow throughput of the network which indicates the presence of controllers in the network and the increase in the ICMP througput which shows the traffic generation inorder for making the router 5 switch between master and slave selection between the two controllers 2 and 3.

5.4 Summary

The ETT implementation for for Software defined MICRONet was succesfully implemented in a test-bed and meaningful results were obtained. The results showed that ETT is a better metric for controller hand-off than ETX or hop-count. Software defined MICRONet therefore deploys ETT as the main controller election algorithm instead of the existing OLSR based ETX. The Software defined MICRONet testbed was setup and experiments were carried out in which the architecture adapts to itself in such an enironment with multiple controllers.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

SDN approach allows us to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed firmware. Incorporating this concept to offshore communication for Indian fishermen, each of the physical boat clusters can be controlled by the network user. Software Defined MICRONet achitecture provides intelligent communication among physical boat clusters in the sea. This will solve the technology challenges faced by the fishermen community in India today, specifically by providing software defined intelligent and adaptable communication and connectivity while they are out at sea. The applications of such an architecure includes examples such as an early warning system for collision avoidance of vessels at sea, location based hierarchical data classification for information dissemination in marine sector, data mining techniques for data dissemination and localization of missing fishing vessels in the deep sea, localization and tracking of multiple mobile boats, design of mobility model for fishing vessels in the arabian sea etc.

The ETT implementation for for Software defined MICRONet was succesfully implemented in a test-bed and meaningful results were obtained. The results showed that ETT is a better metric for controller hand-off than ETX or hop-count. The Software defined MICRONet testbed was setup and experiments were carried out in which the architecture adapts to itself in such an enironment with multiple controllers. A prototype physical cluster was created for this experimental purprose. Software defined MICRONet therefore deploys ETT as the main controller election algorithm instead of the existing OLSR based ETX.

6.2 Future Work

- Development of better controller election metric than ETT
- Analysis of more Software Defined MICRONet scenarios helps to improve the strategy further
- Incorporation of GPS/IRNSS into the Software defined MICRONet for futher intelligent architecture

REFERENCES

[1] openflow-spec v1.3.0.pdf, “Openflow-manual,” http://openvswitch.org/.

[2] sdn-vs traditional, “https://globalconfig.net/,” accessed: 08-07-2015.

[3] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, “Wireless mesh software defined networks (wmsdn),” in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*. IEEE, 2013, pp. 89–95.

[4] N. N. Jennath Hassan, Sethuraman N Rao, “A resilient self-organizing offshore communication network for fishermen,” 2015.

[5] Openflow, “http://archive.openflow.org/,” accessed: 03-06-2015.

[6] P. Dely, A. Kassler, and N. Bayer, “Openflow for wireless mesh networks,” in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*. IEEE, 2011, pp. 1–6.

[7] J. Chung, G. Gonzalez, I. Armuelles, T. Robles, R. Alcarria, and A. Morales, “Experiences and challenges in deploying openflow over real wireless mesh networks,” *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 11, no. 3, pp. 955–961, 2013.

[8] S. Salsano, G. Siracusano, A. Detti, C. Pisa, P. L. Ventre, and N. Blefari-Melazzi, “Controller selection in a wireless mesh sdn under network partitioning and merging scenarios,” *arXiv preprint arXiv:1406.2470*, 2014.

[9] P. M. Esposito, M. E. M. Campista, I. M. Moraes, L. H. M. Costa, O. C. Duarte, and M. G. Rubinstein, “Implementing the expected transmission time metric for olsr wireless mesh networks,” in *Wireless Days, 2008. WD’08. 1st IFIP*. IEEE, 2008, pp. 1–5.

[10] Voyage-linux, “http://www.linux.voyage.hk,” accessed: 03-06-2015.

[11] Openvswitch, “http://osrg.github.io/ryu/http://openvswitch.org/,” accessed: 17-06-2015.

[12] iist mesh met, “https://www.facebook.com/iistmeshnet,” accessed: 08-06-2015.

[13] Wireshark, “https://www.wireshark.org/,” accessed: 04-06-2015.

[14] Alix3d3, “http://www.pcengines.ch/alix3d3.htm,” accessed: 01-06-2015.

[15] Olsr, “http://www.olsr.org,” accessed: 13-06-2015.

[16] Ryucontroller, “http://osrg.github.io/ryu/,” accessed: 23-06-2015.

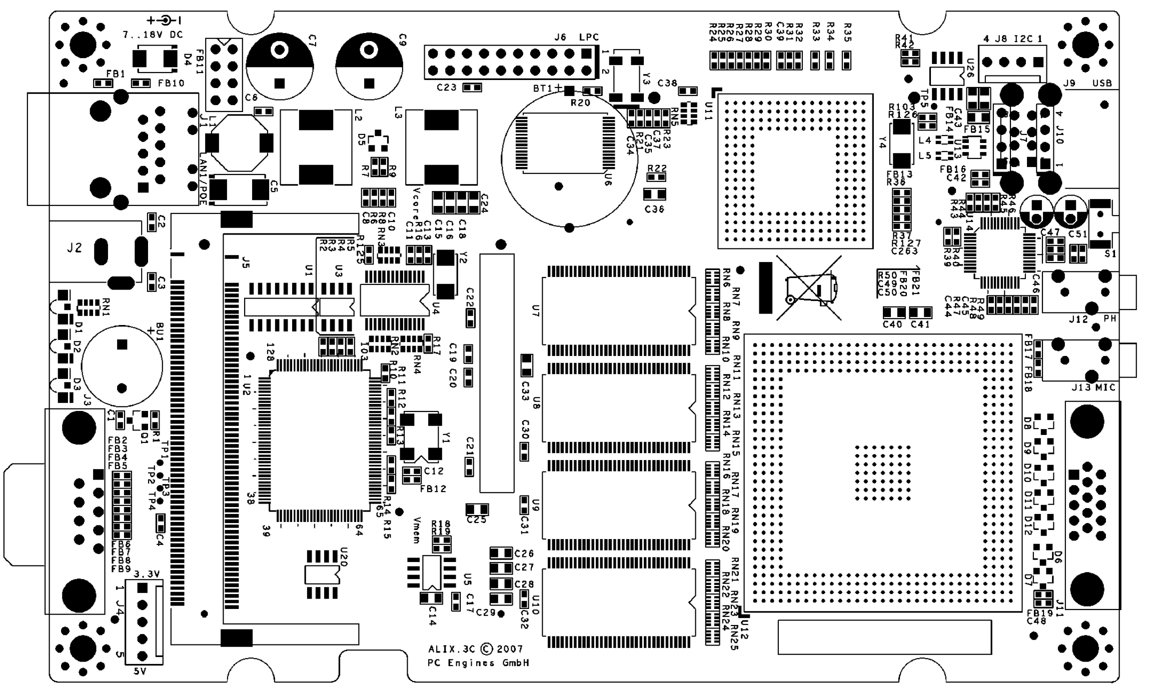
APPENDIX A

ALIX3D3 Datasheet

ALIX.3d3 features

PC Engines ALIX boards are small form factor system boards optimized for wireless routing and network security applications.

- AMD Geode LX CPU, 500 MHz (LX800) 5x86 CPU
- 256 KB cache (64K data + 64K instruction + 128K L2)
- 1 Ethernet channel (Via VT6105M, 10 / 100 Mbit/s)
- 2 miniPCI sockets for 802.11 wireless cards and other expansion
- 256 MB DDR SDRAM, 64 bit wide for high memory bandwidth
- 512 KB flash with Award BIOS
- CompactFlash socket for user's operating system and application
- 7 to 20V (absolute maximum, recommend 18V) DC supply through DC jack or passive power over Ethernet. Suggest a 18V / 15W supply. Center pin = positive, sleeve = ground, 2.1 mm diameter.
- 1 serial port (DB9 male)
- 2 USB 2.0 ports
- Header for LPC bus (use for flash recovery or I/O expansion)
- Temperature range 0 to 50°C
- Dimensions 100 x 160 mm



APPENDIX B

OLSR Configuration File: /etc/olsr.conf

- **line23** “#DebugLevel 1” -> “DebugLevel 1”
- **line232** “#PlParam “accept” “0.0.0.0”” -> “PlParam “accept” “0.0.0.0””
- **line253** “#Interface “<OLSRd-Interface1>” “<OLSRd-Interface2>”” -> “Interface “wlan0””