# Design Concept for a Failover Mechanism in Distributed SDN Controllers

A project presented for the degree of Master of Science

Author: Nathan Kong

Advisor: Dr. Melody Moh

Committee Members: Dr. Teng Moh, and Dr. Robert Chun

Computer Science
San Jose State University
USA
April 2017

# *Abstract*

Software defined networking allows the separation of the control plane and data plane in networking. It provides scalability, programmability, and centralized control. It will use these traits to reach ubiquitous connectivity. Like all concepts software defined networking does not offer these advantages without a cost. By utilizing a centralized controller, a single point of failure is created. To address this issue, this paper proposes a distributed controller failover. This failover will provide a mechanism for recovery when controllers are not located in the same location. This failover mechanism is based on number of hops from orphan nodes to the controller in addition to the link connection. This mechanism was simulated in Long Term Evolution telecommunications architecture.

## Table of Contents

# Acronyms

| | |
|---|---|
| E-UTRA | Evolved Universal Mobile Telecommunication System Terrestrial Radio Access |
| EPC | Evolved Packet Core |
| HLF | Hop and Link Failover |
| LTE | Long Term Evolution |
| ms | milliseconds |
| Mbps | megabits per second |
| SDN | Software Defined Networking |

## 1 Introduction

Software Defined Networking (SDN) is an enabling technology to overcome growing data traffic in dense heterogeneous networks. SDN provides an agile and flexible network. It decouples the control plane from the data plane to create a programmable and centralized software. It allows for scalability and management of a network.

SDN can assume two different architectures. The first assumes a centralized controller over a network. Controllers can be responsible for nodes or other controllers. The second architecture assumes a distributed system of controllers. In either case SDN still maintains the same weakness, a single point of failure. At any time controller failure can transpire. Failure can occur from various events including power outages or lack of connectivity. Once failure transpires, nodes become orphans. They will be unable to transfer data to new connections without a controller. This paper will address the detection of failure and recovery in the distributed architecture.

Failure analysis is an important concept in understanding how and why failure is occurring. Analysis helps to determine the methods for failover. By understanding failure, one can prevent of false positives and determine if failover is truly necessary. Failure analysis is broken into failure types and failure detection.

In order to recover from controller failure, a mechanism must be created to enable orphan nodes to be connected to other controllers. Failover must react instantaneously to maintain network connectivity in a quick efficient manner. Recovery strategies have been researched and are broken into redundant controllers and non-redundant controllers.

This paper proposes a Hop and Link Failover (HLF) mechanism for recovery due to a controller failure. This mechanism was simulated in a Long Term Evolution (LTE) environment to prove efficient failover. The LTE environment was chosen as it is predicted to use SDN in 5G technology. 5G LTE is the next generation of telecommunications that will enable more devices on the network. This is necessary because there is expected to be 50 billion Internet-of-Things devices by 2020 [1].

The remainder of this document is organized as follows: Section 2 presents background concepts of the SDN paradigm, the SDN architecture in a distributed manner, analysis of failure types and detection of controllers, and various strategies for failover. Section 3 presents related works in failover and recovery. Section 4 introduces the HLF mechanism. Section 5 discusses the setup and performance of HLF simulation. Finally the paper is concluded in Section 6.

# 2 Background

The following sections describe the concepts of SDN and how it incorporates into a distributed architecture. This description is followed by the analysis of failure types and detection of controllers. Lastly various strategies for failover are discussed.

## 2.1 Software Defined Networking

Software Defined Networking is a new technology for managing networks. Originally it was thought of as centralized control with an OpenFlow protocol for communications between the control plane and the data plane. Its definition has now changed and can be defined by three features. It is abstracted from the hardware thus having no dependency on physical hardware restrictions. It maintains centralized control to manage forwarding tables and policy delegation. It is programmable to dynamically configure networks based on policy and demand [2].

The SDN architecture consists of three planes and two interfaces for communication between the planes. The components include the application plane, the control plane, and the data plane. The two interfaces include the northbound and southbound interfaces [3]. The SDN architecture is shown in Figure 1.

The data layer is the bottom most layer of the SDN architecture. It consists of network devices that forwards data based on control tables. Although the data layer contains control tables it does not create the table. The table comes from the controller.

The southbound interface is used for communication between the data layer and the control layer. This interface is used to transmit the type and amount of traffic from the data layer to the control layer. It is also used to transmit the flow tables from the controller to the data layer.

The controller layer is in charge of enforcing policies from the business applications. It creates forwarding tables and prioritizes quality of service based on business policies. The control layer sends forwarding tables to the data layer.

The northbound interface is used for communication between the controller layer and the application layer. It provides abstract network views and enables network requirements. This interface can be implemented through an application protocol interface.

The application layer is the top most layer of the SDN architecture. It consists of business applications. These applications programmatically communicate business policies to the network. They determine quality of service and network behavior through service level agreements.

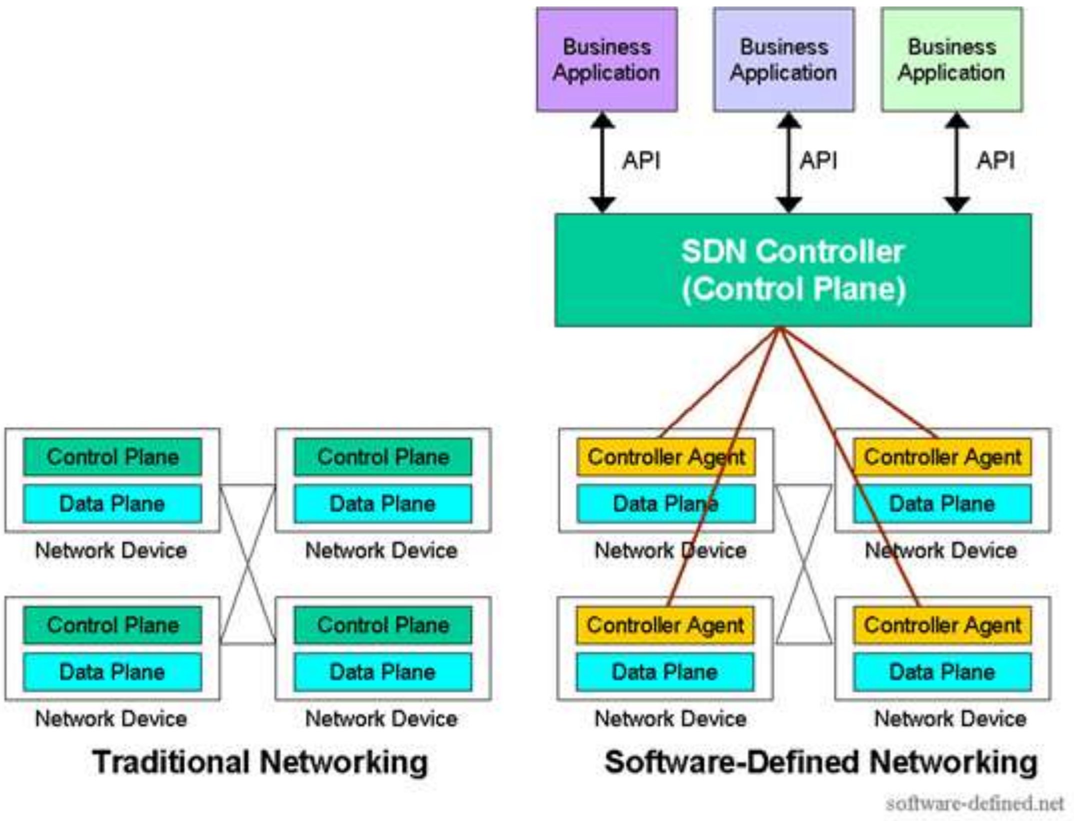

Figure 1: Traditional Network vs. SDN [4]

SDN can improve network communication. It brings new features to networking such as performance, virtualization, orchestration, programmability, visibility, and dynamic scaling. SDN improves performance by managing bandwidth and adjusting the traffic loads. It can be optimized for network capacity and adjust for node failure. It has the ability to orchestrate traffic for a plethora of devices with a few commands. SDN creates visibility in software to monitor resources. SDN is virtualized. It uses network resources regardless of physical location and can dynamically scale according to its need.

## 2.2 Distributed SDN Architecture

SDN can be incorporated in a distributed fashion. This architecture consists of multiple SDN controllers. Each SDN controller is responsible for one sub-network of nodes. These nodes are network devices such as switches or eNodeBs that deliver packets from devices to other devices. The controllers can communicate with one another in a distributed fashion. An example architecture is shown in figure 2.

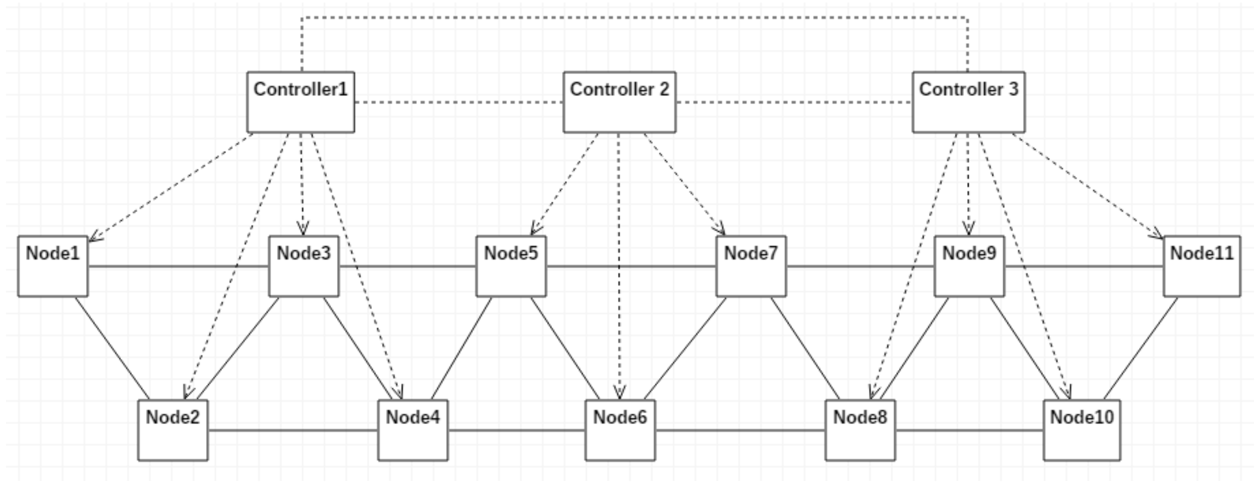

Figure 2: Distributed SDN Controllers

The advantage of a distributed SDN architecture is that it enables reliability in the network by eliminating a single point of failure. It allows other controllers to adopt orphan nodes during controller failure. The distributed architecture allows each network to be dynamically configured.

## 2.3 Long Term Evolution Architecture

Long Term Evolution is a communications standard for telecommunications. It is located in the access portion of the Evolved Packet System to transmit data. LTE uses an architecture where the user equipment communicates with the Evolved Universal Mobile Telecommunication System Terrestrial Radio Access (E-UTRA) [5]. The E-UTRA then transmits data to the Evolved Packet Core (EPC) in order to communicate with the Internet. Figure 3 shows the architecture.

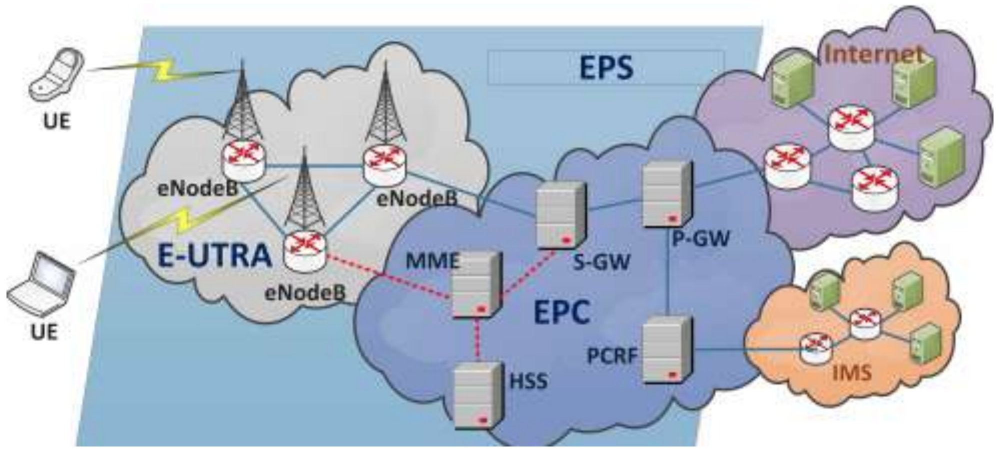

Figure 3: Telecommunications Architecture [6]

User equipment includes items such as computers, mobile phone, tablets, and Internet-of-Things devices. It consists of devices that start the initial transmittal of data. User equipment transmits data through radio waves to towers. These towers are known as eNodeBs. The ENodeBs communicate with one another as well as the EPC.

The EPC is the core network for LTE communications. It is a framework that provides data on an IP network. The EPC consists of five major components: mobility management entity, serving gateway, packet data network gateway, the home subscriber server, and the policy control and charging rules function. These components hold user information, policy, and charging rules. It also allows user equipment to access the Internet.

## 2.4 SDN in LTE

SDN is intended to be included for 5G LTE. According to IMT-2020 [7], testing for SDN in 5G has begun during 2016. The telecommunications industry has already begun incorporating and testing 5G LTE [8].

SDN can be integrated into the LTE architecture by replacing the serving gateway and packet data network gateway with a controller. An example is shown in figure 4. Through this architecture two types of data planes are created. The first data plane is incorporated in switches. These switches connect the EPC components. The second data plane is incorporated in the eNodeBs. The controller would handle the flow from user equipment to other user equipment or to the Internet. It would accomplish this feat by orchestrating the data between eNodeBs and switches.

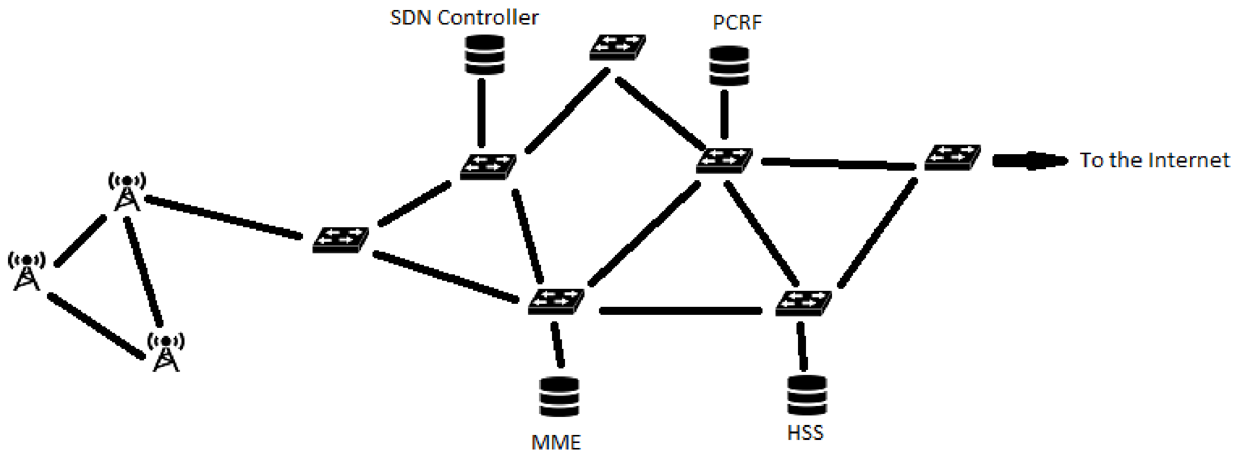

Figure 4: SDN in LTE Architecture

## 2.5 Failure Types

Controller failure can occur from software, network, or hardware failures [9]. During controller failure each node will maintain its current status but not be able to adjust flow control, management of services, or take on new connections until a new controller is adopted.

Software failure can transpire through maintenance issues, bugs, or attacks. Maintenance issues typically occur while software is either being updated or restarted. Software will freeze and be unable to complete tasks. Software bugs can be introduced to a system while testing or going live. Since controllers manage nodes, they are a target for hackers. Malicious software can be installed to take down or control the SDN controllers.

Network failure transpires through lack of connectivity. This issue occurs when the controller and the node cannot communicate properly with one another. Connectivity can be a physical issue or software issue.

Hardware failures happen from lack of maintenance or power. Hardware devices need to be monitored for dust and insects. Insects can cause shorting of integrated circuits. High levels of dust will insulate hardware components and can overheat components. Power failure happens when there are issues powering the hardware.

## 2.6 Failure Detection

Failure detection is important in determining controller failure. Often controllers do not fail. Instead they provide false positives that a failure has occurred. Often failure detection has a high mistake rate [10]. Detection of failure can be discovered in a plethora of ways including heartbeat messages and failure messages.

Failure messages are a graceful way for controllers to fail. If a controller is shutdown due to maintenance, it can send failure messages to nearby neighbors informing them of its status.

Heartbeat messages are a way for neighboring controllers to determine if a controller is alive. Each controller periodically sends messages to nearby neighbors announcing their existence. Reverse heartbeat messages are another way of failure detection. With reverse heartbeat messages, nodes message controllers periodically to determine if the controller is alive. In either case if messages are not received within a certain time frame controllers are assumed to have failed.

With heartbeat messages the issue of false positives occurs. False positives emerge when heartbeat messages happen but are missed by neighboring controllers. Neighboring controllers then try to adopt nodes still associated with the connected controller.

# 3 Related Work

Failover for distributed controllers is an important issue that must be addressed in order for SDN to provide ubiquitous connectivity. Distributed SDN controller failure can occur for a plethora of industries. These industries include: Wide Area Networks, telecommunications, data centers, and big data. Each area has its own constraints thus will require its own set of failover mechanisms.

Failover for SDN has been researched topic. Mathis Obadia et al. [9] explored a greedy mechanism for failover. They implemented failover for distributed SDN controllers. Their simulation involved Mininet with Floodlight controllers. They developed a proactive approach to the greedy mechanism to enhance the speed of recovery.

Li et al [11] proposed a proactive recovery mechanism based on controller load and switch-controller delay. In their mechanism they use scores to reassign switch paths during failover.

A fast and load-aware controller failover mechanism was analyzed by Fang, Wang, and Wang [12]. They proposed an enhanced algorithm to Li et al [11] where controllers pre-compute a recovery plan for failure of other controllers. Their algorithm derives a recovery plan based on switch-controller delay from their mapping in addition to the standard deviation of the controller load.

Failover with redundant controllers has been researched by Pashkov et al [13]. They developed a failover mechanism that incorporates various standby strategies for the redundant controllers. These methods include hot, warm, and cold standby. They proved their hot and warm mechanisms to be effective for failover with relatively low overhead.

Borokhovich and S. Schmid [14] explored failover on OpenFlow switches. They analyzed local fast failover due to link failures. They presented a randomized deterministic algorithm for forwarding sets to show that local fast failover is efficient under randomized link failures.

A method to solve control plane link failures for a single controller was proposed by Beheshti and Y. Zhang [15]. They proposed an algorithm to determine the optimal placement of the controller to maximize resiliency. This allows the network to react to switch and link failures in the data plane by re-routing control traffic.

## 3.1 Redundant Controller Strategies

Recovery mechanisms can be incorporated through redundant controllers. The redundant controller architecture incorporates two or more controllers per controller domain. The first controller will actively control the domain while the other controller(s) remain in a standby mode. In case of failure, the standby controller will replace the active controller. Standby controllers duplicate active controllers in all functionality. Redundant controller strategies provide a fast failover technique. Redundant controllers can be used when extra hardware and software is available and recovery time is crucial. Big data for real-time analytics is a prime candidate for redundant controllers. Three strategies for controller redundancy include cold standby, warm standby, and hot standby [13].

### 3.1.1 Cold Standby

In the cold standby strategy, the standby controller is stateless. During failure of the active controller, the standby controller will become active and take over the failed controller's domain without any knowledge of the network configuration. It will initialize its default configuration of services and applications before starting its topology discovery. In this strategy the standby controller remains unloaded until active. This strategy is ideal for stateless services and applications.

### 3.1.2 Hot Standby

In the hot standby strategy, the active controller is duplicated by the standby controller for all changes that occurs. This duplication runs a full state synchronization. When the active controller fails, the standby controller can replace its position in the current state. This minimizes the recovery time, yet it will require implementing software and hardware redundancy in addition to high communication overhead.

### 3.1.3 Warm Standby

In the warm standby strategy, the standby controller periodically runs a full state synchronization of the active controller to retain its state. During failure of the active controller, the standby controller will operate based on the last state synchronization. With this strategy some states will be lost and some services will be interrupted. This cost will reduce the overhead in communication. It will also eliminate full initialization.

## 3.2 Non-Redundant Controller Strategies

In most cases redundant controllers are unnecessary. They create overhead by requiring software and hardware redundancies as well as additional communication. In non-redundant controller strategies, there is only one controller per controller domain. During failure, controllers can increase their domain and adopt orphan nodes in their network. This reduces the overhead of the network. Non-redundant controller strategies can be effective in data centers or telecommunications. Two current strategies for non-redundant controllers include least loaded failover and greedy failover. A new failover proposed by this paper includes HLF.

### 3.2.1 Greedy Failover

In the greedy failover, controllers try to adopt orphan nodes at the edge of their domain. Greedy failover can be implemented reactively. First, detection of controller failure is conducted by the nodes. They transmit reverse heartbeat messages to their controllers to determine the status of their controller. If the controller fails to respond to a number of messages, failure is detected and the nodes become orphan nodes.

Orphan nodes must then search for a new controller. They send out discovery messages to their connected nodes. The nodes in close proximity forward the discovery message to their controllers. If the nodes in close proximity do not have a controller they will drop the message. The first controller to connect with the orphan node will adopt it. Flow tables contained in the orphan node will remain unchanged until the controller has ample time to discover the topology of its new domain. Once a controller adopts one orphan node it must update its database with information from the node. This must be completed before discovering more orphan nodes. The algorithm for the reactive greedy method is shown in Algorithm1.

Algorithm1: Reactive Greedy

```
1. listenForMessages
2. if receiveMessage then
3.  if hasController = false then
4.   linkCapacity = getLink
5.   hops = getHops
6.   adoptController
7.   hasController = true
8.  end if
9. end if
```

Greedy failover can also be implemented in a proactive approach. In this approach, controllers take turns exchanging their network information with the nodes. This occurs when the nodes are powered up. The nodes use the greedy algorithm during their initialization setup to pre-determine which controllers to connect to in case of failure. The nodes then insert this information into their database. The algorithm for this has been developed by Mathis Obadia et al [9] and completed using the algorithm2.

Algorithm2: Proactive Greedy

```
1.  while setOfNodes ≠ 0 do
2.   for neighborDomain ∈ setOfDomains do
3.     if ∃ node ∈ setOfNodes then
4.       Add node to setOfNeighborDomainBackupNodes
5.       Remove node from setOfNodes
6.     end if
7.   end for
8.   if ∀ neighborDomain ∈ setOfDomains∄node ∈ setOfNodes then
9.   Add first node ∈ setOfNodes to first setOfNeighborDomainBackupNodes
10.  Remove node from setOfNodes
11.  end if
12. end while
```

Detection of controller failure is established the same way as the reactive method above. Once failure is determined, orphan nodes contact controllers in their database. The controllers then establish connections to all orphan nodes simultaneously. Then the controllers updated its database with information from the newly adopted nodes simultaneously.

### 3.2.2 Least Loaded Failover

Least loaded failover [12] is a simple recovery mechanism that is efficient in data centers where controllers are often physically located next to each other. During controller failover, the least loaded controllers increase their controller domain by adopting the orphan nodes.

Detection of controller failure is conducted by heartbeat messages. Controllers interact with each other to determine their nodes. Afterwards they monitor each other through heartbeat messages. When messages fail between controllers, failure is detected. The controllers then determine the least loaded controller. This controller will then adopt all the orphan nodes simultaneously. Flow tables contained in the newly adopted nodes will remain unchanged until the controller can discover the new topology.

Least loaded failover can be combined with greedy failover. This mechanism has been proposed by Li et al. [11]. In this mechanism, scores are created based on controller loading and switch-controller delay. The scores determine how the nodes should be reassigned to controllers during failover.

### 3.2.3 Proposed Hop and Link Failover

HLF is a mechanism proposed by this paper. It takes a different approach to failover for distributed controllers. It incorporates the adoption of orphan nodes based on the physical number of hops from controllers as well as the link capacity from the controller to the nodes.

The previous failover methods are intended for SDN in data centers. They assume controllers and nodes are in close proximity to one another. HLF is intended to provide efficient and quick response for distributed controllers when controllers are not the physical located near each other. HLF would guarantee nodes to be within the least amount of hops to their controllers with the greatest amount of link capacity. HLF can be incorporated either reactively or proactively. This recovery mechanism can be implemented for a telecommunications network.

## 4 Proposed Hop and Link Failover Algorithm

HLF is proposed to combat controller failure when controllers are not physically located to each other. The approach of HLF takes into account the heterogeneity of network connections. Controllers adopt orphan nodes based on the number of hops from the nodes to the controller as well as the link capacity between the nodes. HLF organizes controller domains initially based on the number of hops leading to the controller. If the number of hops is equivalent to another controller, the link connection between the controllers is compared.

HLF is implemented using three algorithms. The first algorithm is for failure detection of the node, the second and third algorithms are for controller adoption. Failure detection is implemented through reverse heartbeat messages. Each node is responsible for sending messages to confirm the controller status. Failure is determined when messages are not received. Failure detection is implemented for each node in the following steps and algorithm3:

Step 1. A node sends a heartbeat message to its controller every five seconds.
Step 2. The node listens for a reply message from the controller. The timestamp from the last reply message is stored by the node.
Step 3. If the node does not receive a reply message from their controller within thirty seconds it deems the controller failed [16].
Step 4. The node becomes an orphan and broadcasts discovery messages to its neighbor nodes.

Algorithm3: Failure Detection (Node)

```
1.  while hasController = true
2.    listenForMessages
3.    if currentTime – lastSentMessage >= 5 seconds then
4.      sendHeartBeat
5.    end if
6.    if currentTime – lastReceivedMessage >= 30 seconds then
7.      hasController = false
8.    end if
9.  end while
10. broadcastOrphan(neighborNodes)
```

Controller adoption occurs when a controller receives a discovery message and tries to connect to the orphan node. The first controller to interact with the orphan node adopts the node. After the adoption, other controllers can also message the newly adopted node. The node will determine if it requires fewer hops than other controllers or is equal distance with more link capacity. If a controller exemplifies these metrics, it will replace the existing controller. Once a controller adopts a node it will then update its database with information from the node.

Controller adoption is implemented in the following steps and algorithms. Algorithm4 is from the controller's perspective while algorithm5 is from the node's perspective.

Controller:

Step 1. A controller receives a discovery message.

Step 2. The controller creates an adoption message with the hops and link capacity to the orphan node.

Step 3. The controller sends the message to the node.

Algorithm4: Controller Adoption (Controller)

```
1. listenForMessages
2. if receiveMessage then
3.  linkCapacity = getLink
4.  hops = getHops
5.  sendAdoptionMessage(linkCapacity, hops)
6. end if
```

Node:

Step 1. The node receives an adoption message from controller1.

Step 2. The node accepts the adoption.

Step 3. The node receives an adoption message from controller2.

Step 4. The node evaluates the connection to controller1 versus the connection to controller2. It takes the connection with the least amount of hops and bandwidth.

Algorithm5: Controller Adoption (Node)

```
10. listenForMessages
11. if receiveMessage then
12.  if hasController = false then
13.   linkCapacity = getLink
14.   hops = getHops
15.   adoptController
16.   hasController = true
17.  else if hasController = true then
18.   newLinkCapacity = getLink
19.   newHops = getHops
20.   if hops $\geq$ newHops and linkCapacity $<$ newLinkCapacity then
21.    dropCurrentController
22.    linkCapacity = newLinkCapacity
23.    hops = newHops
24.    adoptController
25.   end if
```

```
26.   end else if
27.   end if
28. end if
```

An example of HLF is shown in a reactive approach through figures 5 through 8. Figure 5 shows the architecture of three domains. Each domain consists of one controller and three nodes. The connection between each node varies.

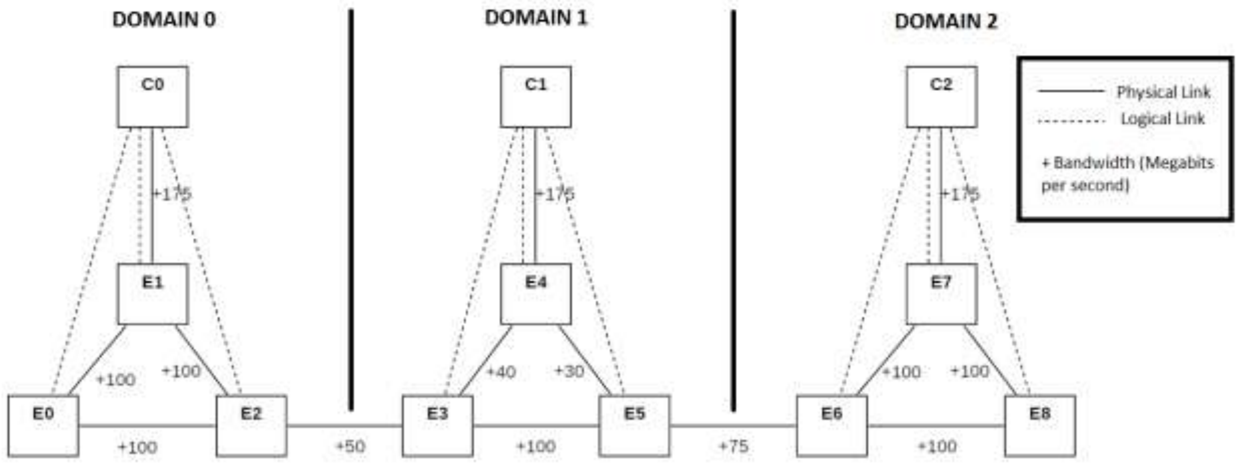

Figure 5: HLF Example Architecture

Reverse heartbeat messages are sent from the nodes to the controller. If controller1 fails the nodes in its domain will announce their orphan status through discovery messages to neighboring nodes. Node3 will send messages to node2, node4, and node5 as shown in figure 6. Since node2 is the only node connected to a controller it will forward the message from node3 to its controller through node1.

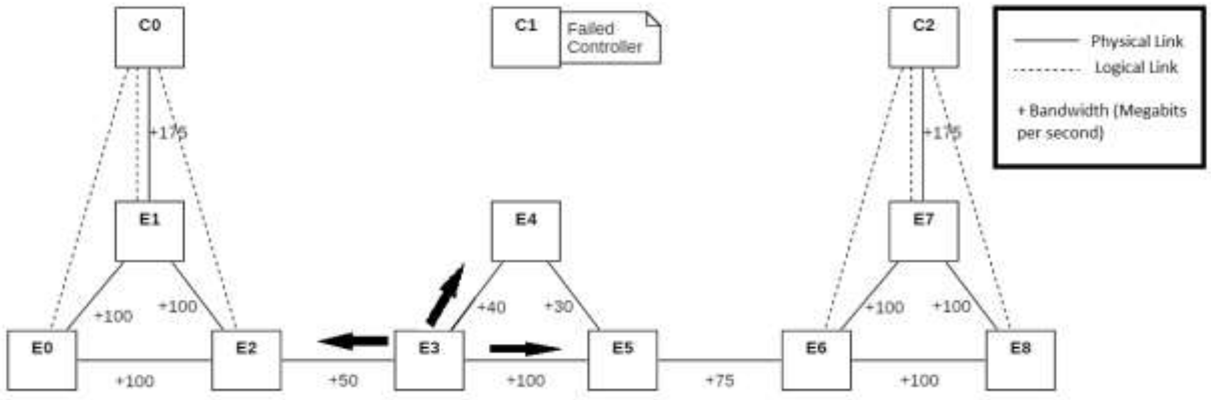

Figure 6: Node3 Orphan Broadcast

Controller0 will reply to the orphan message to adopt node3. It will send an adoption message through node1 and node2. When the message reaches node3, node3 will accept the adoption from controller0 (Figure 7) with three hops and a link connection of 50 megabits per second (Mbps). Node3 will then send data to controller0 for it to update its database.

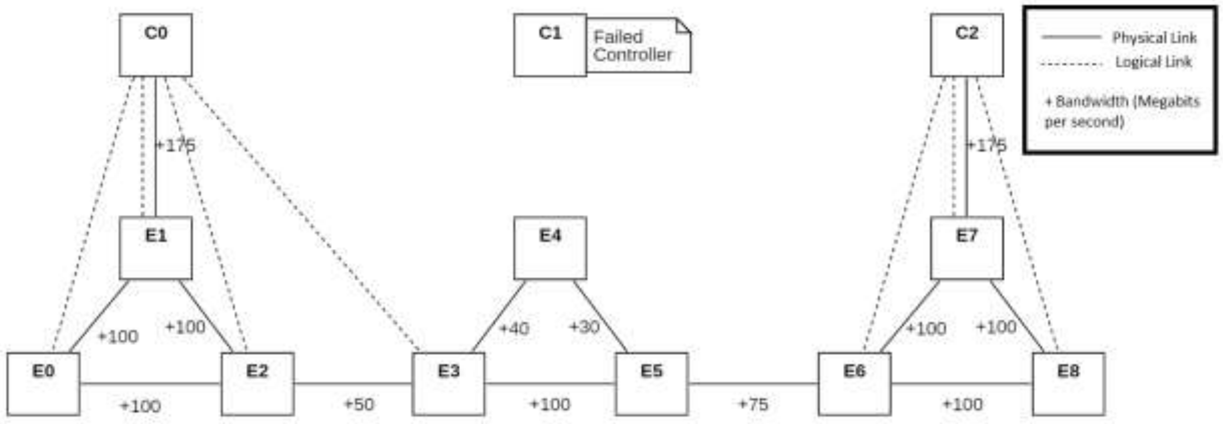

Figure 7: Node3 Adoption

Node5 will concurrently go through the same process as node3 and be adopted by controller2. Controller2 will notify node5 that it is three hops away with a link connection of 70 Mbps. Node4 broadcasts messages to node3 and node5. They forward their messages to their controllers. Node4 can be adopted by controller2 due to the latency of the messages. Controller2 will inform node4 that is it is four hops away with a link connection of 30 Mbps. Node4 will then receive an adoption message from controller0 with four hops and link connection of 40 Mbps. Node4 will then upgrade to controller0. The final architecture for failover is shown in figure 8.

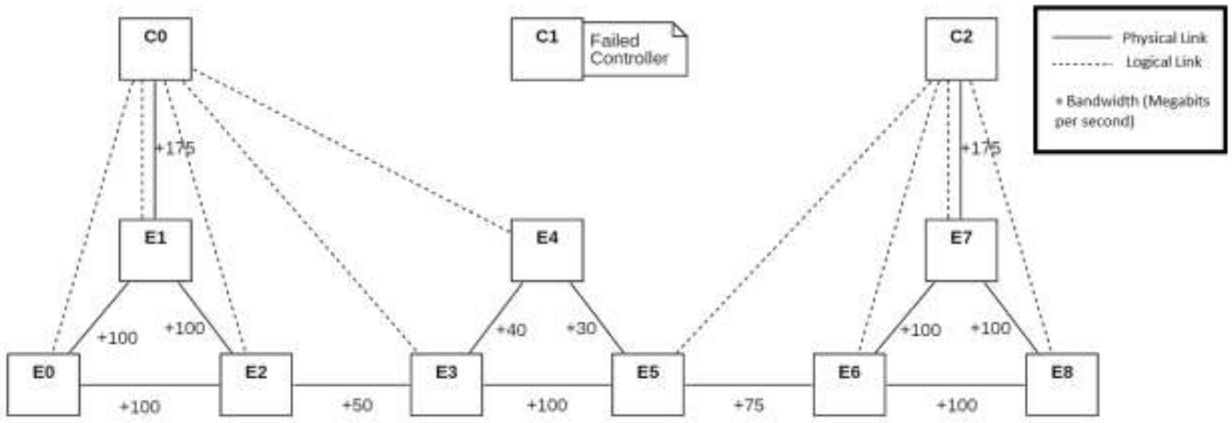

Figure 8: Final Architecture

HLF can be implemented proactively. In the proactive approach, each node pre-determines their backup controller. This process is completed prior to controller failure. During backup controller selection, the first controller to contact the node is initially the backup controller. Other controllers can replace the backup controller’s position if it requires less hops or equal with a higher link capacity.

## 5 Performance Evaluation

HLF can be implemented in any distributed SDN architecture however it is most effective when controller locations are not be physically located nearby. To simulate this, HLF was integrated in a 5G LTE architecture. In this architecture a set number of eNodeBs were used to show temporary failover. The LTE architecture was simulated with the assumption that controllers will reside relatively near their controller domain but may not reside in the same data center as each other.

### 5.1 Implementation and Setup

To evaluate HLF, four failover mechanisms were simulated. The four simulations include HLF as a reactive mechanism, HLF as a proactive mechanism, greedy failover as a reactive mechanism, and greedy failover as a proactive mechanism. These simulations were developed for 5G telecommunications thus all controllers were positioned in different locations. The simulations consisted of two different architectures.

The first failover architecture consists of three controllers and sixteen eNodeBs as shown in figure 9. In this simulation controller1 fails. Controller0 and controller2 must adopt orphan eNodeB5 through eNodeB10.

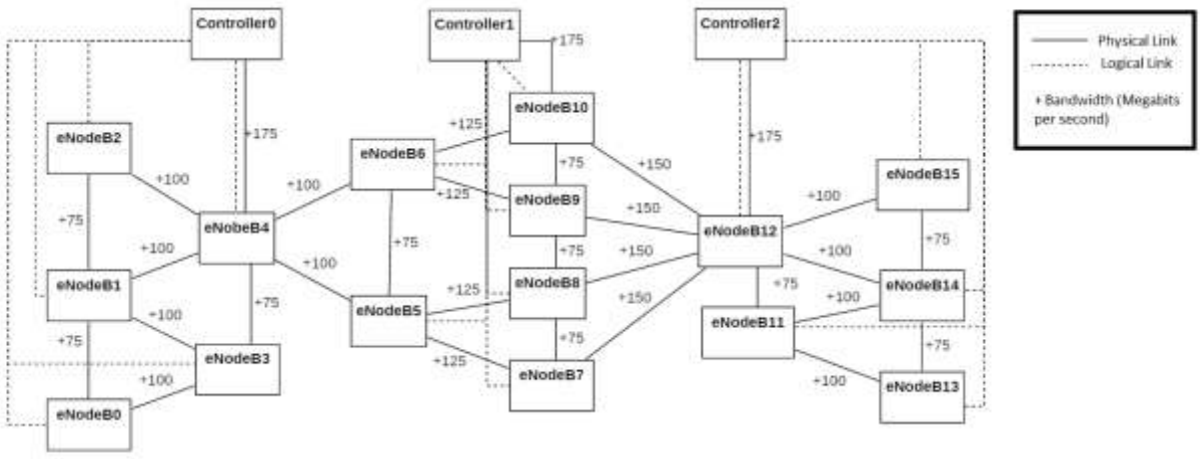

Figure 9: Failover Architecture1

The second failover architecture consists of three controllers and nine eNodeBs as shown in figure 10. In this simulation controller1 fails. Controller0 and controller2 must adopt orphan eNodeB3, eNodeB4, and eNodeB5.

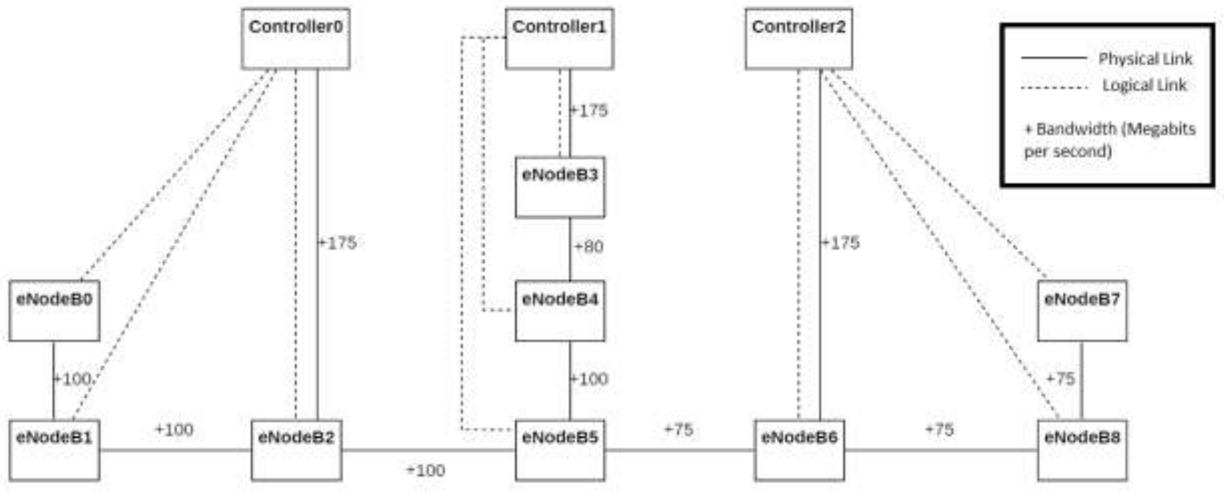

Figure 10: Failover Architecture2

Each simulation began with the initialization of the architecture. Next, controller1 would fail and failover procedures would begin. The orphan eNodeBs would broadcast orphan messages to nearby neighbors. To include latency of network delay with the messages, a variable sleep time was randomly created for both the controllers and eNodeBs. The network delay was determined after analysis of queuing delay, processing delay, propagation delay, and transmission delay.

Queuing delay was determined for the maximum network traffic load of 25%, 50%, 75%, and 95%. These loads were evaluated with a maximum queue of 512 packets [20] per node or controller. With a message size of 750 bytes and rate of transmission of 175 Mbps, one packet was calculated to take 0.034 ms. At a maximum traffic load of 25%, the maximum queuing delay was 4 ms. At 50%, the delay was 9 ms. For 75% the delay was 13 ms and 95% load was 17 ms.

Processing delay was assumed to be the same for every component thus neglected from the simulations.

Propagation delay was calculated by the distance between two eNodeBs divided by the speed of light. The maximum distance for an eNodeB is 32 kilometers [17]. With this distance, the maximum propagation delay is 0.1 milliseconds (ms). Since this number is insignificant, propagation delay was assumed to be negligible.

Transmission delay was calculated by the number of bits in a message divided by the rate of transmission. The message size was determined based on the size of a packet used in videotelephony [19]. This packet was 750 bytes. Rate of transmission was deducted based on the link capacity of a macrocell. A macrocell has a link capacity of approximately 175 megabits per second [18], however path loss will reduce the link connection. The minimum rate of transmission used was 50 megabits per second. Using these assumptions, the transmission delay calculates to 0.1 ms and is considered negligible.

For all failover mechanisms four simulation load variations were created. The simulations incorporated delay between 0 to 4 ms for 25% load, 0 to 9 ms for 50% load, 0 to 13 ms for 75% load, and 0 to 17 ms for 95% load.

For all simulations, each controller was initialized to each eNodeB prior to starting the simulation. For the proactive mechanisms, the eNodeBs begin contacting controllers to determine their backup controller during initialization.

## 5.2 Results and Analysis

Each simulation varied due to the queuing delay time that was randomly generated for each eNodeB and controller. To combat any potential bias in these simulations, each failover mechanism ran twenty five simulations. For each failover mechanism there were five evaluations. These evaluations included eNodeB hops, link capacity, controller load, network recovery time, and controller traffic on the network. Each evaluation was completed for 25%, 50%, 75%, and 95% uniform data traffic as well as non-uniform data traffic.

### 5.2.1 First Failover Architecture

The first architecture (figure 11a) was simulated in both the greedy and HLF mechanisms. This architecture reflects a general design for a telecommunications network. During these simulations the HLF mechanisms maintained the same architecture in every simulation regardless of the data traffic (figure 11b). The greedy mechanisms provided a variation of architectures. The least advantageous architecture was created by the greedy mechanisms. It occurred during the non-uniform data traffic. During this simulation the controller0 adopted all orphan eNodeBs (figure 11c).

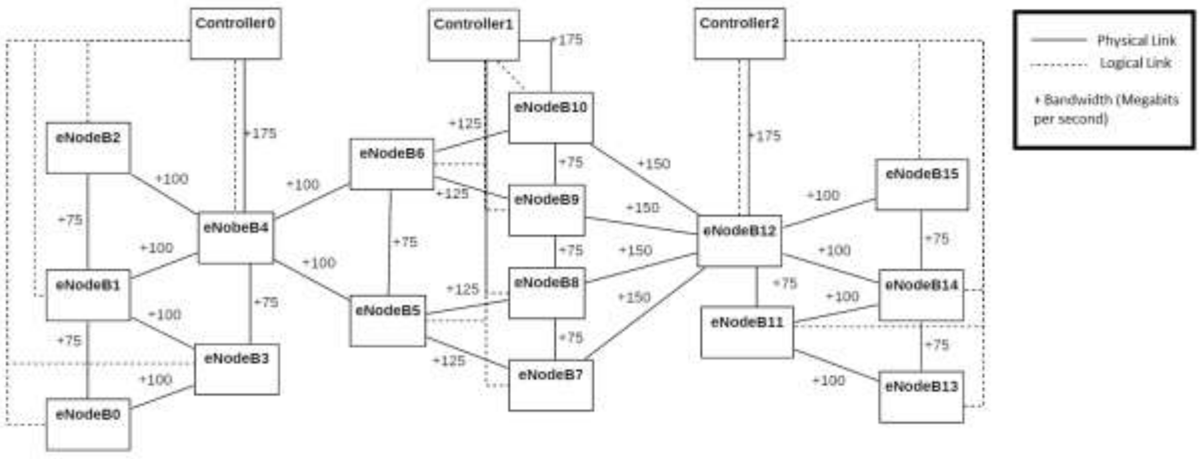

Figure 11a: Starting Architecture1

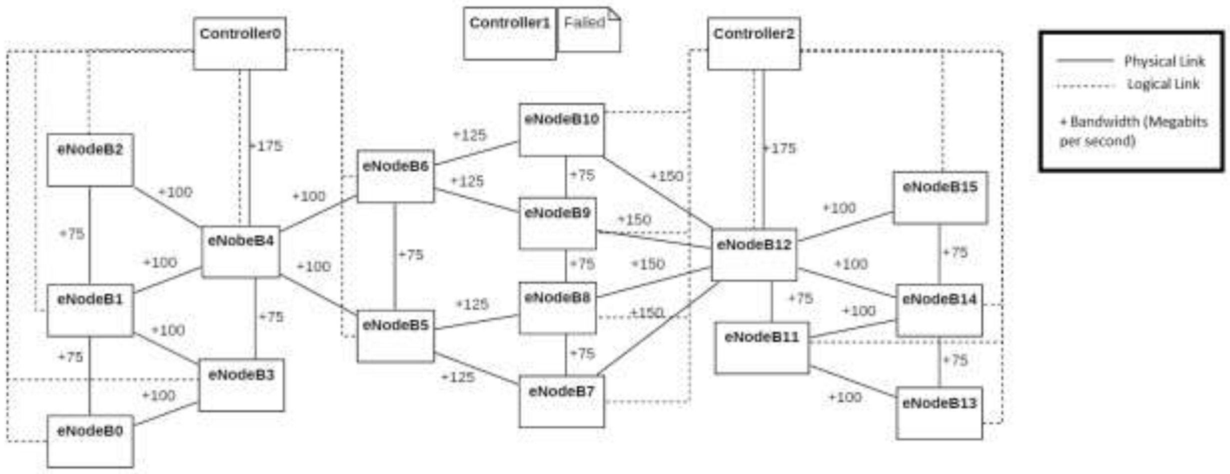

Figure 11b: HLF Resulting Architecture1

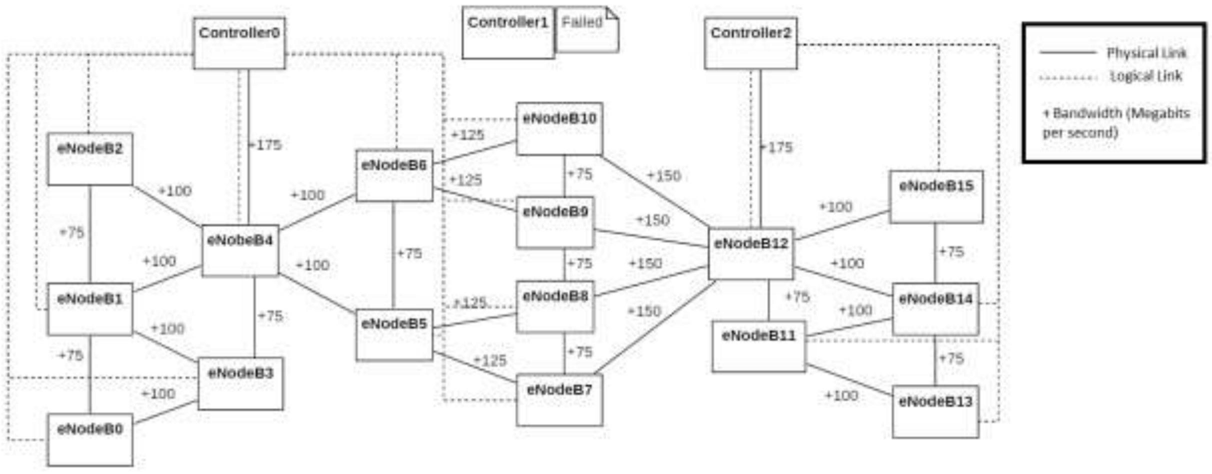

Figure 11c: Greedy Resulting Architecture1

#### *5.2.1.1 eNodeB Hops*

In order to increase the speed of transmission, the reduction of hops to a destination is necessary. Reducing the amount of hops reduces the amount of queues the message must sit in as well as the reduction in the processing delay per eNodeB. This metric was evaluated by analyzing the resulting architecture for each simulation.

Figure 12a shows the analysis of the simulation for architecture1. It shows the average number of hops from the eNodeBs to the controllers including the maximum and minimum number of hops in uniform data traffic. Figure 12b displays the average number of hops from the eNodeBs to the controllers including the maximum and minimum number of hops for 50% and 95% uniform data traffic. The results show that HLF always maintained two hops from the controller to all the orphan eNodeBs.

The greedy failover exhibited a variety of architectures except for the reactive greedy mechanism with 25% uniform data traffic. The variation in the architecture was due to the incorporation of delay. Delay was a factor in the response time for both eNodeBs and controllers in the greedy mechanism. It allowed controllers to adopt eNodeBs that were closer to other controllers. The reactive greedy mechanism with 25% load is believed to have remained constant due to the low amount of delay. The results show the greedy failover mechanisms adopting orphan eNodeBs with an average between 2 and 2.12 hops.

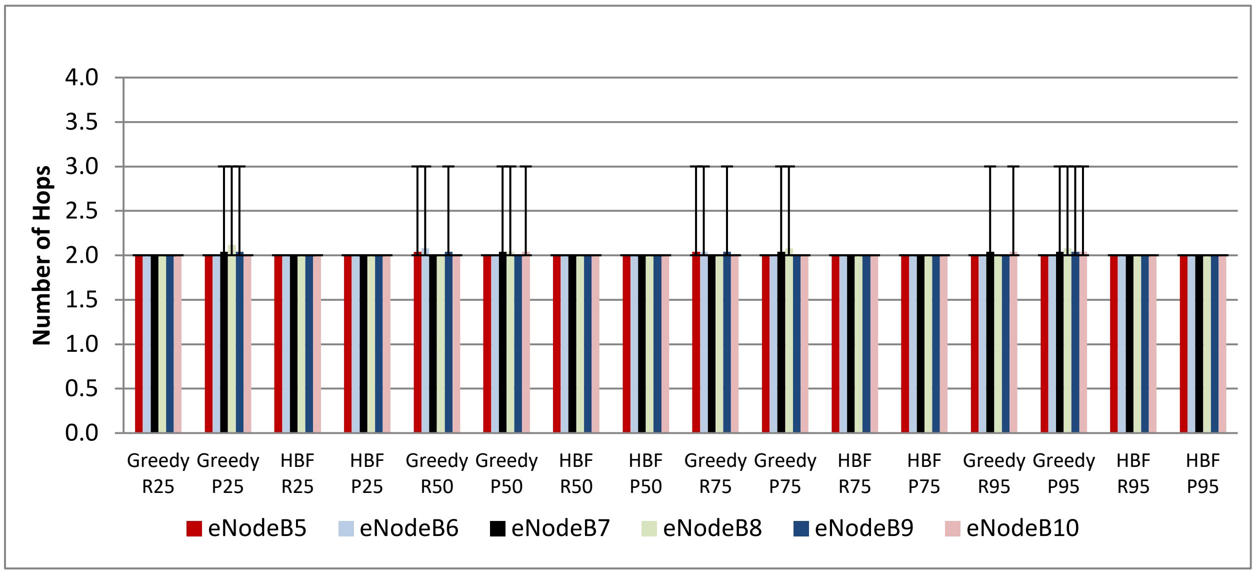

Figure 12a: Average Number of eNodeB Hops to Controller for Architecture1 for all Uniform Data Traffic

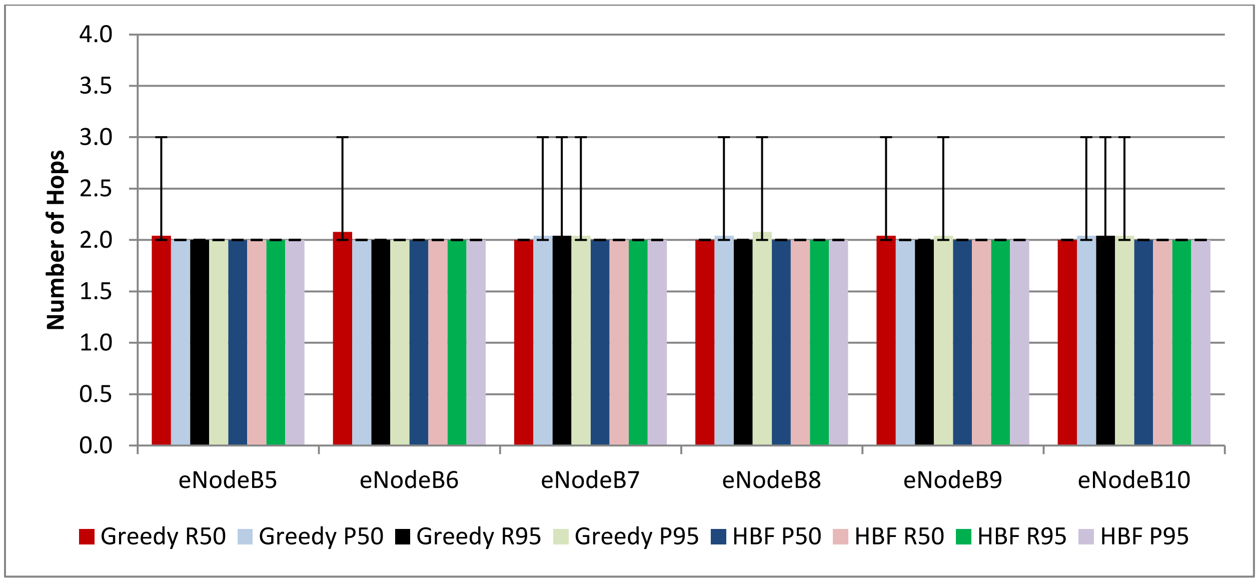

Figure 12b: Average Number of eNodeB Hops to Controller for Architecture1 for 50% and 95% Uniform Data Traffic

The first architecture was additionally simulated with non-uniform data traffic to increase the visibility of the issue of the greedy algorithm with regards to hops. The data traffic was non-uniform thus that the controller domain under controller2 maintained a 95% load while the rest of the system maintained a 25% load. Figure 13 shows the results of an instance that had occurred. During this instance controller0 adopted all the orphan eNodeBs in the greedy mechanism. It increased the amount of hops to the controller for eNodeB7 to eNodeB10. In the HLF mechanism, 2 hops were maintained for all orphan eNodeBs.

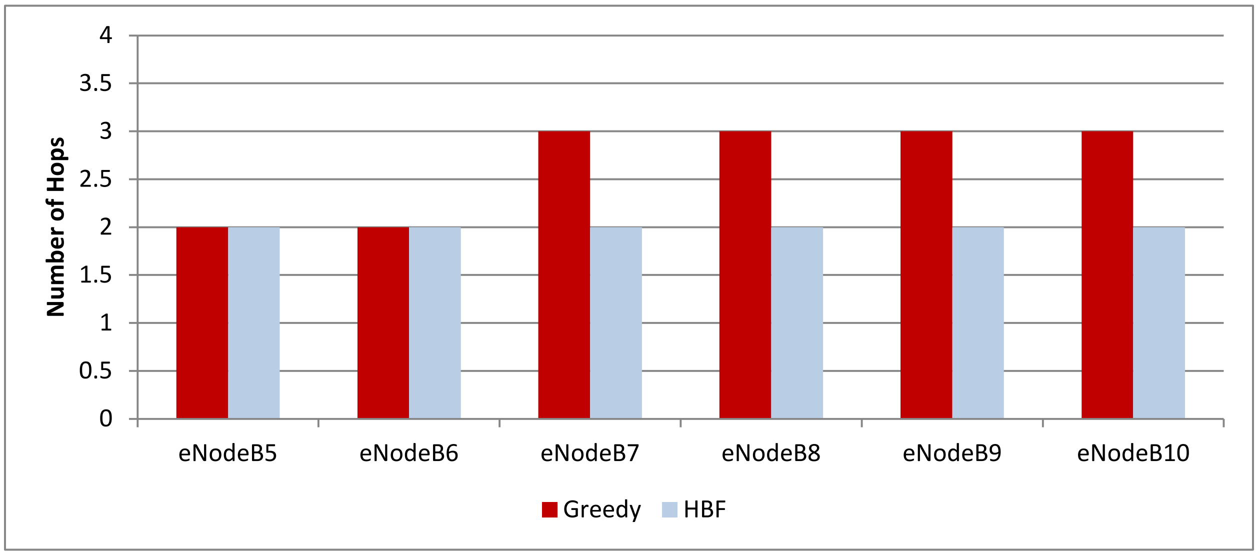

Figure 13: eNodeB Hops to Controller for Architecture1 in Non-Uniform Data Traffic

Figure 11c shows the configuration that occurred when using the greedy mechanisms for the non-uniform data traffic. The figure shows a skewed architecture in which controller0 adopted all orphan eNodeBs. ENodeB7 through eNodeB10 required less hops to controller2, however were not able to be adopted due to the latency of controller2. The latency of controller2 could be temporary yet cause an inefficient ending architecture.

The HLF mechanisms showed an evenly distributed ending architecture for failover (figure 11b). During failover it was possible for controller0 to initially adopt all orphan eNodeBs, however controller2 was able to re-adopt eNodeB7 through eNodeB10 to enable a more balanced architecture. This resulted with each orphan eNodeB having 2 hops to its new controller.

With regards to the amount of hops between the orphan eNodeBs and the controllers, the greedy mechanisms make no guarantees. The possibility of an inefficient resulting architecture can occur. HLF displays a better ending architecture for failover. It enables eNodeBs to be adopted to the controller with the least amount of hops.

### *5.2.1.2 Link Capacity*

To avoid network latency, the connection from a controller to an eNodeB must be as efficient as possible. Therefore orphan eNodeBs should be adopted with the highest available link connection. Figure 14a shows the results of the average, maximum, and minimum link connection from the orphan eNodeBs to the controllers for all uniform data traffic. Figure 14b shows the results of the average, maximum, and minimum link connection from the orphan eNodeBs to the controllers for 50% and 95% uniform data traffic. The greedy mechanisms showed a variation in the link connection per eNodeBs with exception of the reactive greedy mechanism with 25% traffic load. The HLF mechanism maintained the maximum possible link connection during every simulation.

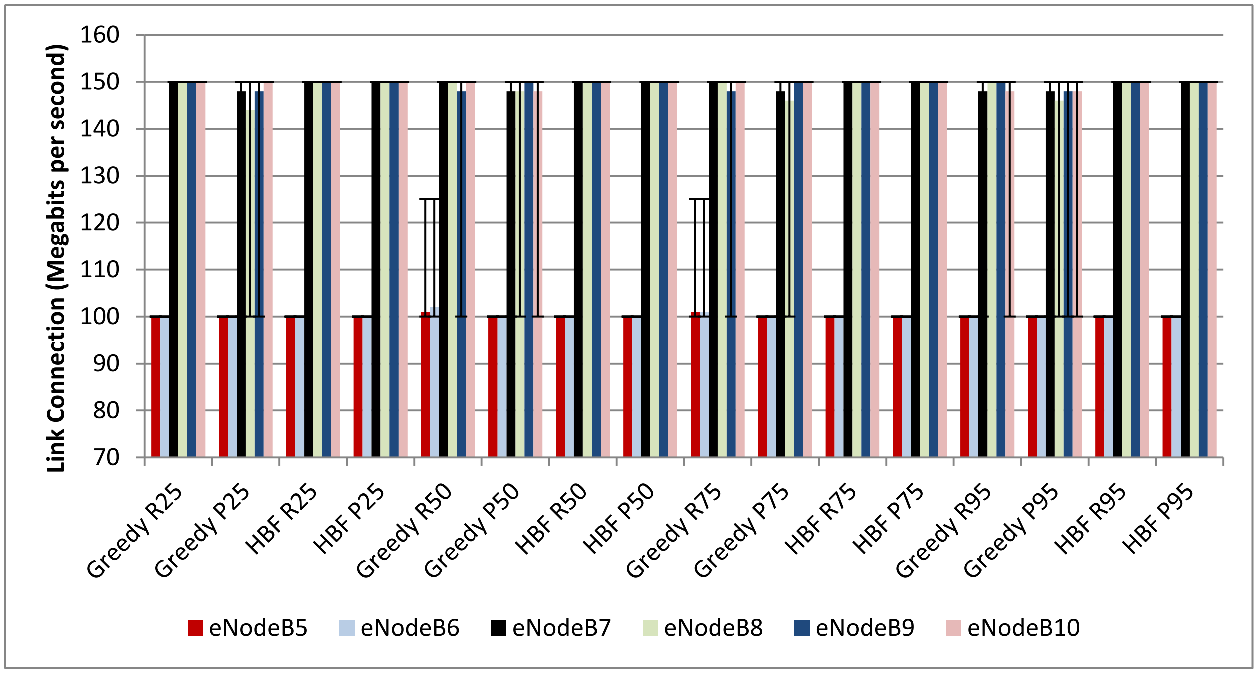

Figure 14a: Average Link Connection for Architecture1 for all Uniform Data Traffic

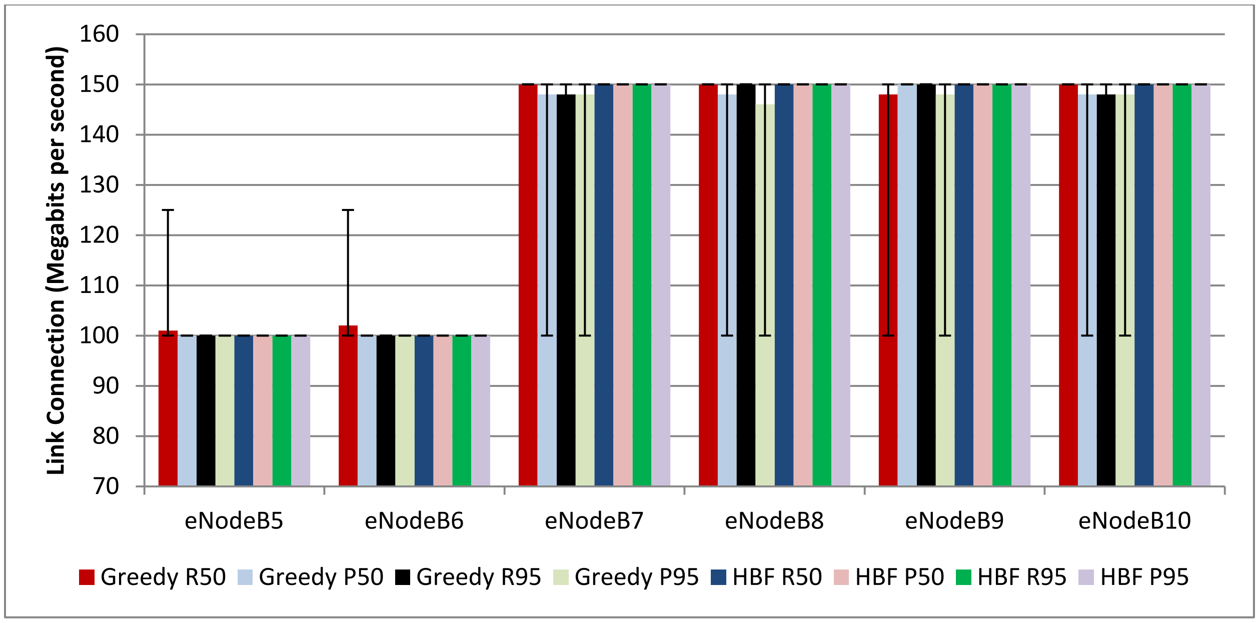

Figure 14b: Average Link Connection for Architecture1 for 50% and 95% Uniform Data Traffic

With non-uniform data traffic the results produced a lower link connection for the greedy algorithm while HLF maintained the maximum possible link connection with the least amount of hops (figure 11c, 15). The resulting architecture for the greedy mechanisms produced 100 Mbps link connection for all orphan eNodeBs. The resulting architecture for the HLF mechanisms produced 100 Mbps for eNodeB5 and eNodeB6 and 150 Mbps for eNodeB7 and eNodeB10.

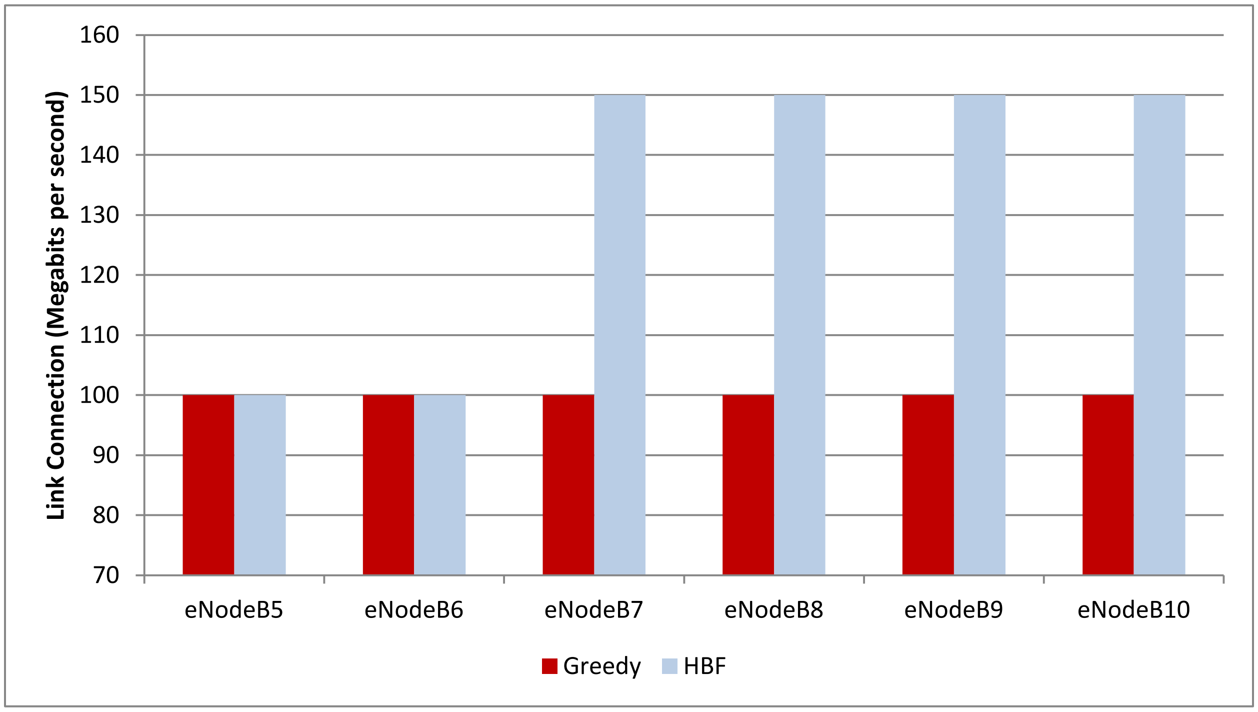

Figure 15: Link Connection for Architecture1 in Non-uniform Data traffic

The results for this architecture show that HLF always provides the ability for the highest link capacity from an eNodeB to its closest controller. The greedy algorithm does not ensure the maximum link connection for failover.

#### *5.2.1.3 Controller Load*

One measure of controller effectiveness can be correlated to the amount of load a controller has. The more eNodeBs communicating with a controller, the more traffic there is in that controller domain. A controller with higher traffic must have the capacity to handle it. Controller load for the architecture1 is shown on figure 16. These results display the average amount of eNodeBs per controller in addition to the maximum and minimum eNodeBs per controller with uniform data traffic.

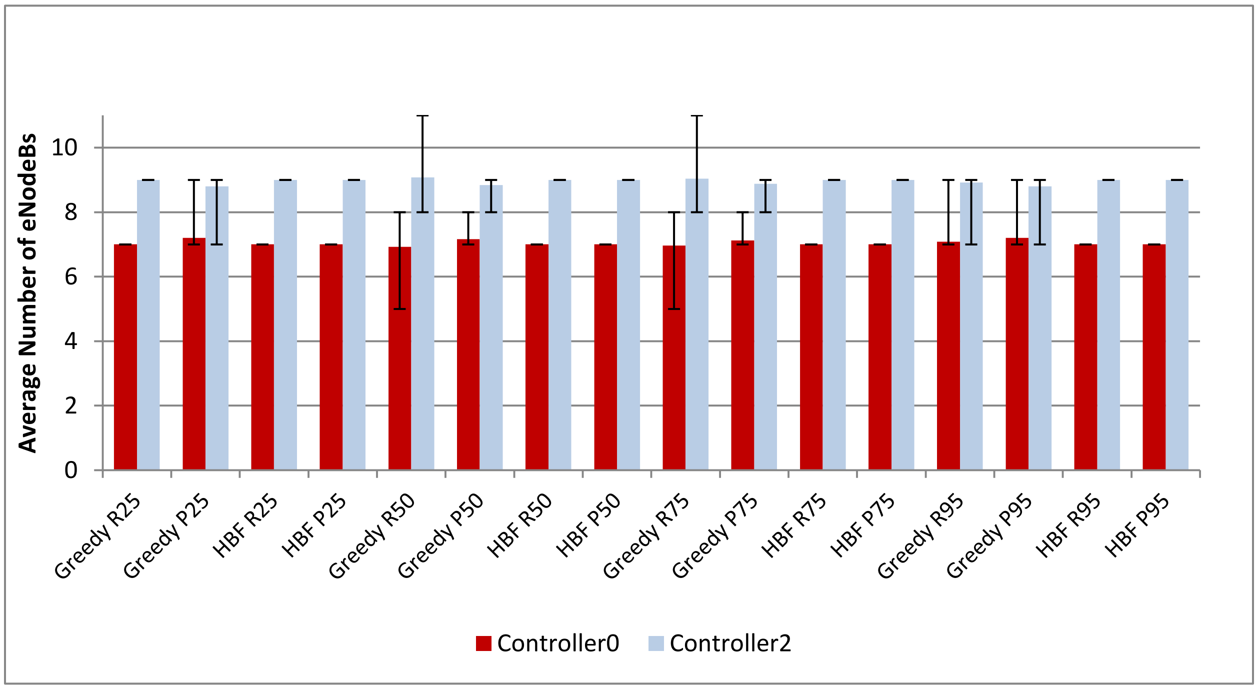

Figure 16: Average Controller Load for Architecture1

On average all mechanisms performed in a similar manner, however the results show that the architecture for the greedy mechanisms varied in all cases except for the reactive greedy mechanism with 25% load. The HLF architecture did not vary. It maintained the same architecture every time. The results show controller0 adopted two eNodeBs in addition to its own eNodeBs and controller2 adopted four eNodeBs in addition to its own eNodeBs. While this architecture is not evenly distributed, it does not overload one controller.

With uniform data traffic the greedy mechanisms showed variation in the amount of eNodeBs adopted per controller. Controller0 adopted between zero to four eNodeBs while controller2 adopted between two to six eNodeBs. This variation was dependent on the network delay. With less delay a controller was able to obtain more eNodeBs. This allowed controllers with less link connection to adopt eNodeBs. In a few simulations controller2 took the entire load of the controller1 and adopted all the orphan eNodeBs. With non-uniform data traffic, the reverse occurred where controller0 adopted all orphan eNodeBs. This additional load is unnecessary for one controller to solely bear when another controller has the available capacity.

For controller loading, the greedy mechanism showed a potential to create the most evenly distributed architecture, however not on a consistent basis. It also showed the potential to overload a controller. With HLF there was a consistent architecture. For this reason, HLF is a superior mechanism.

#### *5.2.1.4 Network Recovery Time*

The impact of failover can be evaluated through network recovery time. This metric is an evaluation of the recovery from the point at which a controller fails to the point at which all eNodeBs have been adopted in a final architecture. In this simulation, the recovery time was calculated from failure of the controller1 to the recovery of the last orphan eNodeB.

Figure 17 displays the results of the average network recovery time in addition to the maximum and minimum recovery time with uniform data traffic. The reactive greedy time proved to be the fastest recovery time while the proactive HLF took the slowest time to recover. Reactive HLF is shown to have a slower recovery time than the reactive greedy mechanism. This is because HLF first runs the reactive greedy mechanism then accepts upgrades to other controllers that incorporate decreased hops or the same hops with increased link capacity. It should be noted that eNodeBs initially recover at the speed of the reactive greedy mechanism but total recovery time is increased due to the upgrades. For this architecture, the proactive mechanisms proved to be slower on average to its counterparts. The reason for this is due to the structure of the architecture as well as how the proactive mechanism works. The benefit of the proactive mechanism is that orphan eNodeBs can be adopted when all the surrounding eNodeBs are in its controller domain. With this first architecture the orphan eNodeBs always have an immediate eNodeB in another domain. The proactive mechanism requires more overhead to operate. It must maintain both a controller and backup controller. This additional overhead creates additional processing time.

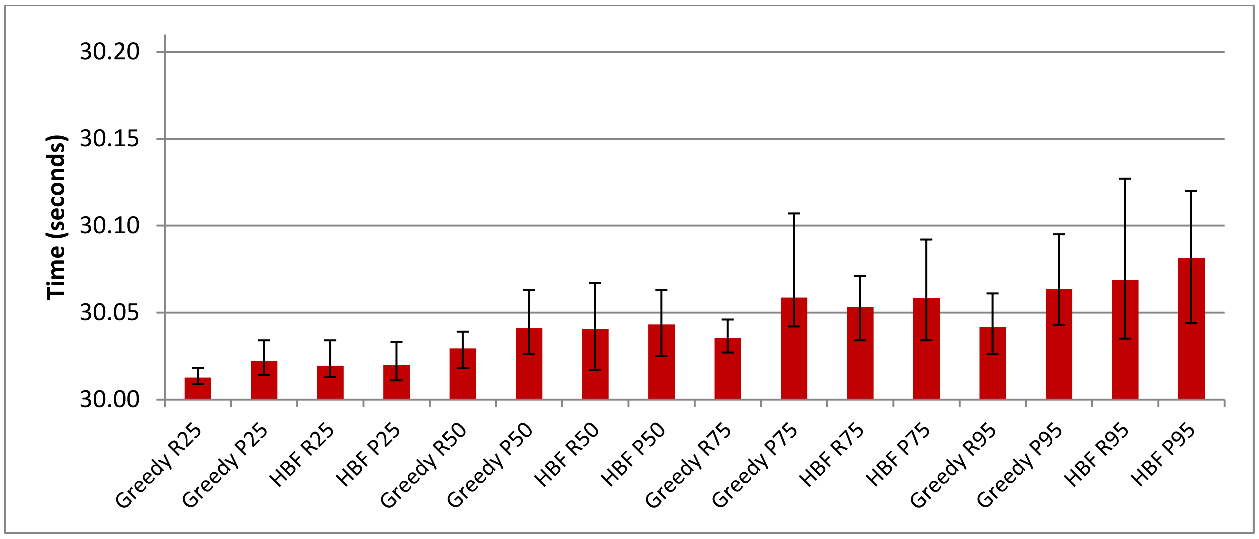

Figure 17: Average Network Recovery Time for Architecture1

For this architecture, the reactive greedy mechanism proves to provide the fastest failover. All other mechanisms have additional overhead which relays to added latency.

### 5.2.2 Second Failover Architecture

Architecture2 (figure 18a) was simulated in both the greedy and HLF mechanisms. This architecture reflects a specific design potentially for rural areas. In rural areas it is expected that mesh networks may not be cost effective as towns are established around nature's topology. ENodeBs may be built following rivers or streams. During these simulations the HLF mechanisms maintained the same architecture in every simulation regardless of the data traffic (figure 18b). The greedy mechanisms resulted in a variation of architectures. The least advantageous architecture occurred when controller2 adopted all orphan eNodeBs (figure 18c).

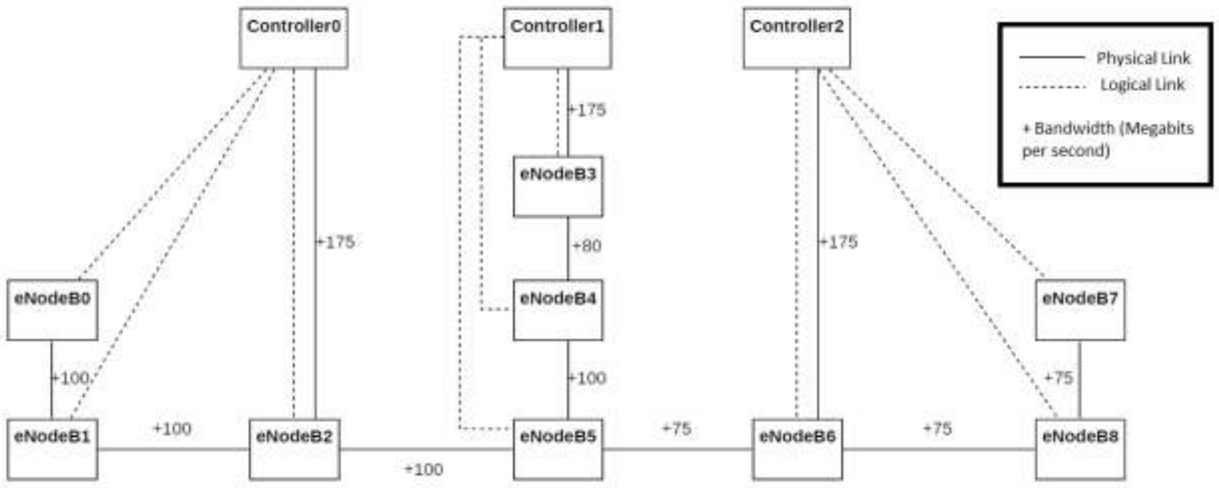

Figure 18a: Starting Architecture2

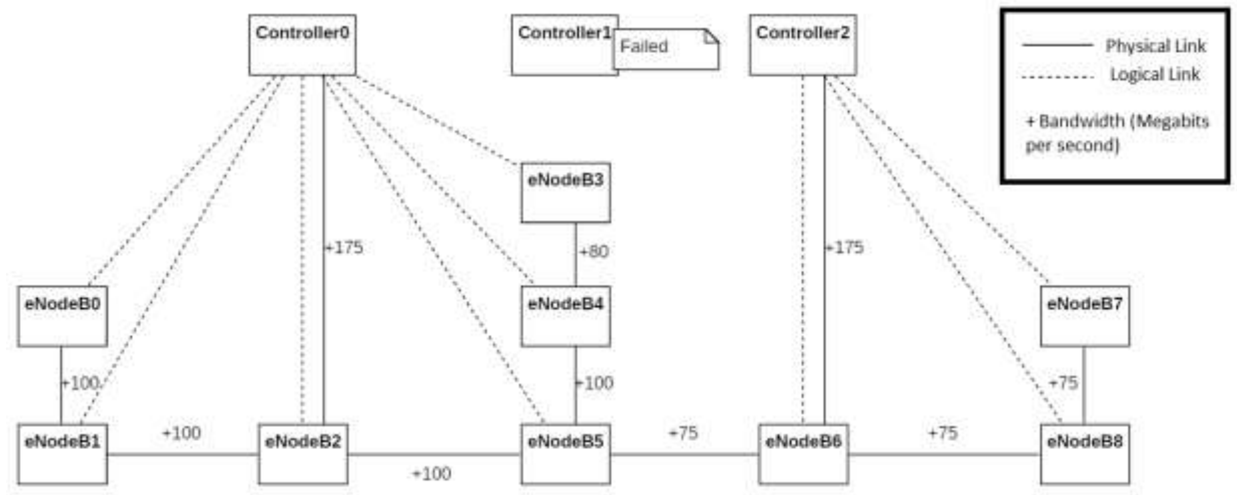

Figure 18b: HBF Resulting Architecture2

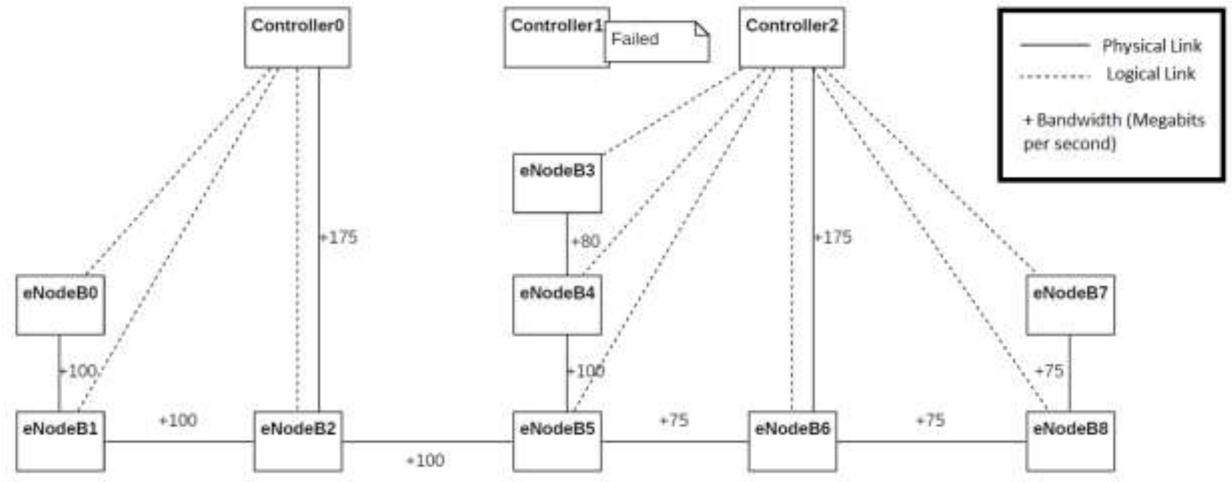

Figure 18c: Greedy Resulting Architecture2

### *5.2.2.1 eNodeB Hops*

The results of the second architecture's simulation for all uniform data traffic are shown in figure 19a. The results display an even amount of hops regardless of the algorithm. This is due to the architecture in which both controller0 and controller2 have an equal number of hops to each orphan node. Since the number of hops is equivalent, HLF does not provide any advantage to the greedy algorithm for this metric. Figure 19b displays the average hops for 50% and 95% data traffic per eNodeB.

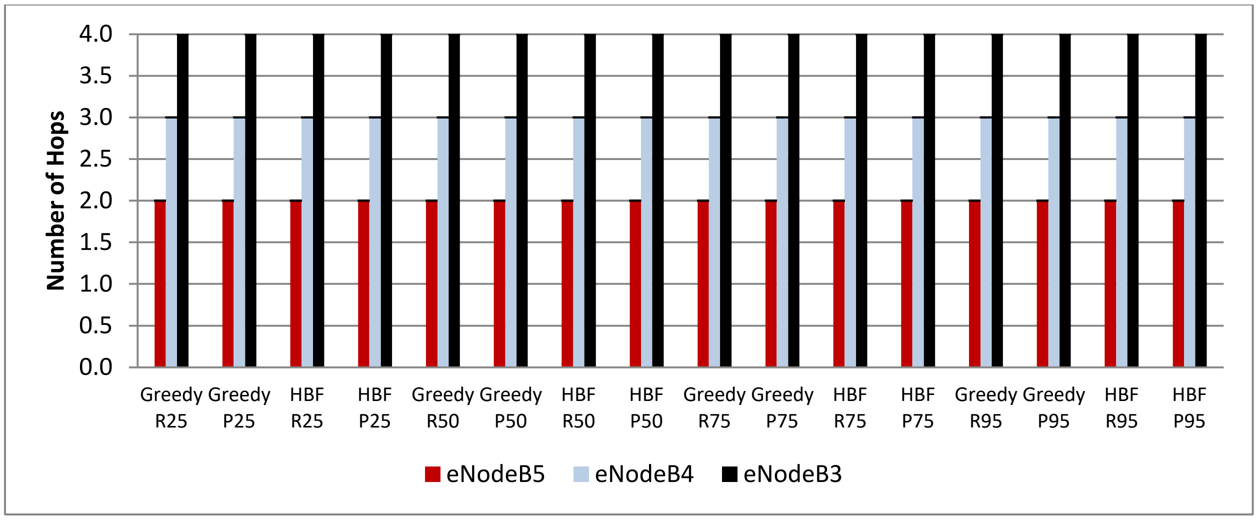

Figure 19a: Average Number of eNodeB Hops to Controller for Architecture2 for all Uniform Data Traffic

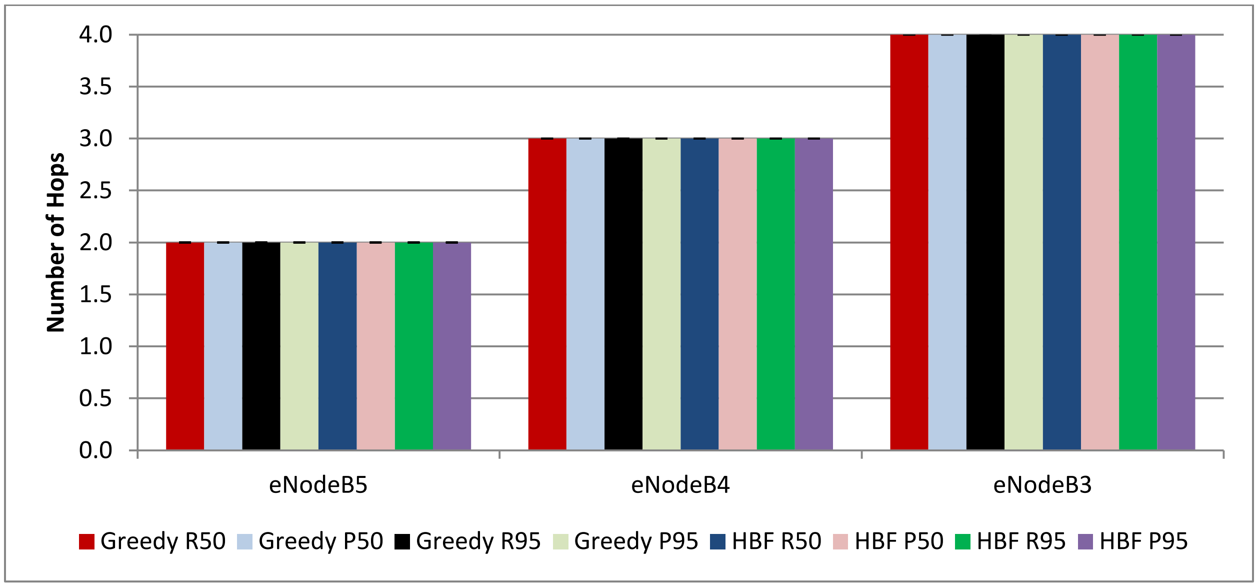

Figure 19b: Average Number of eNodeB Hops to Controller for Architecture1 for 50% and 95% Uniform Data Traffic

This architecture was also simulated with non-uniform data traffic. In this simulation controller0's domain operated with a 95% data traffic while the rest of the network operated at 25% data traffic. In this simulation there was no difference in regards to the number of hops (figure 20).

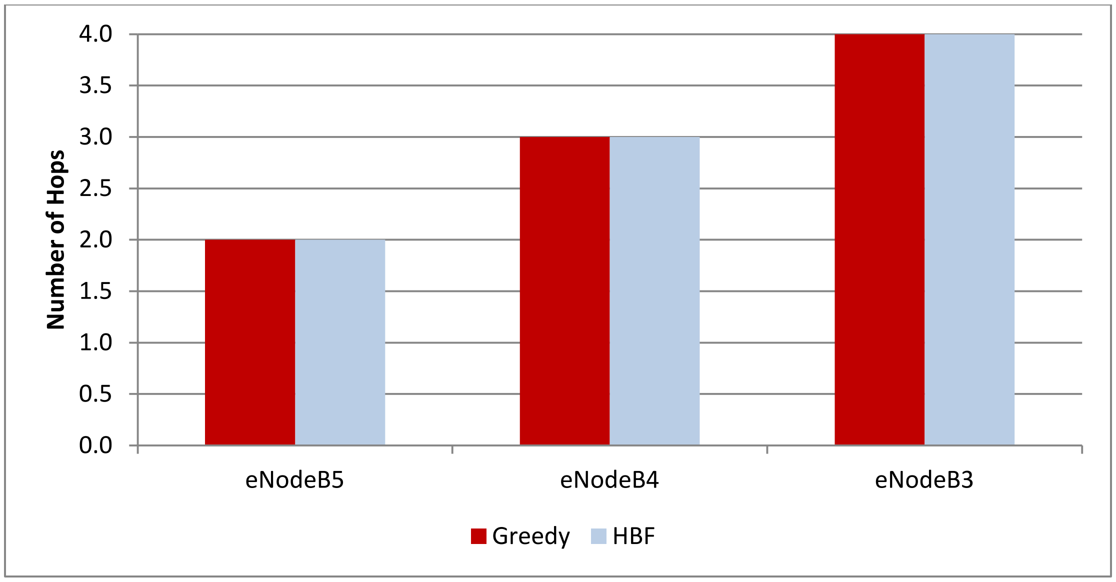

Figure 20: eNodeB Hops to Controller for Architecture2 in Non-Uniform Data Traffic

#### *5.2.2.2 Link Capacity*

The results of the average link connection for all uniform data traffic are shown in figure 21a. Figure 21b shows the average link connection for 50% and 95% uniform data traffic. The greedy mechanisms displayed inconsistent failover architectures. In some simulations controller0 adopted all orphan eNodeBs while in other simulations controller2 adopted all of them. The results show the average link connection was less than the maximum available link connection. The HLF mechanisms did not show a variation. HLF ensured that in the final failover architecture controller0 adopted the orphan eNodeBs 100% of the time. The results prove to enable the maximum link capcacity for all orphan eNodeBs.

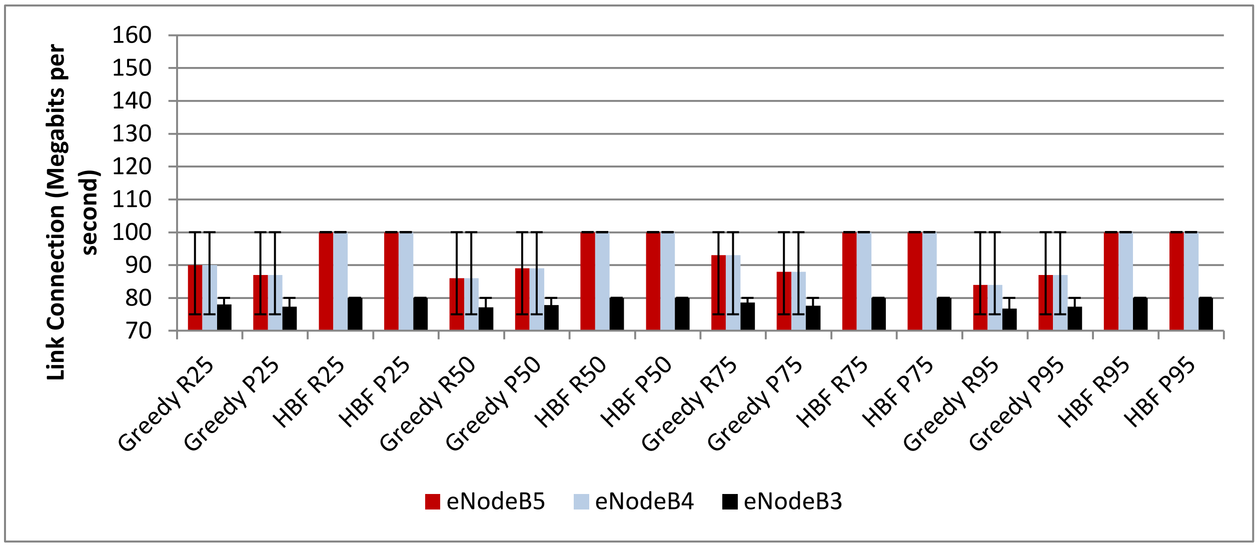

Figure 21a: Average Link Connection for Architecture2 for all Uniform Data Traffic

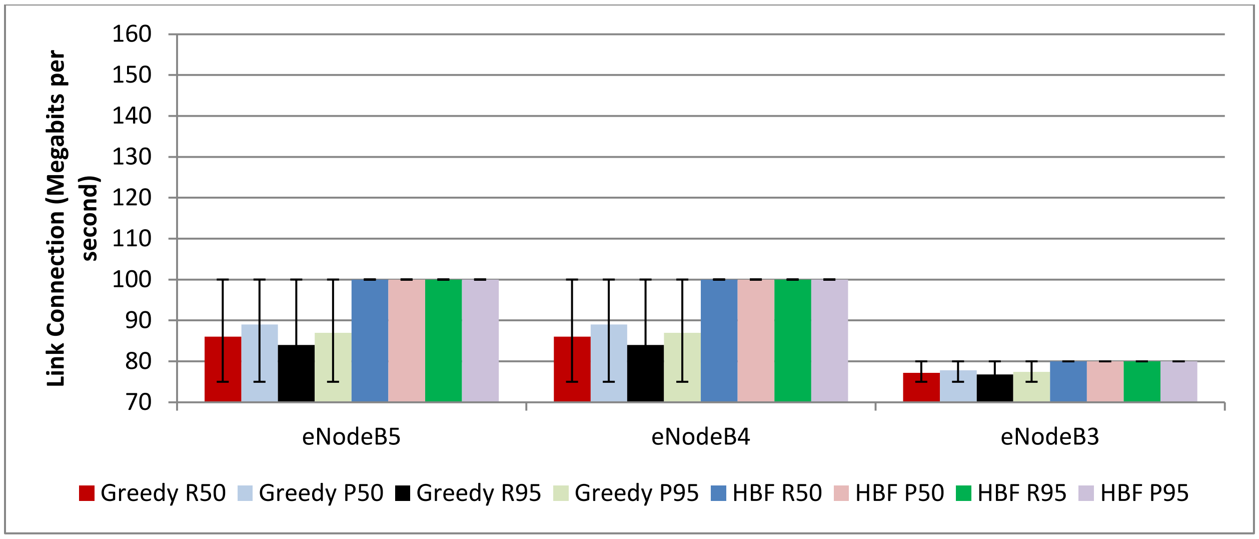

Figure 21b: Average Link Connection for Architecture2 for 50% and 95% Uniform Data Traffic

This architecture was also simulated with non-uniform data traffic as described in the eNodeB Hops section. Figure 22 shows the results of this simulation. The greedy mechanisms adopted the lowest available link connection while HLF adopted the highest available link connection. For the greedy mechanisms all orphan eNodeBs adopted a link connection of 75 Mbps. In HLF eNodeB3 was adopted with a link connection of 80 Mbps while eNodeB4 and eNodeB5 adopted a link connection of 100 Mbps.

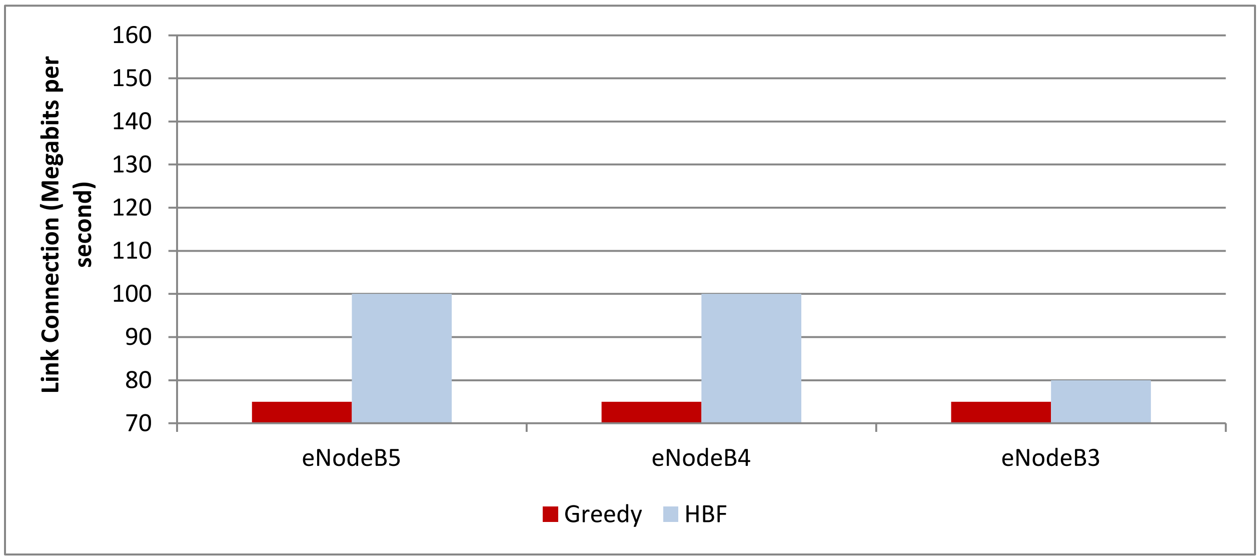

Figure 22: Link Connection for Architecture2 in a Non-Uniform Data traffic

The results for this architecture show that HLF provides the highest link capacity from all the connections to the controllers. The greedy mechanism cannot be trusted to provide the maximum link capacity during failover.

### 5.2.2.3 *Controller Load*

Controller load varied with the greedy mechanisms for all instances of uniform data traffic (figure 23). The HLF mechanisms did not vary the controller load. It maintained the same architecture every time. With HLF the results showed controller0 adopted 3 eNodeBs in addition to its own eNodeBs and controller2 did not adopt any eNodeBs.

With the greedy mechanisms, the amount of eNodeBs adopted per controller varied from 0 to 3 for controller0 and from 0 to 3 for controller2. This variation was dependent on the network delay.

For non-uniform loads, the HLF mechanism resulted with controller0 adopting 3 eNodeBs in addition to its own eNodeBs and controller2 did not adopting any. With the greedy mechanism, the opposite occurred where controller2 adopted 3 eNodeBs and controller0 did not adopted any.

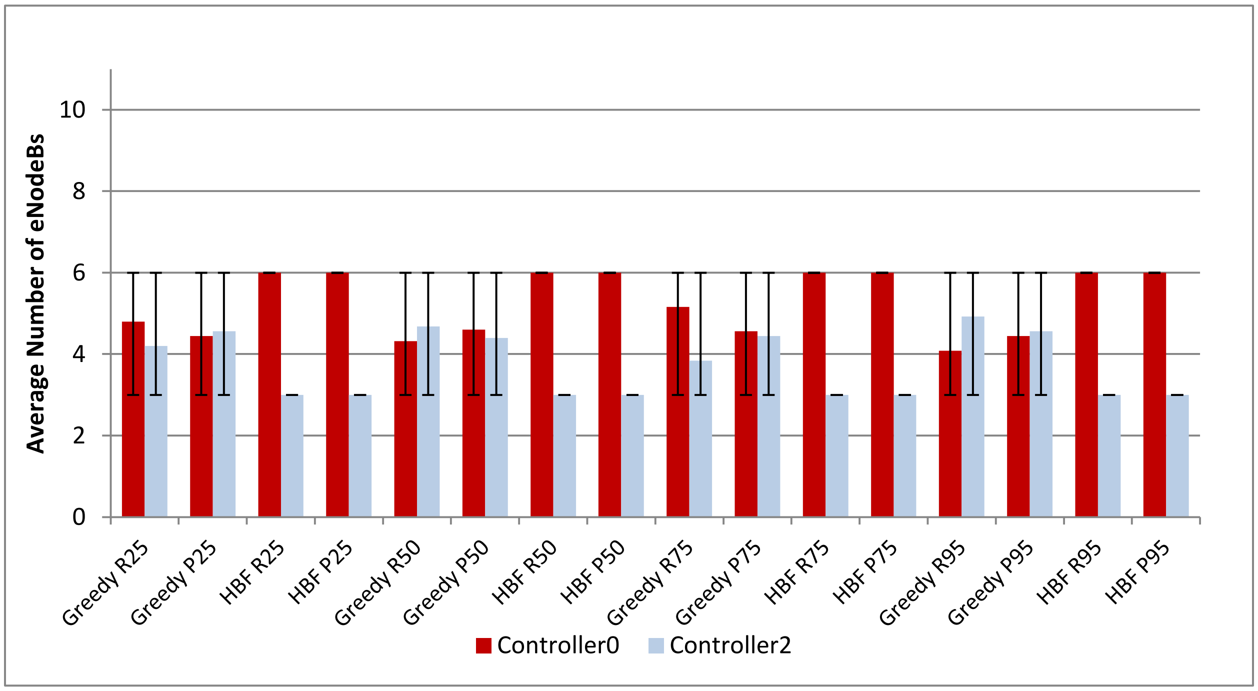

Figure 23: Average Controller Load for Architecture2

In this architecture controller loading is maintained regardless of the mechanism. One controller will always adopt all orphan eNodeBs. This is due to the how both mechanisms operate.

### 5.2.2.4 *Network Recovery Time*

Network recovery time for uniform data traffic is shown in figure 24. The proactive greedy mechanism shows the fastest average network recovery times. The slowest average recovery time is from the HLF reactive mechanism. The proactive mechanisms prove to be faster due to its nature. In the proactive mechanisms, the eNodeBs have a known backup location to another controller. They message the controller for adoption immediately after failure. This behavior allows a fairly linear recovery time regardless of the number of orphan eNodeBs [9].

With the reactive greedy mechanism, the recovery time correlates to number of orphan eNodeBs. This behavior is to be expected as the eNodeBs in the reactive mechanism can only message controllers once their neighbor has is in a different domain. The reactive HLF mechanism takes the longest average recovery time for the final failover architecture. As explained for the first architecture this is due to the fact that it runs the greedy mechanism then potentially upgrades.

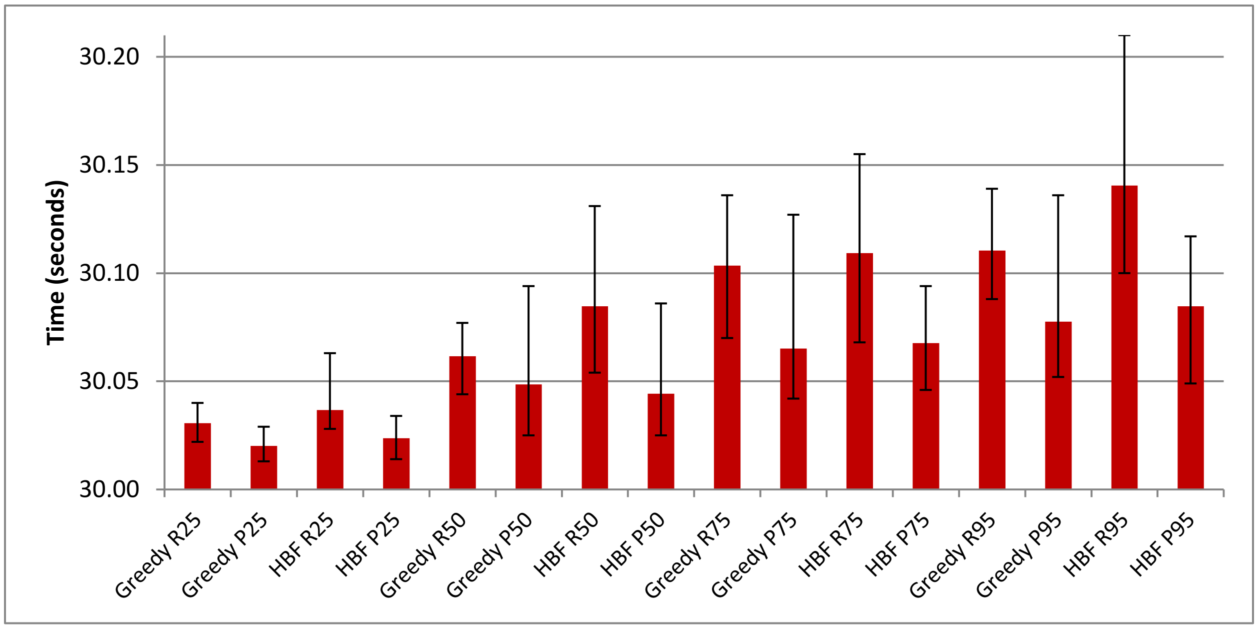

Figure 24: Average Network Recovery Time for Architecture2

For this architecture the proactive mechanisms are the faster mechanisms for failover. The speed at which failover can occur in a proactive mechanism is not dependent on the amount of eNodeBs but the number of hops from an orphan eNodeB to another domain. The greater the number of hops are the faster the proactive mechanisms will recovery when compared to the reactive mechanisms.

### 5.2.3 Control Traffic

Control traffic is a part of network traffic and is defined as the amount of control data transferred between the eNodeBs and the controllers. For theses simulations the control traffic did not vary by the greedy mechanism or HLF mechanism. Control traffic was affected varying on whether the mechanism was reactive or proactive.

Failover mechanisms require heartbeat messages. These messages increase the control traffic on each component in the network. Increased messages correlate to an increase in network delay. In the simulations the reactive mechanisms increased the data traffic by sending reverse heartbeat messages from the eNodeBs to its prospective controller. The proactive mechanisms required additional messaging by sending reverse heartbeat messages to both its controller and backup controller. This doubled the amount of required messaging and increased the network delay.

# 6 Conclusion and Future Work

As SDN is being commercialized for networking, it becomes apparent that failover strategies are needed to remove the issue of a single point of failure and maintain seamless connectivity for users. There are a plethora of failover strategies available. Each mechanism has its own advantages that should be analyzed prior to implementation.

This paper has shown the capabilities of the HLF mechanism to handle failover when controllers are setup in a distributed architecture and controller locations not located in the same location. HLF is a non-redundant controller strategy that can be incorporated in a reactive or proactive mechanism. It proves to be an advantageous mechanism that provides failover for nodes with the least amount of hops and the highest link capacity. It addresses the randomness of the greedy algorithm for node adoption.

Future work will include a weighted algorithm for dynamic adjustment for heterogeneous networks. This mechanism will consider network resources such as controller load, capacity, and bandwidth. This direction will be investigated to further tune failover procedures for distributed controllers.

# References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, Oct. 2016.

[2] Akram Hakiri and Pascal Berthou. “Software Defined Mobile Networks: Beyond LTE Network Architecture,” CH 5 Leveraging SDN for The 5G Networks: Trends, Prospects and Challenges. Wiley Series in *Communications Networking & Distributed Systems, Mobile & Wireless Communications*, ISBN 978-1-118-90028-4 Jun. 2015.

[3] Songlin Sun, Liang Gong, Bo Rong, and Kejie Lu, "An Intelligent SDN Framework for 5G Heterogeneous Networks" in *IEEE Communications Magazine*. Nov. 2015, pp. 142-147.

[4] Software-Defined Networking. (2017) “What is Software-Defined Networking” [Online]. Available: http://www.software-defined.net/networking.php

[5] L. J. Chaves, I. C. Garcia and E. R. M. Madeira, "OpenFlow-based Mechanisms for QoS in LTE Backhaul Networks," in *Symp. on Computers and Comm.*, Messina, Italy, June 2016.

[6] Madhusanka Liyanage et al., “Leveraging LTE security with SDN and NFV,” in *Int. Conf. on Industrial and Information Systems.* Feb. 2016

[7] International Telecommunications Union, “FG IMT-2020 Report on Standards Analysis,” Telecommunication Standardization Sector, 2013-2016

[8] Andrew Berg. (2016, Feb. 12) “At&t Announces Field Trials in Austin,” [Online]. Available: https://www.wirelessweek.com/news/2016/02/t-announces-5g-field-trials-austin

[9] Mathis Obadia et al., “Failover Mechanisms for Distributed SDN Controllers” in *Int. Conf. and Workshop on the Network of the Future*. June 2015.

[10] Tsai-Wei Yang and Kuochen Wang, "Failure Detection Service with Low Mistake Rates for SDN Controllers" in *Asia-Pacific Network Operations and Management Sym*. Oct. 2016. pp. 1-6.

[11] D. Li et al., "High Availability for Non-Stop Network Controller" in *Proc. IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks, Jun*. 2014, pp. 1-5.

[12] Ko-Chih Fang et al., "A Fast and Load-aware Controller Failover Mechanism for SDN" in *Int. Sym. on Comm. Systems, Networks, and Digital Signal Processing*. July 2016.

[13] V. PashKov et al., "Controller Failover for SDN Enterprise Networks" in *Sciences and Technology Conf. on Modern Networking Technologies*. Oct. 2014.

[14] M. Borokhovich and S. Schmid. “How (not) to shoot in your foot with SDN local fast failover” in *Principles of Distributed Systems*. Springer. 2013. pp. 68–82.

[15] N. Beheshti and Y. Zhang. “Fast Failover for Control Traffic in Software Defined Networks” in *IEEE GLOBECOM*. 2012.

[16] Cisco. (2016, Aug. 8) “MME Administration Guide, StarOS Release 20,” [Online]. Available: http://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/20/MME/b_20_MME_Admin/b_20_MME_Admin_chapter_010.html

[17] Mobile Network Guide. “Moblie Base Stations,” [Online]. Available: http://www.mobilenetworkguide.com.au/mobile_base_stations.html

[18] TEcore. (2017, Feb. 3) “4G LTE Radio Access Network,” [Online]. Available: http://www.tecore.com/wp-content/uploads/2017/02/Tecore-Networks_CoreCell-E-LTE-eNodeB-Data-Sheet_02.03.17.pdf

[19] Kristian Kielhofner. (2013, Sept. 20) “Not Just AstLinux Stuff,” [Online]. Available: http://blog.krisk.org/2013/09/apples-new-facetime-sip-perspective.html

[20] Cisco. (2016, Nov. 28) “QoS: Congestion Management Configuration Guide, Cisco IOS XE Release 3S,” [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_conmgt/configuration/xe-3s/qos-conmgt-xe-3s-book/qos-conmgt-qdepth.html