UiO : Department of Informatics
University of Oslo

# Exploring InfiniBand Congestion Control

Ahmed Yusuf Mahamud

Master's Thesis Spring 2015

# Exploring InfiniBand Congestion Control

Ahmed Yusuf Mahamud

May 18, 2015

# Abstract

Congestion Control (CC) is used to achieve high performance and good utilization of network resources during high load in lossless interconnection networks. Without CC a congestion which started from a single node can grow, spread and degrade the performance of the network. Congestion can affect both the contributors of the congestion and also other traffic flows in the network. Infiniband (IB) is one of communication standards providing support for Congestion Control. The IB standard describes the CC functionality for detecting and resolving congestion. The behavior of the IB CC mechanism depends on the values of CC parameters. The given values of the parameters will determine characteristics like how aggressive the congestion detection should be, the rate of feedback from the forwarding node detecting congestion to the contributors of the congestion - and how much and for how long the contributors should lower their injection rates. But there are very few guidelines about how to set the values of the CC parameters for IB CC it to be efficient.

In this thesis, an experiment of a Mesh network topology will be conducted using OmNet++ as a simulation platform. Large amount of traffic will be generated and fed to the network until a congestion is contributed. The performance will be measured when Infiniband congestion control is disable and when it is enabled. The results from those simulations will be compared and analysed. The topology's host-to-switch link capacities are to be increased. There will be a search for proper IB CC parameters and finally, we will learn more about how IB CC parameters influence performance by focusing on some of the parameters.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

According to a report from IHS inc (IHS, 2014), more than 6 billion new Internet-enabled devices were produced in 2014. The number of new products that are connecting to Internet, are increasing beyond expectations. Cellphones, tablets and computers are among the most popular items. It is common that every user has and uses more than one Internet connectable device. These devices have very huge impact to existing network infrastructures. The infrastructure development can not level up to our existing demand (David Ely & Wetherall, 2001).

It is not only that the number of Internet-enabled devices are increasing rapidly, but also our data consumption changed very much in the last few years. Streaming, peer-to-peer file sharing and cloud use, has became part of our everyday activities. Our ever growing demand is deteriorating the performance of the traffic in the network. If our network bandwidth consumption continues like how it is today, it will not take long time before the performance of major networks degrade severely. What was once few computers connected together, grew to be the Internet, and today more than ever, we consume more network bandwidth, and the increase will continue many years in the near future. That is why, Congestion Control in lossless high performance interconnection network in data centers will be focused on in this thesis. Though network hardware is getting more and more powerful, it will soon be impossible to handle all the traffic that is going through, and an increase of network congestion is inevitable.

Congestion is simply the outcome of too much traffic fed into a network link; exceeding link capacity. When a congestion occurs, its effect includes queueing delay, packet loss or blocking new connections. These will have severe consequences to a lossless interconnection network. Congestion Control (CC) which also refers as Congestion Management is a counter measurement for relieving the consequences of congestion in congested interconnected network. CC has been widely studied, debated and solved in traditional networks such as local area networks (LANs) and wide area networks (WANs) according to E. Gran et al., 2010. InfiniBand Congestion Control is among existing congestion counter measurement.

Infiniband Congestion Control (IB CC) is one of communication standards providing support for Congestion Control, (Gusat et al., 2005). IB provides congestion detection, congestion signaling, injection rate reduction and injection rate recovery for congested high performance computing (HPC).

Networks are everywhere and there are different types of networks, but in this thesis, it is subjected to lossless high performance interconnection network in data centers, especially the management of the traffic in such a network during the peak of the traffic and avoiding any kind of possible congestion.

## 1.1 Motivation

The term interconnection network or just network is referred to any infrastructure that allows distributed digital units to communicate (E. Gran, 2014). In this thesis, the term will be referring to the type of network that serves as shared communication infrastructure in a High Performance Computing (HPC) supercomputers or modern data center (Yan, Min, & Awan, 2006; S.-A. Reinemo, Skeie, & Wadekar, 2010). Two requirements which are important for HPC are high throughput and low latency; to achieve lossless interconnection network. Unlikely the lossy network where it is permitted for the forwarding node to drop data if it receives faster than what it can able to forward, lossless network never drops any data during operation. If the forwarding node cannot forward data, it still keeps the data and causes congestion if an action is not taken. It is inefficient and expensive to drop data, and lossless interconnection improves the performance of the network but with a cost.

Lossless Interconnection is achieved by using a link-level flow control mechanism. The forwarding node forwards only after it has ensured that the next node has available buffer space to receive. Until the space is freed nothing will be sent. This procedure makes sure that no data gets dropped, but with a high cost, which is allowing the forwarding node to spread saturation to other nodes, some with plenty of free buffer spaces which otherwise can be used for the delivering of data. The node which is withholding and waiting until the other node gets free buffer space, denies others to forward any data on its way too. The longer this node is occupied, a growing tree of buffer occupancies will block an increasingly larger part of the network. With the use of InfiniBand, it is believed that the blockage will be loosened up and the performance will get improved.

### 1.1.1 Problem statement

The growing interest of environmental and economical efficient industries has its impact also among HPC community. Green computing demand is gradually grown. From last decade, an interest for congestion management in lossless interconnection networks has shown been among both research community and in the industry. In that revolution, Infiniband has become a leading player. According to Top500's last statics from November 2014, InfiniBand leads TOP500 as most used Interconnect technology for High-Performance Computing. InfiniBand has 225 systems in the TOP500, representing 45 percent of the list. Its popularity grow year after year, the number of InfiniBand-based supercomputers increased by 8.7 percent from November 2013 to November 2014 (Top500, 2014) .

Congestion Control (CC) mechanism of Infiniband (IB) is based on a closed-loop feedback control system; where a forwarding node detecting congestion inside the network informs all the nodes injecting traffic contributing the congestion about the situation of the network (Santos, Turner, & Janakiraman, 2003). When a congestion occurs, the node detecting the congestion informs the congestion contributing nodes to slow down its injection rate; until the congestion is removed. As the congestion loosens up and is removed, the source starts injecting in a higher rate until it reaches to its normal injection rate, if no more congestion occurs (E. Gran et al., 2011). The behavior of the IB CC mechanism depends on the values of CC parameters. The given values of the parameters will determine characteristics like how aggressive the congestion detection should be, the rate of feedback from the forwarding node detecting congestion to the contributors of the congestion - and how much and for how long the contributors should lower their injection rates. CC parameters give a great flexibility to the CC IB mechanism (E. Gran, 2014), but there are very few guidelines about how to set the values of CC parameters for it to be efficient. More knowledge is needed on how to use the parameters. It is not also known much about what effect each individual parameter has, on the performance of the network, and relations between the individual parameters.

In this thesis, an experiment of a Mesh network topology will be conducted using OmNet++ (OmNeT++, 2014) as test bed. Hugh amount of traffic will be generated and fed to the network until a congestion is contributed. This project tries to address following research questions:

- How is the performance of the network when IB CC is disabled?
- How is the performance of the network when IB CC is enabled?
- How is the performance of the network when host-to-switch link size is doubled?
- How does IB CC parameters influence performance?
- What is the proper IB CC parameter of this network topology?

## 1.2 Thesis outline

This thesis is divided into six chapters. Most chapters are further divided into sections and subsections. The very first chapter is the Introduction chapter. This chapter contains details about the whole thesis, especially the introduction, the motivation of the thesis and the problem statement. The second chapter is a brief background information of the thesis. The basic characteristics of Congestion, Congestion Control (CC), Infiniband (IB), IB CC, OmNet++ and more, will be described. Chapter 3 consists the approach of the experiment. In it, details of what is intended to do and how it is intended to implement it, is to stated. Chapter 4 is all about the result of the experiments after the experiments have been conducted. Chapter 5 consists an analyse of the result. In chapter 5, there will also be a comparison between the approach and the result. And at last chapter 6, which is the conclusion and future work chapter of the thesis. A list of references and appendices will follow it.

# Chapter 2

# Background

In this chapter, there will be a short introduction to the basic characteristics of network, lossless interconnection network, congestion, congestion management. Infiniband, OmNett++ and other related technologies will also be all briefly explained, in some cases with examples.

## 2.1 Network topology

In communication networks, a topology is a usually schematic description of the arrangement of a network, including its nodes and connecting lines. There are two ways of defining network geometry: the physical topology and the logical (or signal) topology (Rouse, 2014).

The physical topology of a network is the actual geometric layout of workstations, servers, storage nodes and etc. There are several common physical topologies, as described below and as some of them shown in the illustration.

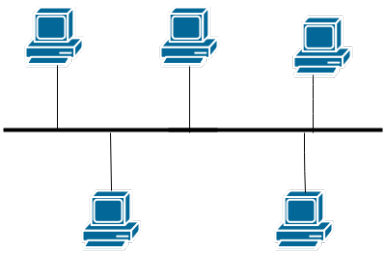


Figure 2.1: Bus network topology

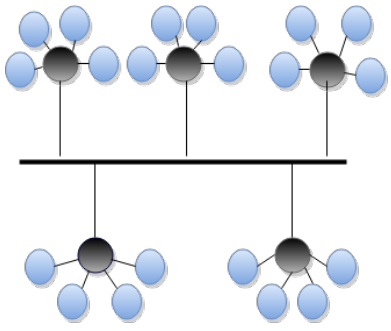


Figure 2.2: Tree network topology

Other well known network topologies are bus network topology, tree network topology, star network topology, ring network topology, star or ring topology and mesh network topology.

## 2.2 Congestion in Lossless Networks

In Lossless Networks, congestion occurs when a link or node is carrying so much data that its quality of service deteriorates. In real life, most of us experience congestion in non-network related occasions everyday especially mornings and afternoons. When most of the people leave their homes to work in the morning, we tend to overload the capacity of the roads and cause congestion (traffic jams). Again, we finish work around the same time, and repeat the same scenario as earlier of that day.

To understand it better, we can use an example of a network topology to describe how a congestion happens. Figure 2.3 shows a network of 4 switches, which is serving for 5 hosts.

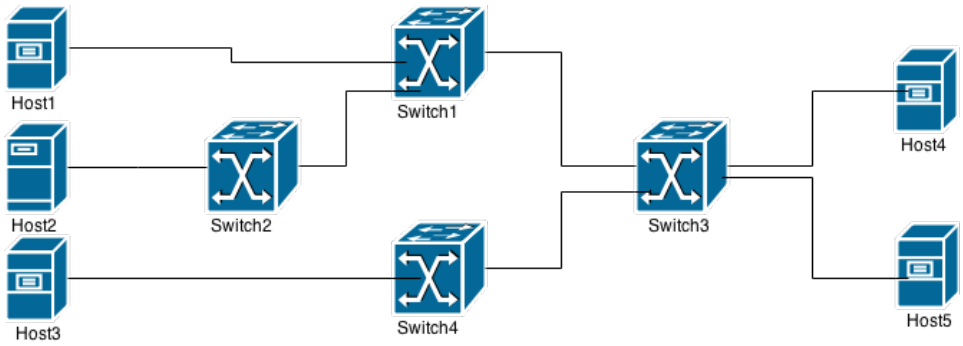



Figure 2.3: A network of four switches and 5 end nodes.

- Switch1 is connected to Host1, Switch2 and Switch3.
- Switch2 to Host2 and Switch1.
- Switch3 to Switch1, Host4, Host5 and Switch4.
- And Switch4 to Host3 and Switch3.

This figure represent normal traffic, flowing as intended without any congestion.

### 2.2.1 Traffic flow between end-nodes

Next figure 2.4 shows the traffic flow of the network topology. This is the traffic flow between the different end-nodes.

- Host1 is sending data to Host4. The traffic is going through Switch1 and Switch3.
- Host3 is also sending traffic to Host4. The traffic is going through Switch4, and Switch3.

- While Host2 is sending to Host5. The traffic is going through Switch2,Switch1 and Switch3.

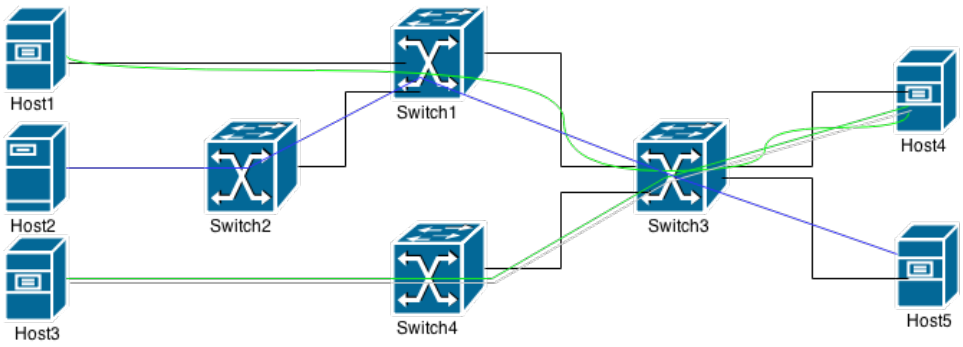



Figure 2.4: Traffic flow between the nodes.

The Green lines represent the traffic going to Host4, while the blue line represent the traffic coming from Host2 and Host5.

### 2.2.2 Congestion occurring

When a node in the network can not manage to serve the traffic it is receiving, the flow control in a lossless network will make buffers in the surrounding nodes fill up as well, then the backpressure effect will continue, until the whole network in congested.

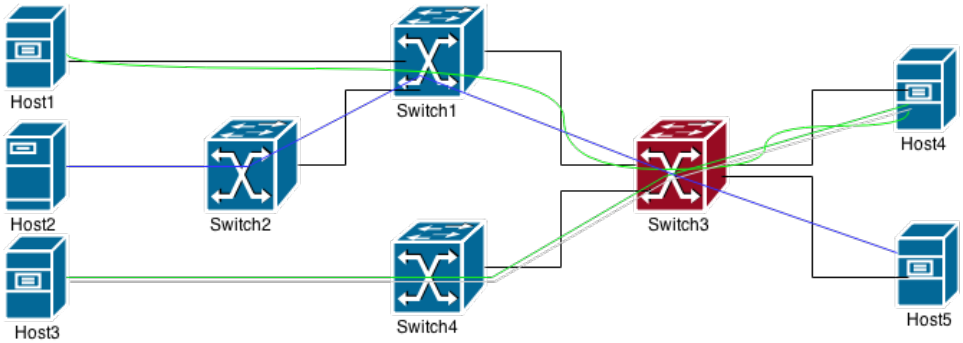



Figure 2.5: A network topology - Congestion tree

In figure 2.5, we can assume that source nodes Host1 and Host3 want to send at the full link capacity of their connected links. The connection between Switch3 and Host4 will then become a bottleneck. In this scenario, each one of the two different sources will get a half of the link. As the sources still continue to send traffic to Host4, traffic will start to pile up at Switch3. Switch3's buffer space will soon fill and the flow control will

notify neighbours Host5, Switch1 and Switch4 that Switch3 is running out of buffer space. The sources will continue to send traffic to the bottleneck link, until no more buffer space is available at Switch3. The source nodes become contributors to congestion.

### 2.2.3 Contributor and Victim

From figure 2.5, it can be seen that the bottleneck exists at the link between Switch3 and Host4, and that the traffic flow coming from Host1 and Host3 contributes the bottleneck. In this scenario, Host1 and Host3 is considered to be the Contributors of the bottleneck while Host2 is the victim. The traffic flow between Host2 and Host5 is not causing any bottleneck and its links until reached Switch3 and between Switch3 and Host5 is empty and can handle more traffic flow. But as a side effect of the bottleneck at the link Switch3-to-Host4, the traffic flow of Host2-to-Host5 is affected. And that is why Host2-to-Host5 is referred as a victim in this example. A traffic from Host1 and Host2 could also cause congestion at switch1.

### 2.2.4 Congestion spreading

Whenever a link cannot handle the traffic passing through in it, a bottleneck gets created there. While the link still struggles to catch up the flow, more and more traffic flow comes, causing it to spread more nodes backward to the sources to the traffic.

Continuing to figure 2.6, it shows when the the link between Switch3 and Host4 got congested, the traffic flow piled up in Switch3 until no more buffer space was available, then the congestion tree grew more backward to Switch1 and Switch4 and then finally to Switch2. Switch2 gets little traffic from Host2 but as it cannot forward any traffic because of Switch1 and Switch3 the little traffic pills up there.

Now all the forwarding nodes in the network is congested. The congestion tree is spread from Switch3 towards Host1, Host2 and Host3 as the spread is one direction.

This is one scenario of how a single bottleneck in one link between two nodes can cause a full congestion in a network. This topology has few forwarding nodes and several hosts but it can represent a network topology in a larger scale.

## 2.3 Congestion management

Congestion management or congestion control concerns controlling traffic entry into a network, and tries to avoid congestive collapse, by attempting

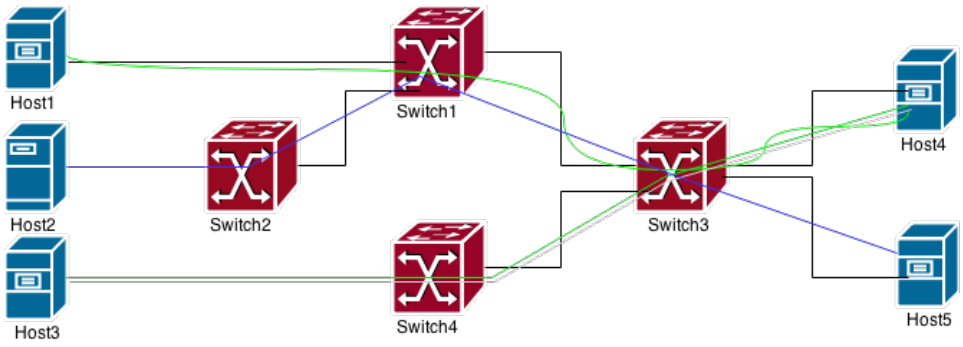



Figure 2.6: A network topology - Congestion spreading to other nodes.

to avoid any thing that will cause oversubscription of any processing or link capacities. The node, which is trying to take more than its share of the resources, gets resource reducing steps such as reducing the rate of sending data (Amol & Rajesh, 2014).

A possible congestion solution will be to increase network resources more than strictly necessary or overprovisioning (Lugones, Franco, & Luque, 2009). Which will mean that by overprovisioning the resources, no congestion will probably ever happen. Competition for resources and contention will be avoided, and the network will function as intended. By overprovisioning, more resources than necessary will be needed. That will be fine for a small interconnection network, but for a large network in High Performance Computing systems, the power consumption and components for the network will cost too much, as well as environmental effect, space and so on. Overprovisioning is not a suitable alternative for HPC systems.

Like overprovisioning, there are other proactive techniques, but they are not appropriate for general HPC systems or modern data centers becuase they do not conduct congestion management in a universal and scalable way (E. Gran, 2014).

According to (Lugones et al., 2009) adaptive congestion control mechanisms typically perform three basic tasks, which are to monitor Network traffic, to detect Congestion and to control or reduce congestion.

Adaptive routing and load balancing techniques (E. Gran, 2014) helps to alleviate congestion by distributing the traffic in the network in a favorable way. The adaptive routing and load balancing techniques do the path selection depending on network status, in that way congestion trees will be removed or made less likely to grow. But its success is not granted every time depending on where the congestion happens. If the root of the congestion tree is located at the last downstream switch prior to a destination,

an attempt to remove the congestion by adaptive routing or load balancing will not remove the congestion tree, it could even make the situation worse. (E. Gran, 2014)

## 2.4 Infiniband

InfiniBand is an industry-standard specification that defines an input/output architecture used to interconnect servers, communication infrastructure equipment, storage and embedded systems (Valentini et al., 2009; Grun, 2010). InfiniBand is a true fabric architecture that leverages switched, point-to-point channels with data transfers today at up to 120 gigabits per second, both in chassis backplane applications as well as through external copper and optical fiber connections (IBTA, 2015).

According to (IBTA, 2015) InfiniBand's advantages are that it has;

- Superior performance
- Low-latency
- High-efficiency
- Cost effectiveness
- Fabric consolidation and low energy usage
- Reliable and stable connections
- Data integrity
- Rich and growing ecosystem
- Highly interoperable environment

### 2.4.1 The CC concept in Infiniband

E. G. Gran and Reinemo, 2011 explained beautifully in their paper what CC concept in Infiniband is; *"The InfiniBand Congestion Control (IB CC) mechanism is based on a closed loop feedback control system where a switch detecting congestion marks packets contributing to the congestion by setting a specific bit in the packet header, the Forward Explicit Congestion Notification (FECN) bit. Then the congestion notification is carried through to the destination by this bit. The destination registers the FECN bit, and returns a packet with the Backward Explicit Congestion Notification (BECN) bit set to the source. The source then temporarily reduces the injection rate to resolve congestion."*

The exact behaviour of the IB CC mechanism depends upon the values of a set of CC parameters governed by a Congestion Control Manager. These parameters determine characteristics like when switches detect

congestion, at what rate the switches will notify destination nodes using the FECN bit, and how much and for how long a source node contributing to congestion will reduce its injection rate (E. Gran et al., 2011). These parameters should enable the network to resolve congestion by avoiding head-of-line blocking, while still utilizing the network resources efficiently.

#### 2.4.1.1 IB CC at a Switch

The switches are the ones that are responsible to detect congestion and notify the destination node using FECN bit. It is the switch that detects congestion on a port or a Virtual Lane depending on a `Threshold` parameter. If the threshold is crossed, it will lead to FECN marking of packets (E. Gran, 2014). The switch's parameters are `Threshold`, `Victim_Mask`, `Marking_Rate`, `Packet_Size`.

`Threshold` has a weight ranging from 0 to 15 in value. And is the same for all port, if it is not specified to each different port. The value zero indicates that no packets should be marked. From 1 to 15 represent a decreasing value of threshold. A value of 1 indicates a high threshold with high possibility of congestion spreading, while 15 is its opposite, it indicates low threshold with a corresponding low possibility of congestion spreading. The first one (a value of 1) risks to move the congestion state (packets eligible for FECN marking) too late. And the later one to move into the congestion state even when the switch is not congested (E. Gran et al., 2012). It is the designer of the switch architecture that is left for the exact implementation of the threshold.

When a port is in the congestion state, its packets are eligible for FECN marking. A packet will then get the FECN bit set depending on two CC parameters at the switch, which are `Marking_Rate` and `Packet_Size`. Packets with a size smaller than the `Packet_Size` will not get FECN bit set. It is `Marking_Rate`, which sets the mean number of eligible packets sent between packets actually being marked. (E. Gran et al., 2010)

| Switch |
|---|
| `Threshold` |
| `Victim_Mask` |
| `Marking_Rate` |
| `Packet_Size` |

#### 2.4.1.2 IB CC at a Channel Adapter

When a Channel Adapter (CA) receives a packet with a FECN bit set from the switch, the CA as soon as possible seeks to notify the corresponding source about the congestion. To do so, the method it uses, is to return a packet with the BECN bit set back to the source of the packet (E. Gran et al., 2011). CA uses a Congestion Control Table (CCT) and a set of CC parameters. The Channel Adapter's Congestion Control parameters are

CCTI_Increase, CCTI_Limit, CCTI_Min, and CCTI_Timer.

| Channel Adapter | Description |
|---|---|
| CCTI_Increase | When traffic injection is to resumed |
| CCTI_Limit | The upper limit of the CCTI |
| CCTI_Min | The minimum limit of the CCTI |
| CCTI_Timer | The time traffic injection is to be halted |

## 2.5 OmNet++

While answering what the OmNet++ is, the official website of OmNet++ (OmNeT++, 2014) stated that " OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queueing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools. There are extensions for real-time simulation, network emulation, database integration, SystemC integration, and several other functions."

It provides a component architecture for models (OmNeT++, 2014). Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

An InfiniBand model has been implemented using the OmNet++ framework. The model consists a set of modules to simulate an InfiniBand network with the support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear forwarding tables (E. Gran et al., 2010).

# Chapter 3

# Approach

This chapter explains how the experiment is going to be implemented. In it, the network topology design, the test bed, and the outcome assumption are all going to be discussed here. Later in the analysis chapter, the content of this chapter and the content of the upcoming result chapter will be compared and analysed. What went as assumed and what have not, will be stated. This chapter is very important as it is defining the whole project. What, why and how to do, is all mentioned here.

## 3.1 The Test Bed

To conduct a simulation of any network topology, either a complete hardware or a software simulator is needed. Getting a whole sett of hardware for an entire network topology in any size can be very expensive and time consuming. That resources are not available for this project. For the software solution, all it is needed, is to get access a good simulation tool. Many of them are available in the marked, most of them have their strengths and weaknesses. The chosen simulator of this project is OmNet++. In earlier projects, OmNet++ has proven to be the right simulation tool. It is an open source software and an InfiniBand (IB) model has been created for it. All the simulation of this project, OmNet++ with its IB model is going to be used.

## 3.2 Network topology

A network topology different from earlier tested topologies, is going to be designed. The topology must not be a very large one because of the thesis time frame and it has to also be large enough to cause a congestion. Many topologies have been designed and tested, either using hardware or software simulation environment. Yet there are too many untested topologies that have to be tested. It has not been published the behaviour of many tested Mesh network topology. The topology in this experiment is going to be a mesh or partial mesh network. It will be different from those earlier

tested and documented.

### 3.2.1 Topology design

First, it has been proposed a topology of four hosts and four switches, each host for one switch and that every switch connected to two other switches. After some testing and discusses, a one more host has been added to the network topology, in order to generate more traffic flow. It is very important in this project to able to inject a huge traffic in the network, so that a congestion can be contributed.

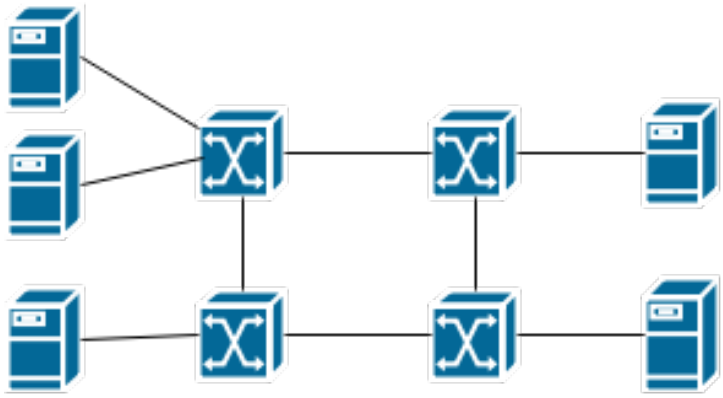


Figure 3.1: Proposal of the network topology

Figure 3.1, shows the proposed 2x2 network topology of the project. Three of the hosts are connecting to each its own unshared switch. Switch one (top left) has two hosts connecting to it. The link between switches and hosts have a 20 Gbit/s capacity. While switch to switch links have 40 Gbit/s capacity.

For this topology, Hosts have names which start with letter H accompanied by a number from 1 to 5, while switches have names which start from letters SW and accompanied by a number from 1 to 4. H1 and H5 are linked to SW1, H2 to SW2, H3 to SW3 and H4 to SW4. All the switches have three active ports except SW1 which has four ports.

For the sake of simplicity, the connection between each host and its closest switch goes through port 1, except H5 which is exceptional. H5-SW1's connection goes through port three because SW1 is the primary switch for two hosts unlikely to the other three switches. The table below shows the whole switch-to-host port connections.

| Switch | H1 | H2 | H3 | 4H | H5 |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 0 | 0 | 3 |
| 2 | 2 | 1 | 0 | 0 | 2 |
| 3 | 0 | 0 | 1 | 2 | 0 |
| 4 | 0 | 0 | 2 | 1 | 0 |

### 3.2.2 Traffic flow

H1 starts to send traffic to H4 right after the simulation is started. The traffic goes through SW1 and SW2 before it gets to SW4 and finally arrives to H4. One second later H2 starts to send traffic to H4 through SW2 and SW4. The two flows do not meet before they reach switch SW4. Two seconds after the simulation started, the third host (H3) starts to send too to the same destination as the other two nodes. Finally and three seconds after the simulation started, H5 starts to send traffic to another destination. Instead of the other three sources, which all send to H4, H5 sends traffic to H3. The traffic from source H5 to destination H3, goes through SW1 and SW2. H4 never sends any traffic during the simulation is running

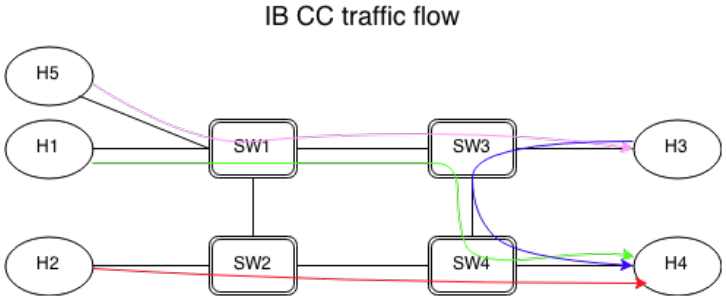



Figure 3.2: InfiniBand Congestion Control - traffic flow directions

| Source name | Switch | Switch | Switch | Destination name |
|---|---|---|---|---|
| H1 | SW1 | SW3 | SW4 | H4 |
| H2 | SW2 | - | SW4 | H4 |
| H3 | SW3 | - | SW4 | H4 |
| H4 | - | - | - | - |
| H5 | SW1 | - | SW2 | H3 |

The table below shows when the hosts start sending traffic flow to their destinations.

| Host name | Start time (in second) |
|---|---|
| H1 | 0 |
| H2 | 1 |
| H3 | 2 |
| H5 | 3 |

## 3.3 InfiniBand Congestion Control OFF

As stated earlier in the introduction chapter, the experiment will first conducted when the Congestion Control is off. Its performance will be analysed, and a plan to improve the performance of the topology will be worked out. Later, the Congestion Control will be turned on, and when the right parameters are discovered, and the final simulation is conducted, the two will be compared. Then the impact of the InfiniBand Congestion Control (IB CC) of the network topology will be presented in the analysis chapter of this thesis.

In the later sections, there will be a further explanations on how to get the proper IB CC parameters of this network. Many experiments will be done and compared before the best suited parameters is gained.

### 3.3.1 Expected performance when IB CC is disabled

Figur 3.3 shows the expected result of the experiment when IB CC is off. This analysis is based on the assumption that all switches are fair.

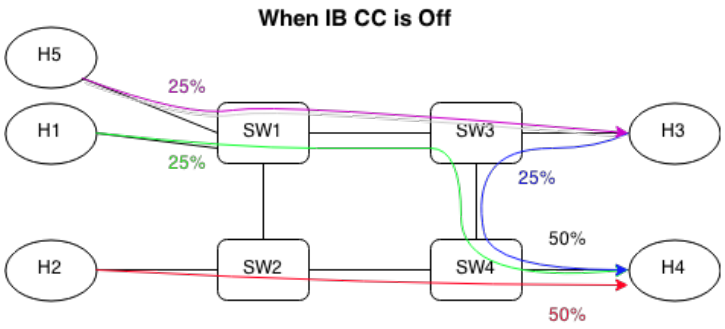



Figure 3.3: Congestion Control Off - flow share in per centage

It is believed that H1 will use almost the maximum link capacity if it can generate all that traffic. The links between switches have 40 GBit/s capacity and the links between hosts have 20 GBit/s. The maximum link capacity H1 can use, is up to 20 GBit/s, but it is improbable that a host can manage to use all the link capacity that are available for it. H1 will start high and continue to use most of the link capacity from zero second to the first second. Then H2 will start to send traffic to destination H4. H2 will try to use its link capacity which is as same as H1. The two traffics do not share any links until reached SW4. The link between SW4 and H4 has a 20 GBit/s capacity and can not handle both traffic flows which combined can be as high 30s GBit/s, because of the configuration and other factors also, it is unlikely to benefit the whole 40 GBit/s capacity.

The link between SW4 and H4 get congested, and the traffic flow is reduced. SW4´s closest switches SW3 and SW2 start sharing the link capacity that is available for them. Each of the switches will send in half of what it was sending. One second later, H3 starts to send to H4. H3 have to share with H1 links from SW3. We know that SW3 is allowed to send in half. Now it has traffic flows coming from two sources, and going to the same destination. SW3 can only use its quota, its two closest forwarding nodes will not be able to send faster due to the link level flow control and congestion. Now, H2 is sending in half while H1 and H3 are sending in each a quarter. After that, H5 starts to send to destination H3. Though H5´s destination is different then the other sources´ destination and have nothing to do with the SW4-H4 link, yet it has to suffer for it. H1, H2 and H3 are contributing the congestion while H5 becomes a victim of the other sources. It happen because H5 shares a switch and a link with H1 (SW1). They are given fair share of the capacity (reduced by congestion) 25% each. They have a sending opportunity window in a quarter of the time, in which SW1 is allowed to send, and another quarter for H3. Yet, we think that both H1 and H5 will manage each of them to send in a quarter because when H5 sends, there is no waiting and the traffic passes through SW3 to H3, so before the window closes H1 sends a quarter and H5 sends a quarter. That is possible because there is nothing hindering H5´s traffic flow after passed SW2.

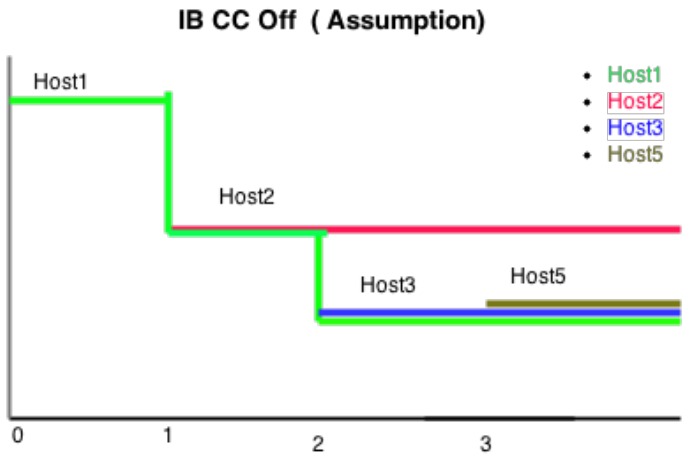



Figure 3.4: Congestion Control Off

In graphics, graph 3.4 shows how H1´s performance degraded from top to half as H2 started to send its traffic. H2 starts in half the level of H1´s performance just before H2 started to send. Then H3 started to send, this time H2´s performance stayed the same as earlier while H1´s performance degraded even more to a quarter of what it started in the beginning. Finally H5 begins to send traffic, H1, H2 and H3 continue the same way and H5 starts sending equal as H1 and H3. There is unfairness between the con-

tributors. H2 gets more bandwidth than its fellow contributors whom each gets half of what H2 gets.

On the other side, the performance of the receivers is expected to seem more like figure 3.5. H3 and H4 are the only receivers of this topology. H4 is the one that start receiving traffic from the start of the simulation. H3 start getting traffic from the last second. H4 receives all the traffic that H1 can generate from the start, then as more nodes try to send more traffic than what H4's link can handle, SW4 start controlling what other forwarding nodes sending. Later H3 start getting traffic from H1 and not much traffic is getting to H3 as most of the traffic is held back/slowed down at SW1.

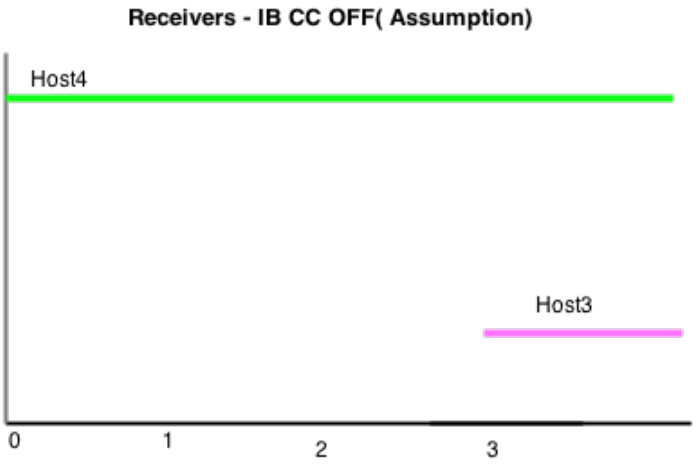



Figure 3.5: The receivers of the traffic flows when CC is disabled

## 3.4 InfiniBand Congestion Control ON

After the first simulation is implemented, in which the CC was off. The CC will enabled. Using CC, network´s performance will be increased as a whole and especially, the victim will get much better performance. When using IB CC, it is the given that the values of the CC parameters will make the difference. These parameters determine characteristics like when the switch detects congestion, and at what rate the switches notify destination nodes using Forward Explicit Congestion Notification (FECN) bit, and how much and for how long a source node contributing to congestion will reduce its injection rate. These parameters should enable the network to resolve congestion and to avoid blocking, while still utilizing the network resources at the root of the congestion tree efficiently.

### 3.4.1 IB CC parameters

For now, the values of the parameters of this network are not known. During the simulations, many experiments will be conducted to determine which parameter values that are best suited for this network. On the switch´s side, the value of the "Threshold" is an important one. The Threshold is represented by a weight ranging from 0 to 15 in value. It is same for all ports. On the Channel Adapter´s parameters, it is the `CCTI_Timer` which is chosen in this case. Channel Adapter relies on the `CCTI_Timer` to increase the injection rate again after it received a BECN bit. In some scenarios, the `Packet_Size` will be changed to see its effect in action.

### 3.4.2 Expected performance when IB CC is enabled

It is expected that the traffic flow will be the same as the previous experiment (IB CC Off). It will start close to maximum of its link capacity. After it run one second, the second host (H2) will start sending traffic and as they have to share the SW4-H4 link capacity which is just 20 GBit/s, a saturation will be risked. When SW4 detects that its buffer space is getting filled up, it will mark packets (marking-rate) going to H4 using Forward Explicit Congestion Notification (FECN) bit. The destination (H4) then notifies the sources (H1 and H2) of the packets about the congestion. That can be done by Backward Explicit Congestion Notification (BECN) bit set back to the sources. For now, the sources have been notified about the congestion and they reduce their sending according to their parameters.

With the right parameters, it is expected that the two forwarding nodes will share the resources evenly. Then the third host (H3) starts to send to destination H4. Now, as two nodes were sharing the resources, the third one will start to share with them too and if the parameters is set right, all of them will use a third of the resources that are available for them. This is unlike expectation from the previous experiment, where H2 maintained to use half of the resources while the other two competed for the other half, each getting one fourth of the total SW4-H4 link capacity.

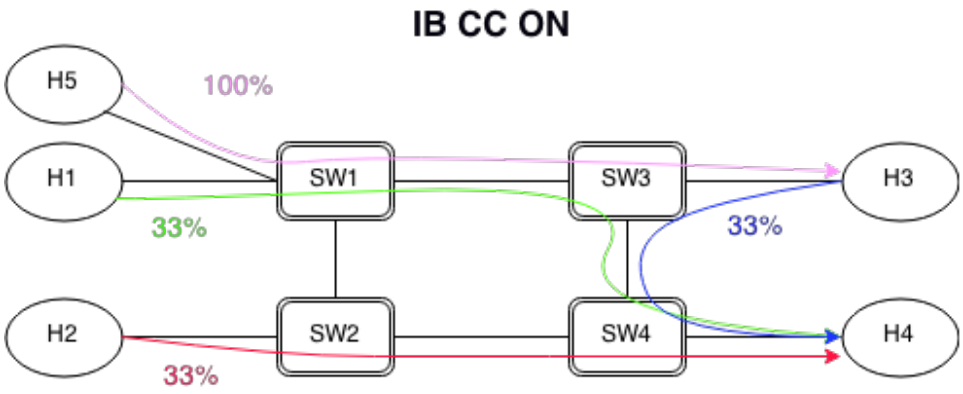



Figure 3.6: An assumption of performance when the Congestion Control enabled - flow share in per centage

The last node (H5) will start sending traffic to H3. From previous experiment, we remember that H5-H3's connection was the victim. As, it is expected that the others got a fairer share of the resources, it is our goal that the victim gets to use more its unused resources. H5 shares H1 with switch SW1 and the link from that switch to SW2. The link has a 40 GBit/s capacity, in which both of them can share comfortably. Now, with the right parameters, the victim will not stay a victim any more and will use up to its maximum link capacity. It is expected that with the help of the enabled IB CC, that other will not be a hinder for H5 to send traffic.

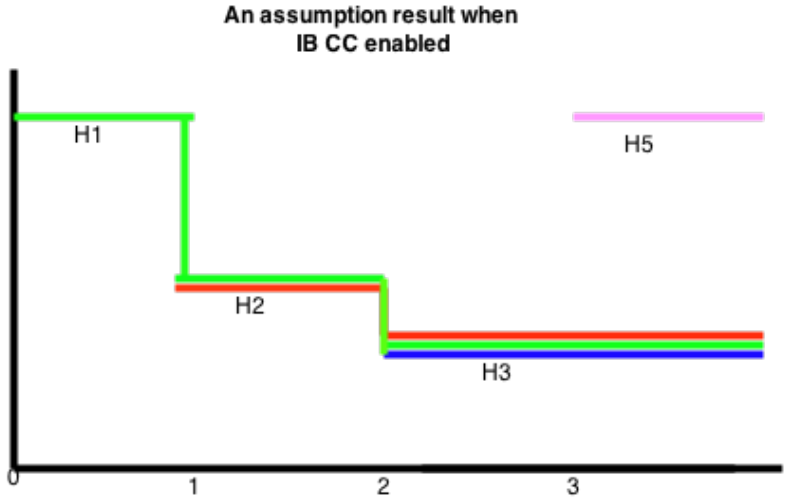



Figure 3.7: Congestion Control ON

Figure 3.7, visualizes the final outcome result of the experiment when the IB CC is enabled. In the analysis chapter, we shall compare and analysis

figure 3.7 and the real result of the simulations, when the IB CC is enabled.

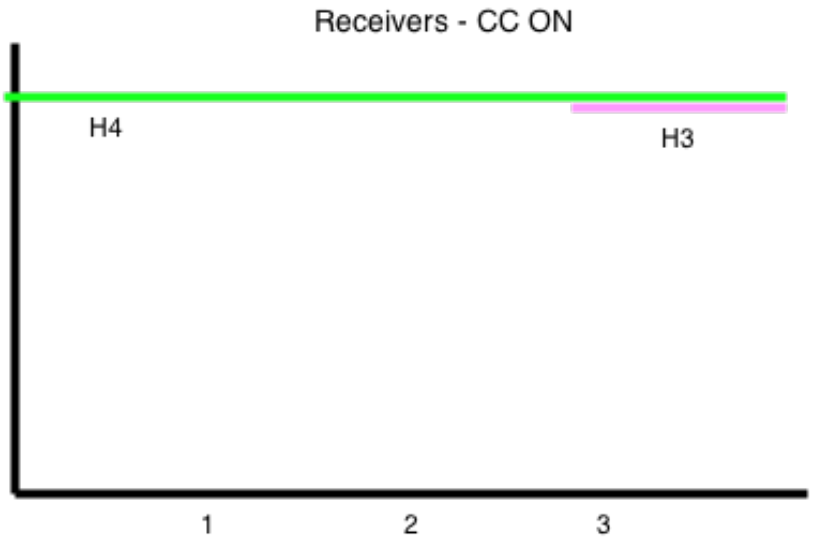



Figure 3.8: The receivers performance when CC enabled

When it comes to, how the receivers are expected to perform while receiving traffic from the sources. It is expected that H4 will maintain to receive all the traffic that it can handle. As well as that H3 is also expected to receive all what H5 can generate. One of the goals of this project is that all sender and receivers to be able to use all their available capacities if they are not contributing any congestion.

## 3.5 Bigger host-to-switch link

The capacity of the links between hosts and switches are increased from 20 GBit/s to 40 GBit/s. Our goal for this part is to learn more about the behaviour of the network if it is equipped with links with bigger capacities. Already the link between switches have a 40 GBit/s capacity. And now, as the rest of the links are replaced, the whole network has links with the same size.

From the last two experiments (when the CC is disabled/enabled), the biggest challenge was that the receiver's link to SW4 (switch) had a smaller capacity and could not handle all traffic going through in it when more than one sender is sending. For now, the receiver has bigger capacity but also the senders have too. What we want to understand is, how much improvement will the new links come with?

From previous experiments, it was the link between H4 and SW4 which could not handle the traffic it was receiving. In this experiment is it be-

lieved that if the sources manage to use all capacities that are available for them, then they can produce up to double of what the source could produce earlier. That will mean, it will not be the link between H4 and SW4 whom will struggle to make the throughput, but also switch-to-switch links will not manage to transfer all traffic. All links have same size capacities, so when the source try to send in full speed, switch-to-switch link gets congested. The later scenario can not be realized in this topology because as H2 injects traffic toward H4, the same as previous experiments will happen. The link-level flow control messages from SW4 to SW2 and SW3 will prevent SW2 and SW3 from sending towards SW4 at full capacity, and then later, when SW2 and SW3 runs out of buffer space, SW2 and SW3 will similarly send link-level flow control messages to their neighbours.

## 3.6 In search of proper IB CC parameters

Finding the proper IB CC parameters for this topology, is the most time consuming and hardest for the whole project. There are no guidelines or formulas of how to get the right parameters. It is known what these different parameters do and there are clear description of them. But The parameters do not have the same effect for every given network topology (no same values for all). Network topology designers must experiment different values of the IB CC parameters and check how the network reacts.

In the background chapter, it has been described the Channel Adapter and Switch CC parameters. Here in this chapter, no further details of the CC parameters will be given. But we will choose some parameters from Channel Adapter and from the Switch. Those chosen ones will get new and different values for every simulations, the outcome of the simulations will be plotted in graphs. The graphs get studied and analysed, in that way, we have a chance to detect the proper IB CC parameter values of this topology.

Probably, dozens if not hundreds of simulations is been expected to be done before the most proper IB CC parameters is gained.

| Parameter | Value |
|---|---|
| `CCTI_Increase` | |
| `CCTI_Limit` | |
| `CCTI_Min` | |
| `CCTI_Timer` | |

| Parameter | Value |
|---|---|
| Threshold | |
| `Marking_Rate` | |
| `Packet_Size` | |

From the Channel Adapter, it is the `CCTI_Timer` which will be tested with different values for many times. And from the switch, Threshold is the one that gets different values. Finally, switch's `Packet_Size` will also be modified in some experiments.

The search for proper IB CC parameters for this topology will not stop until the performance of the network is optimized or proofed that it is not possible to do that while following the approach of this thesis. The goal is to get a performance equal or better the expectations stated above.

## 3.7 How IB CC parameters influence performance

IB CC parameters are difficult to get them right while designing a network topology. The goal of this section is get more insight about how IB CC parameters work and influence performance especially these parameters; `CCTI_Timer`, `Packet_Size` and `Threshold`.

This section gives more insight about how IB CC influence performance. Both the `CCTI_Timer` and the `Threshold` will get values in given intervals between. One of the parameters will remain the same while the other one gets increased or decreased. Whatever the outcome is get presented in the result chapter. This process is going to be repeated many times until sufficient data is gathered. In a fewer times compared to Threshold and `CCTI_Timer`, `Packet_Size's` values gets increased and decreased. Because the later one is easier to understand how it influence the performance.

| Threshold/`CCTI_Timer` | 20 | 40 | 80 | 120 |
|---|---|---|---|---|
| 2 | √ | √ | √ | √ |
| 6 | √ | √ | √ | √ |
| 8 | √ | √ | √ | √ |
| 10 | √ | √ | √ | √ |

The tables lists values of the parameters that we are expecting to experiment. The row on the left contains the values of the `Threshold` and the top column lists the values of the `CCTI_Timer`. A total of 16 simulation will be conducted. And for each of these 16 simulations, a graph containing the performance of the sender and another one for the performance of the receiver will be plotted.

Couple of more experiment will be ran after the value of the `Packet_Size` is changed. `CCTI_Timer` and the `Threshold` will remain the same as one of the simulation from the above table.

## 3.8 Collecting data

A Python script will be developed for extracting desired data and then for plotting graphs. After running a simulation using OmNet++, one ends up a text file containing various information. It will be very tricky to get the desired rows and columns. One will need much times to go through the

output file and to study it well before understanding it. The output file for a topology like this is expected to be around two gigabytes. The size of the file heavily depends on log level, simulation time and parameter values.

Python is chosen as the scripting language for this thesis, because it is the scripting language that I have been using last semesters. Python is very powerful programming and scripting language. It is well suited for the purposes for this kind of assignments.

First, the text output file from OmNet++ is opened, a new file is created and by using regular expressions the rows that are interested, get extracted and saved in the newly created file. Finally, the process of plotting the graph from the content of the file starts. For now, the very first graph for this thesis is expected to get plotted.

## 3.9 Summary of the approach

In this chapter, it was been discussed and decided the most important steps of this project. And it is in this chapter that the authors expectation / assumption of the final result is presented. The answers of all these questions have been answered;

- How will the topology look like?
- How will traffic flow?
- How will data be collected?
- What will be the simulator of the simulations?
- What performance is expected when IB CC is disabled
- What performance is expected when IB CC is enabled

In the same chapter, it has also been stated and explained in details the project's assignments (to do list). After the simulations are done, it is expected that these five different points should be answered;

- How is the performance of the network when IB CC is disabled?
- How is the performance of the network when IB CC is enabled?
- How is the performance of the network when host-to-switch link size is doubled?
- How does IB CC parameters influence performance?
- What is the proper IB CC parameter of this network topology?

# Chapter 4

# Results

In this chapter, the results of all assignments are to be presented. All the simulations have been done by now and all the outputs are ready. There will be some explanation about how the simulations went and what changes that have been made while the implementation of the simulations were taking place. This chapter is divided into five main section, each section contains the answer of one problem statement. For each problem statement, the performance of both senders and receivers are measured and plotted into graphs. The first graph shows the performance of the senders and the second one shows the performance of the receivers. As the network is a lossless, packet dropping is not allowed. The performance of the receivers are going to be studied and later compared to the senders just to make sure that the results are genuine.

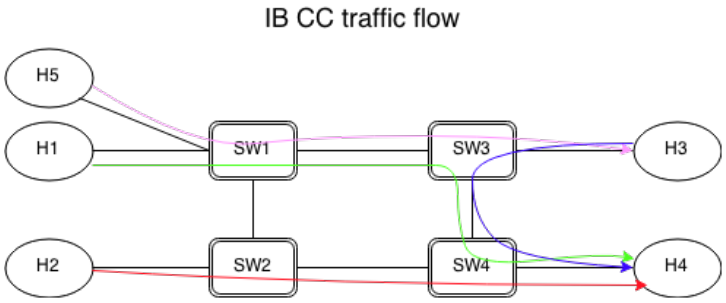



Figure 4.1: InfiniBand Congestion Control - traffic flow directions

## 4.1 IB CC is disabled

The InfiniBand Congestion Control was disabled for the first simulation that was ran. The simulator OmNet++ spent about one and half hour to finish the simulation. A laptop with four cores and with 8 GB of RAM is used for all the simulations. Four of the five hosts send traffic according to the traffic flow design of the network topology. H4 is the only one that is not generating any traffic.

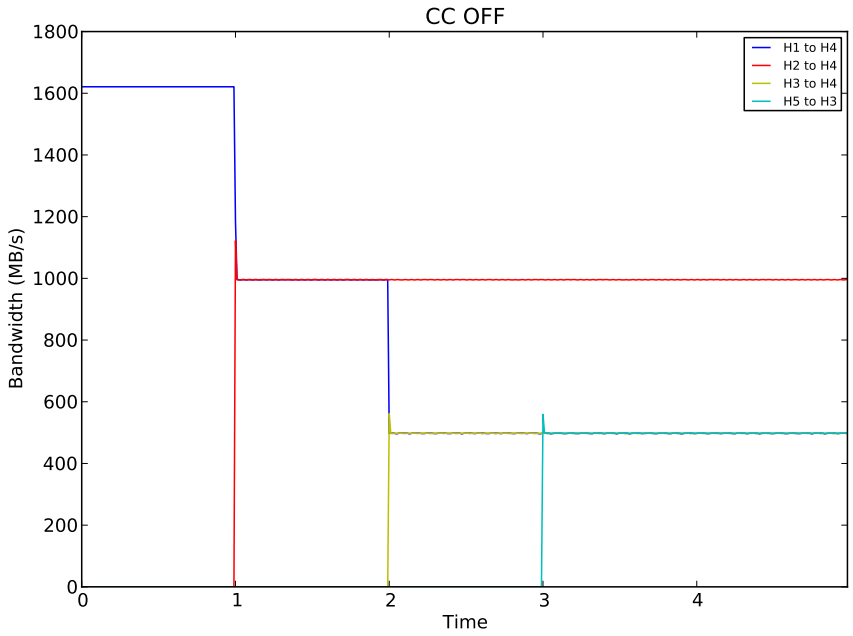



Figure 4.2: Congestion Control OFF with 20 GBit/s link bandwidth

The graph 4.2, shows the result of a simulation in which IB CC is disabled. H1 is represented by the blue colour, starts from the beginning of the simulation at about 1620 MBps and then continued approximately the same level for the first second. From there H2 represented by the red colour started. As H2 started, H1´s performance deteriorated down to about 1000 MBps , while H2 started in the very first time in a higher rate which resulted a congestion but soon after that both of them leveled at the same level. Both of them continued to perform the same way until H3 started to send traffic. From there H1´s performance turned down to about 500 MBps as well as H3 which started just above that number but very soon leveled the same level as H1. Meanwhile H2 kept the same level as it continued before H3 was activated because of the unfair sharing. Finally, the fourth and last host (H5) started to send. As we know from the previous chapter, H5 is the victim node while the other three nodes are the ones that are contributing the congestion. It sends traffic to H3 through SW1 and SW3, while the congestion tree started from the link between SW4 and H4. Yet the victim can only send about 500 MBps equal to H1 and H3.

When it comes to the receivers of the traffic, according to the topology design only H4 and later H3 will receive traffic. If more nodes receiver traffic, it will be a sign of error. In order to make sure that everything is working as it is supposed, it is plotted a graph from receivers´ actives. Graph 4.3 show that from the start, H4 received traffic at a level of above 1600 MBps and a second later, the traffic injection increased to 2000 MBps . Since after that H4 continued to keep on at that level. Three minutes after the simulation is started, H3 began to receive traffic at a level of about 500 MBps .

And it continued to receive in that level after that time.

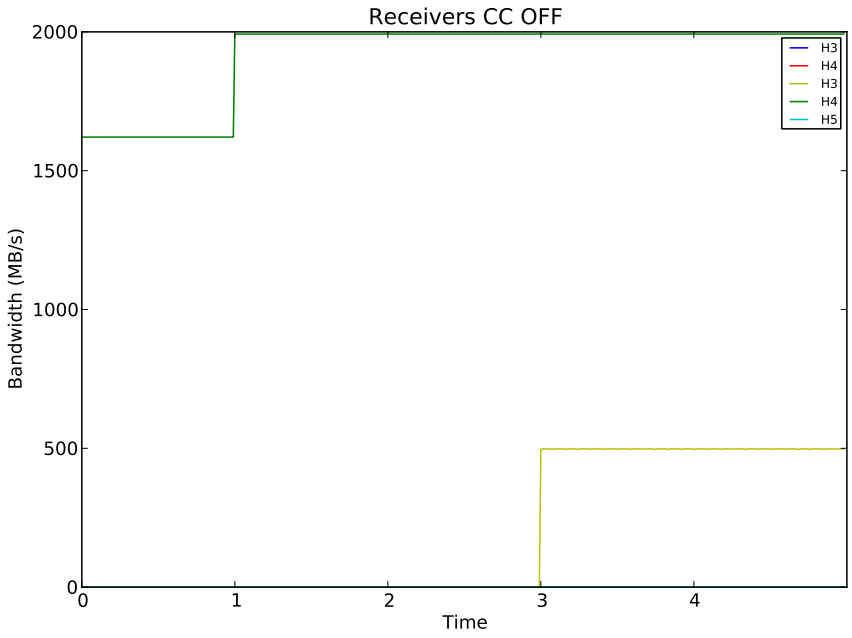



Figure 4.3: Receivers - CC OFF with 20 GBit/s link bandwidth

What this graph is proofing, is that the simulation setup is correct and everything is working is its supposed. It is proofing that all the traffic injections from the senders are actually coming to the right receivers. When it comes to why H1 is not using all of the link capacity that is available for it in the first second, the answer is that, a single nodes can generate traffic up to about 1620 MBps , and a receiver can handle upto 2000 MBps when the host-to-switch link capacity is 20 GBit/s. As the second node start its traffic injection, each of the two nodes send around 1000 MBps which is the receiver‘s full link capacity. Later, H2 continues keep sending in half of the link´s full capacity while H1 and H3 are sharing the other half of the link´s capacity.

## 4.2 IB CC is enabled

In this simulation, the IB CC is to be enabled. For weeks, we have been working for getting the most suited IB CC parameters for this topology. It looks like that we finally managed to do that. The aim for this section is to gain the best possible performance both for the victim node and other nodes too. All nodes will share the resource in a fair way. Only those who are contributing congestion will get their traffic flow reduced. From previous experiment in which the IB CC was disabled, H4 was receiving traffic injections from all of its senders, but their link sharing was not fair for all of them. Instead for each getting a third, one was getting half and the other

two were sharing the other half between them. On the other hand, H1 was using only a fourth of the resources that was available for it. This experiment will show that the victim node is not a victim anymore and that the other three nodes share H4´s resource in a fair way.

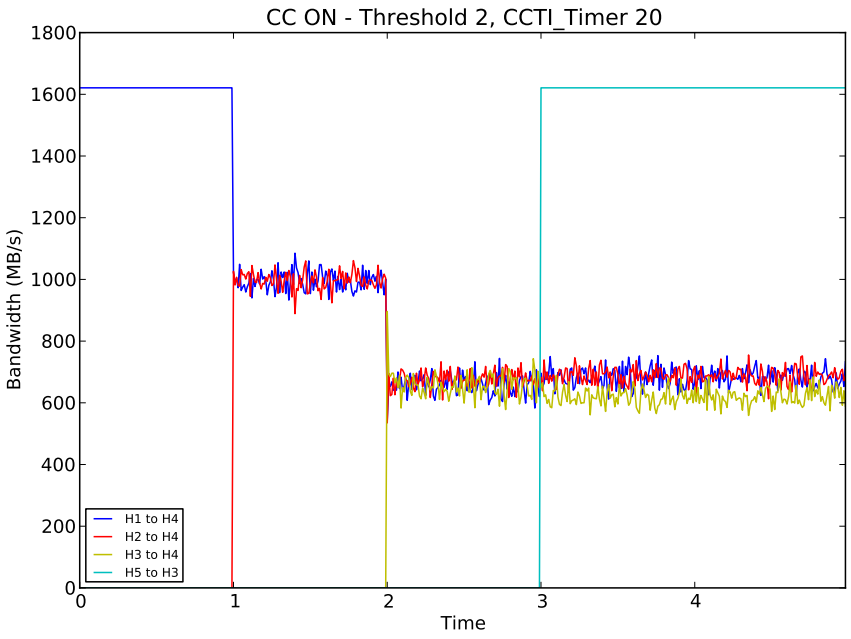



Figure 4.4: Sender - CC ON with 20 GBit/s link bandwidth

Graph 4.4 visualizes how the performance of the network topology improved. In the first two seconds, the performance is the same as the previous experiment. But when the third host (H3) starts to send traffic, something new happens. All of the three nodes gain an equal share of the resources. Each of them a third of the H4-SW4 link capacity. And finally, the last node started to send. From the last experiment, it was the victim node. Now, it is sending in its full capacity. H1 can generate up to 1620 MBps of traffic and H3 is receiving all of it.

On the receivers side, graph 4.5 shows that all the packets sent to H4 and H3, came to right place. H4 receives traffic of 1620 MBps for the first second. That is all the traffic H1 can generate. From there, H4 continued to receive in its full link capacity of 2000 MBps . Later, H3 receives traffic of 1620 MBps and continues in that way. No noticeable changes after that.

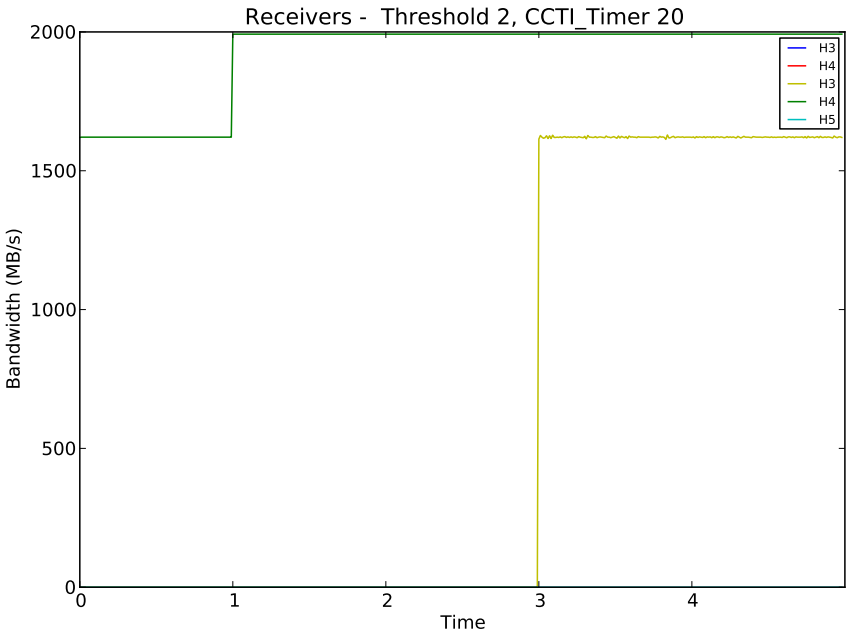



Figure 4.5: Receiver - CC ON with 20 GBit/s link bandwidth

## 4.3 In search of proper IB CC parameters

As, we know, it is needed to get the right IB CC parameters, in order IB CC to be efficient. Without the proper IB CC parameters, the IB CC will not matter much. We have gone through a long process for finding the proper parameters especially `CCTI_Timer` and `Threshold`. Their values have been changed, simulation after simulation. Each time the values got increased or decrease. For each simulation, we learnt more about how to balance them. And finally these values which can be seen below, are accepted as the most proper parameter values for this network topology. With this, the victim node is not a victim anymore and when it comes to fair throughput, all the other three nodes are sharing the link capacity (H4-SW4 link) in a fair way. After all, every node is performing in its best possible way.

```
The IB CC parameters of this experiment.

**.Threshold = 2
**.Marking_Rate = 1
**.Packet_Size = 1
**.CCTI_Timer = 20
**.CCTI_Limit = 128
**.CCTI_Min = 0
**.CCTI_Increase = 1
```

## 4.4 Bigger host-to-switch link

After, we have implemented two simulations where the link between hosts and switches had a capacity of 20 GBit/s, it is decided to conduct an experiment, in which the link capacity between the hosts and switches get increased into 40 GBit/s. The switch to switch link will stay the same as before (40 GBit/s). First, the experiment is done without IB CC then later with IB CC. Both the performance of the senders and the receivers will be measured. These simulations are identical to those previously conducted in the last two sections except the link size.

According to graph 4.6, H1 starts to inject traffic at a level of 2670 MBps , then H2 starts at a level of 1850 MBps but soon after that, it stabilised to 1700 MBps , at the same time H1 went down to the same level as H2. H2's performance stayed there while H1´s performance turned down to 850 as H3 started sending traffic. The rest of the experiment, both nodes continued at the same level. The victim node started finally at 850 MBps .

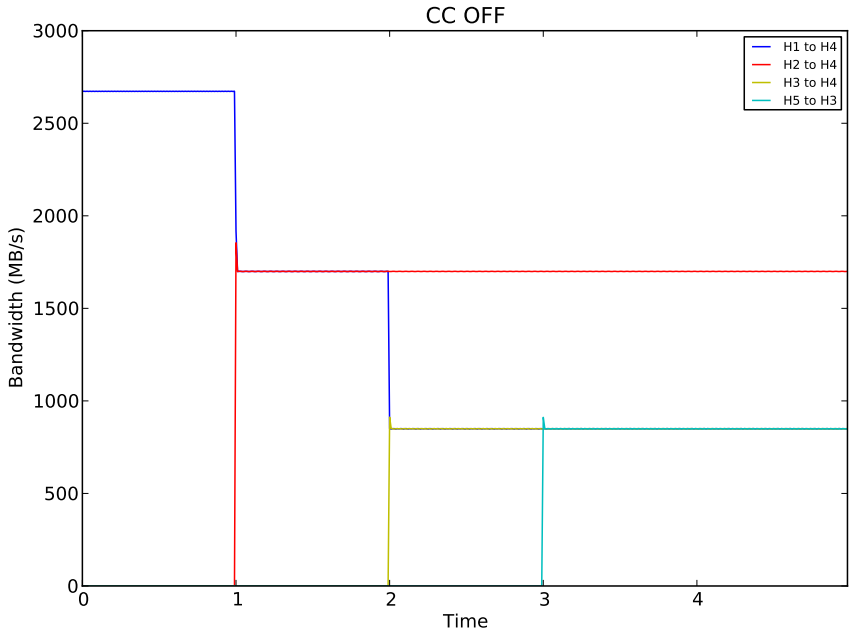



Figure 4.6: Sender - CC OFF with 40 GBit/s link bandwidth

The receivers´performance is shown in graph 4.7, It can be seen that H4 is receiving traffic from one host for the first second, then as second host began to send, the traffic it receives get increased to 3400 MBps and stabilized there for H4. For H3, it got traffic three second after the simulation is ran and the traffic it received leveled at a level of 850 MBps for the rest of the simulation time. Resource sharing of the nodes are not changes at all but when it comes link capacity usage. The performance has increased about 160%. It is still less than the 200% which was the ideal performance.

Read in the analysis and discussion chapter for why it could not maintain to use the whole capacity.

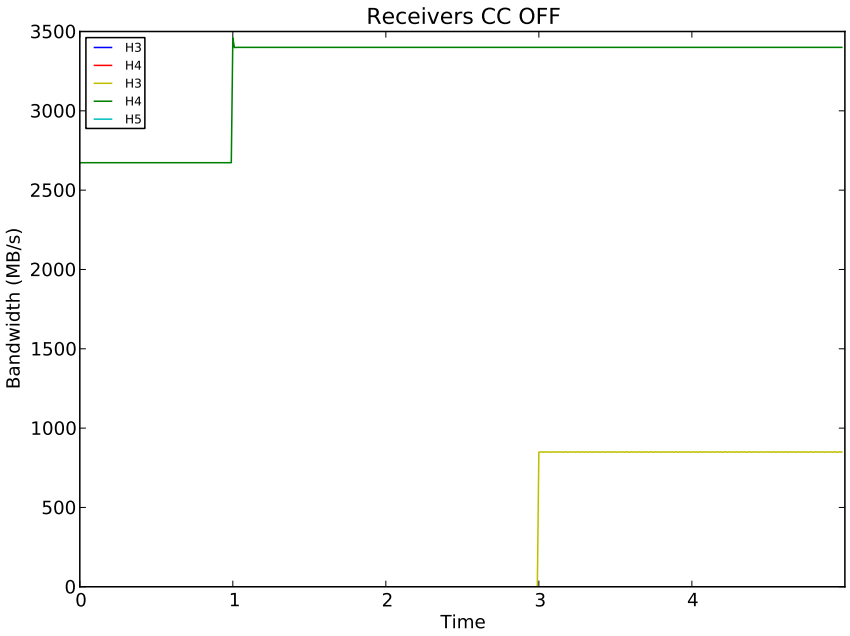



Figure 4.7: Receiver - CC OFF with 40 GBit/s link bandwidth

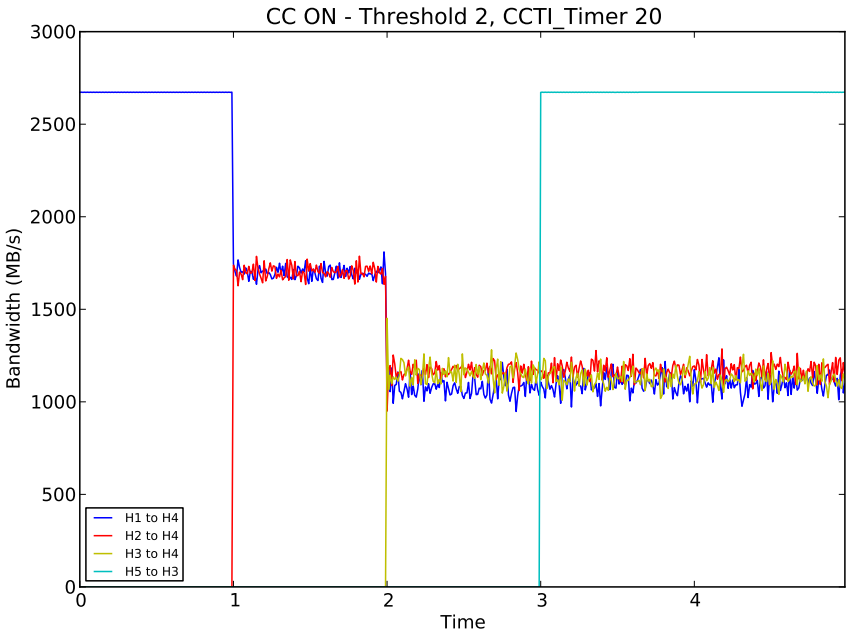



Figure 4.8: Sender - CC ON with 40 GBit/s link bandwidth

When the IB CC is enabled and the host-switch link capacity is increased to 40 GBits/s, it is showing a picture more like the one, we have

explained in previous paragraph. Resource sharing is just like what it was, when the host-switch link was 20 GBit/s, but the link usage level is not proportional to the increase which the host-switch link capacity is double.

H4-SW4 link capacity of 40 GBit/s is never used in its full capacity. According to graph 4.9, at onces, right after H2 started to send, it reached its peak of about 3500 MBps . Soon after that it turned down to 3400 MBps and stayed there the whole time. H5 to H3 traffic flow is also performing very poor. It is sending the whole time about 2600 MBps . One excuse is that a single host does not generate enough traffic to fill the whole link, but still the difference is too big, and that excuse to be taken as the sole cause of the bad performance.

No attempts were tried to experiment the existence of better suited IB CC parameters for this network topology after the host-switch link capacity was increased. The same IB CC parameters as before were used. It is believed as the resource sharing is the same and has not changed at all, that the parameters can not be the root of the poor link usage performance. The cause is probably related to the "in/out" limitations of the PCIe i the simulator internal to the end node. No changes have been done in the .ini file except increasing the host-to-switch link size. So there is a full possibility.

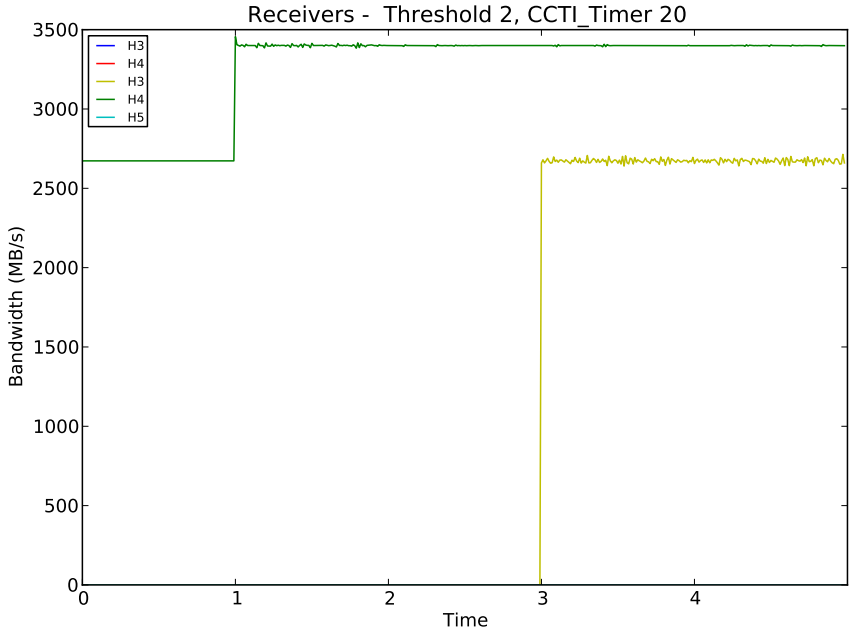



Figure 4.9: Receiver - CC ON with 40 GBit/s link bandwidth

## 4.5 How IB CC parameters influence performance

By now, we are all well aware that the IB CC parameters have a huge impact to the performance of the network. The parameters have a great influence

to the behaviour of the whole network, both on the switch side and also the host side. From chapter two and later chapter three, we learnt much about the most important parameters and got a list of all the parameters. It was decided to focus just two main parameters, one from the Channel Adapter (`CCTI_Timer`) and one from the Switch (`Threshold`). And we will do two simulations, in which the `Packet_Size` parameter from the Switch is changed from one to eight.

First, we started to focus the main two parameters. A total of 16 simulations were conducted to see whether there are common patterns that are present in all of them. Four different values have been given to the `Threshold` and as well as to the `CCTI_Timer`. `Threshold's` given values are two, four, eight and ten. While `CCTI_Timer's` given values are twenty, forty, eighty, and hundred twenty.

After all the simulations were done, a system of comparison was introduced. Two values of the `Threshold` were chosen which are the values; two and eight (Look scenario 1 and 2).

Scenario 1

| Threshold | | CCTI_Timer |
|---|---|---|
| 2 | ---> | 20 |
| | | 40 |
| | | 80 |
| | | 120 |

Scenario 2

| Threshold | | CCTI_Timer |
|---|---|---|
| 8 | ---> | 20 |
| | | 40 |
| | | 80 |
| | | 120 |

The performances of when `Threshold` value is two and the `CCTI_Timer` are twenty, forty, eighty and hundred twenty will be presented side by side to each other. Exactly, the same will happen when the value of the `Threshold` is eight. And from `CCTI_Timer`, values twenty and eighty will each go head to head to every and each of the `Threshold's` values (two, four, eight and ten). See scenario 3 and 4.

Scenario 3

| Threshold | | CCTI_Timer |
|---|---|---|
| 2 | <--- | 20 |
| 4 | | |
| 8 | | |
| 10 | | |

Scenario 4

| Threshold | | CCTI_Timer |
|---|---|---|
| 2 | <--- | 80 |
| 4 | | |
| 8 | | |
| 10 | | |

We are doing this to get a broader picture of how the IB CC parameters influence performance. For each part, one of the parameters stays the same, while the other one changes four times.

### 4.5.1 Scenario 1 - `Threshold` 2

In this part, the parameter Threshold get a value of two, while `CCTI_Timer's` value is twenty for the first time, then forty for the second time, eighty for the third time and finally, the value is hundred twenty. Both senders and receivers get measured. The graph of the left side is for the senders and the one the right side is for the receivers. By putting side by side, a graph of the senders and one for the receivers, we can see and study their performances easily. After that one can see how much the performance changed when the `CCTI_Timer` is changed one level up.

The first line shows the result from when the Threshold is two and `CCTI_Timer` is twenty (graph 4.10 and 4.14). This one, is the one with the best performance. There is no victim at all and the throughput is shared fairly. Each of the three congestion contributors get a third of the H4-SW4 link capacity.

Below it, graph 4.11 and graph 4.15 shows how the performance changed after the value of the `CCTI_Timer` is changed to forty. There are noticeable decline on the performances of H1, H2 and H3. When H1 is the only one sending, there is no problem, but as H2 and later H3 starts to send, they will never manage to use all the capacity that are available for them. H4's receiving peak is about 1900 MBps of the 2000 MBps capacity it can receive. H5 (the victim) maintained to keep up in its maximum capacity that it can generate and is unchanged.

Graph 4.12 and graph 4.16 show when the value of the `CCTI_Timer` is increased to eighty. The performance decline is more noticeable than before. H1 starts fine, then H2 starts and the performance deteriorates instead of increasing. And as H3 starts to send, their common traffic injection going to H4, is measured to be as low as about 1450 MBps . H5 is not effected by the change. It is performing as it did last two times.

The last two parallel graphs are graph 4.13 and 4.17, H5 is the same as before but the first second of H1 is the same as before too. Beside that, the performance is the worse that have been experienced so far. H4 receives as low as 1300 MBps from when H2 starts to send and goes down to 1100 MBps as H3 starts to send and keeps on the same for the rest of the simulation.

Senders

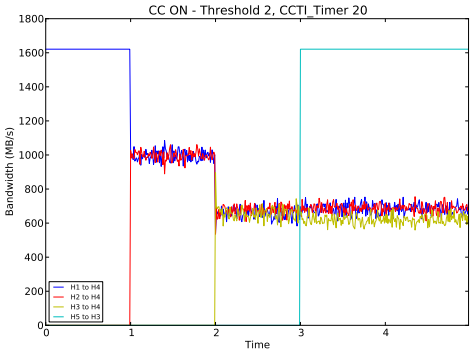



Figure 4.10: Threshold=2, Timer=20

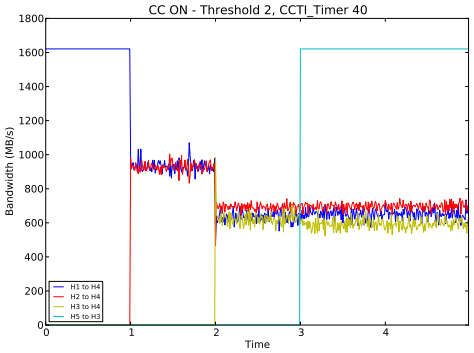



Figure 4.11: Threshold=2, Timer=40

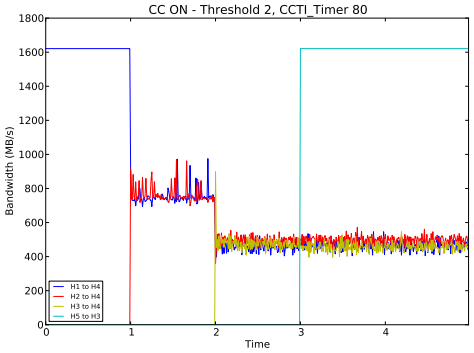



Figure 4.12: Threshold=2, Timer=80

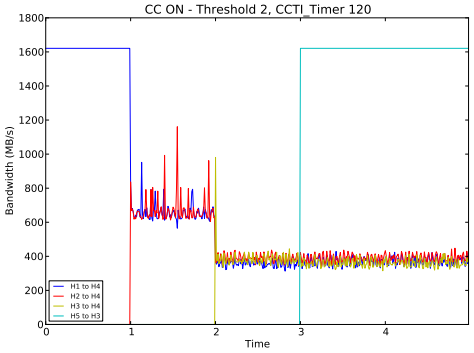



Figure 4.13: Threshold=2, Timer=120

Receivers

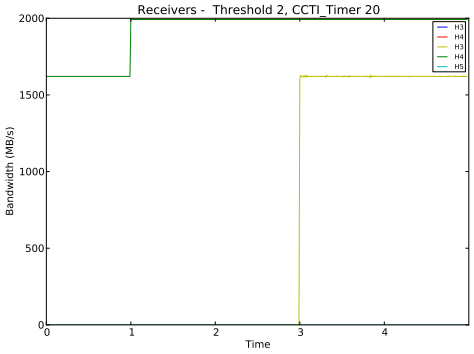



Figure 4.14: Threshold=2, Timer=20

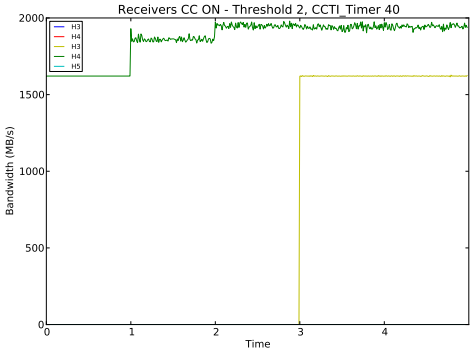



Figure 4.15: Threshold=2, Timer=40

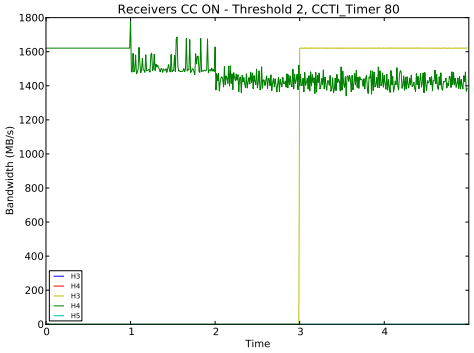



Figure 4.16: Threshold=2, Timer=80

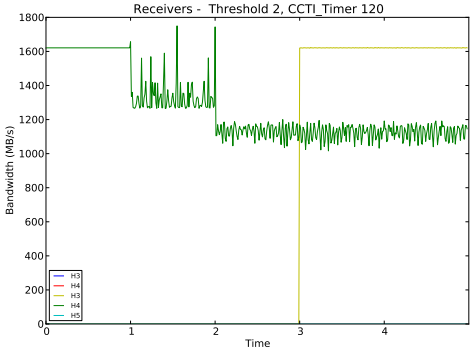



Figure 4.17: Threshold=2, Timer=120

### 4.5.2 Scenario 2 - `Threshold` 8

The value of parameter Threshold is increased to eight in this part. We keep to the same system as previous part. The `CCTI_Timer` values are twenty, forty, eighty and hundred twenty. The Threshold will stay the same the whole time in this part. As the previous part, we start with when the Threshold is eight and `CCTI_Timer` is twenty.

Graph 4.18 and graph 4.22 show how the performance is well in the first second, after that, the performances of both H1 and H2 decline to about 950 MBps each, not the 1000 MBps which was supposed to. And as H3 starts to inject traffic in the network. H4 receives about 1900 MBps , slightly less than the 2000 MBps , it can receive. This experiment's result is good but not good enough. All the available capacity of the link between H4 and SW4, was never used to its maximum. For H3, it receives all the traffic, H5 can generate.

Next in line are, Graph 4.19 and graph 4.23, whose show us the result of when the Threshold is eight and the `CCTI_Timer` is forty. Almost the whole time during the simulation, H1, H2 and H3 have a combined throughput of slightly less than 1600 MBps . H2 can be noticed to perform little better than the other two as H3 started to send traffic. H5 is doing well, there is no victim in this simulation as well as the last simulation were.

Graph 4.20 and graph 4.24, and also graph 4.21 and graph 4.25, both simulations show how receiving performance of H4 deteriorates as the value of parameter `CCTI_Timer` is increased more. The first of these two simulations, the `CCTI_Timer` is eighty and the second one the `CCTI_Timer` is hundred twenty. The first simulation H4 receives at low as 1100 MBps , while for simulation two, H4 receives in its lowest point (two third of the time) as little as 800 MBps . Every time, H1 is sending in its full capacity for the first second and there is no victim node at any time.

Senders

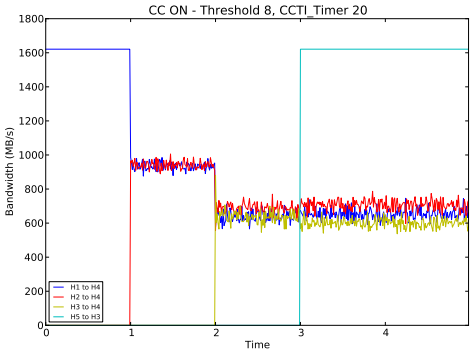



Figure 4.18: Threshold=8, Timer=20

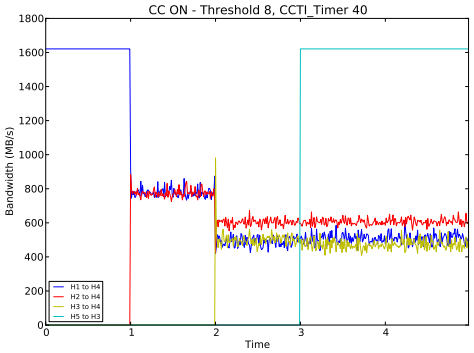



Figure 4.19: Threshold=8, Timer=40

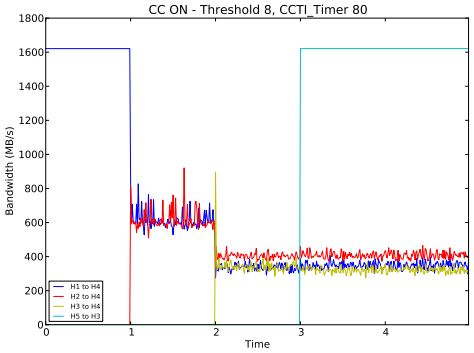



Figure 4.20: Threshold=8, Timer=80

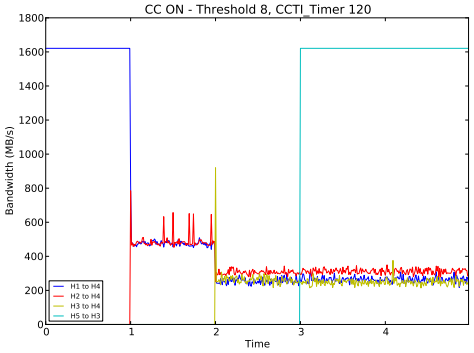



Figure 4.21: Threshold=8, Timer=120

Receivers

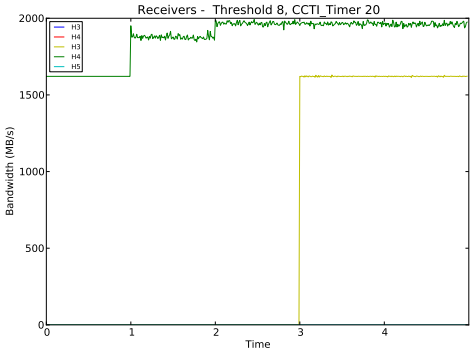



Figure 4.22: Threshold=8, Timer=20

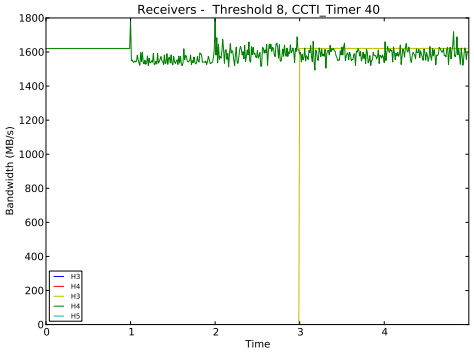



Figure 4.23: Threshold=8, Timer=40

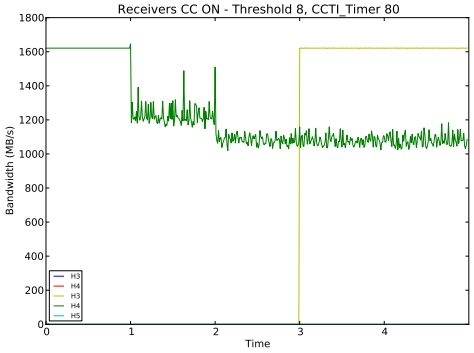



Figure 4.24: Threshold=8, Timer=80

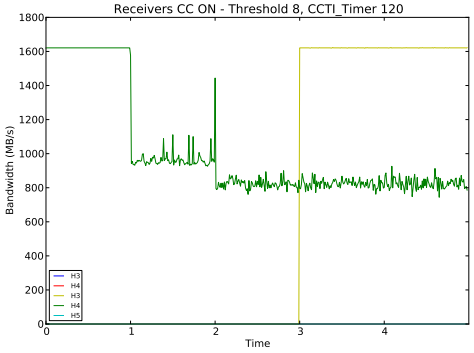



Figure 4.25: Threshold=8, Timer=120

### 4.5.3 Scenario 3 - `CCTI_Timer` twenty

In scenario 1 and 2, we have experienced how `CCTI_Timer` influence the performance of the network. We kept the Threshold the same in each scenario. Scenario 1, the Threshold was two and the second one, it was eight. For this scenario and the next one, we will do the opposite of what we have done in scenario 1 and 2. The `CCTI_Timer` will stay the same but the Threshold will be changed in each simulation.

`CCTI_Timer` is twenty and Threshold is two for this simulation (see graph 4.26 and graph 4.30). The outcome of this simulation have been explained before, two times in the past sections, but it can be said brief that it is the best among all the outcomes. And graph 4.27 and graph 4.31 show what happens when the CCTI_Timer stays the same but the Threshold is changed to four. No, major changes happen. The performances are very much the same as when the Threshold was two. The throughput is shared by H1, H2 and H3 almost equally and H5 is sending all the traffic, it can generate to H3.

For graph 4.28 and graph 4.32, in which the Threshold is eight. It can be seen that unlike, the last two simulations, H4 is not receiving all it can receive. When H2 starts to send, its and H1's combine traffic is not reaching the 2000 MBps which H4 reached earlier. Their combine injection is about 1850 MBps until H3 starts to send, in which the trios combined injection is about 1950 MBps . It is not bad but not as good as the last two results where all H4's received close to 2000 MBps . For H5 no changes have been registered.

And for last, graph 4.29 and graph 4.33 visualises the performance when the Threshold is ten. First second the performance is the best it could be, and for the second second, the combined performances of H1 and H2 is about 1750 MBps and for the third second, the trios combined performances went up to at most 1900 MBps . This is the worst so far of the performances in when `CCTI_Timer` is twenty. The performances got worse as the Threshold increased from two to finally ten.

Senders

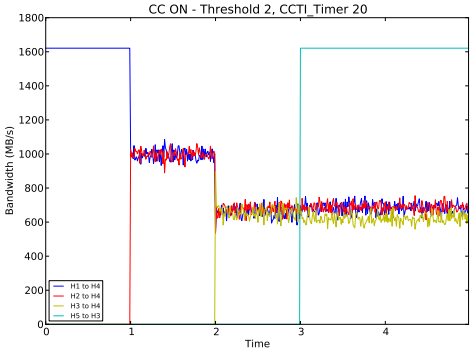



Figure 4.26: Threshold=2, Timer=20

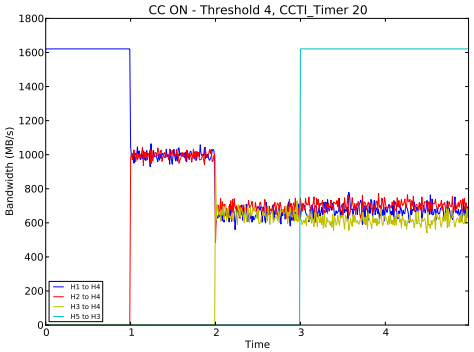



Figure 4.27: Threshold=4, Timer=20

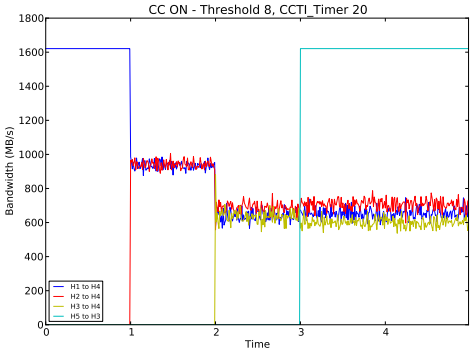



Figure 4.28: Threshold=8, Timer=20

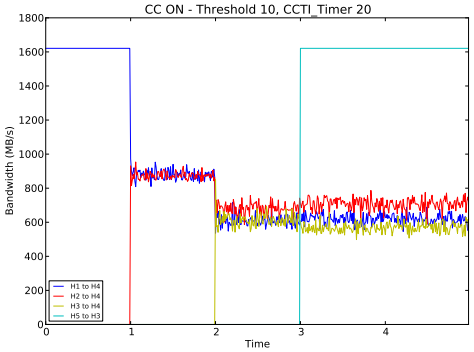



Figure 4.29: Threshold=10, Timer=20

Receivers

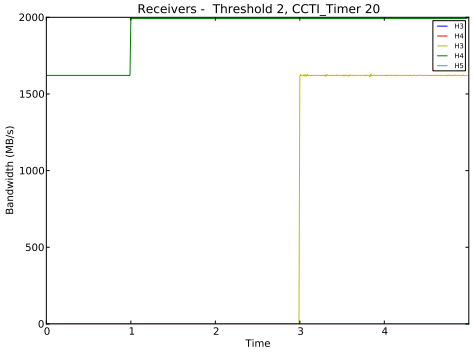



Figure 4.30: Threshold=2, Timer=20

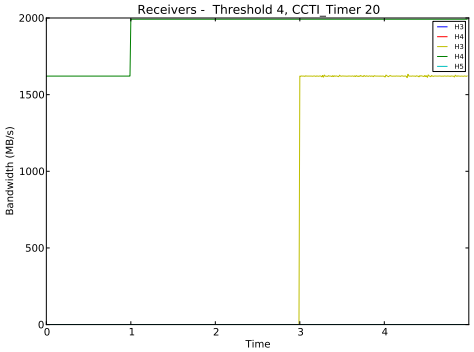



Figure 4.31: Threshold=4, Timer=20

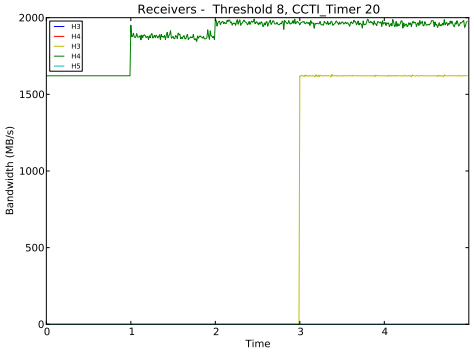



Figure 4.32: Threshold=8, Timer=20

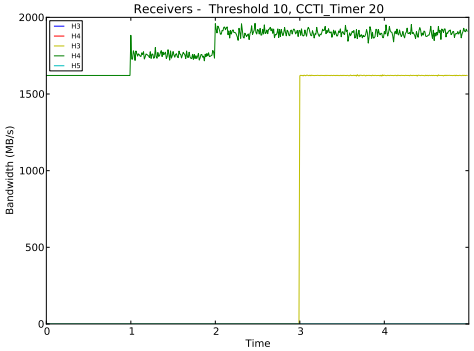



Figure 4.33: Threshold=10, Timer=20

### 4.5.4 Scenario 4 - `CCTI_Timer` eighty

The `CCTI_Timer` will remain eighty in the next four simulations while the Threshold will be two, then four, eight and finally ten. The first graphs (4.34 and 4.38) show how the performance is when `CCTI_Timer` is eighty and Threshold is two. As usual from last simulations, H1 starts at about 1650 MBps which is the maximum, it can generate. H4, the receiver receives all. Then H2 starts to send. In which both of them, H1 and H2 start sending little bit less than 800 MBps . As, H3 starts to send traffic, all three of them, send around 500 MBps each. H4 receives around 1500 MBps of traffic. H5 performance fine. So, no victim node has been observed.

Graph 4.35 and 4.39, fellows the same pattern as the last simulations. In this simulation the Threshold is increased to four. Its performance is close to last simulation's performance. There is about 100 MBps of decrease in all the performances of H1, H2 and H3, except the very first second, which always is the same for all the simulations that have been conducted until now.

When we increased the value of the Threshold one more time to eight, the performance get even worse than ever since swiping the Threshold started. H2 starts to send and both H1 and H2 send around 600 MBps each and as H3 starts to send, all of them send around 400 MBps . H4's receiving rate goes down from 1650 MBps to 1200 MBps and then to 1100 MBps . H5 still does well as it was supposed to. In graphics see graph 4.36 and graph 4.40

And finally, the last simulation that will be presented in this section is that in which the Threshold is changed to ten, still the `CCTI_Timer` is eighty, as it has been last three simulations. Graph 4.37 and 4.41 show that from when H2 starts to send, its and H1's rate turn to be around 500 MBps each. When H3 activated, Its and H1's become around 300 MBps each and H2 performed slightly better in about 350 MBps . H5 stays as it always is, around 1650 MBps . It can be seen that H4 from the second second receives only 1000 MBps and then it is receiving rate turn down to around 950 MBps . That rate is less than half of what it has capable to receive. With these parameters, much of the bandwidth is unused.

On the appendices, some more simulation results like these but with different parameters can be seen. We believe that one can easily view the rest of the graphs and continue to build their understand of how IB CC parameters influence performance.

Senders

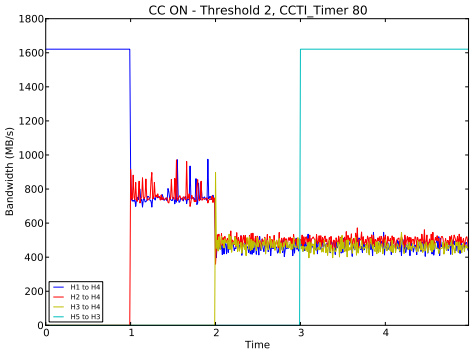



Figure 4.34: Threshold=2, Timer=80

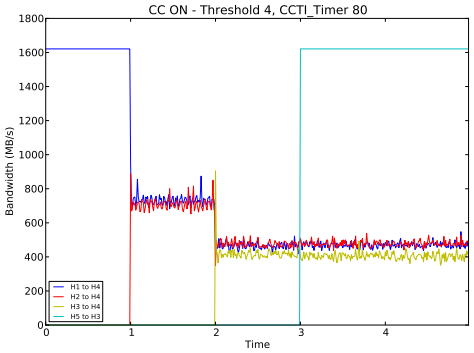



Figure 4.35: Threshold=4, Timer=80

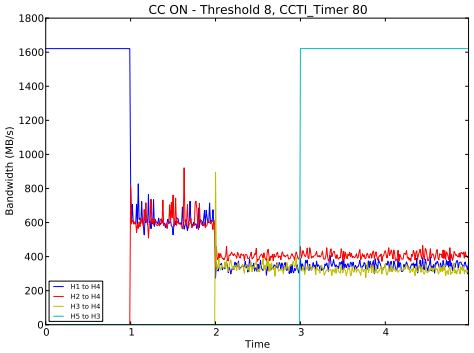



Figure 4.36: Threshold=8, Timer=80

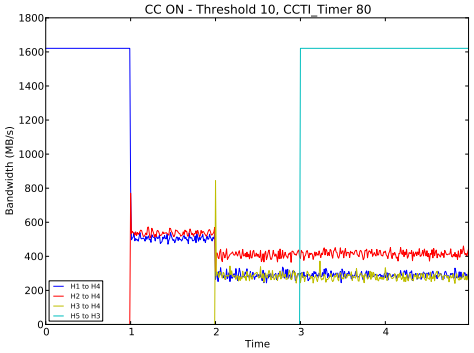



Figure 4.37: Threshold=10, Timer=80

Receivers

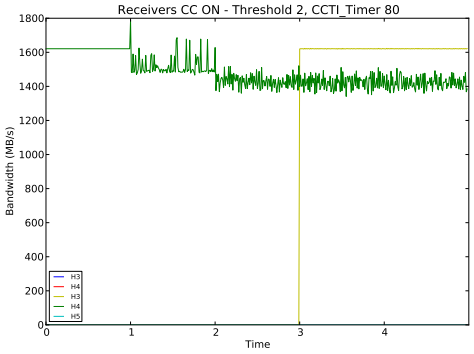



Figure 4.38: Threshold=2, Timer=80

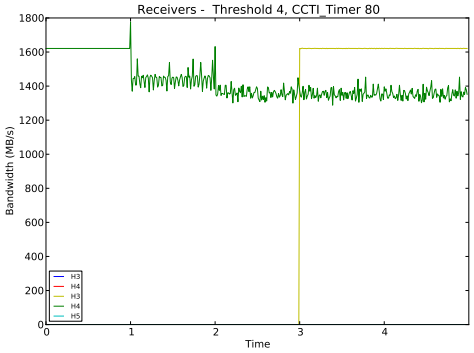



Figure 4.39: Threshold=4, Timer=80

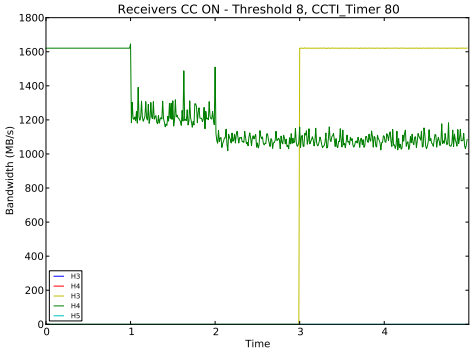



Figure 4.40: Threshold=8, Timer=80

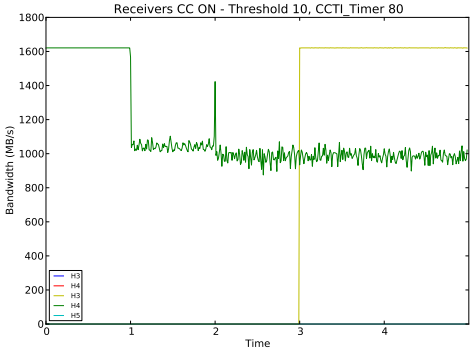



Figure 4.41: Threshold=10, Timer=80

### 4.5.5 Influence of `Packet_Size` parameter

We have seen how the Threshold and `CCTI_Timer` influence performance. Now, `Packet_Size` and its influence will be examined here. For the next four simulations, each simulation has the same IB CC parameters with one other simulation, except the `Packet_Size` which is different. In that way, each two are paired together, a comparison of every pair will be done. The goal is to see how much influence parameter `Packet_Size` has. For all the previous simulation and those on the appendices have all of them a `Packet_Size` of one. Two from the previous simulations which has a `Packet_Size` of one, will go head to head two new simulations with a `Packet_Size` of eight.

The first to go, are the ones who have a Threshold of two and a `CCTI_Timer` of twenty. From Graph 4.42 and graph 4.44, it can be viewed how it performed when it has a `Packet_Size` of one. Below it, graph 4.43 and 4.45 show when it has a `Packet_Size` of eight.

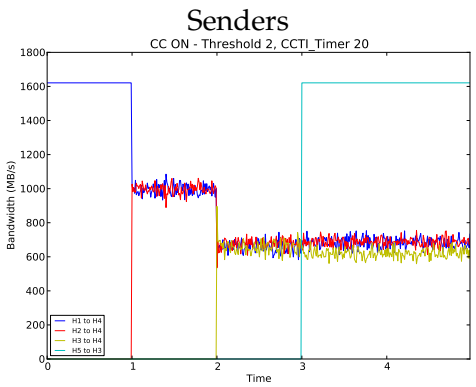



Figure 4.42: `Packet_Size` 1

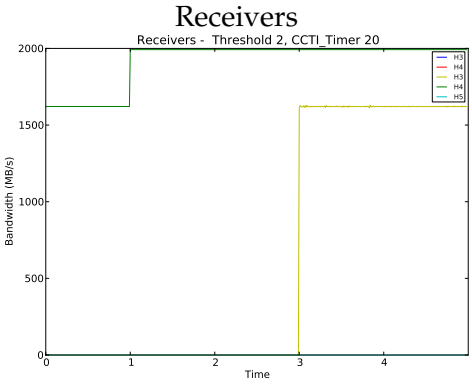



Figure 4.44: `Packet_Size` 1

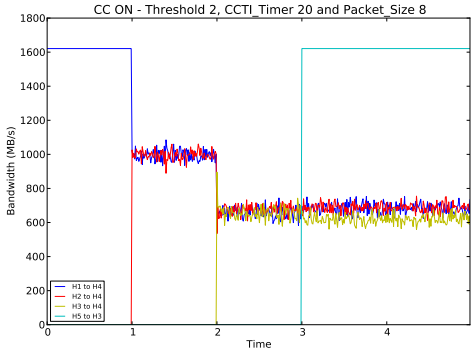



Figure 4.43: `Packet_Size` 8

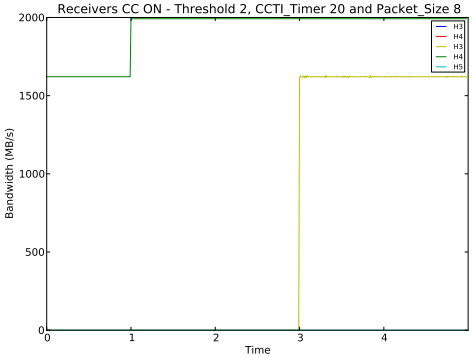



Figure 4.45: `Packet_Size` 8

No, major difference can seen from the graphs. The numbers are the same or very close to each other. Viewing both the senders graphs and the receivers graphs, no difference been detected.

One single simulation can not proof or disproof about any thing, so one more simulation has been conducted. Still the Threshold and `CCTI_Timer` are the same for both the simulation that we will compare to each other. The Threshold is ten and `CCTI_Timer` is 20 for both of them. The `Packet_Size` is once one and again eight.

Viewing the result of these two simulations when the `Packet_Size` is one (4.46 and 4.48) and when it is eight (4.47 and 4.49), seem very much the same. Every up-swing and down-swing can be found both of the results.

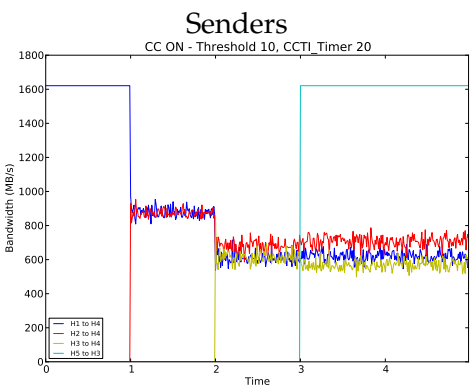



Figure 4.46: `Packet_Size` 1

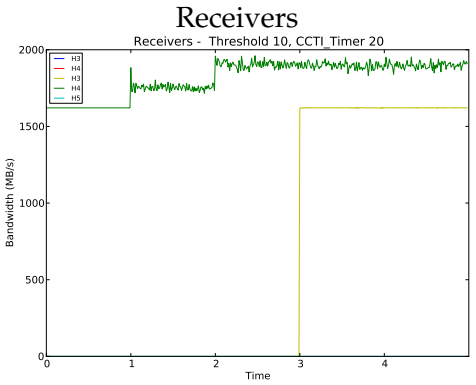



Figure 4.48: `Packet_Size` 1

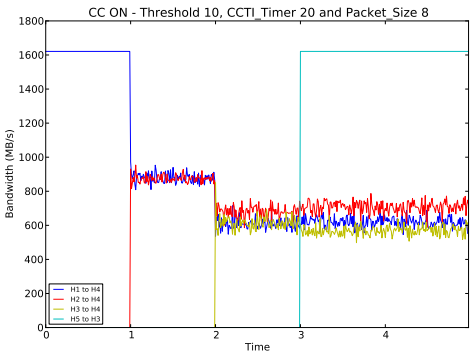



Figure 4.47: `Packet_Size` 8

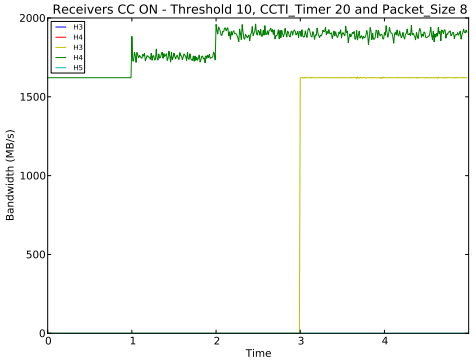



Figure 4.49: `Packet_Size` 8

As, stated earlier, the result of one simulation is not enough to proof anything and two simulations can still not be much of evidence, but we have to acknowledge that it must mean something as every pair of results shown to be identical. We went back to the configuration file and we found out that in the configuration file, the packet length in bytes was 2176 bytes. And that both the values that we tested are smaller than the packet length size, so increasing the `Packet_Size` from one to eight do not do any difference.

Two more simulations have been conducted again, these simulations have a `Packet_Size` value of 2200. This value is bigger than the packet length size in the configuration file which is 2176 bytes.

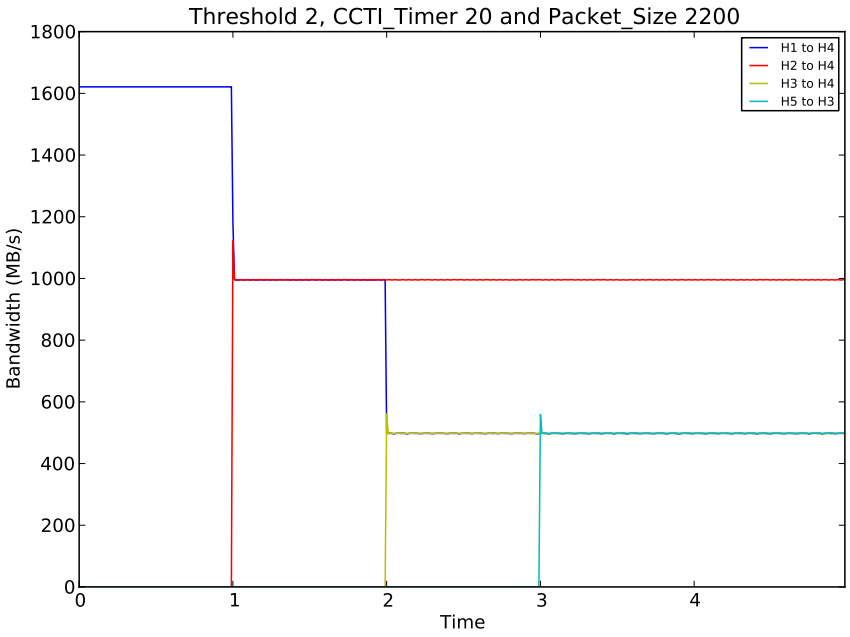



Figure 4.50: `Packet_Size` 2200

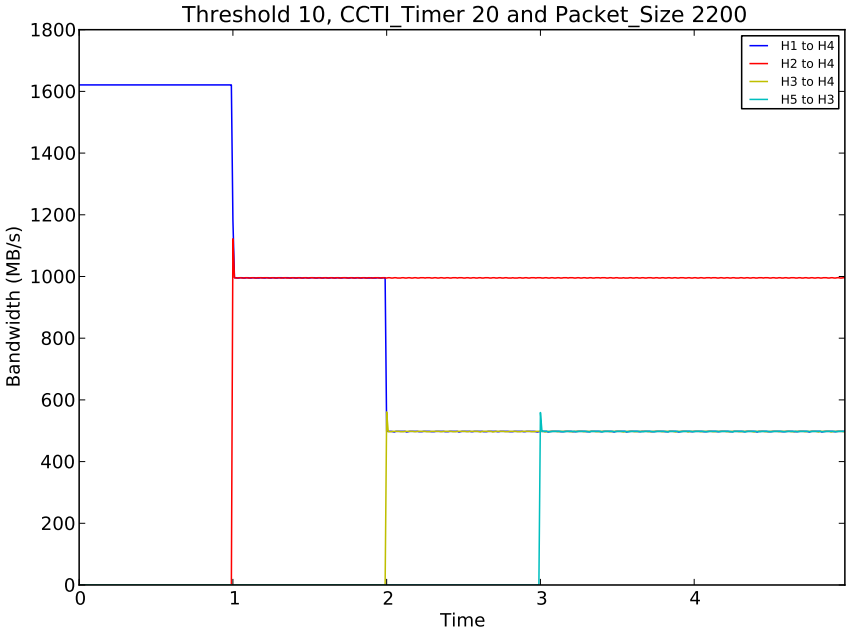



Figure 4.51: `Packet_Size` 2200

These two graphs 4.50 and 4.51 are clearly different from graph 4.46 and 4.47. We can see that the new `Packet_Size` value has changed the

whole result. These new results are identical to the result of when the IB CC is off (see graph 4.2). By having a value which is bigger the one from the configuration file shown to disable the effect of the IB CC. For more analyses, see the analyses chapter, `Packet_Size` section.

# Chapter 5

# Analysis and Discussion

In this chapter, we analyse and discuss the results and explain our findings. Results from when IB CC was enabled and when it was disabled, will be compared and analysed, both for throughput and fairness. Also, the results gained after the host-to-switch link size was increased will too be analysed. The kind of influences parameters `Threshold`, `CCTI_Timer`, and `Packet_Size` have to congestion contributors, congestion victim and traffic receiving nodes, will be studied and presented in this chapter. In chapter three "the approach", some expectations toward the results have been presented, those expectations are going to be compared the real results from chapter 4, "the result chapter". And finally, we are also going to see the values that are considered to be the best for this topology.

This chapter consists six different sections, each section is supposed to address one or several relating issues. We are hoping that the content of this chapter, will answer all issues that have been touched in the last four chapters.

## 5.1 Throughput and Fairness

The results of two different simulations (section 4.1 and 4.2), in which one of them has IB CC enabled, while the other one has not, was conducted. The one without IB CC was first conducted and as its result was recorded, the second one with IB CC was also conducted. Four graphs have been plotted from their results (see sections 4.1 and 4.2). From that graphs, it is very easy to see the improvements caused by the IB CC.

When it comes to the throughput, the receiving rate of H4 stayed the same, as there were no room for an improvement, but the throughput of the victim flow (H5-to-H3) have improved. From previous simulation, it could only benefit from a quarter of the resources. But now, all the traffic that H5 is capable to generate, gets to destination H3. No more victim flow exists after IB CC was enabled. All the negative impact the congestion had on the victim flow is removed.

And for the fairness, the contributors (H1, H2 and H3) get each a third of H4's available receiving capacity. When the IB CC was disabled, H1 and H3 shared a half and the other half was for H2. An eight percent improvement for each of them (H1 and H3) is resulted. H2's unfair share was eliminated.

Table 5.1: A comparison of when IB CC is on and when it is off

| Node | Throughput | Fairness | Expectations |
|---|---|---|---|
| H1 - sending | Better | Fairer | As expected |
| H2 - sending | Worse | Fairer | As expected |
| H3 - sending | Better | Fairer | As expected |
| H5 - sending | Better | Fairer | As expected |
| H3 - Receving | Better | | As expected |
| H4 - Receving | Same | | As expected |

It is a good news that all the nodes are using their fair share of the bandwidth and the congestion has not any impact to any node rather than those contributing. The receiving rate of both the receiving nodes (H3 and H4) are on top depending on the traffic injections sent toward them. When this result is compared to the expectations that was presented in the approach chapter, they are very close to each other (See the last column of the tables 5.1.

## 5.2 Bigger host-to-switch link

The capacity of the host-to-switch links have been increased from 20 GBit/s to 40 GBit/s (section 4.3), while the switch-to-switch link size stayed the same as previous (40 GBit/s). Two new simulations have been implemented from this slightly modified network topology. Once IB CC is disabled and the second time it is enabled. These two results is going to be compared to those of when the host-to-switch link size were 20 GBit/s.

Table 5.2: Improvements in % when host-to-switch link is bigger

| Node | Throughput improvement | Fairness |
|---|---|---|
| H1 - sending | 160 percent | Same |
| H2 - sending | 160 percent | Same |
| H3 - sending | 160 percent | Same |
| H5 - sending | 160 percent | Same |
| H3 - Receiving | 160 percent | Same |
| H4 - Receiving | 160 percent | Same |

A major throughput improvement has been observed, but this improvement is not in the reach of what one may expect (two times the previous

throughput). All the sending and receiving nodes improved their throughput performances around 160 percent. The throughput improvement is constant all the time and for all the nodes in both the two simulations. But the bandwidth fair sharing is exactly the same as their counterparts.

The reason why performances never reached its peak of around 40 GBit/s is that there is a limitation in the configuration. 34 GBit/s is the upper limit for the receivers. While we were implementing this simulation nothing else was changed except the host-to-link size. For future simulations, it will be nice to do some more modifications in the configuration file, when the link capacity is increased. In that way, there can be a chance to further improve the performance.

## 5.3 `CCTI_Timer` influence

According to the simulations that are presenting in this project, `CCTI_Timer` proofed to be the most influential when compared to the influences of `Threshold` and `Packet_Size`. Table 5.3 displays how each of four simulations with different values of `CCTI_Timer` perform against the result of a simulation without IB CC. In a recap the performance rate of a simulation without IB CC (NO CC) was 2000 MBps for H4 and 500 for H3. The values of H4's performance rate is when all the contributing nodes start to send, or the last three seconds of the simulation's total five second running time. The improvement of the victim flow is constant when IB CC is on. And it is the performance of the contributors which changes each time.

Table 5.3: `CCTI_Timer` against when IB CC is off

| Simulation | H4 receiving rate | H3 receiving rate | Total Improvement |
|---|---|---|---|
| NO CC | 2000 | 500 | 100% |
| 20 - 2 | 2000 | 1650 | 146% |
| 40 - 2 | 1900 | 1650 | 142% |
| 80 - 2 | 1450 | 1650 | 124% |
| 120 - 2 | 1300 | 1650 | 110% |

In this comparison, a noticeable improvement is registered. Best performing one is when the Threshold is two and the `CTI_Timer` is twenty, with a total improvement of 146%. Then 142%, 124% and 110%. All of them shown to increase the overall performance.

Table 5.4: `CTI_Timer` against when IB CC is off

| Simulation | H4 receiving rate | H3 receiving rate | Total Improvement |
|---|---|---|---|
| NO CC | 2000 | 500 | 100% |
| 20 - 8 | 1900 | 1650 | 142% |
| 40 - 8 | 1600 | 1650 | 130% |
| 80 - 8 | 1100 | 1650 | 110% |
| 120 - 8 | 800 | 1650 | 98% |

When another four simulations were conducted, this time with a bigger `Threshold` (eight). Still the `CTI_Timer` influences the performance into a positive way. Three of the four simulation shown improvement while the last one's performance about the same as when IB CC is off.

The reason why `CCTI_Timer` has so much of influence is that when the destination registries a FECN bit from the switch and it returns a packet with BECN bit to the source. The source puts a gap between its injection rates. That gap widens as more BECN bits arrive at the source. If no new BECN arrives, it starts to narrow down, until normal injection rate resumes again. To increase injection rate again, the source relies on the `CCTI_Timer`. The higher value `CCTI_Timer` has, the longer time it takes to reach normal injection rate, even though the congestion has been removed a long time ago.

## 5.4 `Threshold` Influence

Threshold shown also to have a big influence to the performance of the network topology with the enabled IB CC. A great increase for all the different Threshold values when the `CCTI_Timer` is twenty. When the Threshold's values are two and four, a total overall improvement of 146% is registered that is the highest point, performance can reach. Followed by 144% and 142%.

Table 5.5: Threshold influence - `CCTI_Timer` twenty

| Simulation | H4 receiving rate | H3 receiving rate | Total Improvement |
|---|---|---|---|
| NO CC | 2000 | 500 | 100% |
| 2 - 20 | 2000 | 1650 | 146% |
| 4 - 20 | 2000 | 1650 | 146% |
| 8 - 20 | 1950 | 1650 | 144% |
| 10 - 20 | 1900 | 1650 | 142% |

The closeness of the results are little worrying, compared to the previous section where the `CCTI_Timer` was focused. Each time the value of the `CCTI_Timer` is changed, the performance differentiated very much. From table 5.5, we can see that changing the Threshold value have an impact but

not as big as when the CCTI_Timer is changed.

Table 5.6 displays the overall performances of the receivers in percentage when the Threshold varies and the CCTI_Timer is eighty. The results vary from 126% to 104%. That is also a good improvement, but far less than the previous one.

Table 5.6: Threshold influence - CCTI_Timer eighty

| Simulation | H4 receiving rate | H3 receiving rate | Total Improvement |
|---|---|---|---|
| NO CC | 2000 | 500 | 100% |
| 2 - 80 | 1500 | 1650 | 126% |
| 4 - 80 | 1400 | 1650 | 122% |
| 8 - 80 | 1100 | 1650 | 110% |
| 10 - 80 | 950 | 1650 | 104% |

From these simulations, we have learnt that modifying Threshold parameter is important and can make an improvement to the network. Each time the values get increased noticeable changes can be observed in the result, that shows the influence Threshold parameter has. We have also found out that smaller Threshold values have the best performances. And whenever a smaller Threshold value is combined with a smaller CCTI_Timer value, the performance shown to the best.

## 5.5 Packet_Size influence

Four simulations have been compared into pairs (section 4.4.5). Two with a Packet_Size of one and two with a Packet_Size of eight. All the other parameters were the same for every opposite ones except their Packet_Size values. In that way, we have paired them into two pairs with two different Packet_Size values. The goal was to see the influence Packet_Size has.

From the results, no differences between each pair, have been detected in any of the two pairs. We later found out that in the configuration file, the packet length in bytes was 2176 bytes. And that both the values we used in the simulations, are smaller than the packet length size value in the configuration file, because of that no changes have been recorded.

Packet_Size works this way. When a congestion state occurs, packets get FECN marked. The FECN bit set, depends on Packet_Size and Marking_Rate. Packets with a size smaller than the Packet_Size will not get a FECN bit set. Both of the values are smaller and because of that the Packet_Size parameter did not have any impact in these simulations. We believe if the Packet_Size value was bigger than 2176 bytes then the result will have been totally different. All packets will have got FECN bit, in which could have destroyed the whole effect of the IB CC and the result

will have been similar when the IB CC is disabled.

From that theory, we decided to do the same simulation again but this time, the Packet_Size value is bigger than the packet length size in the configuration file (2176). The Packet_Size value of these two new simulations are 2200 bytes.

After both the simulations are finished. We can with a confidence say that the theory mentioned above is true. When the Packet_Size value is greater than the packet length size in the configuration file, the IB CC effect will not exist anymore. The result is the same as when the IB CC was disabled.

## 5.6 Best IB CC values for this topology

The best IB CC parameters values so far are the ones shown below. It is only the Threshold, CCTI_Timer and Packet_Size which have been tested in these simulations. When Threshold is two, CCTI_Timer is twenty and Packet_Size is one (or less than its given value in the configuration file) performed best of all the different values which have been tested during this thesis period.

```
The best IB CC parameters so far for this topology.

**.Threshold = 2
**.Marking_Rate = 1
**.Packet_Size = 1
**.CCTI_Timer = 20
**.CCTI_Limit = 128
**.CCTI_Min = 0
**.CCTI_Increase = 1
```

These values eliminated all negative impacts that the congestion had on the victim flow. And they contributed a fairness to the flows that otherwise would be treated unfairly due to their positioning in the network topology.

# Chapter 6

# Conclusion and Future Work

In this thesis, we studied and explored the influence and effect of InfiniBand Congestion Control in interconnection networks. Simulations have been conducted when InfiniBand Congestion Control (IB CC) is enabled and when disabled. There have also been a search for finding proper IB CC parameters and to understand the influence of different parameters, especially `Threshold`, `CCTI_Timer`, and `Packet_Size` . Not only that but we have also increased the size of some of the links in the network topology, in order to see what kind of effect that action would have to the overall performance of the network.

We found out that when IB CC is enabled the throughput is better and there is a fairness among all nodes. The node's positions in the network will no longer have negative effects on fairness; every node get its fair share of the bandwidth. For the victim flow, all the negative impacts of the congestion have been removed without affecting congestion contributors' performances.

There are strong indications that the parameters `Threshold` and `CCTI_Timer`, have great influence on the performance. In both cases, whenever one parameter is changed and all others are kept constant, a noticeable change get registered in the result of the simulation. Furthermore, the smaller values are shown to produce a better performing results. And for the `Packet_Size` When the `Packet_Size` value is bigger than the packet length size in the configuration file. All packets will get FECN bit, in which destroys the whole effect of the IB CC and the result will be similar to when the IB CC is disabled.

A major throughput improvement has been observed when the host-to-switch capacity was doubled. All the sending and receiving nodes improved their throughput performances around 160 percent. The throughput improvement is constant. There is also a fairness among all the nodes, where every node get its fair share of the bandwidth.

For future work, there are much to do. Only three IB CC parameters have been explored, all the other IB CC parameters are still unexplored and needs more studying. The parameters do not have the same affect to all network topologies, so for each time, a system architecture is designing a network topology, more testing for that given topology must be done. But, the content of this project and future similar work will help others to understand the IB CC parameters better.

For future simulations, it will be nice to do some more modifications in the configuration file, when the link size is modified. In that way, there can be a chance to further improve the performance. As, there are few available guidelines, any kind of exploration of this technology is recommended.

# Bibliography


Amol, D. & Rajesh, P. (2014, January). A review on active queue management techniques of congestion control. In *Electronic systems, signal processing and computing technologies (icesc), 2014 international conference on* (pp. 166–169). doi:10.1109/ICESC.2014.34

David Ely, S. S. & Wetherall, D. (2001). Alpine: a user-level infrastructure for network protocol development. Retrieved from http://cseweb.ucsd.edu/~savage/papers/Usits01.pdf

Gran, E. (2014). Congestion management in lossless interconnection networks. Retrieved from https://www.simula.no/publications/congestion-management-lossless-interconnection-networks

Gran, E., Eimot, M., Reinemo, S.-A., Skeie, T., Lysne, O., Huse, L., & Shainer, G. (2010, April). First experiences with congestion control in infiniband hardware. In *Parallel distributed processing (ipdps), 2010 ieee international symposium on* (pp. 1–12). doi:10.1109/IPDPS.2010.5470419

Gran, E., Reinemo, S.-A., Lysne, O., Skeie, T., Zahavi, E., & Shainer, G. (2012, May). Exploring the scope of the infiniband congestion control mechanism. In *Parallel distributed processing symposium (ipdps), 2012 ieee 26th international* (pp. 1131–1143). doi:10.1109/IPDPS.2012.104

Gran, E., Zahavi, E., Reinemo, S.-A., Skeie, T., Shainer, G., & Lysne, O. (2011, May). On the relation between congestion control, switch arbitration and fairness. In *Cluster, cloud and grid computing (ccgrid), 2011 11th ieee/acm international symposium on* (pp. 342–351). doi:10.1109/CCGrid.2011.67

Gran, E. G. & Reinemo, S.-A. [Sven-Arne]. (2011). Infiniband congestion control: modelling and validation. In *Proceedings of the 4th international icst conference on simulation tools and techniques* (pp. 390–397). SIMUTools '11. Barcelona, Spain: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). Retrieved from http://dl.acm.org/citation.cfm?id=2151054.2151122

Grun, P. (2010). *Introduction to infiniband™ for end users*.

Gusat, M., Craddock, D., Denzel, W., Engbersen, T., Ni, N., Pfister, G., ... Duato, J. (2005, August). Congestion control in infiniband networks. In *High performance interconnects, 2005. proceedings. 13th symposium on* (pp. 158–159). doi:10.1109/CONECT.2005.14

IBTA, T. I. T. A. (2015). About infiniband. Retrieved from http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband

IHS. (2014). Ihs: 6b new internet-enabled devices to be produced this year. Retrieved from http://www.bloomberg.com/bb/newsarchive/ah0WUGiYdsrI.html

Lugones, D., Franco, D., & Luque, E. (2009, September). Adaptive multipath routing for congestion control in infiniband networks. In *Parallel processing workshops, 2009. icppw '09. international conference on* (pp. 222–227). doi:10.1109/ICPPW.2009.38

OmNeT++. (2014). What is omnet++? Retrieved from http://omnetpp.org/intro

Reinemo, S.-A. [S.-A.], Skeie, T., & Wadekar, M. (2010, July). Ethernet for high-performance data centers: on the new ieee datacenter bridging standards. *Micro, IEEE*, *30*(4), 42–51. doi:10.1109/MM.2010.65

Rouse, M. (2014). Network topology. Retrieved from http://whatis.techtarget.com/definition/network-topology

Santos, J., Turner, Y., & Janakiraman, G. (2003, March). End-to-end congestion control for infiniband. In *Infocom 2003. twenty-second annual joint conference of the ieee computer and communications. ieee societies* (Vol. 2, 1123–1133 vol.2). doi:10.1109/INFCOM.2003.1208949

Top500. (2014). Mellanox infiniband leads top500 as most used interconnect technology for high-performance computing for the 2nd consecutive time. Retrieved from http://www.mellanox.com/page/press_release_item?id=1425

Valentini, A., Di Biagio, C., Batino, F., Pennella, G., Palma, F., & Engelmann, C. (2009, February). High performance computing with harness over infiniband. In *Parallel, distributed and network-based processing, 2009 17th euromicro international conference on* (pp. 151–154). doi:10.1109/PDP.2009.64

Yan, S., Min, G., & Awan, I. (2006, April). An enhanced congestion control mechanism in infiniband networks for high performance computing systems. In *Advanced information networking and applications, 2006. aina 2006. 20th international conference on* (Vol. 1, pp. 845–850). doi:10.1109/AINA.2006.87

# Appendices

# Appendix A

# Appendix

## A.1 OmNet++ configuration files

### A.1.1 the Ned file

```
   network FABRIC extends h5_s4
{
    parameters:
}
package networks.h5_s4;

import src.HCA;
import src.Switch;
import src.IDGenerator;

module h4_s4
{
     @display("bgb=946,315");

    submodules:
        H_1: HCA {
            @display("p=108,116");
        }
        H_2: HCA {
            @display("p=108,209");
        }
        H_3: HCA {
            @display("p=526,116");
        }
        H_4: HCA {
            @display("p=526,209");
        }

  H_5: HCA {
            @display("p=108,46");
        }
```

```
    SW1: Switch {
        parameters:
            numSwitchPorts = 4;
            @display("p=248,116");
        gates:
            out[4];
            in[4];
    }
    SW2: Switch {
        parameters:
            numSwitchPorts = 3;
            @display("p=248,209");
        gates:
            out[3];
            in[3];
    }


    SW3: Switch {
        parameters:
            numSwitchPorts = 3;
            @display("p=366,116");
        gates:
            out[3];
            in[3];
    }
    SW4: Switch {
        parameters:
            numSwitchPorts = 3;
            @display("p=366,209");
        gates:
            out[3];
            in[3];
    }


    IDGen: IDGenerator {

        @display("p=311,23");
        gates:
            gate[5];
    }

connections:
    //SW1 --> HCAs
    SW1.out[1] --> {  delay = 5ns; } --> H_1.in;
    H_1.out --> {  delay = 5ns; } --> SW1.in[1];
```

```
        SW1.out[3] --> {  delay = 5ns; } --> H_5.in;
        H_5.out --> {  delay = 5ns; } --> SW1.in[3];

        SW2.out[1] --> {  delay = 5ns; } --> H_2.in;
        H_2.out --> {  delay = 5ns; } --> SW2.in[1];

        SW3.out[1] --> {  delay = 5ns; } --> H_3.in;
        H_3.out --> {  delay = 5ns; } --> SW3.in[1];

        SW4.out[1] --> {  delay = 5ns; } --> H_4.in;
        H_4.out --> {  delay = 5ns; } --> SW4.in[1];


        SW1.out[0] --> {  delay = 5ns; } --> SW3.in[0];
        SW3.out[0] --> {  delay = 5ns; } --> SW1.in[0];

        SW1.out[2] --> {  delay = 5ns; } --> SW2.in[2];
        SW2.out[2] --> {  delay = 5ns; } --> SW1.in[2];

        SW2.out[0] --> {  delay = 5ns; } --> SW4.in[0];
        SW4.out[0] --> {  delay = 5ns; } --> SW2.in[0];

        SW3.out[2] --> {  delay = 5ns; } --> SW4.in[2];
        SW4.out[2] --> {  delay = 5ns; } --> SW3.in[2];

        IDGen.gate[0] <--> {  delay = 5ns; } <--> H_1.gateIDGen;
        IDGen.gate[1] <--> {  delay = 5ns; } <--> H_2.gateIDGen;
        IDGen.gate[2] <--> {  delay = 5ns; } <--> H_3.gateIDGen;
        IDGen.gate[3] <--> {  delay = 5ns; } <--> H_4.gateIDGen;
        IDGen.gate[4] <--> {  delay = 5ns; } <--> H_5.gateIDGen;

}

network FABRIC extends h5_s4
{
    parameters:
}
```

### A.1.2 The fdbs file

```
file.fdbs

0: 255 1 2 0 0 3
1: 255 2 1 0 0 2
2: 255 0 0 1 2 0
3: 255 0 0 2 1 0
```

### A.1.3 The fdbs.ini file

```
file.fdbs.ini

 [General]
**.SW1.fdbIndex = 0
**.SW1.DRGroups = " "
**.SW2.fdbIndex = 1
**.SW2.DRGroups = " "
**.SW3.fdbIndex = 2
**.SW3.DRGroups = " "
**.SW4.fdbIndex = 3
**.SW4.DRGroups = " "
```

### A.1.4 The .ini file

```
The .ini file

# This file is shared by all ib_credits simulations.
# Lines beginning with '#' are comments

[General]

sim-time-limit = ${simtime=0.5}s # max number of simulation seconds to run
network = FABRIC # this line is for Cmdenv
print-undisposed = false
debug-on-errors = true

cmdenv-express-mode = true
#cmdenv-express-mode = false
# cmdenv-runs-to-execute=1

##########################################################################
# "speed"
##########################################################################

**width = 4 # link width 4x, 8x, or 12x
```

```
**speed = 5.0 # link speed 2.5, 5.0, or 10.0
# interface width, should we set to match the link width
#**vlarb.width = 4 # switch core frequency should be 200, 400, and 800 for SDR, DDR, and
**SW**vlarb.coreFreq_MH = 800
#**.SW**.port[0].obuf.width = 4
#**.SW**.port[0].ibuf.width = 4
# the next one is not used(?)
#**L**vlarb.coreFreq_MH = 200
# HCAs frequency is the wire frequency, should be 125, 250, and 500 for SDR, DDR, and QDR
**H_**vlarb.coreFreq_MH = 250

######################################################################
# GENERATOR
######################################################################
**.gen.floodVLs = "0"
**.gen.dstHotSpotPerc = 0

**.H_1.gen.genStartTime = 0s
**.H_2.gen.genStartTime = 1s
**.H_3.gen.genStartTime = 2s
**.H_5.gen.genStartTime = 3s

#**.H_1.gen.slowStart = 0
#**.H_1.gen.genFullTime = 0.1s  # time after genStartTime to be at full

#never start these
**.H_4.gen.genStartTime = 10s # relates to sim time - must be higher to avoid traffic gen

#################
# DESTINATION
#################


# possible values are: dstLid, dstSeqOnce, dstSeqLoop, dstRandom, dstHotSpot, dstBurstHot
**.dstSeqMode = "dstLid"
# send by the given dstLid parameter
**H_1**.srcLid = 1
**H_2**.srcLid = 2
**H_3**.srcLid = 3
**H_4**.srcLid = 4
**H_5**.srcLid = 5


**H_1**.dstLid = 4
**H_2**.dstLid = 4
**H_3**.dstLid = 4
**H_4**.dstLid = 1
**H_5**.dstLid = 3
```

```
#################
# TRAFFIC
#################
**.gen.trafficDist = "trfFlood"
**.gen.dstSeqVecFile = ""
**.gen.sizeSeqVecFile = ""
**.gen.dstSeqIndex = 0

###########################################################################
# SWITCH
###########################################################################
# generated file holding port groups and FDBs vector indexes
**.SW**.fdbsVecFile = "h4_s4.fdbs"
include h4_s4.fdbs.ini

#################
# TESTING (Tuning)
#################
#**gen.srcLid = 1
**gen.burstLength = intuniform(20,100)
**gen.burstNumber = 128
**gen**.interBurstDelay = 0

###########################################################################
# IBUF
###########################################################################
**H**.ibuf.maxStatic* = ${hibuf=1000}
**H**.ibuf.totalBufferSize = ${hibuf}
#**ibuf.maxStatic0 = 96 # in credits
**ibuf.maxStatic* = ${sibuf=1000} #
**ibuf.totalBufferSize = ${sibuf} # in credits
# speedup
**ibuf.maxBeingSent = 1

**.Victim_Mask = 1

#####################################
# RUNS
#####################################
**.logInterval = ${logint=1}ms #5
#**.vlarb.recordVectors = 0
**.recordVectors = 0
include ../../src/modules.ini

[Config flood]
**gen**.trafficDist = "trfFlood"
description = "Simple Run"
```

```
#fast interswitch link:
**SW**.port[0].obuf.width = 4
**SW**.port[0].obuf.speed = 10.0 #toban ayay ahayd
**SW**.port[0].ibuf.maxBeingSent = 2
**SW**.port[0].ccmgr.Victim_Mask = 0
**SW**vlarb.coreFreq_MH = 800
#**SW**.port[0].ibuf.totalBufferSize = ${swsw=2000}
#**SW**.port[0].ibuf.maxStatic* = ${swsw}
#**.SW2.port[3].ibuf.totalBufferSize = ${redbuf=500}
#**.SW2.port[3].ibuf.maxStatic* = ${redbuf}
#**.SW2.port[4].ibuf.totalBufferSize = ${redbuf}
#**.SW2.port[4].ibuf.maxStatic* = ${redbuf}
#---
# start of hosts:
#**.H_1.gen.genStartTime = 10s
**.H_2.gen.genStartTime = 0.1s
**.H_3.gen.genStartTime = 0.2s
**.H_4.gen.genStartTime = 0.6s
**.H_5.gen.genStartTime = 0.3s

#---
# Gen speed
**gen**.interBurstDelay = 0 # Update this one to slow down a sender...
**gen**.intraBurstDelay = ${intrabd=26}
**gen**.dstHotSpotPerc = 0
#---
# Sink must not be too efficient
**sink.pciExpWidth = 8
**sink.pciExpTransferRate = 5 # 2.5 equals PCI 1.1, 5.0 equals PCI 2.0, and 8.0 PCI 3.0
**sink.hiccupDuration_us = ${hdur=0.0125}
#**.H_5.sink.hiccupDelay_us = 0.001
**sink.hiccupDelay_us = 0.1
#---
# Msg and packet sizes:
**gen**.msgLength = 65536 # message length in number of bytes
**gen**.msgLengthInMTUs = 32 #message length in number of mtus
**mtu = 34 # number of credits in one mtu
**gen**.packetlengthbytes = 2176 # packetlength in bytes
**gen**.packetlength = 34 # packetlength in number of credits
**gen**.floodVLs = "0"
**gen**.PktLenDist = "2176"
**gen**.PktLenProb = "100"
#---
#slow node
#**SW2**.port[2].**.speed = 2.5
#**SW2**.port[2].**.width = 1
#---
#dynamic hotspot support
```

```
#**.dynamicHSFile = ${dyn=true}  # no file provided
#**.dynHSSeqVecFile = "dynHS"
#**.gen.dstSeqVecFile = "dstSeq"
#**.dstSeqMode = "dstHotSpotR"
**.H_1.gen.dstSeqIndex = 4
**.H_2.gen.dstSeqIndex = 4
**.H_3.gen.dstSeqIndex = 4
**.H_4.gen.dstSeqIndex = 1
**.H_5.gen.dstSeqIndex = 3

#---
# CC:
**.cc_enabled = true
**.cc_extended_ibuf_log = false
**.qp_level = ${qp=true}
**.cc_single_thr_per_outport = ${singleThr=true}
**.cc_single_thr_devide = ${singleThrDev=false}
#**.SW**.port[0]**.Threshold = ${SWThres=8..13 step 4}
**.Threshold = ${thr=2} #15
# REMARK: Reset_Threshold needs to be HIGHER than Threshold for the settings to m
**.Reset_Threshold = ${thrlow=3}
#**.Reset_Threshold = ${ResetTH=0..13 step 4}
**.Marking_Rate = ${mr=1}
**.Packet_Size = 1 # originaly 8
**.CCTI_Timer = ${cctitimer=20}us
**.CCTI_Limit = 128
**.CCTI_Min = 0
**.CCTI_Increase = ${cctiinc=1}
**.CCT_Algorithm = 0 # 1
**.CCT_Div = ${cctdiv=95}
**.num_nodes = ${nn=10} #..70 step 10} # value originaly 50
output-vector-file = C:/cc1/1-20/dynHS/${resultdir}/${configname}-AllRandom-dynHS
output-scalar-file = C:/cc1/1-20/dynHS/${resultdir}/${configname}-AllRandom-dynHS
#output-vector-file = d:/tempresults/dynHS/${resultdir}/${configname}-NoCC-AllRan
#output-scalar-file = d:/tempresults/dynHS/${resultdir}/${configname}-NoCC-AllRan
#output-vector-file = ${resultdir}/fairness2xThr/${configname}-ib${sibuf}_${hibuf
#output-scalar-file = ${resultdir}/fairness2xThr/${configname}-ib${sibuf}_${hibuf
```

## A.2 More scenarios of IB CC parameters

### A.2.1 Scenario 5

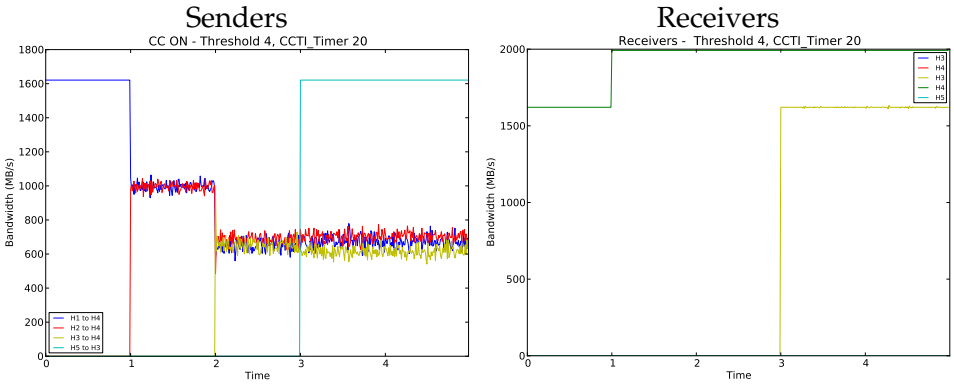



Figure A.1: Threshold=4, Timer=20

Figure A.5: Threshold=4, Timer=20

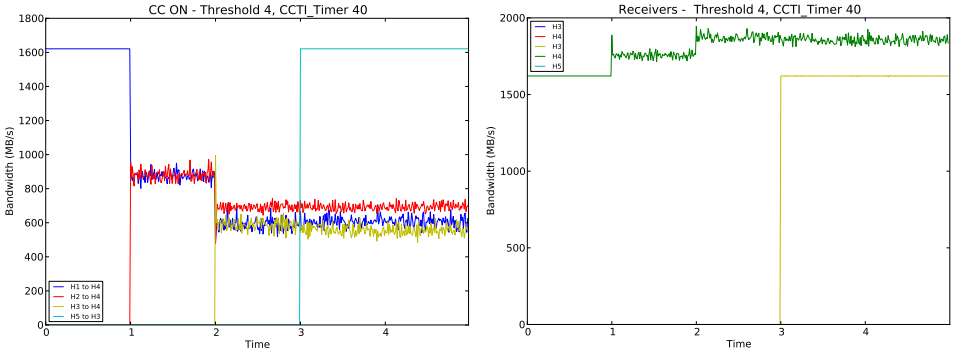



Figure A.2: Threshold=4, Timer=40

Figure A.6: Threshold=4, Timer=40

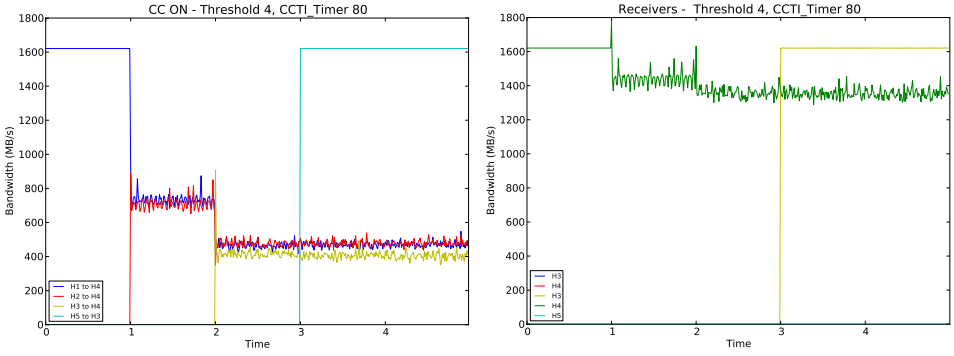
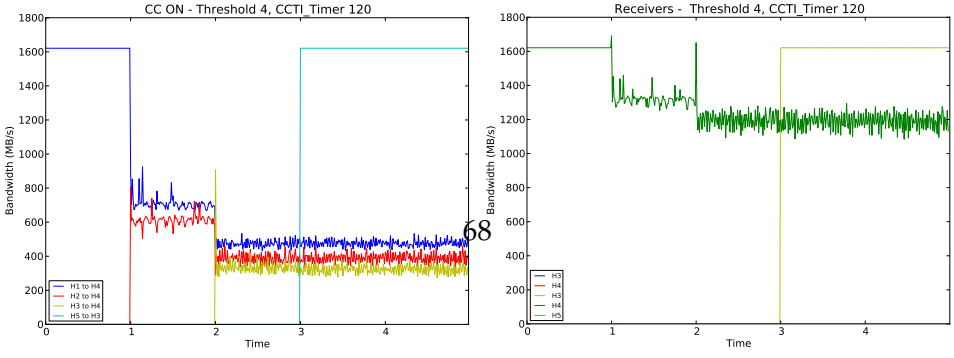



Figure A.3: Threshold=4, Timer=80

Figure A.7: Threshold=4, Timer=80



### A.2.2 Scenario 6

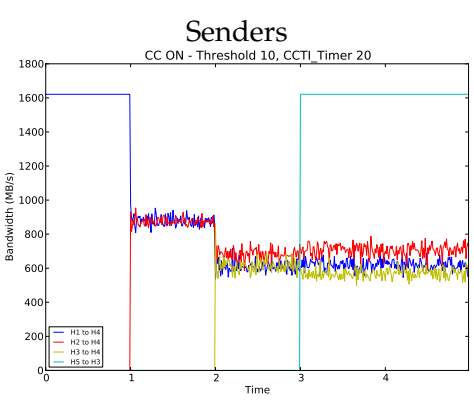



Figure A.9: Threshold=10, Timer=20

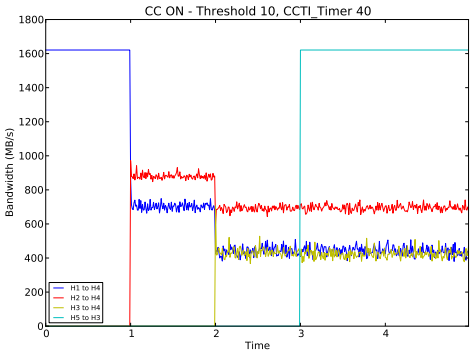



Figure A.10: Threshold=10, Timer=40

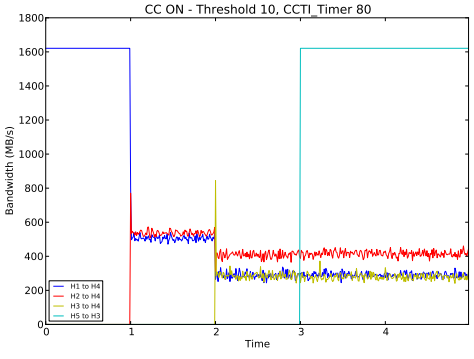
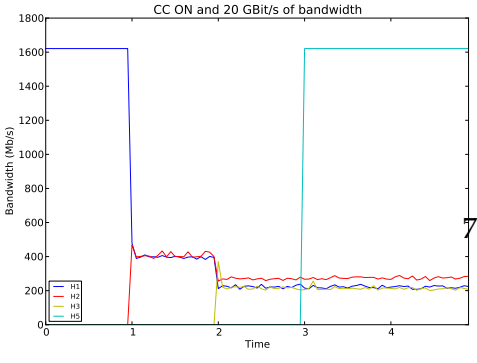



Figure A.11: Threshold=10, Timer=80




Receivers

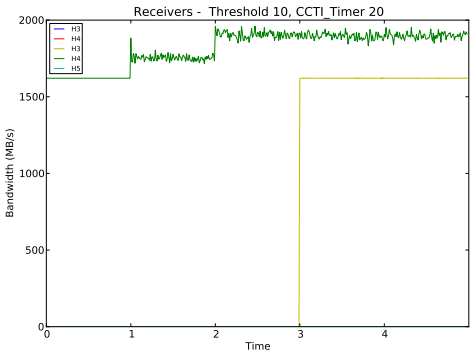



Figure A.13: Threshold=10, Timer=20

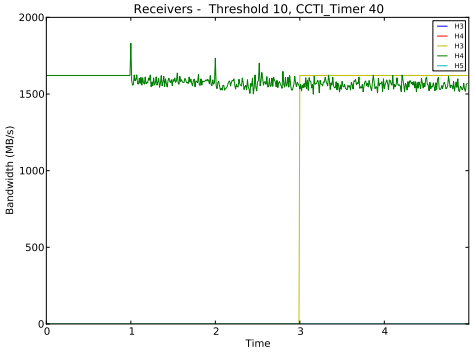



Figure A.14: Threshold=10, Timer=40

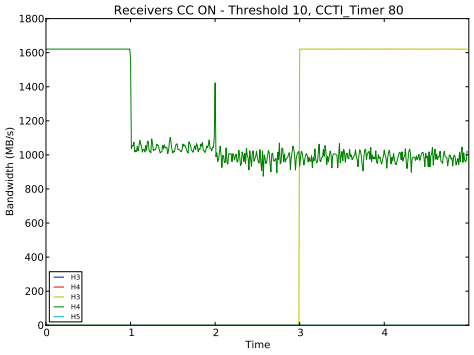
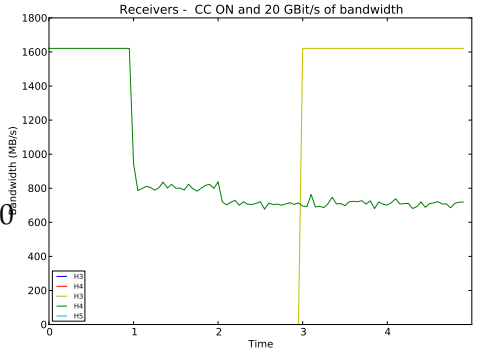



Figure A.15: Threshold=10, Timer=80



## A.3 Extracting and graph plotting scripts

### A.3.1 For the sender graph

```
#! /usr/bin/python

import re
import matplotlib
matplotlib.use("Agg")
from matplotlib import pylab as plt

def extractLines():
f=open("8-120.vec")
lines=f.readlines()

H1pattern = r'''^(?P<H1>(1\s+))'''
H2pattern = r'''^(?P<H2>(724\s+))'''
H3pattern = r'''^(?P<H3>(1447\s+))'''
H4pattern = r'''^(?P<H4>(2170\s+))'''
H5pattern = r'''^(?P<H5>(2893\s+))'''

H1lines = re.compile(H1pattern, re.VERBOSE)
H2lines = re.compile(H2pattern, re.VERBOSE)
H3lines = re.compile(H3pattern, re.VERBOSE)
H4lines = re.compile(H4pattern, re.VERBOSE)
H5lines = re.compile(H5pattern, re.VERBOSE)

for line in lines:
H1result = H1lines.match(line)
H2result = H2lines.match(line)
H3result = H3lines.match(line)
H4result = H4lines.match(line)
H5result = H5lines.match(line)


if H1result or H2result or H3result or H4result or H5result:
file=open("cc1floodlines.vci","a")
file.write(line)
extractLines()

def extractValues():
f=open("cc1floodlines.vci")
        lines=f.readlines()

H1pattern = r'''^(1\s+\d+\s+(?P<H1time>\d+\.\d+)\s+(?P<H1speed>\d+))'''
        H2pattern = r'''^(724\s+\d+\s+(?P<H2time>\d+\.\d+)\s+(?P<H2speed>\d+))'''
        H3pattern = r'''^(1447\s+\d+\s+(?P<H3time>\d+\.\d+)\s+(?P<H3speed>\d+))'''
        H4pattern = r'''^(2170\s+\d+\s+(?P<H4time>\d+\.\d+)\s+(?P<H4speed>\d+))'''
```

```
        H5pattern = r'''^(2893\s+\d+\s+(?P<H5time>\d+\.\d+)\s+(?P<H5speed>\d+))''

H1values = re.compile(H1pattern, re.VERBOSE)
H2values = re.compile(H2pattern, re.VERBOSE)
H3values = re.compile(H3pattern, re.VERBOSE)
H4values = re.compile(H4pattern, re.VERBOSE)
H5values = re.compile(H5pattern, re.VERBOSE)

       gb = 1000000

groupdict={}
valueDict={}
for line in lines:
                H1result = H1values.match(line)
H2result = H2values.match(line)
H3result = H3values.match(line)
H4result = H4values.match(line)
H5result = H5values.match(line)


if H1result:
groupdict=H1result.groupdict()
valueDict.setdefault('H1speed',[]).append(int(groupdict['H1speed'])/gb)
valueDict.setdefault('H1time',[]).append(float(groupdict['H1time']))
elif H2result:
                        groupdict=H2result.groupdict()
                        valueDict.setdefault('H2speed',[]).append(int(groupdict['
                        #valueDict.setdefault('H2time',[]).append(float(groupdict
      elif H3result:
                        groupdict=H3result.groupdict()
                        valueDict.setdefault('H3speed',[]).append(int(groupdict['
elif H4result:
                        groupdict=H4result.groupdict()
                        valueDict.setdefault('H4speed',[]).append(int(groupdict['
elif H5result:
                        groupdict=H5result.groupdict()
                        valueDict.setdefault('H5speed',[]).append(int(groupdict['

return valueDict
extractValues()

def plotGraph():
values=extractValues()
print values

H1 = values['H1speed']
     H2 = values['H2speed']
     H3 = values['H3speed']
```

```
     H4 = values['H4speed']
     H5 = values['H5speed']

H1time = values['H1time']

     #Draw Graph
     fig, ax = plt.subplots()
     x=xrange(len(H1time))
     LABELS = H1time

plt.plot(x, H1, color='b', linewidth=1.0, label='H1 to H4')
     plt.plot(x, H2, color='r', linewidth=1.0, label='H2 to H4')
     plt.plot(x, H3, color='y', linewidth=1.0, label='H3 to H4')
#     plt.plot(x, H4, color='g', linewidth=1.0, label='H4')
      plt.plot(x, H5, color='c', linewidth=1.0, label='H5 to H3')


#plt.xticks(x, LABELS, rotation=45)
     plt.xticks(xrange(0, len(LABELS), 100), xrange(0, 5, 1))
     ax.set_xlim([0, len(LABELS)-1])
#        ax.set_ylim([0,18.0])
     plt.xlabel('Time')
     plt.ylabel('Bandwidth (MB/s)')
     plt.title('CC ON - Threshold 8, CCTI_Timer 120')
     plt.legend(loc='best', prop={'size':8})
     plt.subplots_adjust(bottom=0.2)
     plt.tight_layout()
     plt.savefig('cc1-8-120.pdf',bbox_inches='tight')
plotGraph()
```

### A.3.2 For the receiver graph

```
#! /usr/bin/python

import re
import matplotlib
matplotlib.use("Agg")
from matplotlib import pylab as plt

def extractLines():
f=open("10-80.vec")
lines=f.readlines()

H1pattern = r'''^(?P<H1>(4\s+))'''
H2pattern = r'''^(?P<H2>(727\s+))'''
H3pattern = r'''^(?P<H3>(1450\s+))'''
H4pattern = r'''^(?P<H4>(2173\s+))'''
H5pattern = r'''^(?P<H5>(2896\s+))'''

H1lines = re.compile(H1pattern, re.VERBOSE)
H2lines = re.compile(H2pattern, re.VERBOSE)
H3lines = re.compile(H3pattern, re.VERBOSE)
H4lines = re.compile(H4pattern, re.VERBOSE)
H5lines = re.compile(H5pattern, re.VERBOSE)

for line in lines:
H1result = H1lines.match(line)
H2result = H2lines.match(line)
H3result = H3lines.match(line)
H4result = H4lines.match(line)
H5result = H5lines.match(line)


if H1result or H2result or H3result or H4result or H5result:
file=open("sink-cc1floodlines.vci","a")
file.write(line)
extractLines()

def extractValues():
f=open("sink-cc1floodlines.vci")
        lines=f.readlines()


H1pattern = r'''^(4\s+\d+\s+(?P<H1time>\d+\.\d+)\s+(?P<H1speed>\d+))'''
        H2pattern = r'''^(727\s+\d+\s+(?P<H2time>\d+\.\d+)\s+(?P<H2speed>\d+))'''
        H3pattern = r'''^(1450\s+\d+\s+(?P<H3time>\d+\.\d+)\s+(?P<H3speed>\d+))'''
        H4pattern = r'''^(2173\s+\d+\s+(?P<H4time>\d+\.\d+)\s+(?P<H4speed>\d+))'''
        H5pattern = r'''^(2896\s+\d+\s+(?P<H5time>\d+\.\d+)\s+(?P<H5speed>\d+))'''
```

```
H1values = re.compile(H1pattern, re.VERBOSE)
H2values = re.compile(H2pattern, re.VERBOSE)
H3values = re.compile(H3pattern, re.VERBOSE)
H4values = re.compile(H4pattern, re.VERBOSE)
H5values = re.compile(H5pattern, re.VERBOSE)

       gb = 1000000
ten = 1

groupdict={}
valueDict={}
for line in lines:
                H1result = H1values.match(line)
H2result = H2values.match(line)
H3result = H3values.match(line)
H4result = H4values.match(line)
H5result = H5values.match(line)


if H1result:
groupdict=H1result.groupdict()
valueDict.setdefault('H1speed',[]).append(int(groupdict['H1speed'])/gb)
valueDict.setdefault('H1time',[]).append(float(groupdict['H1time'])*ten)
elif H2result:
                        groupdict=H2result.groupdict()
                        valueDict.setdefault('H2speed',[]).append(int(groupdict['H2speed'
                        valueDict.setdefault('H2time',[]).append(float(groupdict['H2time'
      elif H3result:
                        groupdict=H3result.groupdict()
                        valueDict.setdefault('H3speed',[]).append(int(groupdict['H3speed'
valueDict.setdefault('H3time',[]).append(float(groupdict['H3time'])*ten)
elif H4result:
                        groupdict=H4result.groupdict()
                        valueDict.setdefault('H4speed',[]).append(int(groupdict['H4speed'
valueDict.setdefault('H4time',[]).append(float(groupdict['H4time'])*ten)
elif H5result:
                        groupdict=H5result.groupdict()
                        valueDict.setdefault('H5speed',[]).append(int(groupdict['H5speed'
valueDict.setdefault('H5time',[]).append(float(groupdict['H5time'])*ten)

return valueDict
extractValues()

def plotGraph():
values=extractValues()
print values
```

```
H1 = values['H1speed']
     H2 = values['H2speed']
     H3 = values['H3speed']
     H4 = values['H4speed']
     H5 = values['H5speed']

H1time = values['H1time']
H2time = values['H2time']
  H3time = values['H3time']
         H4time = values['H4time']
         H5time = values['H5time']

#Draw Graph
     fig, ax = plt.subplots()
     x=xrange(len(H1time))
x1=xrange(len(H2time))
x2=xrange(len(H3time))
         x3=xrange(len(H4time))
x4=xrange(len(H5time))
     LABELS = H1time

plt.plot(H1time, H1, color='b', linewidth=1.0, label='H3')
     plt.plot(H2time, H2, color='r', linewidth=1.0, label='H4')
     plt.plot(x, H3, color='y', linewidth=1.0, label='H3')
     plt.plot(x, H4, color='g', linewidth=1.0, label='H4')
      plt.plot(x, H5, color='c', linewidth=1.0, label='H5')


#plt.xticks(x, LABELS, rotation=45)
     plt.xticks(xrange(0, len(LABELS), 100), xrange(0, 5, 1))
  #   ax.set_xlim([0, len(LABELS)-1])
#         ax.set_ylim([0,18.0])
     plt.xlabel('Time')
     plt.ylabel('Bandwidth (MB/s)')
     plt.title('Receivers CC ON - Threshold 10, CCTI_Timer 80')
     plt.legend(loc='best', prop={'size':8})
     plt.subplots_adjust(bottom=0.2)
     plt.tight_layout()
     plt.savefig('sink-10-80.pdf',bbox_inches='tight')
plotGraph()
```