Minor changes that have been made are listed below:

1. **Address why GPS is not a solution for clock synchronization? (Page. 9-10)**

IEEE 1588-2008 PTP also provides an evolutionary step towards the deployment of the next generation synchronization architecture [15]. Next generation network (NGN) infrastructure combining traditional TDM (Time Division Multiplexing) core networks and packet-based IP backhaul networks based on Ethernet is seen as the future in the telecommunication networks. In addition, there are situations where GPS is not an appropriate synchronization source for traditional synchronization methods particularly for fast-moving objects and in-building Pico-cell applications. PTP is an excellent candidate for GPS backup and as a redundant synchronization source for CDMA/LTE Macro-cells. The PTP system allows for setting/maintaining time/phase synchronization of Radio Access Networks (RAN) such as CDMA, LTE and in-building Pico-cell synchronization. Moreover, IEEE 1588-2008 has been widely studied in various fields such as power distribution networks, wireless sensor network, telecommunications networks, and military applications [20]. The detailed functionalities of IEEE 1588-2008 including the node, system and necessary communication properties to support PTP will be discussed in the next section.

2. **Rewrite the contribution for multiple master clock (Page. 3)**

The proposed solution further extends to coordinate multiple master clocks through a single slave clock, which may be connected through multiple networks. The rationale of the extension is to support multiple master clocks instead of selecting a grandmaster clock. The proposed solution ensures that the slave clock receives at least one good offset sample from one of these

master clocks within defined update frequency interval for maintaining high synchronization accuracy between the slave and the master clocks.

3. **Why I choose NS-2 clock Model (page. 16-17)**

Network Simulator 2 (NS-2) is an open source network simulation tool [25]. NS-2 is one of the prominent tools particularly designed for conducting research in computer communications networks. The primary use of NS-2 is to simulate various types of network protocols such as TCP, UDP, multicast protocols, routing algorithms, traffic source behavior, queue management mechanisms, etc. over wired and wireless environments. It is an object-oriented, discrete event driven simulator written in C++ and the Otcl language [36]. We choose NS-2 because it facilitates to study network effects elaborately such as network congestion, temporary network outage, network path reconfigurations and many more, which are very difficult to model in other simulation tools, for instance MatLab.

Furthermore, other simulation tools do not have a clock model which is affected by temperature and aging effects. As described in Section 1 of this chapter, a clock oscillator starts to drift due to the environmental changes such as temperature and aging effect. Keeping this in mind, a clock agent is implemented by a prior student in NS2 via a C++ class hierarchy. Our proposed work extends the current model of such a clock agent. The details of the implementation of such a clock agent can be found in [4].

4. **Clear the conflict of the name ‘ R’ test and the statistical test of the same name (Page 35)**

It is worth mentioning that the term ‘R’ test mentioned in the thesis does not refer to the statistical test of the same name.

5. **Justify the boundary condition ? Page. 39**

We believe that it is possible to achieve high synchronization accuracy of the slave clock in the presence of bursty traffic by using 32% of all samples within 3% variance. Since the distribution is plotted under high traffic load, there is a good possibility to receive more samples under less traffic loads.

6. **A summary of contributions is added in the end of Chapter 4.**

This chapter proposes a Delay Asymmetry Correction (DAC) Model to enhance the traditional IEEE 1588 synchronization protocol for asymmetric communication links. The proposed solutions are summarized as follows:

- The DAC model is proposed to achieve high synchronization accuracy by determining the correct offset value in a slave clock for asymmetric communication link delays. The initiative revolves around the idea of incorporating the DAC model with the conventional IEEE 1588 synchronization protocol. The proposed DAC model relies on two consecutive filtering methods named as the 'R' test and Update sample filter, which make sure that only good samples are used to update the slave clock.
- The filtering process of the proposed work does not only filter out bad samples, it also saves a notion of good updates for calculating a saved offset value. The latter value is used when the slave clock does not receive (good) samples from the IEEE 1588 protocol.
- Furthermore, the proposed solution further extends to support multiple master clocks updating a single slave clock. The rational of the extension is to support multiple master clocks instead of selecting a grand master clock. To do so, a new equation is developed. The solution suggests that the master clocks will initiate IEEE 1588 message exchange according to that equation for achieving high synchronization accuracy between the slave and the master clocks.
- The proposed solution also integrates the Adaptive Oscillator Correction Model (AOCM) in order to compensate temperature and aging effects of the oscillator and hence, to improve the stability of the slave clock during holdover mode.

7. **Temperature: Give reference when listing temperature coefficients. Discuss why temperature ranges of 60 degrees make sense outdoors. Also explain why the temperature profile has three cycles?**

The temperature profile represents the temperature values (i.e. coefficients) observed at the clock oscillator [4, 26]. The range of the temperature variation is 60°C over an 8 hour cycle. We are using 3 cycles of temperature variations to study the effect of temperature over shorter simulation time periods. Thus, we artificially condensed the daily temperature cycle. The temperature range is large enough to represent a real working environment. For instance, if a BTS located in a desert, the average temperature might be very hot in day time. The 8 hour cycle guarantees that the simulation results are obtained fast enough.

8. **Performance result: Why no comparison to other solutions? (Page 59)**

From Figure 11, the result shows that the maximum of the maximum slave clock difference is about 99.80 nanoseconds level. The resultant difference implies that the slave clock accuracy improved significantly in the presence of heavy traffic load in the network, much lower than the one shown in Figure 10. It is worth mentioning that we are not comparing the performance of the proposed DAC model with other solutions except IEEE 1588 message sequence only as shown in Figure 10. Because if we get the slave synchronization accuracy to 100 ns, that is as good as we can make it considering given parameters (i.e. 1 second synchronization interval, 100 ppb faster drift of the slave clock w.r.t the master clock). Thus, every time we get a result close to 100 ns, we got the best possible performance of the proposed DAC model.

9. **TCL Scripts in Appendix A**

Appendices

NS-2 TCL Examples

The NS-2 TCL scripts presented in this appendix use a master node n0 connected with a slave node n3 with two intermediate nodes n1 and n2, resulting in the 3 hops topology as shown in Figure 8. Two traffic sources are also introduced. One of the traffic sources is node n4, which sends traffic to node n5 and vice versa. All the clocks used have no initial offsets in the rate i.e. initially all tick at the same time as the reference clock (simulation time), but they are not drifting at the same rate. Clock m1 is a master clock agent connected with node n0 and clock s1 is a slave clock agent connected with node n3. The slave clock s1 is drifting 100 ppb faster than the master clock m1. No temperature and aging effects are enabled in the clock agents. The detail descriptions of the clock parameters can be found in [4]. The traffic profile described in Section 5.3.1.1 is implemented in the scripts. Node n4 generates 80% static packet load, which sends traffic to node n5 (i.e. Master-to-Slave). Similarly, node n5 generates 20% static packet load, which sends traffic to node n4 (Slave-to-Master).

NS-2 TCL Script Code for Data Centric Traffic Model

In this test case, a TCL script for data centric traffic model, which is described in Section 5.1.1.1, is presented with the static traffic load described in Section 5.3.1.1.

NS-2 TCL Script

```
set ns [new Simulator]

#Open a trace file

set tracefd [open clktest1d.tr w]
$ns trace-all $tracefd
set nf [open clktest1d.nam w]
$ns namtrace-all $nf
set graphData [open graphData.txt w]

#Define a 'finish' procedure

proc finish {} {
        global ns graphData nf tracefd
        $ns flush-trace
        close $nf
        close $tracefd
        close $graphData
        exit 0
}

#Create nodes

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

#Link configurations

$ns duplex-link $n0 $n1        1Mb    100ns   DropTail
$ns duplex-link $n1 $n2        1Mb    100ns   DropTail
$ns duplex-link $n2 $n3        1Mb    100ns   DropTail
$ns duplex-link $n1 $n4        1Mb    100ns   DropTail
$ns duplex-link $n2 $n5        1Mb    100ns   DropTail

#Define a 'timeout' function for the class 'Agent/Clock'
```

```
Agent/Clock instproc timeout {ts rate eff_rate} {
# NOTE: rate is natural rate of clock and eff_rate is effective rate (after applying RLS correction) of the
clock
        global ns graphData
        $self instvar node_
        variable nodeID [$node_ id]
        puts $graphData "[$node_ id]  $eff_rate $ts [$ns now] "
}



# Generate bursts lasting between 0.1 s to 3 s

set rng1 [new RNG]
$rng1 seed [lindex $argv 0]
set burstDuration [new RandomVariable/Uniform]
$burstDuration set min_ 0.1
$burstDuration set max_ 3
$burstDuration use-rng $rng1


# ***** 80% "Static" packet load in the Forward Direction (Master-to-Slave)*****


# 60% of the load based on maximum sized packet (1518 bytes) generated at node n4, which sends
traffic to node n5


#Create a UDP agent and attach it to node n4

set udp4a [new Agent/UDP]
$udp4a set packetSize_ 1518
$ns attach-agent $n4 $udp4a


# Create a CBR traffic source, which will generate bursty traffic and attach it to udp4a

set cbr4a [new Application/Traffic/CBR]
$cbr4a set packetSize_ 1518
$cbr4a set rate_ 0.96Mb
$cbr4a attach-agent $udp4a


#Create a Null agent (a traffic sink) and attach it to node n5

set null5 [new Agent/Null]
$ns attach-agent $n5 $null5


#Connect the traffic source with the traffic sink

$ns connect $udp4a $null5
```

 # 30% of the load based on small sized packet (64 bytes) generated at node n4, which sends traffic to node n5

#Create a UDP agent and attach it to node n4

```
set udp4b [new Agent/UDP]
$udp4b set packetSize_ 64
$ns attach-agent $n4 $udp4b
```

Create a CBR traffic source and attach it to udp4a

```
set cbr4b [new Application/Traffic/CBR]
$cbr4b set packetSize_ 64
$cbr4b set rate_ 0.24Mb
$cbr4b attach-agent $udp4b
```

#Connect the traffic source with the traffic sink

```
$ns connect $udp4b $null5
```

 # 10% of the load based on medium sized packet (576 bytes) generated at node n4, which sends traffic to node n5

#Create a UDP agent and attach it to node n4

```
set udp4c [new Agent/UDP]
$udp4c set packetSize_ 576
$ns attach-agent $n4 $udp4c
```

Create a CBR traffic source and attach it to udp4c

```
set cbr4c [new Application/Traffic/CBR]
$cbr4c set packetSize_ 576
$cbr4c set rate_ 0.08Mb
$cbr4c attach-agent $udp4c
```

#Connect the traffic source with the traffic sink

```
$ns connect $udp4c $null5
```

 # ***20% "Static" packet load in the Reverse Direction (Slave-to- Master)**

 # 60% of the load based on maximum sized packet (1518 bytes) generated at node n5, which sends traffic to node n4

#Create a UDP agent and attach it to node n5

```
set udp5a [new Agent/UDP]
$udp5a set packetSize_ 1518
$ns attach-agent $n5 $udp5a
```

Create a CBR traffic source, which will generate bursty traffic and attach it to udp5a

```
set cbr5a [new Application/Traffic/CBR]
$cbr5a set packetSize_ 1518
$cbr5a set rate_ 0.48Mb
$cbr5a attach-agent $udp5a
```

#Create a Null agent (a traffic sink) and attach it to node n4

```
set null4 [new Agent/Null]
$ns attach-agent $n4 $null4
```

#Connect the traffic source with the traffic sink

```
$ns connect $udp5a $null4
```

 # 30% of the load based on small sized packet (64 bytes) generated at node n5, which sends traffic to node n4

#Create a UDP agent and attach it to node n5

```
set udp5b [new Agent/UDP]
$udp5b set packetSize_ 64
$ns attach-agent $n5 $udp5b
```

Create a CBR traffic source and attach it to udp5b

```
set cbr5b [new Application/Traffic/CBR]
$cbr5b set packetSize_ 64
$cbr5b set rate_ 0.06Mb
$cbr5b attach-agent $udp5b
```

#Connect the traffic source with the traffic sink

```
$ns connect $udp5b $null4
```

 # 10% of the load based on medium sized packet (576 bytes) generated at node n5, which sends traffic to node n4

```
#Create a UDP agent and attach it to node n5
set udp5c [new Agent/UDP]
$udp5c set packetSize_ 576
$ns attach-agent $n5 $udp5c

# Create a CBR traffic source and attach it to udp5c
set cbr5c [new Application/Traffic/CBR]
$cbr5c set packetSize_ 576
$cbr5c set rate_ 0.02Mb
$cbr5c attach-agent $udp5c

#Connect the traffic source with the traffic sink
$ns connect $udp5c $null4

#Create master clock agent
set m1 [new Agent/Clock]
$m1 set offset 1
$m1 set rate 1
$m1 set timeToDisplayInfo 1
$m1 set masterClock 1
$m1 set gpsSignal 0
$m1 set enableRLS 0
$m1 set enableLockedRLS 0
$m1 set RLSFreq 0
$m1 set tempProfileName -1
$m1 set ageing 0
$m1 set driftInterval 1
$m1 set enable1588Logs 1
$ns attach-agent $n0 $m1

# Create a slave clock agent
set s1 [new Agent/Clock]
$s1 set offset 1
#slave clock rate is 100 ppb (100 ns) faster than the master clock
$s1 set rate 1.000000100
$s1 set timeToDisplayInfo 1
$s1 set masterClock 0
$s1 set gpsSignal 0
$s1 set enableLockedRLS 0
$s1 set enableRLS 0
$s1 set RLSFreq 0
$s1 set tempProfileName -1
$s1 set driftInterval 1
$s1 set timeStampReqFreq 1
$s1 set masterAddr 0
$s1 set masterPort 0
$s1 set enable1588Logs 1
```

```
$s1 set enable1588Delays 0
$ns attach-agent $n3 $s1
```

#***Schedule events: Traffic in the Forward Direction (Master-to-Slave)*****

```
set rng [new RNG]
$rng seed [lindex $argv 0]
set u [new RandomVariable/Uniform]
$u set min_ 0
$u set max_ 1
$u use-rng $rng

#Schedule events for bursty traffic as cbr4a agents
set simTime4a_ 1
while {$simTime4a_  < 86400.00 } {
        set burstTime4a_ [$burstDuration value]
        set startTime [expr [expr ([$u value]) + {$simTime4a_ }]]
        $ns at  $startTime "$cbr4a start"
        set simTime4a_ [expr $startTime + $burstTime4a_ ]
        #puts " CBR Traffic Stops at $simTime4a_"
        $ns at $simTime4a_ "$cbr4a stop"
        set simTime4a_ [expr {$simTime4a_ + $burstTime4a_}]
}

#Schedule events for cbr4b and cbr4c agents will continue until 24 hours
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr4b start"
$ns at 86400.00 "$cbr4b stop"
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr4c start"
$ns at 86400.00 "$cbr4c stop"
```

#***Schedule events: Traffic in the Reverse Direction (Slave-to-master)*****

```
#Schedule events for bursty traffic as cbr5a agents
set simTime5a_ 1
while {$simTime5a_ < 86400.00 } {
        set burstTime5a_ [$burstDuration value]
        #puts [format " Burst Time at Node n5 is %-4.3f  " $burstTime5a_]
        set startTime [expr [expr ([$u value]) + {$simTime5a_ }]]
        $ns at  $startTime "$cbr5a start"
        set simTime5a_ [expr $startTime + $burstTime5a_ ]
        $ns at $simTime5a_ "$cbr5a stop"
        set simTime5a_ [expr {$simTime5a_ + $burstTime5a_}]
}

#Schedule events for cbr5b and cbr5c agents will continue until 24 hours
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr5b start"
$ns at 86400.00 "$cbr5b stop"
```

```
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr5c start"
$ns at 86400.00 "$cbr5c stop"


#Schedule events for clock agents
$ns at 1.0 "$m1 start"
$ns at 1.0 "$s1 start"

$ns at 86401.05 "$m1 stop"
$ns at 86401.05 "$s1 stop"
$ns at 86401.1 "finish"


#Run the simulation
$ns run
```

NS-2 TCL Script Code for Voice Centric Traffic Model

In this test case, a TCL script for voice centric traffic model, which is described in Section 5.1.1.2, is presented with the static traffic load described in Section 5.3.1.1. The results of this test case are discussed in Appendix C.

NS-2 TCL Script

```
set ns [new Simulator]


#Open a trace file
set tracefd [open clktest1v.tr w]
$ns trace-all $tracefd
set nf [open clktest1v.nam w]
$ns namtrace-all $nf
set graphData [open graphData.txt w]




#Define a 'finish' procedure
proc finish {} {
        global ns graphData nf tracefd
        $ns flush-trace
        close $nf
        close $tracefd
        close $graphData
        #exec nam clktest1v.nam &
        exit 0
}


#Create nodes
```

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]


#Link configurations
$ns duplex-link $n0 $n1          1Mb     100ns   DropTail
$ns duplex-link $n1 $n2          1Mb     100ns   DropTail
$ns duplex-link $n2 $n3          1Mb     100ns   DropTail
$ns duplex-link $n1 $n4          1Mb     100ns   DropTail
$ns duplex-link $n2 $n5          1Mb     100ns   DropTail


#Define a 'timeout' function for the class 'Agent/Clock'
Agent/Clock instproc timeout {ts rate eff_rate} {
        global ns graphData
        $self instvar node_
        variable nodeID [$node_ id]
        puts $graphData "[$node_ id]  $eff_rate $ts [$ns now] "
}


# Generate bursts lasting between 0.1 s to 3 s
set rng1 [new RNG]
$rng1 seed [lindex $argv 0]
set burstDuration [new RandomVariable/Uniform]
$burstDuration set min_ 0.1
$burstDuration set max_ 3
$burstDuration use-rng $rng1


# ****** 80% traffic load in the forward direction (Master-to-slave) ******


# 15% of the load based on maximum sized packet (1518 bytes) generated at node n4, which sends traffic to node n5
#Create a UDP agent and attach it to node n4
set udp4a [new Agent/UDP]
$udp4a set packetSize_ 1518
$ns attach-agent $n4 $udp4a


# Create a CBR traffic source, which will generate bursty traffic and attach it to udp4a
set cbr4a [new Application/Traffic/CBR]
$cbr4a set packetSize_ 1518
$cbr4a set rate_ 0.24Mb
$cbr4a attach-agent $udp4a
```

```
#Create a Null agent (a traffic sink) and attach it to node n5
set null5 [new Agent/Null]
$ns attach-agent $n5 $null5


#Connect the traffic source with the traffic sink
$ns connect $udp4a $null5


# 80% of the load based on minimum sized packet (64 bytes) generated at node n4, which sends traffic to node n5


#Create a UDP agent and attach it to node n4
set udp4b [new Agent/UDP]
$udp4b set packetSize_ 64
$ns attach-agent $n4 $udp4b


# Create a CBR traffic source, which will generate bursty traffic and attach it to udp4a
set cbr4b [new Application/Traffic/CBR]
$cbr4b set packetSize_ 64
$cbr4b set rate_ 0.64Mb
$cbr4b attach-agent $udp4b


#Connect the traffic source with the traffic sink
$ns connect $udp4b $null5


# 5% of the load based on medium sized packet (576 bytes) generated at node n4, which sends traffic to node n5


#Create a UDP agent and attach it to node n4
set udp4c [new Agent/UDP]
$udp4c set packetSize_ 576
$ns attach-agent $n4 $udp4c


# Create a CBR traffic source and attach it to udp4c
set cbr4c [new Application/Traffic/CBR]
$cbr4c set packetSize_ 576
$cbr4c set rate_ 0.04Mb
$cbr4c attach-agent $udp4c


#Connect the traffic source with the traffic sink
$ns connect $udp4c $null5


# ***20% "Static" packet load in the reverse direction ( Slave-to-Master)***
```

 # 15% of the load based on maximum sized packet (1518 bytes) generated at node n5, which sends traffic to node n4

```
#Create a UDP agent and attach it to node n5
set udp5a [new Agent/UDP]
$udp5a set packetSize_ 1518
$ns attach-agent $n5 $udp5a

# Create a CBR traffic source, which will generate bursty traffic and attach it to udp5a
set cbr5a [new Application/Traffic/CBR]
$cbr5a set packetSize_ 1518
$cbr5a set rate_ 0.06Mb
$cbr5a attach-agent $udp5a

#Create a Null agent (a traffic sink) and attach it to node n4
set null4 [new Agent/Null]
$ns attach-agent $n4 $null4

#Connect the traffic source with the traffic sink
$ns connect $udp5a $null4
```

 # 80% of the load based on minimum sized packet (64 bytes) generated at node n5, which sends traffic to node n4

```
#Create a UDP agent and attach it to node n5
set udp5b [new Agent/UDP]
$udp5b set packetSize_ 64
$ns attach-agent $n5 $udp5b

# Create a CBR traffic source and attach it to udp5b
set cbr5b [new Application/Traffic/CBR]
$cbr5b set packetSize_ 64
$cbr5b set rate_ 0.16Mb
$cbr5b attach-agent $udp5b

#Connect the traffic source with the traffic sink
$ns connect $udp5b $null4
```

 # 5% of the load based on medium sized packet (576 bytes) generated at node n5, which sends traffic to node n4

```
#Create a UDP agent and attach it to node n5
set udp5c [new Agent/UDP]
$udp5c set packetSize_ 576
```

```
$ns attach-agent $n5 $udp5c


# Create a CBR traffic source and attach it to udp5c
set cbr5c [new Application/Traffic/CBR]
$cbr5c set packetSize_ 576
$cbr5c set rate_ 0.01Mb
$cbr5c attach-agent $udp5c


#Connect the traffic source with the traffic sink
$ns connect $udp5c $null4


#Create master clock agent
set m1 [new Agent/Clock]
$m1 set offset 1
$m1 set rate 1
$m1 set timeToDisplayInfo 1
$m1 set masterClock 1
$m1 set gpsSignal 0
$m1 set enableRLS 0
$m1 set enableLockedRLS 0
$m1 set RLSFreq 1
$m1 set tempProfileName -1
$m1 set ageing 0
$m1 set driftInterval 1
$m1 set enable1588Logs 1
$ns attach-agent $n0 $m1


#  Create slave clock agent
set s1 [new Agent/Clock]
$s1 set offset 1
#slave clock rate is drifting100 ppb (100 ns) faster than the master clock
$s1 set rate 1.000000100
$s1 set timeToDisplayInfo 1
$s1 set masterClock 0
$s1 set gpsSignal 0
$s1 set enableLockedRLS 0
$s1 set enableRLS 0
$s1 set RLSFreq 0
$s1 set tempProfileName -1
$s1 set driftInterval 1
$s1 set timeStampReqFreq 1
$s1 set masterAddr 0
$s1 set masterPort 0
$s1 set enable1588Logs 1
$s1 set enable1588Delays 0
$ns attach-agent $n3 $s1
```

```
#*****Schedule events: Traffic in the Forward Direction (Master-to-Slave)*****
set rng [new RNG]
$rng seed [lindex $argv 0]
set u [new RandomVariable/Uniform]
$u set min_ 0
$u set max_ 1
$u use-rng $rng

#Schedule events for bursty traffic as cbr4a agents
set simTime4a_ 1
while {$simTime4a_  < 86400.00 } {
        set burstTime4a_ [$burstDuration value]
        set startTime [expr [expr ([$u value]) + {$simTime4a_ }]]
        $ns at  $startTime "$cbr4a start"
        set simTime4a_ [expr $startTime + $burstTime4a_ ]
        #puts " CBR Traffic Stops at $simTime4a_"
        $ns at $simTime4a_ "$cbr4a stop"
        set simTime4a_ [expr {$simTime4a_ + $burstTime4a_}]
}


#Schedule events for cbr4b and cbr4c agents will continue until 24 hours
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr4b start"
$ns at 86400.00 "$cbr4b stop"
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr4c start"
$ns at 86400.00 "$cbr4c stop"


#*****Schedule events: Traffic in the Reverse Direction (Slave-to-master)*****


#Schedule events for bursty traffic as cbr5a agents
set simTime5a_ 1
while {$simTime5a_ < 86400.00 } {
        set burstTime5a_ [$burstDuration value]
        #puts [format " Burst Time at Node n5 is %-4.3f  " $burstTime5a_]
        set startTime [expr [expr ([$u value]) + {$simTime5a_ }]]
        $ns at  $startTime "$cbr5a start"
         set simTime5a_ [expr $startTime + $burstTime5a_ ]
        $ns at $simTime5a_ "$cbr5a stop"
        set simTime5a_ [expr {$simTime5a_ + $burstTime5a_}]
}
#Schedule events for cbr5b and cbr5c agents will continue until 24 hours
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr5b start"
$ns at 86400.00 "$cbr5b stop"
$ns at [expr [expr ([$u value]) + 1.0]] "$cbr5c start"
$ns at 86400.00 "$cbr5c stop"


#Schedule events for clock agents
$ns at 1.0 "$m1 start"
$ns at 1.0 "$s1 start"
```

```
$ns at 86401.05 "$m1 stop"
$ns at 86401.05 "$s1 stop"
$ns at 86401.1 "finish"

#Run the simulation
$ns run
```