Utility Accrual Real-time Channel Establishment in Multi-hop Networks

Karthik Channakeshava

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Binoy Ravindran, Chair
Abdul R. Habayeb
Scott F. Midkiff
Ali H. Nayfeh

August 29, 2003
Blacksburg, Virginia

Keywords: real-time systems, multi-hop networks, real-time channels, time utility functions, utility-aware routing

Utility Accrual Real-time Channel Establishment in Multi-hop Networks

Karthik Channakeshava

(Abstract)

Real-time channels are established between a source and a destination to guarantee in-time delivery of real-time messages in multi-hop networks. In this thesis, we propose two schemes to establish real-time channels for soft real-time applications whose timeliness properties are characterized using Jensen's Time Utility Functions (TUFs) that are non-increasing. The two algorithms are (1) Localized Decision for Utility accrual Channel Establishment (LocDUCE) and (2) Global Decision for Utility accrual Channel Establishment (GloDUCE). Since finding a feasible path optimizing multiple constraints is an NP-Complete problem, these schemes heuristically attempt to maximize the system-wide accrued utility. The channel establishment algorithms assume the existence of a utility-aware packet scheduling algorithm at the interfaces. The route selection is based on delay estimation performed at the source, destination, and all routers in the path, from source to destination.

We simulate the algorithms, measure and compare their performance with open shortest path first (OSPF). Our simulation experiments show that for most of the cases considered LocDUCE and GloDUCE perform better than OSPF. We also implement the schemes in a proof-of-concept style routing module and measure the performance of the schemes and compare them to OSPF. Our experiments on the implementation follow the same trend as the simulation study and show that LocDUCE and GloDUCE have a distinct advantage over OSPF and accrue higher system-wide utility. These schemes also react better to variation in the loading of the links. Among the two proposed approaches, we observe that GloDUCE performs better than LocDUCE under conditions of increased downstream link loads.

Acknowledgements

I would like to express my deepest appreciation and thanks to my advisor, Dr. Ravindran, for his constant support, guidance and encouragement in my thesis research. I have learned a lot from this interaction and know that it will go a long way in shaping my career.

I would also like to thank Dr. Midkiff, for his constant guidance and valuable comments, in helping me to improve my research. I also thank Dr. Nayfeh and Dr. Habayeb for their support and comments on my work.

I would like to thank all the members of the Real-time Systems Lab, for their support and company throughout this association with the lab. Special thanks to Peng Li, Haisang Wu and former member Jinggang Wang for all the help and support that have gone a long way in improving my work. I would also like to thank the members of the Networking Lab, for being very supportive and encouraging. Special thanks to Kaustubh Phanse, Tao Lin and Palan Annamalai for their support and guidance.

I would like to express my deepest thanks to my family and friends, both here in Blacksburg, elsewhere in the US and in India, for their moral support and encouragement throughout this study and research.

I would also like to acknowledge the Office of Naval Research for sponsoring this research through the Navy Collaborative Integrated Information Technology Initiative (NAVCIITI) program.

Contents

1 Introduction **1**

1.1 TUFs and UA Scheduling 2

1.2 UA Channel Establishment 3

1.3 Thesis Contribution and Organization 5

2 Motivating Examples and Models **7**

2.1 Example Applications 7

2.1.1 TUFs in AWACS 8

2.1.2 TUFs in the Coastal Air Defense System 10

2.2 The Models 10

2.2.1 The Message Model 11

2.2.2 Timeliness Model 12

2.2.3 System Model 13

2.3 Problem Statement 15

3 Channel Establishment Algorithms **16**

3.1 The LocDUCE Algorithm 16

3.1.1 Algorithm Complexity 19

3.1.1.1 Channel Establishment Complexity 19
3.1.1.2 Runtime Complexity 19
3.1.2 Illustrative Example 19
3.2 The GloDUCE Algorithm 23
3.2.1 Algorithm Description 24
3.2.2 Complexity 25
3.2.2.1 Channel Establishment Complexity 25
3.2.2.2 Runtime Complexity 26
3.2.3 Example Scenario Walk-through 26
3.3 Channel Teardown 28
3.4 Summary 29

4 Simulation Study **30**

4.1 The Simulation Model 30
4.1.1 Topology Generation 31
4.1.2 System Model 33
4.1.3 Simulation Modules 34
4.2 Simulation Setup and Parameters 35
4.2.1 Homogeneous TUFs 36
4.2.1.1 Varying Message Inter-arrival Times: Rectangular TUFs . 37
4.2.1.2 Varying Message Sizes: Rectangular TUFs 40
4.2.1.3 Varying Message Deadline Values: Rectangular TUFs . . . 41
4.2.2 Heterogeneous TUFs 41
4.2.2.1 Varying Message Inter-arrival Times 43

4.3 Downstream Load 45
4.4 Summary 48

5 Implementation **49**

5.1 Channel Establishment Algorithms 49
5.1.1 Implementation Architecture: Routing Module 50
5.1.2 LocDUCE 52
5.1.3 GloDUCE 53
5.1.3.1 k-Shortest Path Algorithm 54
5.2 Real-time Application and Middleware 55
5.2.1 Real-time Middleware 55
5.2.1.1 Client Initialization Interface (CII) 55
5.2.1.2 Communication Interface (CI) 56
5.2.1.3 Embedding Application Characteristics in the Packet . . . 56
5.3 Message Scheduler 57
5.4 Summary 57

6 Experimental Evaluation **59**

6.1 Testbed Configuration 59
6.2 Experimental Scenarios 61
6.2.1 Static Scenario 62
6.2.2 Dynamic Scenarios 64
6.2.2.1 Increasing Packet Arrival Rate 64
6.2.2.2 Importance Factor (Rectangular TUFs) 67

6.2.2.3 Downstream Load 68
6.2.2.4 Message Latency 70
6.2.3 Overhead Measurement 71
6.3 Summary 73

7 Delay Analysis **74**
7.1 Life of a Packet through Operating System Stack 74
7.1.1 Receiving a Packet 74
7.1.2 Sending a Packet 76
7.2 Delay Analysis with Linux Implementation 76
7.2.1 Single Hop Delay 76
7.2.1.1 Delay at First Output Queue 76
7.2.1.2 Delay at Second Output Queue 79
7.2.2 Total Delay at a Node 80
7.2.3 End-to-End Delay 80
7.3 Delay in Actual Implementation 81

8 Past and Related Efforts **82**
8.1 Real-time Channel Establishment 82
8.2 Fault Tolerance for RTCs 84
8.3 Related Work at Virginia Tech 85

9 Conclusions and Future Work **86**
9.1 Advantages and Disadvantages of LocDUCE 86
9.2 Advantages and Disadvantages of GloDUCE 87

9.3 Future Work 87

A Simulation Topologies **94**

A.1 Simulation Topologies 94

B Simulation Results **99**

C Confidence Interval Tables **105**

C.1 Simulation Experiments 105

C.2 Implementation Results 107

List of Figures

1.1 Distributable Threads 2

1.2 Deadline and Example Soft Timing Constraints Specified Using Jensen's Time/Utility Functions 3

2.1 AWACS Association TUF 9

2.2 Avg. Dropped Tracks Under Decreasing Assocn. Capacity 9

2.3 TUFs of Three Activities in GD/CMU Coastal Air Defense 10

2.4 Hard and Soft TUFs 11

2.5 Example Unimodal and Multimodal TUFs 12

2.6 Example Non-Increasing Unimodal TUF 13

2.7 System Model 13

2.8 Custom Components of the System Model 14

3.1 Snapshot of the Network with just DT-1 Messages 21

3.2 Snapshot of the Network with both DT-1 and DT-2 Messages 23

3.3 Traffic of Real-time DT-2 in the Network 27

3.4 Traffic of Real-time DT-1 and DT-2 in the Network 28

4.1 Modelling Philosophy of OMNET++ 30

4.2 Tools used for the Simulation Study 32

4.3 Conversion of Generated Topology into the Simulation Model 33

4.4 System Model: Snapshot of the `.ned` File 34

4.5 OMNET++ Compound Modules of Router and Node 34

4.6 TUFs for Homogeneous Class of Simulation Experiments 36

4.7 Varying Inter-Arrival Times (`RECT` TUFs): Performance Comparison 38

4.8 Performance Comparison of LocDUCE and GloDUCE for Ratio `R4` 39

4.9 Partial Topology for Edge to Node Ratio `R4` 39

4.10 Varying Message Sizes (`RECT` TUFs): Performance Comparison 41

4.11 Varying Message Deadlines (`RECT` TUFs): Performance Comparison 42

4.12 TUFs for Heterogeneous Class of Simulation Experiments 42

4.13 Varying Inter-Arrival Times: Performance Comparison 44

4.14 Total Utility Obtained for Different TUFs in LocDUCE and GloDUCE . . 44

4.15 Utility Comparison for DT with `SOFT_RECT` TUF 45

4.16 Network Topology for Downstream Load Simulation 46

4.17 Increasing Downstream Load: Performance Comparison for DT-2 47

5.1 Implementation Architecture of the Utility Accrual Routing Module 51

5.2 State Diagram of the Single-hop UA Channel Establishment Algorithm . . 52

5.3 State Diagram for Multi-hop Utility Accrual Channel Establishment 54

5.4 Sample Client Implementation Code Showing the CIIs and CIs 56

6.1 Example Network Topology 60

6.2 TUFs for DTs used in Experiments 61

6.3 Performance in the Static Scenario: Percentage Utility 62

6.4 End-to-End Delay Comparison between LocDUCE and GloDUCE 63

6.5 Varying Inter-Arrival Times: Utility Obtained Comparison 66

6.6 Varying Inter-Arrival Times: Deadline Misses Comparison 66

6.7 Performance Comparison of DTs under Different Algorithms 67

6.8 Importance Factor: Percentage Deadline Misses Comparison 68

6.9 Importance Factor: System-Wide Utility Comparison 69

6.10 Scenario for Loading Downstream Links 69

6.11 Load on Downstream Links: Performance Comparison for DT-1 70

6.12 Percentage of Deadline Misses for DTs Under LocDUCE 70

6.13 Increasing Latency: Performance Comparison among DTs 71

6.14 Comparisons under GloDUCE for Increasing Latency 72

6.15 Overhead Comparison of Channel Establishment Algorithms 72

7.1 Interrupt Levels for scheduling of Network Tasks in the Operating System . 75

7.2 Communication between the Layers for Input and Output 75

7.3 Node Input/Ouput Queues 77

7.4 UPA's Packet Interference 78

A.1 Network Topology: Ratio-0 95

A.2 Network Topology: Ratio-1 95

A.3 Network Topology: Ratio-2 96

A.4 Network Topology: Ratio-3 96

A.5 Network Topology: Ratio-4 97

A.6 Network Topology: Ratio-5 97

A.7 Network Topology: Ratio-6 98

A.8 Network Topology: Ratio-7 98

B.1 Message Inter-Arrival Times: Distributions 100

B.2 Heterogeneous TUFs Comparison: Message Inter-Arrival Times 101

B.3 Utility Obtained for Heterogeneous TUFs: Varying Message Inter-Arrival Times 101

B.4 Varying Message Sizes: Distributions 102

B.5 Varying Message Deadline Values: Distributions 103

B.6 Utility Comparison for Heterogeneous TUFs: Varying Message Deadlines . 104

C.1 Confidence Intervals: Overhead Measurement 108

List of Tables

3.1 DT Characteristics for the Illustrative Example 20

4.1 Simulation Study Parameters 36

4.2 Function Properties for Different DTs 37

4.3 Function Properties for Different DTs 43

4.4 Function Properties for Different DTs 46

6.1 DT Characteristics for Static Scenario 62

6.2 Routes Taken by DTs Under OSPF, LocDUCE and GloDUCE 63

6.3 DT Characteristics for Varying Arrival Rate Experiment 65

6.4 DTs for Importance Factor Experiments 68

C.1 Percentage Utility Comparison for Homogeneous TUFs: Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.7(a), Page 38) 105

C.2 Percentage Deadline Misses Comparison for Homogeneous TUFs: Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.7(b), Page 38) 106

C.3 Percentage Utility for Homogeneous TUFs (Ratio-4): Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.8(a), Page 39) 106

C.4 Percentage Deadline Misses for Homogeneous TUFs (Ratio-4): Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.8(b), Page 39) 106

C.5 Utility Obtained for Static Scenario (Heterogeneous TUFs): Confidence Intervals for Figure 6.3 107

C.6 End-to-End Delay for Static Scenario (Heterogeneous TUFs): Confidence Intervals for Figure 6.3 107

C.7 System-Wide Percentage Utility (Increasing Message Arrivals): Confidence Intervals for Section 6.2.2.1, Page 64 107

List of Algorithms

1 Overview of LocDUCE 18
2 Overview of GloDUCE 25
3 DT Registration in LocDUCE: $registerApp(Utility, ThreadChr)$ 53

Chapter 1

Introduction

The Object Management Group's recently adopted Real-Time CORBA 2.0 (Dynamic Scheduling) standard [1] (abbreviated here as RTC2[1]) specifies *distributable threads* as a programming and scheduling abstraction for system-wide, end-to-end scheduling in real-time distributed systems. Distributable threads first appeared in the Alpha OS [2] and later in Alpha's descendant, the MK7.3 OS [3, 4].

A distributable thread is a single thread of execution with a globally unique identifier that transparently extends and retracts through an arbitrary number of local and remote objects. A distributable thread is thus an end-to-end control flow abstraction, with a logically distinct locus of control flow movement within/among objects and nodes. Concurrency is at the distributable thread-level. Thus, a distributable thread always has a single execution point that will execute at a node when it becomes "most eligible" as deemed by the node scheduler.In the rest of the thesis, we will refer to distributable threads as *threads* except as necessary for clarity.

A thread carries its execution context as it transits node boundaries, including information such as the thread's scheduling parameters (e.g., time constraints, execution time, importance), identity, and security credentials. Hence, threads require that Real-Time CORBA's *Client Propagated* model be used, not the *Server Declared* model. Figure 1.1 cited from [1] shows the execution of threads.

[1]Real-Time CORBA 2.0 has been recently renamed as Real-Time CORBA 1.2.

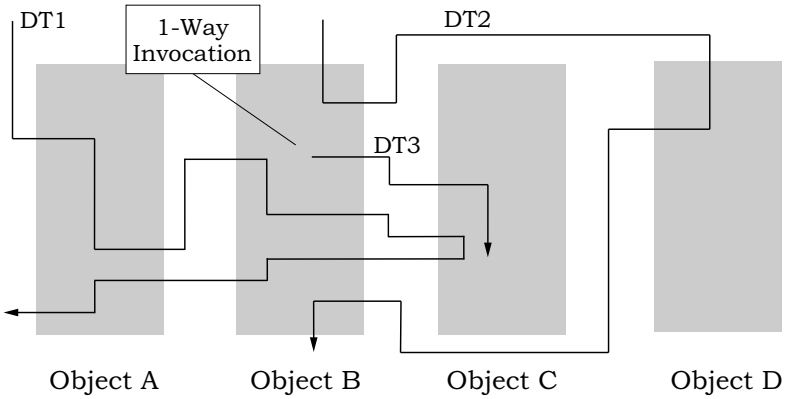


Figure 1.1: Distributable Threads

The propagated parameters are used by the schedulers on each of the nodes the thread transits, for resolving all node-local resource contentions among threads and for scheduling threads on nodes to satisfy the system's timeliness optimality. Using the same optimality criterion, with the same parameters on each node that a thread transits, results in approximate, system-wide timeliness optimality. RTC2 explicitly supports this distributed scheduling approach, called *Distributed Scheduling: Case 2* in the RTC2 specification, due to its simplicity and capability for coherent end-to-end scheduling.[2]

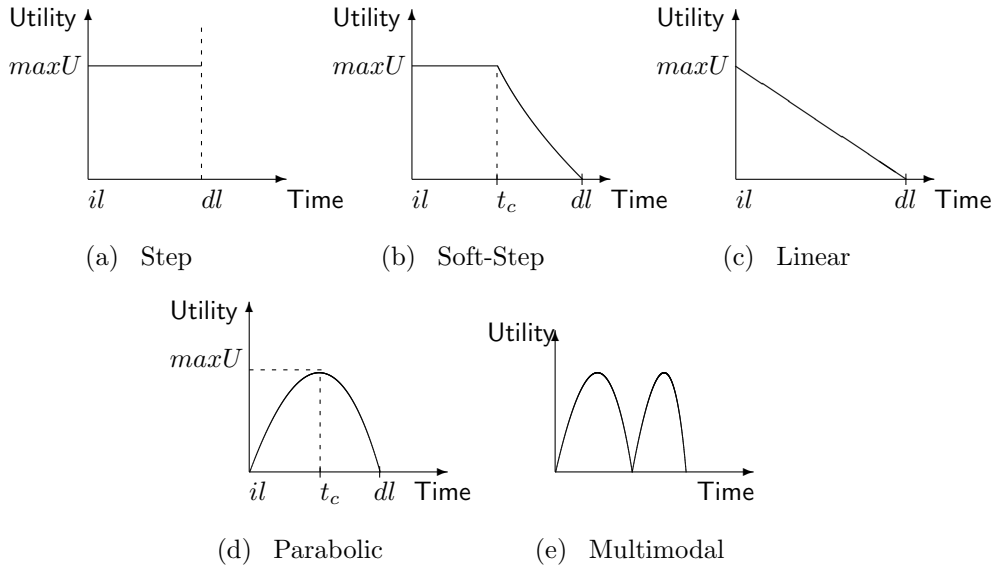
1.1 TUFs and UA Scheduling

In this thesis, we focus on complex, dynamic, adaptive real-time systems at any level(s) of an enterprise—e.g., in the defense domain, from devices such as multi-mode phased array radars [5] to battle management [6]. Such systems include "soft" as well as hard time constraints in the sense that completing a time-constrained activity at any time will result in some utility to the system, which depends on the activity's completion time. Such soft real-time constraints may be as important or mission-critical as hard deadlines.

Jensen's time/utility functions [7] (or TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF specifies the utility to the system that results from

[2]RTC2 also describes Cases 1, 3, and 4, which describe non real-time, global and multilevel distributed scheduling, respectively [1]. However, RTC2 does not support Cases 3 and 4.

the completion of an activity as a function of its completion time. Figure 1.2 shows the conventional deadline (downward step) and several soft time constraints specified using TUFs.



(a) Step (b) Soft-Step (c) Linear

(d) Parabolic (e) Multimodal

Figure 1.2: Deadline and Example Soft Timing Constraints Specified Using Jensen's Time/Utility Functions

When time constraints are expressed with TUFs, the scheduling optimality criteria are based on factors that are in terms of maximizing accrued utility from those activities—e.g., maximizing the sum, or the expected sum, of the activities' attained utilities. Such criteria are called Utility Accrual (UA) criteria, and sequencing (scheduling, dispatching) algorithms that consider UA criteria are called UA sequencing algorithms. In general, other factors may also be included in the optimality criteria, such as resource dependencies and precedence constraints. Several UA scheduling algorithms are presented in the literature [8–14]. RTC2 has IDL interfaces for the UA scheduling discipline, besides others such as fixed-priority, earliest deadline first, and least laxity first.

1.2 UA Channel Establishment

When a multi-hop network – i.e., one where end-hosts are interconnected by multiple switches and routers – is considered as the underlying platform for an RTC2 application,

distributable threads will compete for node-local resources as well as network resources. Node-local resources are those resources that are local to a system "node" such as an end-host or a router. Such resources include physical resources (e.g., processor, disk, input/output) and logical resources (e.g., locks).

Network resources include *real-time channels*. Traditionally, a real-time channel is a uni-directional virtual circuit that is established for application-level messages in a multi-hop network with guaranteed timeliness properties [15]. In the context of an RTC2 application, application-level messages include those that are generated when distributable threads invoke operations on remote objects and thus transcend nodes. Thus, such messages contend for real-time channels in multi-hop networks. Moreover, they are indirectly subject to the timeliness properties of their "parent" distributable threads, on which time constraints and timeliness optimality criteria are explicitly expressed.

While scheduling of threads on nodes and resolution of node-local resource contention among threads is performed by a scheduling algorithm, real-time channels are established by a channel establishment algorithm. Thus, when threads are subject to TUF time constraints and UA optimality criteria, UA scheduling algorithms and UA channel establishment algorithms must be used for coherent system-wide resource management and for improved timeliness optimization.

In this thesis, we consider the problem of UA channel establishment in multi-hop networks. We consider thread time constraints that are specified using TUFs and the optimality criterion of maximizing the sum of threads' attained utilities. We focus on messages of distributable threads. Thus, thread messages are indirectly subject to TUF time constraints. Toward maximizing the sum of threads' attained utilities, we consider the problem of establishing channels for thread messages such that the sum of *messages'* attained utilities are maximized.

Note that timeliness optimization at the message-level is completely consistent with RTC2's Case 2 approach, as thread messages, on behalf of threads, contend for real-time channel resources. This contention is simply resolved (for the messages) by using the thread scheduling parameters (that messages propagate) and by considering the thread-

level optimality criterion at the message-level. Thus, message-level timeliness optimization contributes to thread-level timeliness optimization.

The UA channel establishment problem can be shown to be $\mathcal{NP}$-hard. Thus, we present (1) Local Decision for Utility accrual Channel Establishment (or LocDUCE), and (2) Global Decision for Utility accrual Channel Establishment (or GloDUCE) that heuristically compute channels, seeking to maximize the sum of messages' attained utilities as much as possible. We implement LocDUCE and GloDUCE in a prototype test-bed and experimentally compare them with the Internet standard Open Shortest Path First (OSPF) routing algorithm. Our experimental measurements reveal that both LocDUCE and GloDUCE accrue significantly higher utility than OSPF. In some cases GloDUCE performs better than LocDUCE.

1.3 Thesis Contribution and Organization

The contribution of the thesis includes: (1) the LocDUCE and GloDUCE UA channel establishment algorithms that establish real-time channels in multi-hop networks, maximizing system-wide, summed timeliness utility; (2) construction of a TUF-driven multi-hop network using the algorithms; and (3) timeliness feasibility conditions of the algorithms. To the best of our knowledge, there are no other UA channel establishment algorithms, implementations, and feasibility analysis.

The rest of the thesis is organized as follows: In Chapter 2, we discuss example supervisory real-time applications to provide the motivating context for TUF's. We also discuss the message and system models in Chapter 2. In Chapter 3 we discuss the proposed algorithms, LocDUCE and GloDUCE. We also provide an example scenario for the working of the algorithms and a section on complexity. In Chapter 4 we provide a detailed discussion regarding the simulation modules built to simulate and compare the performance of LocDUCE and GloDUCE with OSPF. We also provide the analysis and explanation for the results obtained in the simulation study in this Chapter. Chapter 5 details the implementation of the two proposed algorithms. The experimental setup, the experimental

evaluations that were preformed for the proposed approaches are presented in Chapter 6. We also provide plots for the results and analyze them. The delay analysis is provided in Chapter 7. In Chapter 8, we overview past research in real-time channel establishment and contrast with our work. The thesis concludes in Chapter 9.

Chapter 2

Motivating Examples and Models

What is the motivation for considering time/utility functions to characterize applications? Are there any systems that have been built using them? This chapter provides an answer to these questions by describing two example systems that have been designed and tested and use TUFs. In this chapter we also discuss the models: *message*, *timeliness* and *system* models that have been assumed. We also discuss the problem here.

2.1 Example Applications

As example real-time systems requiring the expressiveness and adaptability of TUF time constraints, we summarize TUFs of two applications. These include: (1) AWACS (Airborne WArning and Control System) surveillance mode tracker system [16] built by The MITRE Corporation and The Open Group (TOG); and (2) a coastal air defense system [17] built by General Dynamics (GD) and Carnegie Mellon University (CMU). We only summarize some of the application time constraints here; other details can be found in [16, 17], respectively.

2.1.1 TUFs in AWACS

The AWACS is an airborne radar system with many missions, including air surveillance. Surveillance missions generate aircraft tracks for command and control (C2) and battle management (BM). The surveillance tracker consists of several different activities. Its most demanding computation, called *association*, associates sensor reports to aircraft tracks. The tracker employs two sensors that sweep 180 degrees out of phase with a ten second period. Thus, association has a "critical time" at the period length. If the computation can process a sensor report for a track in under five seconds (half the sweep), that will provide better data for the corresponding report from the out-of-phase sensor. Thus, prior to critical time, utility of association decreases as critical time nears.

After the critical time, the utility of association is zero, because newer sensor data has probably arrived. Thus, if the processing load in one sensor sweep period is so heavy that it cannot be completed, probably the load will be about the same in the next period. So there will not be any resources to also process sensor data from the previous sweep.

This timeliness behavior, which requires the expressiveness and adaptability of soft yet mission-critical time constraints, would be difficult to describe using priorities. An effective solution is to describe it using TUFs.

The described semantics establish association's TUF shape: a critical time t_c at the sweep period; utility that decreases from a value U_1 to a value U_2 until t_c; and an utility value U_3 after t_c. U_1, U_2, and U_3 are determined using Application QoS (AQoS) metrics such as: (1) track quality, which is a measure of the amount of sensor data incorporated in a track record; (2) track accuracy, which is a measure of the uncertainty in the estimate of a track's position and velocity; and (3) track importance, which is measure of track attributes such as its threat. Figure 2.1 shows the association thread's TUF.

The tracker creates threads for each airborne object that it tracks. The threads perform a sequence of activities, including association. The TUFs of all threads have the same basic shape shown in Figure 2.1, but use different values for U_1, U_2, and U_3. The system's UA scheduling algorithm resolves the resource contention among all the association (and

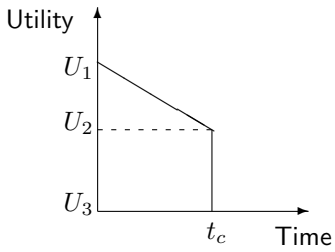
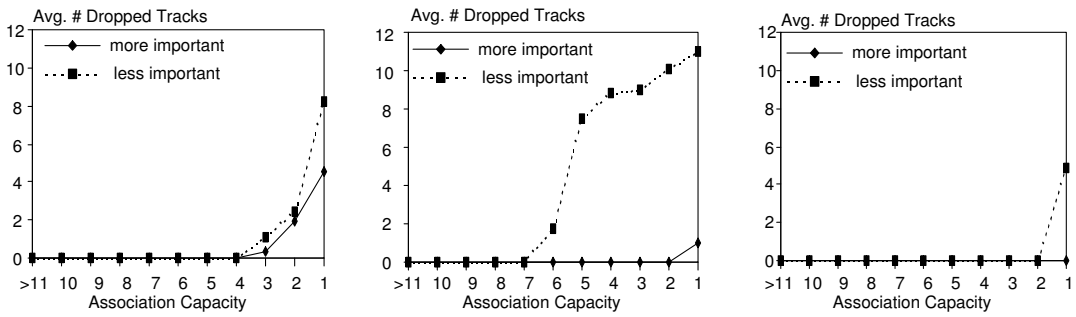


Figure 2.1: AWACS Association TUF

other) threads and schedules system resources to maximize the total summed utility.

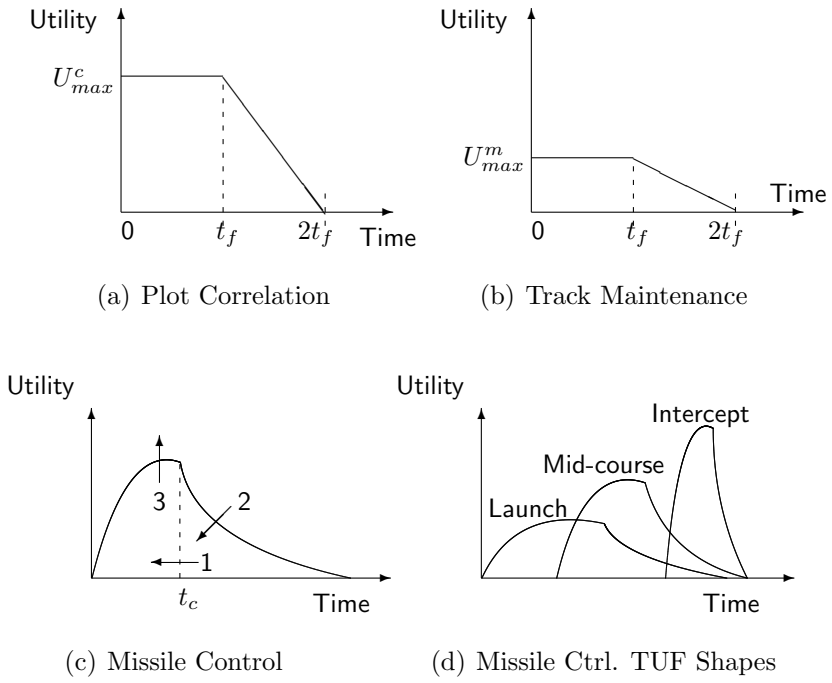
The AWACS surveillance tracker implementation was done using TOG's MK7 operating system [3]. MK7 contains the UA scheduling algorithm described in [9]. To understand how well MK7's UA algorithm is able to schedule system resources in a mission-oriented way, significant performance measurements were made. Different scheduling algorithms, including FIFO and fixed priority, were compared with [9].

Figure 2.2 shows the average number of dropped tracks for the three scheduling policies under decreasing association capacity. The figure illustrates that the UA algorithm minimizes the number of dropped tracks, thereby illustrating the adaptivity of the TUF/UA paradigm.



(a) FIFO Priority (b) Fixed Priority (c) UA Algorithm

Figure 2.2: Avg. Dropped Tracks Under Decreasing Assocn. Capacity



(a) Plot Correlation (b) Track Maintenance

(c) Missile Control (d) Missile Ctrl. TUF Shapes

Figure 2.3: TUFs of Three Activities in GD/CMU Coastal Air Defense

2.1.2 TUFs in the Coastal Air Defense System

Time constraints of three activities in the GD/CMU coastal air defense system, called *plot correlation*, *track maintenance*, and *missile control* are shown in Figures 2.3(a), 2.3(b), and 2.3(c), respectively. Figure 2.3(d) illustrates how the shape of the missile control activity's TUF dynamically changes. Details of how the TUFs were derived, application implementation, and adaptive timeliness measurements can be found in [17].

2.2 The Models

In this section we describe the models that we consider: (1) Message Model (2) Timeliness Model and (3) System Model. The solutions that we propose apply to these models, where in the messages are generated as described in the message model. Time constraints of the generated messages are characterized as described in the timeliness model. The system model describes the system for which the solution applies.

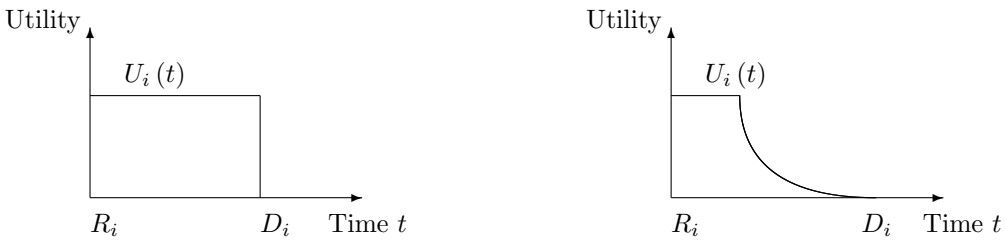
2.2.1 The Message Model

We consider a message model, where messages arrive, as a result of DTs traversing from one node to another. Thus, a message traverses from a source node to a destination node carrying the DT's time constraints. These time constraints are used for the scheduling of the node-local resources in the remote node. The messages arrive at the data link-layer of end-hosts and need to be transmitted to their destination hosts and desire end-to-end timeliness. We denote the set of messages as $m_i \in M, i \in [1, n]$.

The bit length of a message m_i at the data link-layer is denoted as $b(m_i)$. The physical framing overheads increase this size into an actual bit length $b'(m_i) > b(m_i)$ for transmission. Thus, the transmission latency of a message m_i is given by $l_i = b^{'}(m_i)/\psi$, where ψ denotes the nominal throughput of the underlying network medium, e.g., 10^9 bits/s for Gigabit Ethernet.

We consider the *unimodal arbitrary arrival model* for messages, as this model "dominates" other arrival models including aperiodic, sporadic, and periodic arrival models due to the "strength" of the "adversary" embodied in the model [18]. For a message m_i, the unimodal arrival model defines the size of a sliding time window $w(m_i)$ and the maximum number of arrivals $a(m_i)$ that can occur during any window $w(m_i)$.

The timing requirements are expressed using Jensen's time/utility function [7]. We denote the utility function of a message m_i as $U_i(t)$. Thus, the arrival of a message m_i at its destination host (since the release of the message at the source host) at a time t_1 will yield a benefit $U_i(t_1)$. Example utility functions are shown in Figure 2.4.



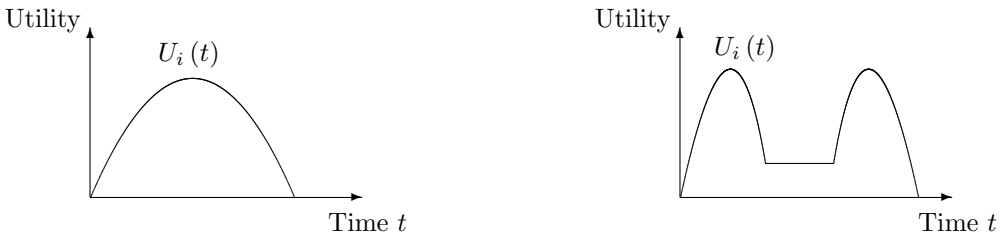
(a) Hard Utility Function (b) Soft Utility Function

Figure 2.4: Hard and Soft TUFs

2.2.2 Timeliness Model

The classical "hard" deadline requirements are "step" utility functions such as the one shown in Figure 2.4(a), where the arrival of a message at anytime before its deadline will result in uniform utility equal to the maximum possible utility; the arrival of the message after the deadline will result in no utility. All other utility functions express soft timing requirements [7].

Though utility functions can take arbitrary shapes, in this thesis, we restrict our focus to *unimodal* functions that are *non-increasing*. Unimodal functions are those functions for which any decrease in utility cannot be followed by an increase [7]. Utility functions which are not unimodal are called *multimodal*. Example unimodal and multimodal functions are shown in Figure 2.5.



(a) Unimodal Function (b) Multimodal Function

Figure 2.5: Example Unimodal and Multimodal TUFs

Unimodal functions that are non-increasing are simply those utility functions for which utility never increases when time advances. Figure 2.6 shows an example. The class of non-increasing unimodal functions allow the specification of a broad range of timing constraints, including hard constraints and most soft constraints.

We denote the release time and deadline of a message M_i as R_i and D_i, respectively. We assume that utility functions take positive values and $U_i(t) = 0, \forall t \notin [R_i, D_i], i = 1, \ldots, n$.

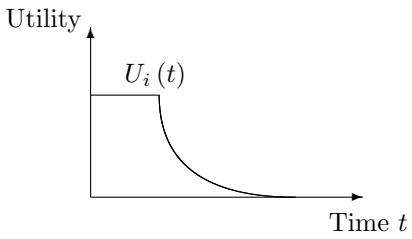


Figure 2.6: Example Non-Increasing Unimodal TUF

2.2.3 System Model

We consider a network with an arbitrary topology and consisting of several networks of various types, e.g. local area networks (LANs) interconnected together by a set of routers, forming a wide area network (WAN). In this network, a message can travel from source to destination passing through a number of intermediate nodes (routers) inter-connecting networks. A source and destination path goes through a set of nodes, where transmitted information can be stored and forwarded to the appropriate next hop. Figure 2.7 shows an example of such a network.

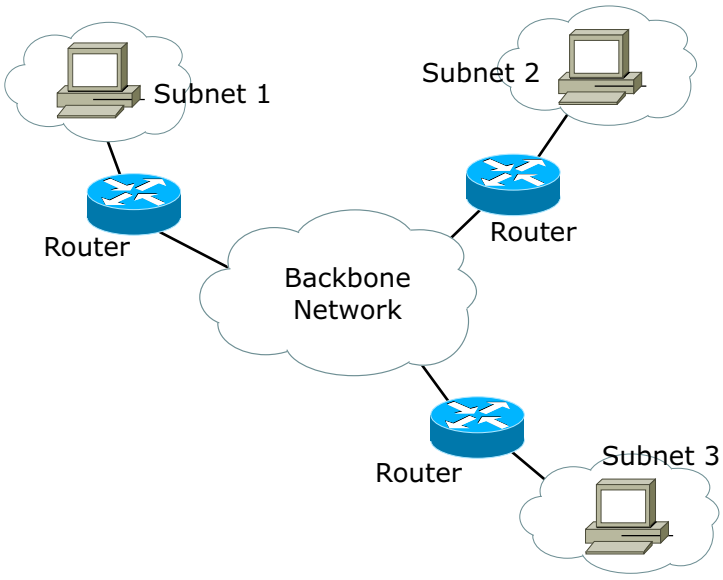
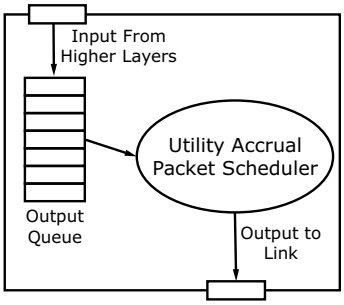


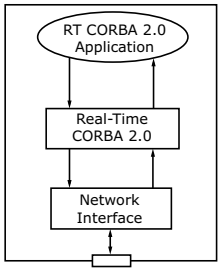
Figure 2.7: System Model

Figure 2.8 shows the components used in the system model and the implementation at each node. In this system model, the end hosts and routers are altered from the usual

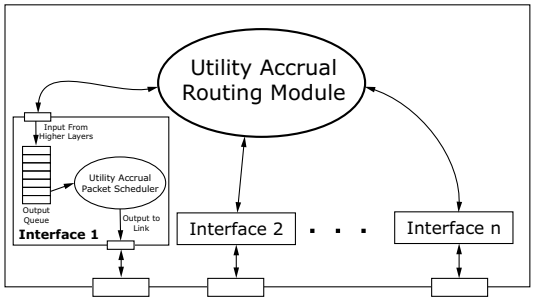
implementation. The end hosts and router operating system kernels are equipped with the utility accrual packet scheduling algorithm (UPA) [14]. The end hosts have the real-time application and middleware to generate the packets. The routers also have the proposed utility-aware routing protocol. The Figure 2.8(a) shows the UPA implemented at the link layer. The same interface component is used on both the end-hosts and the router. The Figure 2.8(b) shows the end-hosts composition The Figure 2.8(c) shows the router implementation in this system model.



(a) Interface Implementation



(b) End-Host System



(c) Router System

Figure 2.8: Custom Components of the System Model

A packet sent from an application process in a source host is first converted into a real-time packet by the host middleware and placed on the output queue. When the link between the host and the router becomes free, the packet is forwarded to the router by the scheduler. There may be an aggregation of several networks at the router, with several other sources sending real-time traffic. There might also be contention for the same outgoing link on the router. The routing module maintains a list of applications and a routing table. When an application packet arrives, it forms a channel for the packet and forwards the packet on to that interface. This packet leaves the router after being scheduled by

the packet scheduler.

We assume that a message is an aggregation of a number of packets. We denote the set of packets, of a message m_i as $P_i = p_1^i, p_2^i, ..., p_n^i$. Thus, the set of all messages from an DT can be denoted by $P = \cup_{i=1}^{n} P_i$. The i^{th} packet from a node j can be denoted as p_j^i. But, for simplicity, we will not use the the superscript i, unless necessary. The length of the packet p_j, in bits is denoted as b_i and the transmission latency $l_i = \frac{b^{'}(p)}{\Psi}$, where Ψ denotes the nominal throughput for the network.

2.3 Problem Statement

Given the timeliness and system models in Section 2.2, our objective is to maximize the system-wide accrued utility by the arrival of all the application packets at their destinations. Thus, the problem we are addressing in this thesis and proposing solutions to, is maximizing $\sum_{k=1}^{n} U_k(t_k)$, where t_k is the arrival time for the message M_k at the destination, for a distributed real-time application across large point-to-point networks, eg. WANs.

The proposed two schemes try to achieve this by establishing real-time channels from source to the destination. Based on the application packet utility characteristics, this routing decision is made and will always try to achieve a higher utility. The reasoning of these schemes is that "if the routes are allocated in the order of the utility, and if routes are optimized based on delay metric rather than by, hop count or fixed cost, then a packet is likely to achieve a higher system-wide accrued utility". In order to increase the system-wide utility for applications using large networks we are proposing two routing schemes. Both these schemes achieve a higher system-wide utility when compared to existing shortest path routing algorithms.

Often, the problem of finding a feasible path subject to multiple constraints is inherently hard. Polynomial-time algorithms may not exist [19]. For example, finding a feasible path with two or more independent constraints is NP[1]-Complete [20].

[1]non-deterministic polynomial-time

Chapter 3

Channel Establishment Algorithms

In this chapter we provide an explanation of the proposed two algorithms for real-time channel establishment in multi-hop networks for applications characterized by TUFs. We provide the high level algorithm description with an illustrative example. We also discuss the complexity of the algorithms.

3.1 The LocDUCE Algorithm

The key heuristic employed by the algorithm is to allocate channels for messages in decreasing order of their potential "return of investments." The return of investment for a message is simply the timeliness utility that can be obtained when the message is delivered to the destination.

To obtain an estimate of the maximum possible return of investment from a message, LocDUCE first allocates channels to each message, assuming that the messages will not interfere with each other (i.e., under zero contention between message packets). Thus, the algorithm considers the network as an un-weighted graph (or equivalently, all edges have the same weight), where each vertex represents a system node and an edge represents a connection between a pair of nodes. For each message, LocDUCE determines the shortest path-distance from the source to destination (i.e., one with the smallest hop-count) by

running the breadth-first-search (BFS) algorithm. The path (or channel) with the shortest path-distance for a message will yield the maximum possible utility for the message, since all TUFs considered are *non-increasing*.

Note that this utility is the theoretically maximum possible utility and cannot be achieved in practice because of message contention.

LocDUCE computes channels for messages in decreasing order of the maximum possible message utility. For the k^{th} message (in the decreasing maximum possible utility order), denoted m_k, the algorithm at any node i first updates the network graph such that, the edge connecting i to the next-hop node $(i+1)$ is annotated with the interference m_k may suffer because of m_{k-1}. LocDUCE includes message m_{k-1} in P_i, where P_i is a set of messages that arrive at node i and are transmitted to the next-hop node $i+1$. Thus, once P_i has been updated with m_{k-1}, any message channel that includes the edge between nodes i and $(i+1)$ may suffer interference from m_{k-1}.

LocDUCE now runs Dijkstra's shortest-path algorithm to determine the shortest path-distance channel for message m_k. Again, the shortest path-distance channel will yield as maximum utility for the message as possible, since all TUFs are non-increasing. For determining the shortest path-distance, the weight of any outgoing edge from any node i in m_{k-1}'s channel is determined using Equation 7.7 (Chapter 7, Page 80). Note that Equation 7.7 gives an upper bound on the total delay incurred by the message to arrive at the next intermediate node by traveling through the edge. The algorithm repeats the process for each message $m_i \in M, i \in [1, n]$.

Note that while LocDUCE computes the channel for message m_k, it considers the potential interference that m_k can suffer from messages $m_i, 1 \leq i \leq k-1$. However, it does not consider the interference that m_k can cause on messages $m_i, 1 \leq i \leq k-1$ and accordingly update their channels that were computed in earlier steps. This is precisely due to the heuristic nature of the algorithm. LocDUCE ignores such "backward" interference and reasons that it will not be significant, as each message considered for channel establishment at any given time is the one with the largest (theoretically) maximum possible utility from the remaining unexamined ones. Further, correcting such backward interferences will be

Algorithm 1: Overview of LocDUCE

```
1  graph: Network Graph; t_trans: Transmission time;
2  P: set of all messages in the network ordered by "Return of Investment" (roi);
3
4  for m_i ∈ M, i ∈ [1, n] do
5      /* For Node k */
6      if m_i ∉ P_k then
7          hops := BFS (graph, m_i);
8          t_d := t_cur + (hops × t_t);
9          /* Calculate roi */
10         roi := U_{m_i}(t_d);
11         P_k[j] := (m_i, roi) : (P_k[j].roi > P_k[j + 1].roi);
12         /* Update network graph with m_{k-1} */
13         for ∀p ∈ P_k, p ∈ [1, (j − 1)] do
14             UpdateGraph (graph, P_k[p]);
15         for ∀p ∈ P_k, p ∈ [j, ...] do
16             /* Store the nextHop for future use */
17             P_k[p].nextHop := Dijkstra (graph, P_k[p].msg);
18             UpdateGraph (graph, P_k[p]);
19     else
20         /* Route already exists */
21         nextHop := P_k[m_i].nextHop;
```

computationally expensive. Thus, the algorithm takes this heuristic approach and seeks to maximize the total utility attained by all messages as much as possible. A high level description of LocDUCE is described in Algorithm **??**.

3.1.1 Algorithm Complexity

The complexity of LocDUCE can be divided into (1) Runtime complexity and (2) Channel Setup Complexity. Runtime complexity is the complexity of the algorithm when performing only packet forwarding. This happens after the channel establishment process. The Channel Setup Complexity is the complexity during the channel establishment phase, when the first application packet comes along.

3.1.1.1 Channel Establishment Complexity

Let k be the number of routes stored for each destination.
Let n maximum applications registered on each interface of any router in the path.

Since each router has to determine the lowest delay interface among k different routes and among each of the n applications registered on the interface, the complexity of determining the delay is $O(nk)$.

3.1.1.2 Runtime Complexity

Since the algorithm has to determine whether the application has already been registered, assuming that there are a applications registered, the complexity is $O(n)$. The Chapter 6 gives more details on the actual values in terms of the time taken with respect to a fixed number of applications registered.

3.1.2 Illustrative Example

Consider the network topology as shown in the Figure 3.1. Consider the DTs as shown in Table 3.1 using the network. The Figures 3.1 and 3.2 show the network state at different

Table 3.1: DT Characteristics for the Illustrative Example

	DT-1	DT-2
Arrival Rate (a pkts/sec.)	4	7
Window Size (w sec.)	1	1
Packet Size ($b(p)$ Bytes)	1000	1000
Deadline (d sec.)	2.0	2.0
Function Type	`SOFT_RECT`	`LINEAR`
Maximum Utility	100	100

times, when there are different DTs in the system.

To start with, for sake of simplicity, let us assume that there is no DT performing a network transition. Real-time DT 1 arrives at router R_1 and has to make a transition to a destination node connected to R_d. The DT characteristics are shown the the Table 3.1 and TUF of the DT is shown in the Figure 3.1. We are also assuming that all the links have the same bandwidth of 128 kbps. The worst-case execution time of UPA (UPA_{WCET}) is 5×10^{-3} seconds and the delay incurred for transfering packets from one queue to another (δ) is $500\mu sec$[1]. The DT message size $b(p)$ is assumed to be 1000 Bytes. This means that the actual size of packets on the wire, $b'(p)$, is 1070 Bytes.

$$\delta = 500\mu sec. \tag{3.1}$$

$$UPA_{WCET} = 5 \times 10^{-3} sec. \tag{3.2}$$

$$b(p) = 1000\text{Bytes} \Rightarrow b'(p) = 1070\text{Bytes} \tag{3.3}$$

$$\Psi = 128\text{kbps} \tag{3.4}$$

The steps that the LocDUCE algorithm is going to go through for DT-1 messages are as follows:

1. Add DT-1 to the *DTlist*.

[1]Average values are obtained by actual execution of the UPA algorithm [14] and the routing module.

2. The time when the message is likely to reach the destination $t_d^1 = \frac{1070\times8}{128\times10^3} \times 2 = 0.1337$. This is determined using the shortest path and the shortest path has 2 hops to the destination.

3. Utility, $U_1(t_d^1) = 100 - (\frac{100}{4.0} \times 0.1337) = 93.315$.

4. Since, this is the only DT, the shortest path is allocated to DT-1. Hence, the next hop is router R_2. The table in Figure 3.1 shows the decision made.

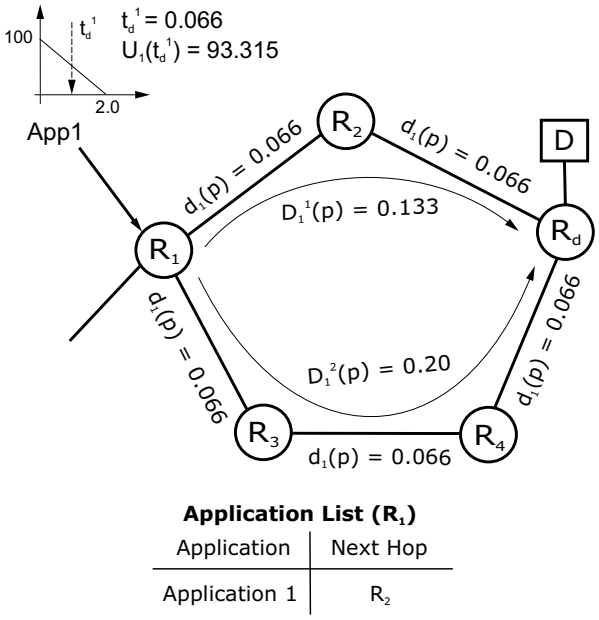


Figure 3.1: Snapshot of the Network with just DT-1 Messages

Figure 3.1 shows the network when DT-1 is making a transition (as seen from router R_1). For the incoming traffic at router R_1, there are two routes available. In the Figure 3.1, the links between the routers are marked with the estimated delays, which in the above case, is equal to the transmission delay of the message ($d_1(p) = 0.066$ seconds). We estimate $U_1(t_d^1)$ for DT-1 using transmission delays on the 2 hop route, through R_2. The values of the real-time channel delay $D_1^1(p)$ and $D_2^1(p)$ of DT-1, along the two possible paths are illustrated in the Figure 3.1. Hence the path chosen for DT-1 is through R_2, i.e. the smaller delay path.

Now, at a later time, another DT, DT-2 arrives at R_1 destined to the same destination network connected to R_d. The value of $U_2(t_d^2)$, utility of DT-2 at t_d^2, is greater than that

of DT-1 (Figure 3.2) and hence the order of the list is changed, with DT-2 before DT-1. The allocation of the routes should now begin with DT-2 and then proceed to DT-1. The total estimated delays in each of the two paths are as shown in the Figure as $D_1^1(p)$ and $D_1^2(p)$ respectively. The router R_1 decides to choose the shortest path i.e. R_2 as the next hop.

1. When DT-2 arrives at R_1, the time when a message of DT-2 is 'predicted' to reach the destination is $t_d^2 = \frac{1070 \times 8}{128 \times 10^3} \times 2 = 0.1337$.
2. Utility, $U_2(t_d^2) = 100$
3. $U_2(t_d^2) > U_1(t_d^1)$
4. Allocate the route, first to DT-2 and then to DT-1. DT-2 is allocated the link with next hop R_2.
5. The messages from DT-2 will interfere (when messages are being scheduled at the interface) with messages of DT-1 (as they have higher utility). The interference delay on the link from R_1 to R_2, is 0.308 as shown in the Figure 3.2. This is calculated using Equation **??**. Note that the delay on the link between R_2 and R_d is the same as the transmission delay (Router R_1 is not intelligent enough or does not have information regarding the interfaces of R_2).
6. The entire channel delay ($D_1^1(p) > D_1^2(p)$)
7. Hence, the route chosen for DT-1 messages is through R_3.

At a different time instant, DT-2 arrives in the system. The same procedure of calculating $U_2(t_d^2)$ is followed and it is found to be 100. Since, the utility is higher, the $DTList$ is ordered as shown in the Figure 3.2. The values of the link delay estimates for the two applications are also shown in the Figure 3.2. First, DT-2 is allocated a route, as it yields a higher utility and then, other DTs are allocated their routes. But, once messages from DT-1 are allocated to the link with next hop R_2, DT-2 will experience interference from DT-1. This increases the worst-case delay for DT-1 and hence needs to be re-routed through the router R_3.

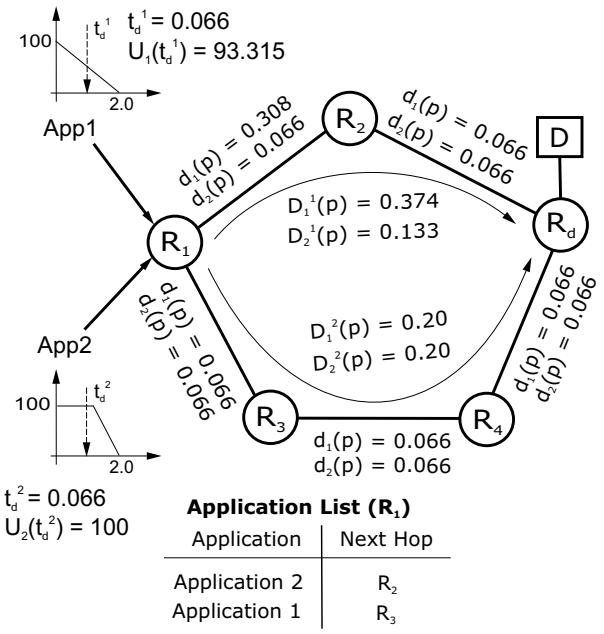


Figure 3.2: Snapshot of the Network with both DT-1 and DT-2 Messages

3.2 The GloDUCE Algorithm

LocDUCE makes routing decisions based on the local information maintained by each router. There are cases when the local decision affects the performance of the DTs. In the same example as illustrated in Figure 3.1, if DT-1 arrives at R_2 before DT-2 the algorithm would choose the shortest route available, i.e. through the next hop R_d. When DT-2 arrives at R_1, at a later time, router R_1 is not aware of the DTs in the network. So, when it makes a routing decision for DT-2 messages, it would select the lowest delay path, i.e. through R_2. This affects the performance of both the DTs. The overall utility reduces. The results from experiments as discussed in Chapter 6 illustrate the effect of this problem.

To address this problem, each router has to maintain a global picture of the DTs and the channels that are established for these DTs. The solution we propose is called, Global Decision for Utility Accrual Channel Establishment (GloDUCE) algorithm. The term "Global decision" illustrates the fact that the GloDUCE algorithm utilizes the global information of link states along k different paths from a source to a destination to decide on the best path. The algorithm performs delay analysis on each of the k available

paths using this information pertaining to the routers along the path. In addition it also maintains global information regarding the channels established elsewhere and uses that information in choosing the best possible path to achieve a higher system-wide utility accrual. It also, just as LocDUCE, assumes the existence of a utility-aware scheduling algorithm at the interface for scheduling application packets. GloDUCE is somewhat more complicated as it also uses a *k-shortest path algorithm* for its optimization. In this chapter, we describe the algorithm in detail, provide a step-by-step walk through of an example network scenario and also discuss the complexity of the algorithm.

3.2.1 Algorithm Description

GloDUCE also maintains information regarding the applications as LocDUCE, and in addition, GloDUCE enabled router has to maintain the the following information:

1. A list of routers in the network and, applications on each of them
2. k shortest paths to a particular destination

This information is used to compute the worst-case delays on every path to the destination before choosing the shortest delay path.

Similar to LocDUCE, each router maintains the application list ordered in the decreasing order of the application utility. The utility of the application at time t_d ($U_i(t_d)$) is obtained, where t_d is the time when the application packet reaches the destination. t_d is determined by assuming transmission delays ($b_i^{'}(p)/\Psi$) for the packet for the number of hops to the destination. This order is used to allocate the network resource to each application.

Unlike a LocDUCE router, which selects the lowest delay interface without knowing about the downstream routers, GloDUCE evaluates the entire path from source to destination. It determines the 'k' paths during the process of forming the routes and selects the lowest delay path. We have used the k-shortest path algorithm [21] for determining k paths, and a modified implementation of this algorithm [22][2](refer Section 5.1.3). The GloDUCE

[2]GNU implementation in 'C' of the algorithm by V. Jimenez and A. Marzal

algorithm is described in Algorithm 2 below.

Algorithm 2: Overview of GloDUCE

```
1  graph: Network Graph; t_trans: Transmission time;
2  P: set of all messages in the network ordered by "Return of Investment";
3
4  for m_i ∈ M, i ∈ [1, n] do
5      /* For any Node */
6      if m_i ∉ P then
7          hops := BFS (graph, m_i);
8          t_d := t_cur + (hops × t_trans);
9          /* Calculate return of investment (roi) */
10         roi := U_i(t_d);
11         P[j] ← (m_i, roi) : (P[j].roi > P[j + 1].roi);
12         /* Update network graph with m_{k-1}s */
13         for ∀p ∈ P, p ∈ [1, (j − 1)] do
14             UpdateGraph (graph, P[p]);
15         for ∀p ∈ P, p ∈ [j, ···] do
16             P[p].nextHop :=
                       Dijkstra (graph, P[p].msg);
17             UpdateGraph (graph, P[p]);
18             BroadcastUpdate (P[p]);
19     else
20         /* Route already exists */
21         nextHop := P[m_i].nextHop;
```

3.2.2 Complexity

GloDUCE is more complicated than LocDUCE because of the global nature of the algorithm. Similar to LocDUCE, the complexity of GloDUCE can be divided into runtime and channel setup complexity. GloDUCE also requires more state information regarding the network topology, paths and the real-time channels established in the network.

3.2.2.1 Channel Establishment Complexity

Let k be the number of paths stored for each destination.

Let h be the maximum number of hops to the destination from the router performing the analysis, to the destination.

Let a maximum DTs registered on each interface of any router in the path.

Since each router has to determine the end-to-end delay a packet might incur in going through each of the k paths, through each of the h routers in the path and among each of the a DTs registered on the interface, the complexity of determining the delay is $O(ahk)$.

3.2.2.2 Runtime Complexity

Since the algorithm has to determine whether the DT has already been registered, assuming that there are a DTs registered, the complexity is $O(a)$. The Chapter 6 gives more details on the actual values in terms of the time taken with respect to a fixed number of DTs registered.

3.2.3 Example Scenario Walk-through

Let us consider the same example network as in Figure 3.1. The DT characteristics are also the same as in Table 3.1. We will change the routers on which each of these DT's arrive to analyze the difference between the algorithms.

To start with, for sake of simplicity, let us assume that there are no DTs using the network. Real-time DT-2 arrives at router R_2 destined to a network connected to R_d. The Figure 3.3 shows the TUF of DT-2. We are also assuming that all the links have the same bandwidth of 128 kbps. The worst-execution time of BPA (BPA_{WCET}) is assumed to be 5×10^{-3} seconds. The application packet sizes are also assumed to be 1000 Bytes. This means that the actual size of packets on the wire, $b'(p)$, is 1070 Bytes.

The steps that the GloDUCE algorithm is going to go through for DT-2 packets are as follows:

1. Add DT-2 to the DT list of R_2.
2. The time when the packet is likely to reach the destination $t_d^2 = \frac{1070 \times 8}{128 \times 10^3} \times 1 = 0.066$. This is determined for the shortest path and 1 is the number of hops to the destination.

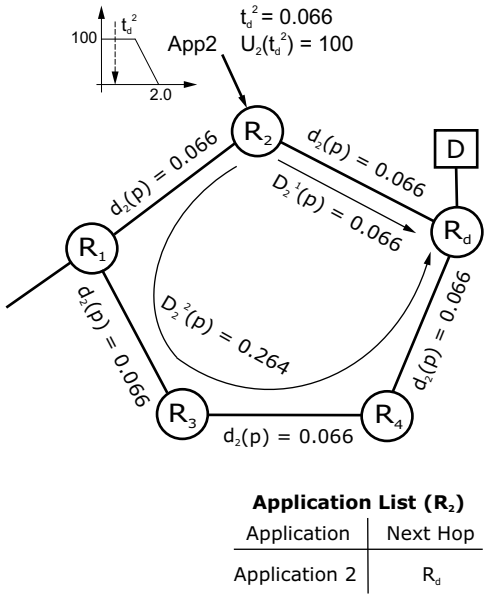


Figure 3.3: Traffic of Real-time DT-2 in the Network

3. Utility, $U_2(t_d^1) = 100$.

4. Since, this is the only DT, the shortest path is allocated to this DT. Hence, the next hop is router R_d. The table in Figure 3.3 shows the decision made.

5. In addition to the above router R_2 sends an update to the neighbors regarding the DT that it registered and the route it choose for it.

Now, at a later time, another thread, DT-1 arrives at R_1 destined to the same destination network connected to R_d. The value of $U_1(t_d^1)$, utility of DT-1 at t_d^1, is less than that of DT-2 (Figure 3.4). DT-1 being the first thread in R_1, we register it in the list. While allocating the route to DT-1 we examine the two paths leading to the destination and evaluate both of them. The path with the least delay is chosen. Also, R_1 maintains a list for each of the routers in the network and has registered DT-2 for R_2 from the update packet it receives from R_2. The total estimated delays in each of the two paths are as shown in the Figure as $D_1^1(p)$ and $D_1^2(p)$ respectively. The router R_1 decides to choose the shortest path i.e. R_3 as the next hop.

1. When Application 1 arrives at router R_1 the Figure 3.4 shows the view of the network at R_1

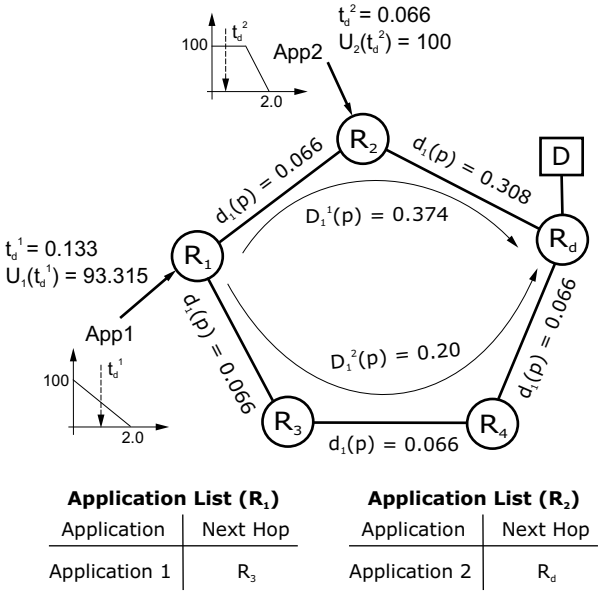


Figure 3.4: Traffic of Real-time DT-1 and DT-2 in the Network

2. The delay a application 1 packet is 'predicted' to incur is $t_d^1 = \frac{1070 \times 8}{128 \times 10^3} \times 2 = 0.1337$.

3. Utility, $U_1(t_d^1) = 100 - (100/2.0) \times 0.1337 = 93.315$. This is needed to order the applications in the list.

4. R_1 evaluates the two paths to reach the destination. It maintains information about other router interfaces and hence knows that Application 2 is routed through R_2.

5. The interference Application 1 packets are going to incur is shown in the Figure 3.4, along the link between R_2 and R_d.

6. The channel delay $D_1^1(p) > D_1^2(p)$ as shown in Figure 3.4.

7. Hence, the route chosen for DT-1 packets is through R_3.

8. Sends an update packet updating the routers about the DT-1 that is registered.

3.3 Channel Teardown

In both Section 3.1 and Section 3.2, we have discussed in detail about channel establishment algorithms. But, both these algorithms do not mention about the channel tear-down.

It is also important to address this problem so that allocated resources can be reclaimed and used for other flows. This may be necessary in the event of a failure of either the source, other nodes (routers) or links in the network, or a normal DT termination.

Here we discuss some of the approaches that are taken in other resource reservation methods. Since, the main focus of this thesis is channel establishment we provide a brief discussion about tear down. We discuss the procedure employed by ReSource reserVation Protocol (RSVP) [23]. RSVP uses something called *soft-state signalling*, i.e. "soft" in the sense that, if the state information is not refreshed within a certain period of time, it is discarded and resources reclaimed. In our case, (though this has not been implemented here) a similar mechanism can be used. Each time a timeout happens, the routes are recomputed for the remaining DTs. There are performance trade-offs in selecting a particular timeout value. One approach is that this initial timeout value can be provided by the DT. Once the timeout occurs, either the router picks a known timeout value or the DT re-initializes it.

3.4 Summary

In this chapter we have discussed the proposed approaches LocDUCE and GloDUCE. LocDUCE is based on local information and is less complex when compared to GloDUCE. The performance of LocDUCE is comparable to GloDUCE for most cases. But there are other situations where GloDUCE performs slightly better than LocDUCE. Although, GloDUCE provides a much better performance in all the cases, the amount of information maintained to perform the route selection is large. This is clearly not scalable for large networks, as the more processing capacity is required for selecting a channel. In this thesis we do not address the issues related to channel tear-down except making a note that it is also required in addition to channel establishment. To measure and compare the performance of LocDUCE and GloDUCE, we have conducted simulation and implementation based comparisons with OSPF. The Chapter 4 and Chapter 5 present the results that were obtained from them.

Chapter 4

Simulation Study

In order to study and measure the performance of the proposed algorithms, LocDUCE and GloDUCE, and compare them with OSPF, a simulation model was constructed. OMNET++, a discrete event simulation environment[1] was used for conducting this study. This chapter describes in detail the simulation model, the simulation experiments that were conducted to verify the performance and results that we have obtained.

4.1 The Simulation Model

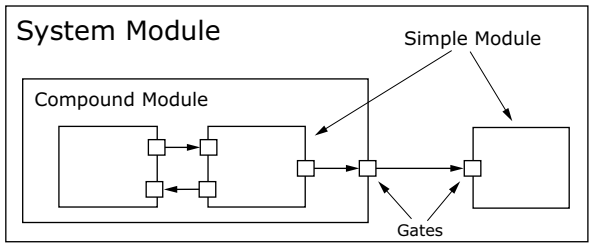


Figure 4.1: Modelling Philosophy of OMNET++

OMNET++ implements a modelling philosophy where a system consists of various entities (called *modules*) communicating by exchanging messages. Modules can be nested; several modules can be grouped together to form a bigger unit (*a compound module*). Figure 4.1

[1]OMNET++ is a free software available for academic/non-profit use from `http://www.omnetpp.org`.

describes this philosophy pictorially.

A graphical editor, "`gned`" can be used to design the compound modules. OMNET++ also provides a topology description language to define the model structure. The active components of the system can be described as concurrent processes using C++. OMNET++ facilitates this by providing a simulation class library. A configuration file can be used to specify various options to OMNET++ and also holds parameters required by the modules in the model. The modules built can communicate by sending messages through *gates* which form the input/output interfaces of the modules (can be seen in Figure 4.1).

4.1.1 Topology Generation

In order to generate a system model with a certain number of nodes and also to obtain a random topology, Boston university Representative Internet Topology gEnerator (BRITE) was employed. BRITE was developed by the QoS Networking Laboratory of Boston University[2]. BRITE provides the user with the flexibility to specify diverse topology generation parameters. A screen shot of BRITE is shown in the Figure 4.2(a). To visualize the topology generated by BRITE, Otter[3] [24], a general purpose network visualization tool was employed. Otter was developed by the Cooperative Association for Internet Data Analysis (CAIDA). The Figure 4.2(b) shows a screen-shot of the program displaying a network.

BRITE is used for the initial topology generation. It takes the number of nodes in the network, the model for topology generation and the bandwidth distribution, as arguments. BRITE provides RouterWaxman and RouterBarabasiAlbert models for router-level topology generation. From among them, we have selected the model proposed by Barabási and Albert in [25]. These models differ in the way the network grows and the node interconnection method they use. The BarabasiAlbert model provides preferential connectivity, i.e., a tendency of a new node to connect to the existing nodes that are highly connected or popular. Thus, this model provides additional paths for source-destination pairs with-

[2]BRITE is available for free download from `http://www.cs.bu.edu/brite`.

[3]Otter can be obtained from `http://www.caida.org/tools/visualization/otter/`

out increasing connectivity to edge nodes. This means that from an initial topology, no significant changes in the shortest paths occur as a result of increase in edge to node ratio. According to the BarabásiAlbert model, when a node i joins the network, the probability that it connects to a node j already in the network is given by Equation 4.1.

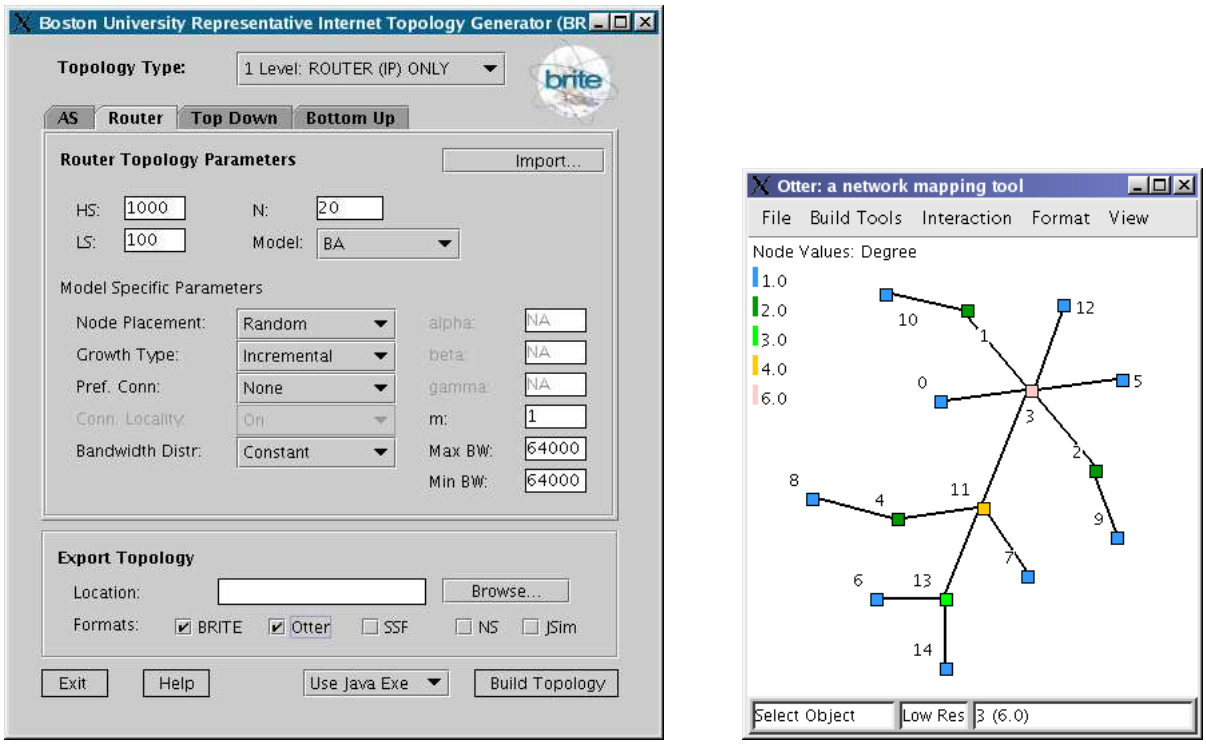
$$P(i,j) = \frac{d_j}{\sum_{k \in V} d_k} \tag{4.1}$$

where,

d_j – degree of the target node

V – is the set of nodes that have joined the network

$\sum_{k \in V} d_k$ – is the sum of outdegrees of all the nodes that previously joined the network



(a) Screen-shot of BRITE

(b) Screen-shot of Otter

Figure 4.2: Tools used for the Simulation Study

A perl script (`brite2omnet.pl`) is used to transform the topology from BRITE format to formats required for running the simulations. It also generates the random topologies with increasing node to edge ratios by randomly selecting new edge end-points. In addition

to converting the initial topology into the other formats needed the script also generates Otter file formats for the newly created topologies. The Figure 4.3 illustrates this process. The output in Otter format is used for visualization of the topologies generated. The graph file output is used by the Eppstein's algorithm for generation of the routes and paths to be used by the simulation modules to route the messages. The output in `.ned` format is used by OMNET for configuring the system model used in the simulation. This is needed during compile time and OMNET produces the system according to this file. The `.ini` file is used for simulation initialization for configuring parameters before running the simulation experiments.

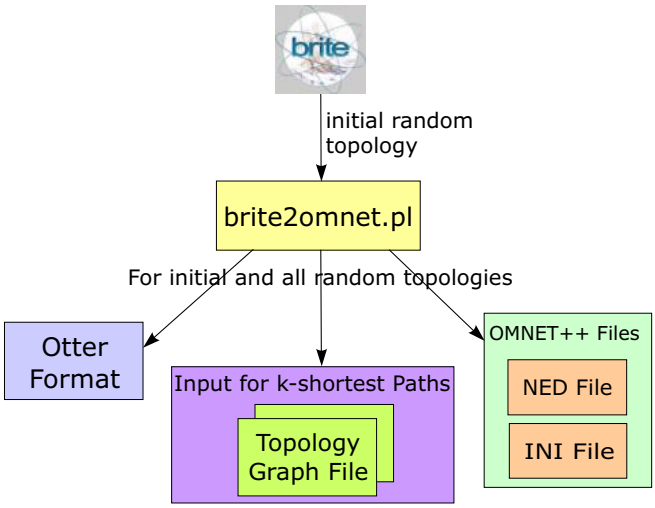


Figure 4.3: Conversion of Generated Topology into the Simulation Model

4.1.2 System Model

The system model built for the simulation study consists of the nodes and routers interconnected by links. The node and router are compound modules each consisting of several simple modules connected through gates. The system module (as shown in Figure 4.4) consists of a number of nodes and routers interconnected through links. The links between the nodes and routers are configured with a certain capacity during the simulation runs. The model in Figure 4.4 was generated from the `.ned` file created by the script.

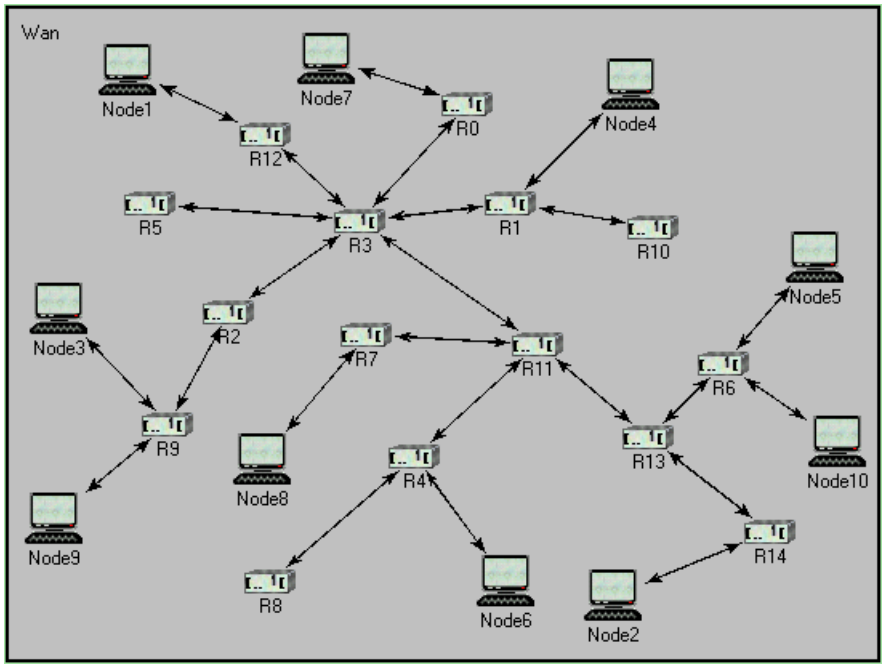
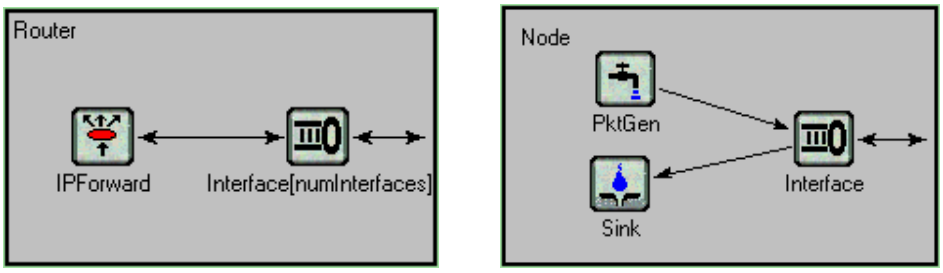


Figure 4.4: System Model: Snapshot of the `.ned` File

4.1.3 Simulation Modules

The Figure 4.5 shows the node and router modules used in the simulation. Both these modules are compound and are composed of one or more simple modules. Each node (as shown in Figure 4.5(b)) further consists of simple modules `PktGen`, `Sink` and `Interface`. Similarly, the router module (Figure 4.5(a)) is composed of one `IPForward` and a number of `Interface` modules. `Interface` (in nodes as well as routers) is representative of the network interfaces. A router has multiple interfaces and is represented as an array of `Interface`. The simulation model can be configured to have multiple router interfaces.



(a) Router Module (b) Node Module

Figure 4.5: OMNET++ Compound Modules of Router and Node

The simple modules `Interface` implements different scheduling algorithms and these algorithms can be configured from the OMNET++ configuration file. The `PktGen` module implements a message source that can generate messages in different ways as needed by the experiment. `Sink` destroys and logs the messages received at the destination. All these modules can log trace information for deeper analysis, if needed. The `IPForward` module forwards the messages based on the forwarding table of the router. Different algorithms for forwarding to abstract the behavior of the CE algorithms and OSPF are implemented in this module.

4.2 Simulation Setup and Parameters

To study the performance of the CE algorithms in a large data space, we randomly generate network topologies with increasing edge to node ratio. This increases the network connectivity and provides a basis for comparison of the two proposed approaches and OSPF. Some of the DT characteristics like the TUF type of the DT, message sizes and message deadline values are varied and the performance is measured. The Table 4.1 shows the parameters that were varied during the simulation experiments. The topology used in the simulation experiments consisted of 15 routers interconnecting 5 source-destination node pairs. The links connecting the nodes have the same bandwidth ($64kbps$). The value of the bandwidth is chosen, such that the links are loaded when more than one message flow uses the same link. The simulation experiments can be broadly classified into two categories:

1. Homogeneous TUFs – All DTs have similar type of TUF
2. Heterogeneous TUFs – Each DT has a different type of TUF

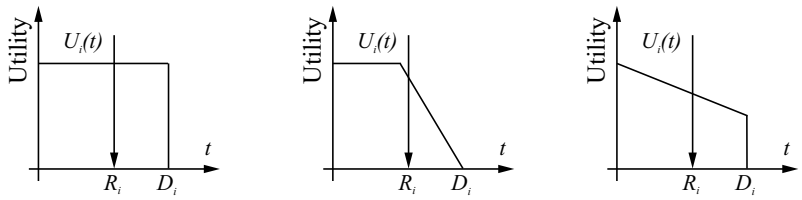
Under each of these broad classes of experiments, experiments were conducted by varying (a) the inter-arrival times of messages, (b) the message sizes, and (c) message deadlines. These classes of experiments were repeated for different values of edge to node ratios varying from 0.933 to 1.867. Since, we are using a limited set of DTs for the simulation

Table 4.1: Simulation Study Parameters

Message Properties	**Parameters**
Inter-arrival Time Mean – 0.25	$Constant(0.25)$, $Uniform(0.125, 0.375)$, $Normal(0.25, 0.0722)$, $Max(0.050, Exp(0.25))$, $Pareto(2.0, 0.125)$
Message Length Mean – 2000	$Constant(2000)$, $Uniform(1500, 2500)$, $Normal(2000, 288.675)$, $Max(800, Exp(2000))$, $Pareto(2.0, 1000)$
Message Deadline Mean – 1.5	$Constant(1.5)$, $Uniform(0.75, 2.25)$, $Normal(1.5, 0.443)$, $Max(0.75, Exp(1.5))$, $Pareto(2.0, 0.75)$
System Properties	**Parameters**
Edge/Node Ratio	0.933–1.867

experiments, increasing this ratio beyond 1.867 does not alter the performance by much. The Table 4.1 shows these parameters and the distributions that were employed for the experiments. Each simulation experiment run was repeated for different seed values and the average and standard deviation were determined.

4.2.1 Homogeneous TUFs



(a) `RECT` (b) `SOFT_RECT` (c) `TRK_ASOC`

Figure 4.6: TUFs for Homogeneous Class of Simulation Experiments

Under the homogeneous case, the simulation comparison was conducted with all DTs having similar TUFs. For example, all DTs having either rectangular TUFs or all having

Table 4.2: Function Properties for Different DTs

DTs	**Maximum Utility**	**Deadline (s.)**
DT-1	1.5	3.0
DT-2	2.0	3.0
DT-3	2.5	3.0
DT-4	3.0	3.0
DT-5	3.5	3.0

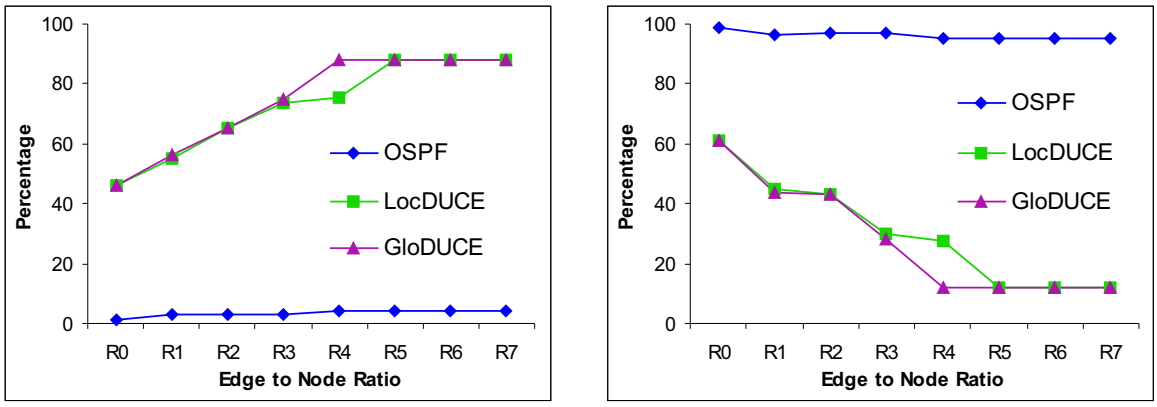
soft-rectangular TUFs. In this case the value of maximum utility is different among the DTs. The experiments were performed with all the DTs characterized by either Rectangular 4.6(a) or Soft Rectangular 4.6(b) or Track Association 4.6(c) TUFs. The maximum utility value and the deadlines for each of DTs used in these simulation runs are outlined in Table 4.2. The same characteristics apply for all TUFs in the homogeneous case.

4.2.1.1 Varying Message Inter-arrival Times: Rectangular TUFs

In this set of simulation runs, the inter-arrival times of the messages are varied using various distributions as outlined in the Table 4.1. The results corresponding to these distributions are presented in the Appendix C. In this section we present the results for rectangular TUFs corresponding to the Pareto Distribution. All the other results pertaining to different TUFs are also presented in the Appendix C. In the Appendix C we also present the confidence intervals for the results presented in this Chapter.

In the Figure 4.7 we plot both the system-wide utility obtained and the system-wide deadline misses with respect to the system topology for OSPF, LocDUCE and GloDUCE. Each point on the x-axis indicated by `R0`, `R1`, etc., represent the edge to node ratio for the topology used in the simulation. Table 4.1 shows the actual range used for this ratio. In this section we only present the results for rectangular TUFs.

The utility obtained in OSPF is not affected by increase in the connectivity. This shows that even when there was an increase in the edge to node ratio, there was no noticeable



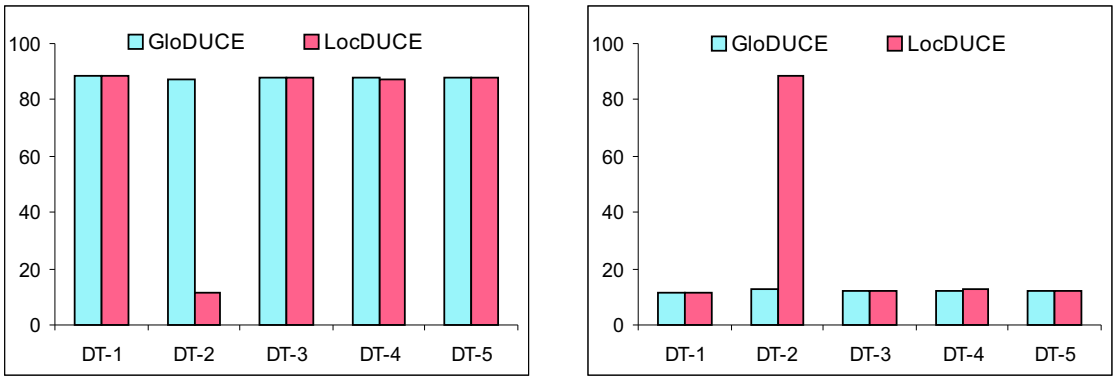
(a) Percentage Utility

(b) Percentage Deadline Miss

Figure 4.7: Varying Inter-Arrival Times (`RECT` TUFs): Performance Comparison

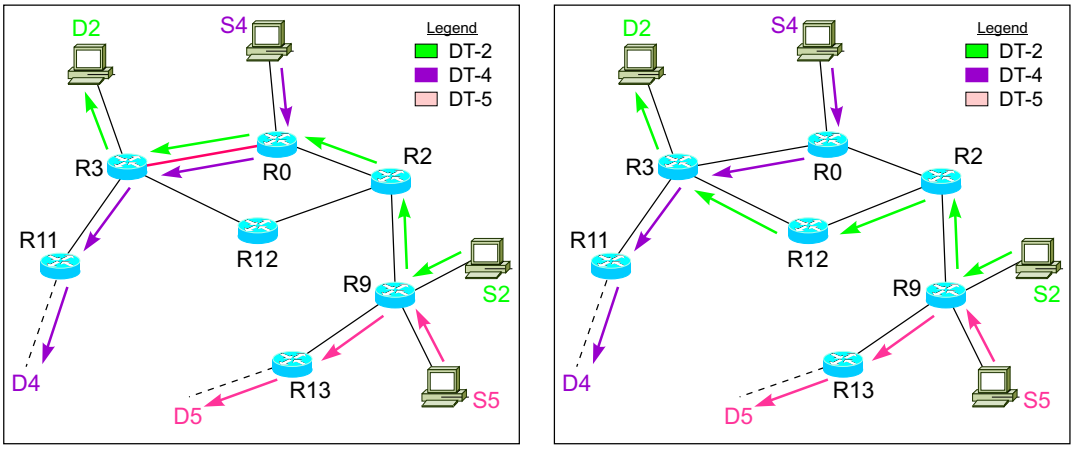
change in the shortest paths. Hence, the system-wide utility obtained remains low and the deadline misses remains high. This offers a very good comparison scenario with respect to LocDUCE and GloDUCE. From the results we find that both LocDUCE and GloDUCE trace a similar performance except at the point where the ratio is `R4`. LocDUCE has a lower system-wide utility than GloDUCE at this point. Figure 4.8 shows the performance comparison between LocDUCE and GloDUCE for the individual DTs in the system for the ratio value `R4`. Observe the performance for DT-2 for the case of LocDUCE and GloDUCE. The utility obtained for DT-2 under LocDUCE is very less when compared to the utility under GloDUCE (Figure 4.8(a)). In order to analyze the reason for this performance difference, we have to consider the topology under which the DTs are executing and the routes that they take.

Figure 4.9 shows the partial topology for ratio `R4` along with the routes taken for DT-2, DT-4 and DT-5. The other DTs are not necessary to analyze the performance of DT-2 as they take diverse paths and do not affect DT-2. Also, from Table 4.2 we know the value for the maximum utility we consider for each DT. The routes shown for messages of DT-2, DT-4 and DT-5 in Figure 4.9(a) correspond to the shortest paths (in terms of the number of hops) available for the destinations D2, D4 and D5 respectively. These paths may not be the only paths or for that matter the only shortest paths. But these are according to the routing table entries of the routers along the path.



(a) Percentage Utility (b) Percentage Deadline Miss

Figure 4.8: Performance Comparison of LocDUCE and GloDUCE for Ratio R4



(a) Route Selected by LocDUCE (b) Route Selected by GloDUCE

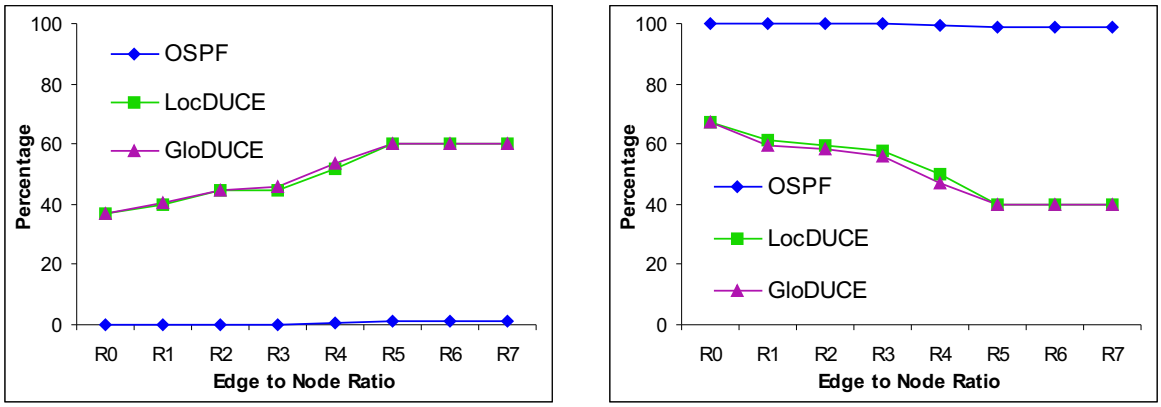
Figure 4.9: Partial Topology for Edge to Node Ratio R4

When the routers employ LocDUCE for channel establishment, router R9 only maintains local information pertaining to its interfaces. Since, DT-5 has a higher utility than DT-2, it is allocated its shortest route through router R13. This decision is shown in the pink arrows. DT-2 gets the route allocated along the other available path through router R2. DT-4 messages arrive at router R0 and R0 allocates the shortest route through R3 as next hop. The path chosen for DT-4 is marked by the arrows in violet. This decision of router R0 is local and does not get conveyed to R2. When messages from DT-2 arrive at R2, the shortest path that R2 has for DT-2 is through R0. Even though there is another path of the same hop count, the implementation chooses the first shortest entry in the routing table and hence chooses R0. Thus, there is contention for the link R0–R3 and DT-2 suffers more than DT-4 as it has a lower maximum utility than DT-4.

Figure 4.9(b) shows the routes allocated to the DTs when GloDUCE is used for establishing channels. GloDUCE employs the global information of the real-time channels and performs its decisions. In this case, the router R2 is aware of the decision made by router R0, and that DT-4 having a higher utility is using the link R0–R3 and chooses the other available path through R12. This path is marked in Figure 4.9(b) by the arrows in green. Here, it so happens that the hop counts for both routes are the same. Even if this is not the case, GloDUCE will decide on a particular route based on delay incurred along that route, in comparison with the other available routes.

4.2.1.2 Varying Message Sizes: Rectangular TUFs

In this set of simulation experiments, the message sizes were varied with the inter-arrival times of messages and the deadline values kept the same. The variation in the message sizes was according to the Table 4.1. The performance of the various algorithms when the message sizes are distributed using Pareto Distribution, is shown in Figure 4.10. Here, we only provide the performance for rectangular TUFs as the performance is same for the case of both soft-rectangular and track association TUFs (plots shown in the Appendix C). We observe the same general trend in the performance results. Both, LocDUCE and GloDUCE perform better than OSPF.



(a) Percentage Utility

(b) Percentage Deadline Misses

Figure 4.10: Varying Message Sizes (`RECT` TUFs): Performance Comparison

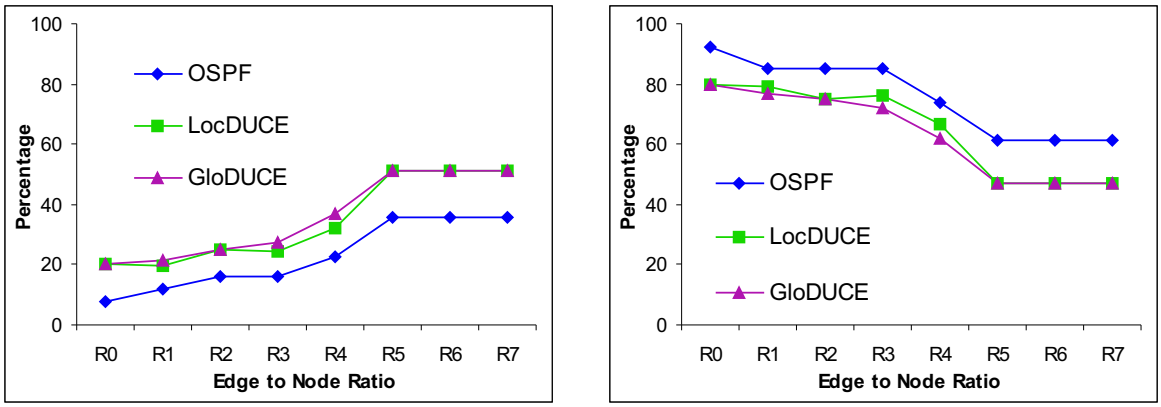
4.2.1.3 Varying Message Deadline Values: Rectangular TUFs

In this simulation study, the messages had varying deadlines. The deadlines of the messages were varied around a mean value and the performance of the algorithms were measured. The performance is as shown in Figure 4.11. Here, even though we observe the same trend as before, i.e. both LocDUCE and GloDUCE out-perform OSPF, the magnitude of the differences is reduced considerably. This can be attributed to the fact that the deadline values are randomly chosen in the different distributions considered and affect all the algorithms in a similar manner. LocDUCE and GloDUCE perform better than OSPF even under this condition. In some cases GloDUCE performs slightly better than LocDUCE.

4.2.2 Heterogeneous TUFs

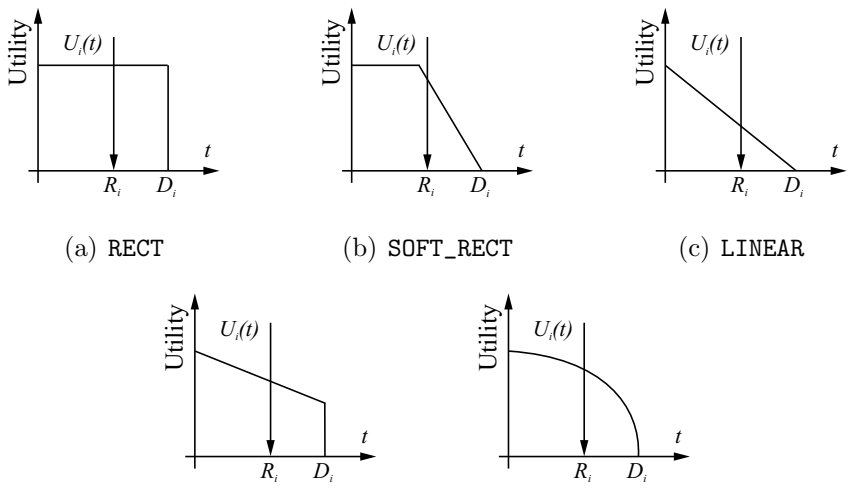
Under the heterogeneous case, the simulation comparison was conducted with DTs having different TUFs. For example, the DTs with rectangular, soft-rectangular and linear TUFs all executing simultaneously. The value of maximum utility is kept the same with the TUF type varied. The experiments were performed with the DTs characterized by TUFs as shown in Figure 4.12.

In order to offset the effect of the location of the DTs on particular source-destination



(a) Percentage Utility (b) Percentage Deadline Misses

Figure 4.11: Varying Message Deadlines (`RECT` TUFs): Performance Comparison



(a) `RECT` (b) `SOFT_RECT` (c) `LINEAR`

(d) `TRK_ASOC` (e) `QUAD`

Figure 4.12: TUFs for Heterogeneous Class of Simulation Experiments

Table 4.3: Function Properties for Different DTs

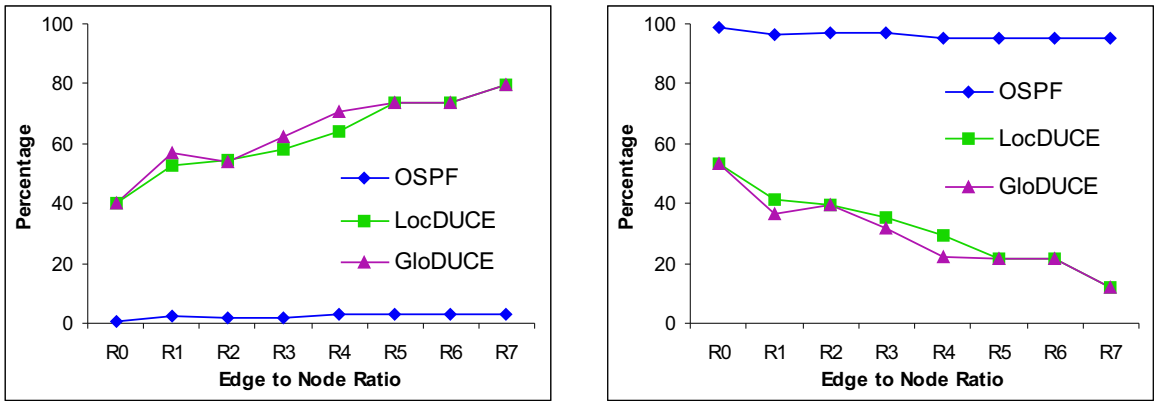
DTs	**TUF Type**	**Maximum Utility**	**Deadline (s.)**
DT-1	`RECT`	1.0	3.0
DT-2	`SOFT_RECT`	1.5	3.0
DT-3	`LINEAR`	2.0	3.0
DT-4	`QUAD`	2.5	3.0
DT-5	`TRK_ASOC`	3.0	3.0

pairs, the simulation runs were conducted by rotating the DTs around, so that the DT with a certain TUF executed in all the possible source nodes. For example, DT-1 having a rectangular TUF and executes from source S1 in one run, in another run executes from S2 and so on till it is run on all the possible source nodes. The same procedure is repeated for other DTs used in the simulation. The Table 4.3 shows the DT characteristics, i.e., the TUF type, maximum utility and deadline.

4.2.2.1 Varying Message Inter-arrival Times

The same set of simulation experiments outlined in Section 4.2.1.1 were repeated, but with DTs having time constraints as listed in Table 4.3. In this case the percentage utility measurement was performed in a different way. The average value of the minimum delay incurred by a message from a DT was determined from the experiments and used for determining the utility obtained. This value is assumed to the maximum possible utility that can be obtained from this message. The total possible utility from all messages is determined using this value and then the percentage utility obtained is calculated. Figure 4.13 shows the comparison in performance of the algorithms.

In Figure 4.14 we show the comparison of the utility obtained for various TUFs between LocDUCE and GloDUCE for a ratio value of `R4`. For this topology, GloDUCE performs better than LocDUCE and this shows the difference among the two algorithms. In the case of ratio `R4`, GloDUCE prefers the `RECT` and `SOFT_RECT` TUFs over `LINEAR` TUF and achieves a better system-wide utility than LocDUCE.



(a) Percentage Utility (b) Percentage Deadline Misses

Figure 4.13: Varying Inter-Arrival Times: Performance Comparison

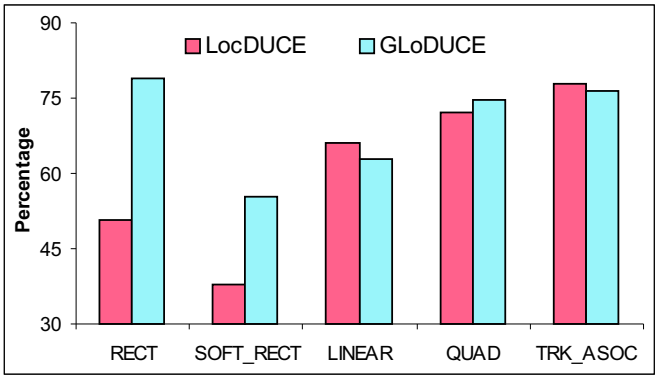


Figure 4.14: Total Utility Obtained for Different TUFs in LocDUCE and GloDUCE

The results for heterogeneous TUFs where the message size is varied follow the same trend as observed for the heterogeneous TUFs as described above. Hence these results are not repeated here and are presented in the Appendix C. But, for the case of varying deadlines there are certain results that need to be highlighted.

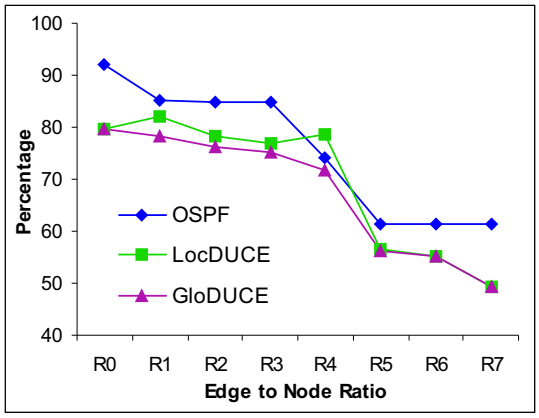


Figure 4.15: Utility Comparison for DT with `SOFT_RECT` TUF

In Figure 4.15 we can see the performance of OSPF, LocDUCE and GloDUCE. From this, we can see that for ratio `R4` the performance of OSPF is better than LocDUCE. OSPF always selects the shortest path to a particular destination and schedules packet on a first-come-first-serve basis. Thus the delay experienced by the DT having a `SOFT_RECT` TUF is somewhat lesser than when there is a utility-accrual algorithm scheduling the packet. This is because, the utility accrual algorithm always tries to maximize the accrued utility, there-by choosing the message with a higher utility. Thus it always prefers the DT with the `QUAD` TUF which has a higher utility (Table 4.3). It should also be noted that the system-wide utility in the case of OSPF is always lower than both LocDUCE and GloDUCE. Also, it might so happen that the utility obtained for some DTs may be higher than LocDUCE under certain situations.

4.3 Downstream Load

To observe the difference between LocDUCE and GloDUCE, a specific experiment was conducted in simulation. The Figure 4.16 shows the topology considered for conducting

this simulation experiment. The experiment was performed to show the difference in performance of both LocDUCE and GloDUCE algorithms. In this experiment there are three source nodes generating traffic. Sources `S1` and `S2` generate the traffic for DTs considered for showing the difference in performance. Source `S3` generates increasing rate of background traffic to load the link R1–R2. The destination nodes `D1`, `D2` and `D3` form the sinks for these messages. Table 4.4 shows the DT characteristics for this experiment. In these experiments all DTs have rectangular TUFs and same value of deadline.

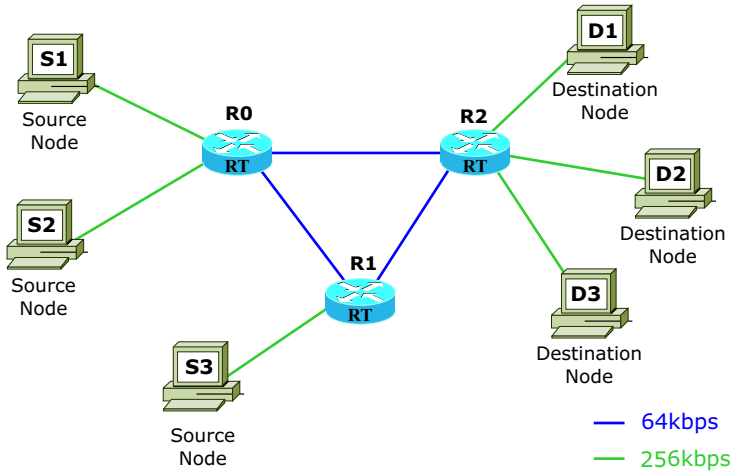


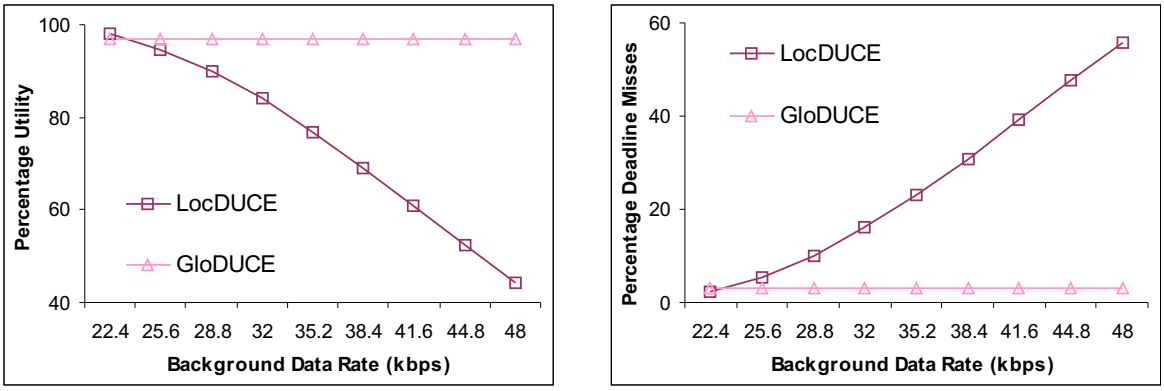
Figure 4.16: Network Topology for Downstream Load Simulation

Table 4.4: Function Properties for Different DTs

DTs	**Time Constraint**	**Data Rate (kbps)**
DT-1	Max Utility: 4.0 Deadline: 3.0 *sec.*	24
DT-2	Max Utility: 1.0 Deadline: 3.0 *sec.*	32
DT-3	Max Utility: 2.5 Deadline: 3.0 *sec.*	9.6–48

Figure 4.17 shows the performance of DT-2 for OSPF, LocDUCE and GloDUCE algorithms. It shows the percentage utility obtained in Figure 4.17(a) and percentage deadline misses in Figure 4.17(b). The background data rate in the Figure 4.17 is the data rate for DT-3. From these plots, we find that OSPF performs better than LocDUCE and

GloDUCE for DT-2. OSPF chooses the shortest path for DT-1 and DT-2, i.e., the path R0–R2–D1 and R0–R2–D2 respectively. Table 4.4 shows the data rate of DT-1 and DT-2 and Figure 4.16 shows the link capacity. From this we can see that the link is not overloaded and DT-1 and DT-2 perform identically. Also, OSPF does not differentiate among the two.



(a) Percentage Utility

(b) Percentage Deadline Misses

Figure 4.17: Increasing Downstream Load: Performance Comparison for DT-2

GloDUCE performs better than LocDUCE because of its global nature. Initially, GloDUCE routes DT-1 along the R0–R2 path and DT-2 along the R0–R1–R2 path. When, DT-3 having a higher utility than DT-2 arrives at R1, it chooses the shortest available path through R1–R2 and creates a channel for DT-3. R1 also broadcasts the information of the real-time channel to other routers in the network. Now, router R0 re-evaluates the routes already allocated for DT-2 and chooses the path R0–R2 instead of the earlier path through R1. This is because of the delay analysis it performs including DT-3 in the link R1–R2. In a similar manner LocDUCE initially allocates the channel for DT-2 through R1. Since, it does not maintain global information about the channels, even when DT-3 arrives at router R1, it does not change the routes for DT-2. Thus suffers a significant degradation in performance. This is more so because DT-3 has a higher utility when compared to DT-2. In either case, depending upon the actual load on the link R1–R2, the performance of the DTs using that link gets affected.

4.4 Summary

The simulation study was conducted to study the performance of the channel establishment algorithms for a wide spectrum of conditions, like varying topologies, message inter-arrival times, message sizes and deadline values. This Chapter describes in detail the simulation study and discusses the tools that were employed for conducting this study. The Chapter also presents the various experiments that were conducted, the results obtained and provides analysis for the results. From the results obtained we can see that GloDUCE and LocDUCE outperform OSPF for all the simulation experiments that were considered. When the network connectivity is increased without any significant change in the shortest paths, the GloDUCE and LocDUCE algorithms perform better. There are certain drawbacks in considering only local channels (as LocDUCE does), without any information regarding the channels established between other sources and destinations. In some cases presented in the Chapter these points have been highlighted and reasons for this performance analyzed.

It is clear from this that both the channel establishment algorithms, GloDUCE and LocDUCE, perform better and achieve higher system-wide utility when compared to OSPF. In the Chapter 5 we present a proof-of-concept implementation and conduct some experiments on the implementation.

Chapter 5

Implementation

A prototype of the Channel Establishment Algorithms was implemented in the application layer of the Linux Operating System. The various parts that were implemented were: (1) Application Layer Utility Accrual Routing Module (URM) (2) Utility Accrual Message Scheduler in the Linux Kernel. The routing module captures the packet from the interface and processes them and selects the proper routes for the packets. These packets are scheduled using the packet scheduler.

5.1 Channel Establishment Algorithms

The routing module uses the `PF_PACKET` type socket, to capture the packets from the interface and process them. In order to limit the number of packets that are captured, a Berkeley Packet Filter (BPF) also known as Linux Socket Filter (LSF) [26] is installed on the socket using the `setsockopt()` (`SO_SOCKET_FILTER` option) call of the socket API. The filter employed here, filters the application packets from the rest of the packets on the interface. Each real-time application packet is characterized by the presence of the IP Options field, identified by the Option Code (`0xa0`) which is used to recognize the packets by the filter. The filter code is generated in assembly using the `dd` option of `tcpdump`. After the packets are processed and a route identified, the packet is placed

on the corresponding link using `libnet`[1] library calls. Internally, the libnet libraries use the `PF_PACKET` socket to write to the link. Figure 5.1 shows the actual architecture of the implementation and the sub-components. Since, the packets were being sniffed off the interface, and forwarded by the routing module, it is necessary to block the packet going through the operating system stack. The packet capturing process creates a copy of the original packet and hands it to the `PF_PACKET` socket created by the routing module. In order to avoid duplicate packets, we have to drop the original packet after capturing it from the interface. This is achieved using the Linux firewall tool, `ipchains`. The `ipchains` module is configured to drop any packet destined to the destination ports used by the real-time applications, in the `forward` chain.

The implementation being a "proof-of-style" one, the freely available Open Shortest Path First (OSPF) implementation for Linux was employed to provide the channel establishment module with the basic routing table OSPF creates, using its packet exchanges about the link state. The implementation of OSPF that is used is GNU Zebra[2]. Only the OSPFv2 protocol was used for performance comparison with the two proposed approaches. The URM queries the OSPF daemon for the routing table and uses this routing table for performing the route decisions. By default, most implementations of OSPF maintain multiple routes internally, and store only the *shortest path* in the kernel routing table. So, by querying the OSPF daemon, the routing module obtains the entire table (including multiple routes) and selects the least delay path. The overview of the general design architecture is provided in Section 5.1.1 in detail. Both LocDUCE and GloDUCE have been implemented in the URM and their performance studied and compared with OSPF.

5.1.1 Implementation Architecture: Routing Module

Figure 5.1 shows the framework in which the routing module has been implemented. It also shows the module interfaced with the networking components in the operating system.

[1]Libnet library version 1.1.0 can be obtained from `http://www.packetfactory.net/libnet/`

[2]GNU Zebra is distributed under the GNU Generic Public License at (`http://www.zebra.org/`). Zebra implements TCP/IP based routing protocols BGP-4, RIPv1, RIPv2 and OSPFv2.

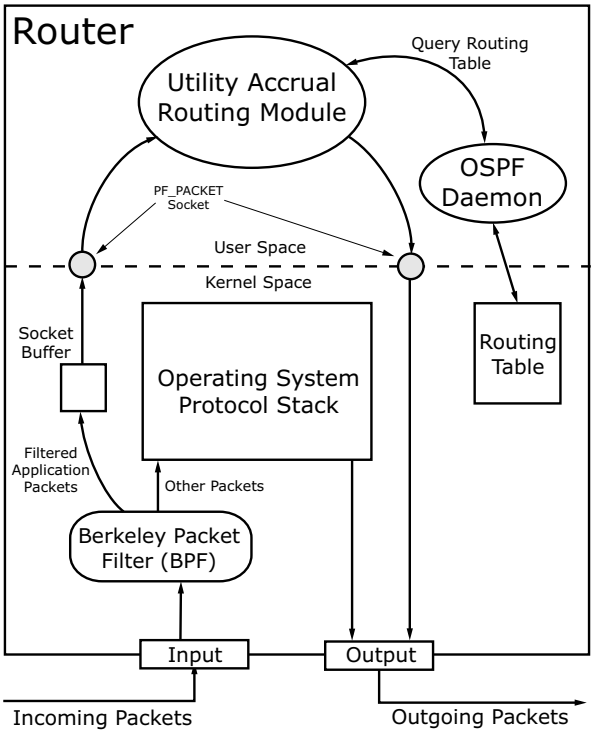


Figure 5.1: Implementation Architecture of the Utility Accrual Routing Module

Internally, the routing module is multi-threaded, creating a different thread for each interface to handle DT messages arriving at the router. The module maintains an updated, mutex protected list of DTs and associates each DT with an output interface on which the messages from the DT are being forwarded. When a message from a DT arrives for the first time, it is registered with the router. The registration process is necessary as the router makes decisions based on DTs already registered with it. When a route is identified, it is stored for further use. If there is no change in the DT parameters, or, no new DTs arrive, this route remains unaltered. The implementation of the two proposed schemes follow the same overall architecture as outlined above but use different heuristics. LocDUCE, makes routing decisions based only on the local information where as GloDUCE uses global information. The implementations of the schemes are discussed below in Sections 5.1.2 and 5.1.3 respectively.

5.1.2 LocDUCE

The LocDUCE algorithm is based on performing the route selection based on the local information regarding the applications. Figure 5.2 shows the finite state machine of LocDUCE.

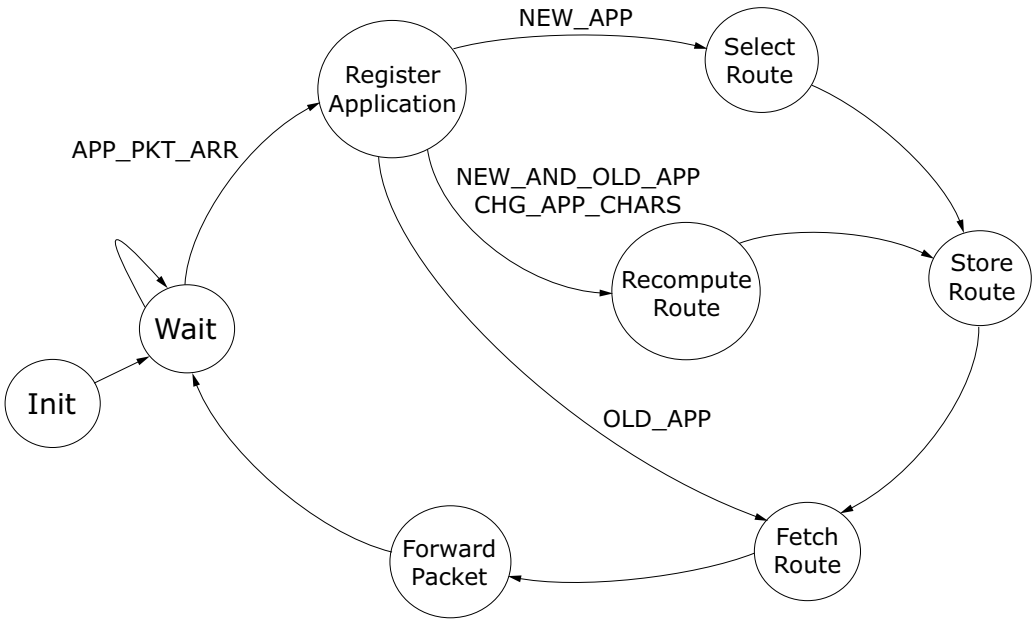


Figure 5.2: State Diagram of the Single-hop UA Channel Establishment Algorithm

In Figure 5.2, each oval represents a state in the decision process, the text along the arrows refers to the event that triggers the transition from one state to another. Each state is implemented as a single (or multiple) function call(s). For example, when a new application packet arrives at a router, `APP_PKT_ARR` event occurs and the state transitions from `Wait` to `Register Application`. Similarly, the other events are `NEW_APP`, `OLD_APP` and `NEW_AND_OLD_APP`, and transitions are from `Register Application` state, to `Select Route`, `Recompute Route` and `Fetch Route` states respectively. The LocDUCE algorithm heuristic is provided in Algorithm 1 (Chapter 3, Page 18). In this chapter we provide a more detailed discussion for two important functions, `registerApp()` (Algorithm 3) and `selectRoute()`.

The `Register Application` state registers the DT with the router and adds new thread to the *ThreadList*. This is implemented in the function `registerApp()`. The `register-`

Algorithm 3: DT Registration in LocDUCE: $registerApp(Utility, ThreadChr)$

```
1  Input:Maximum Possible Utility(Utility), Thread Characteristics.(ThreadChr);
2  graph: Network Graph; t_trans: Transmission time;
3  /* Insert Thread at the position of Utility */
4  for ∀p ∈ P_i, p ∈ [1, n] do
5      if P[p].util > Utility then
6          /* Insert at next position of p */
7          Insert (p + 1, ThreadChr);
8          return (SUCCESS);
9  Insert (n, ThreadChr);
10 return (SUCCESS);
```

`App()` function returns `NEW_APP`, `OLD_APP` or `NEW_AND_OLD_APP` depending upon whether its a new DT or its a DT already registered with the router. `NEW_AND_OLD_APP` is returned when there is a change in the characteristic of a DT already registered. The DT in inserted in the decreasing order of the maximum possible utility ($U_i(t_d)$), where the utility value is determined considering transmission delay ($b^{'}(p_i)/\psi$) on all links leading to the destination and the hop count metric.

The `selectRoute()` function selects the route for the DT. It is called only when there is a new DT that has been registered (i.e. `NEW_APP` event), or a change in the DT characteristics (i.e. `CHG_APP_CHARS`/`NEW_AND_OLD_APP` event(s)). This function executes the Dijkstra's algorithm to determine the route.

5.1.3 GloDUCE

Figure 5.3 shows the finite state machine of GloDUCE. Most of the states are the same as LocDUCE (Figure 5.2). In addition, there are some additional states, "`Send Update`" and "`Register Other Application`." `Send Update` state broadcasts the update information regarding the DTs and their routes to the other routers in the network. `Register Other Application` state is triggered by the update packet arrival (`UPD_PKT_ARR`) and registers the DTs in the update packet. The update packet has information list regarding the DTs that are registered on the other router and the routes they employ. A high level overview of the GloDUCE algorithm is provide in Algorithm 2 (Chapter 3, Page 25).

The implementation of the `registerApp()` function is the same as LocDUCE and hence

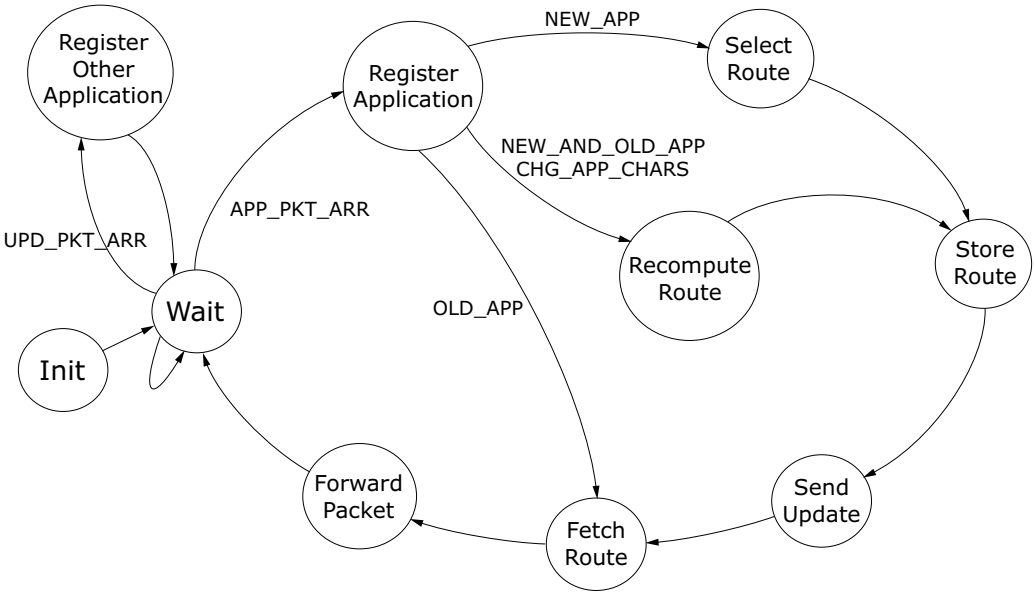


Figure 5.3: State Diagram for Multi-hop Utility Accrual Channel Establishment

is not repeated here. The implementation of `selectRoute()` is the same in the sense that it also employs Dijkstra's. But, the only difference being that the graphs on which the algorithm is applied is slightly. Under LocDUCE, the network graph is updated such that, the outgoing links of a router allocating the channel have the new cost taking the interference of existing channels into account. In GloDUCE, all the links along the path of a channel are updated with the interference. In addition to the `registerApp()` function, there is another function which is triggered by the arrival of the update packet. This function `registerNeighApp()` registers/updates the DT list for the router to include the information contained in the update packet. This function updates the receiving router with the DTs information of the updating router. Thus it maintains the global information of the channels.

5.1.3.1 k-Shortest Path Algorithm

The GloDUCE algorithm employs a k-shortest path algorithm for computing the k different paths to the destination. It then evaluates each path to find the smallest delay path from them, in the decreasing order of utility. The k-shortest path algorithm implemented by Jimenez and Marzal [22] has been altered to configure interfaces and generate the shortest paths in terms of the router identification number and this information is

read by the routing module through a file. The algorithm that has been implemented in [22] implements Eppstein's algorithm in [21]. Eppstein's algorithm has a complexity of $O(m + n \log n + kn)$, where m and n are the number of arcs and vertices in a graph and k is the number of shortest paths that is being determined [21]. In an actual implementation this algorithm has to be embedded into the routing daemon to maintain the multiple paths when it actually determines the shortest path from the network tree it maintains.

5.2 Real-time Application and Middleware

In order to measure the performance of the CE algorithms, we had to design real-time applications and middleware to generate the traffic. The real-time application was a simple client program which uses the middleware's APIs in order to generate the real-time packets.

5.2.1 Real-time Middleware

The real-time middleware consists of the API functions provided for the clients at the application layer. The API functions that the middleware exports and be divided into:

1. Client Initialization Interface
2. Communication Interface

5.2.1.1 Client Initialization Interface (CII)

The Client Initialization Interface (CII) are functions provided to perform the client initializations. This involves initializing the middleware with the DT characteristics and the timeliness properties. This interface is required so that the middleware can embed that information when the DT sends messages. The inputs arguments to these functions are: (1) DT Data Rate (2) TUF Type (3) Maximum Utility, and (4) DT Message Deadline.

These interface functions register the DT with the middleware so that the middleware is aware of the DT characteristics and can embed this information while forwarding its

messages to their destination. The Figure 5.4 shows a sample client implementation code using CIIs and CIs.

```
enum FunctionType {RECT, LINEAR, SOFT_RECT, ...};

typedef struct __dtchars {
    long relativeDeadline;      // Seconds
    int TUF_Type;
    double maxUtility;
} DTCharacteristics;

DTCharactertics DtChars;

void main() {
    ...
    ...
    // Initialize the DT Characteristics
    DtChars.relativeDeadline = 2; // 2 seconds from the start time
    DtChars.TUF_Type = RECT;      // Rectangular TUF
    DtChars.maxUtility = 100;     // Maximum Utility = 100

    // Register DT Characteristics using CIF with the Middleware
    RegisterApplication (DtChars);
    ...

    while (1) {
        ...
        ...
        // Send the data by calling the CF provided by Middleware
        rtSendTo (sock, &to, &data, len);
        ...
    }
}
```

Figure 5.4: Sample Client Implementation Code Showing the CIIs and CIs

5.2.1.2 Communication Interface (CI)

Once the DTs are "registered" with the middleware, using the CII functions it can use the CI functions for the actual communication. The CI functions are actually using the usual socket calls with additional functionality to embed the DT characteristics of the clients into their messages. For example, a function `rtSendTo()` would use the socket API's `sendto()` function. In addition it uses the `setsockopt()` to actually set the user-defined options onto the IP packets. This information is used in the underlying message scheduler (Section 5.3) to perform packet scheduling.

5.2.1.3 Embedding Application Characteristics in the Packet

The API functions of the middleware embed the DT characteristics in the packet. An user defined structure holds this information (DT Identification, the window size ($w(p)$), the number of messages arriving in $w(p)$, $a(p)$, the type of TUF, the maximum utility value

and the message time stamp). This structure is built by the middleware and embedded into the IP Options field, using the `setsockopt()` call. The intermediate routers and message schedulers use this information in making routing and scheduling decisions.

5.3 Message Scheduler

We have implemented the Utility Accrual Packet Scheduler Algorithm (UPA) as given in [14], inside the Linux kernel, at the link layer. This message scheduling algorithm maximizes the accrued utility at a local level. When a message arrives at the output interface queue, the scheduler inserts the message in the queue in the decreasing order of pseudo-slopes. The pseudo-slopes is computed as $\frac{MaximumUtility}{RelativeDeadlineValue}$. When the link becomes available, the dequeuing function examines two consecutive messages, performs a bubble-sort style swap and dequeues the message at the head. The swapping is done only when the total utility obtained by transmitting the second packet before the first. This process is repeated for all packets in the queue, for one iteration. The algorithm also performs a feasibility check on the packet and forwards the messages only if it can meet the deadline. Since, it is a local algorithm it only estimates this using the transmission delay on the output link.

5.4 Summary

In this chapter we have discussed the proof-of-concept style implementation of the routing modules. Routing modules implemented in Linux are typically application level programs, maintaining a kernel level routing table. The `ip_forward()` function, inside the Linux kernel, forwards application messages to the appropriate next-hop, using this routing table. Here, we have followed an approach where the utility accrual routing module is at the application layer, along with the OSPF daemon. This is similar to moving the forward function to the applications layer and passing the incoming application messages to the application layer for the forward decision to be made. Though, this approach has its own disadvantages, it still provides a very strong indication of the performance of the channel

establishment algorithms. In order to measure and compare its performance, we have implemented these modules in a testbed. Several experiments were performed on these implementations and performance measured and analyzed. These are presented next in Chapter 6.

Chapter 6

Experimental Evaluation

In this chapter we discuss the experimental setup for evaluating the channel establishment algorithms proposed here. We describe the testbed configuration and the experimental scenarios that were used to measure the performance of the algorithms. We also present the results and analyze them.

6.1 Testbed Configuration

The testbed consists of Linux machines configured as routers, interconnecting subnetworks, to form a wide area network. The utility accrual routing modules are implemented on these Linux machines running the Redhat Distributions with a 2.4 version of the kernel. In the current implementation, the routing module is a "user process" performing routing functionality. The source and destination nodes form the subnetworks, with the source nodes generating real-time DT traffic and a destination node receiving these messages and analyzing the results. Figure 6.1 shows the topology of the system used to conduct experiments.

The machines S_1 and S_2 formed the source nodes with D serving as the destination node. The machines marked R_1, R_2, R_3 and R_4 are the routers. In Figure 6.1, the letters '`RT`' marked on the routers denote that they are equipped with the UPA message scheduler and the utility accrual routing module. In some cases (e.g. for OSPF experiments) these modules are disabled and OSPF routing algorithm and first-in-first-out (FIFO) queues

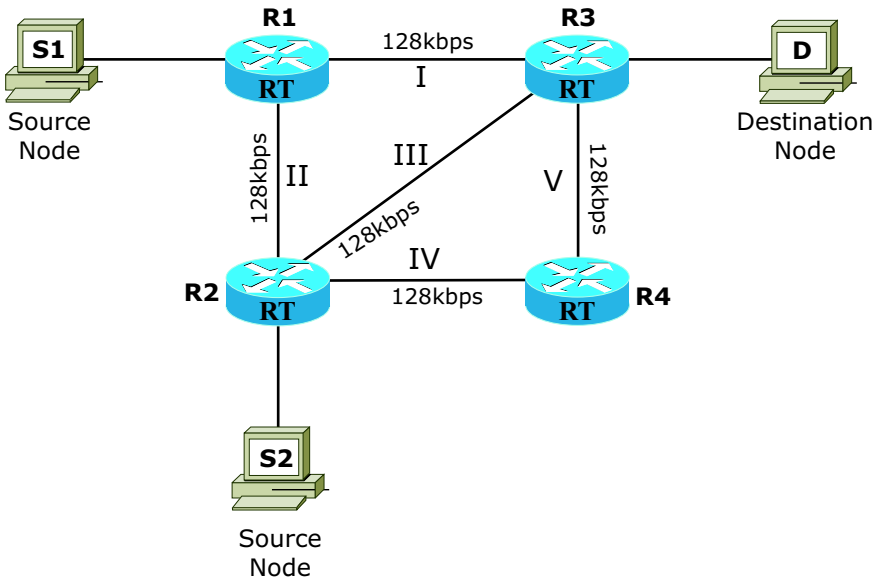
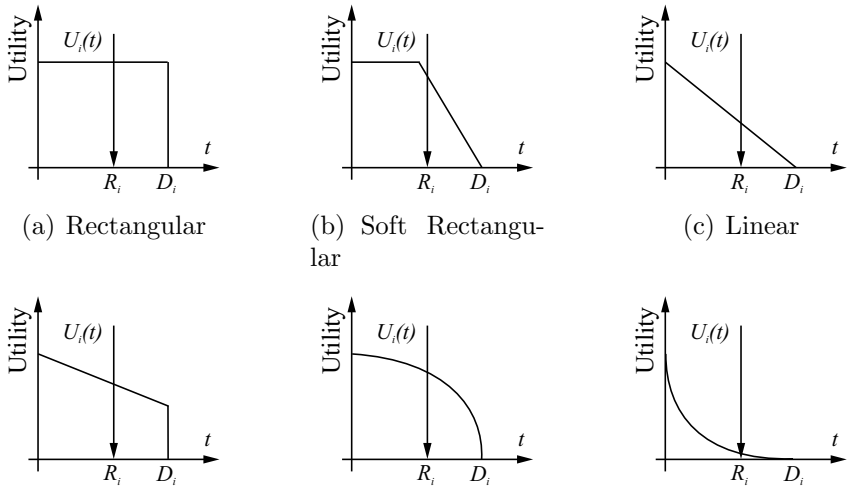


Figure 6.1: Example Network Topology

are used. The links in Figure 6.1 are marked as I. II, to aid in referring to these links in the explanation provided.

To observe the effect of the algorithms and for a fair comparison among the schemes, it is necessary to load the network as is the case in a real-life backbone network. Hence, a hierarchical token bucket (HTB) [27] queuing discipline was employed to limit the bandwidth to 128 kbps, even though the links were capable of transmitting at a rate of 100 Mbps. As, we are using a single machine to generate the messages for a real-time application, the link from the source to the router is not congested. In this prototype implementation we are using the OSPF Routing Daemon to generate the routing table for the utility accrual routing module. In order to measure the performance of the real-time applications, we need to determine the utility obtained and deadline misses. This requires that all the machines in the network to be time synchronized. The clocks of all machines in the testbed are synchronized using network time protocol (NTP) [28]. We use one of the machines in the testbed as the NTP server (i.e., router R_1).



(a) Rectangular (b) Soft Rectangular (c) Linear

(d) Track Association (e) Quadratic (f) Exponential

Figure 6.2: TUFs for DTs used in Experiments

6.2 Experimental Scenarios

The real-time DTs that are used for the experiments are characterized by TUFs as shown in Figure 6.2. There were several scenarios under which the experiments were conducted on the prototype implementation. They can be broadly divided into: (1) Static Scenario (2) Dynamic Scenarios and (3) Overhead Measurement. In the static scenario, the DT characteristics (i.e. the data-rate, message size, inter-arrival times of messages) are kept the same, but each DT had a different TUF type (Figure 6.2). The DTs are distributed on two sources and experiments are conducted to compare OSPF[1], LocDUCE and GLoDUCE. The performance of the three were measured for all the TUFs. In the dynamic scenarios, experiments were conducted by varying message arrival rates, message sizes, varying laxity of deadlines and varying maximum utility. In the overhead comparison experiments, the overhead in the LocDUCE and GloDUCE algorithms are compared. We use a fixed number of DTs and establish channels and measure the channel establishment as well as runtime overheads. Though, the overhead is implementation specific and also depends on the number of DTs, we use this to get a feel of the differences in the complexity of LocDUCE and GloDUCE. In this scenario, we are taking the overhead of OSPF to be "zero". This is because, OSPF being a proactive routing protocol, has no cost associated

[1]In all experiments involving OSPF, the default FIFO queues were used and UPA was disabled.

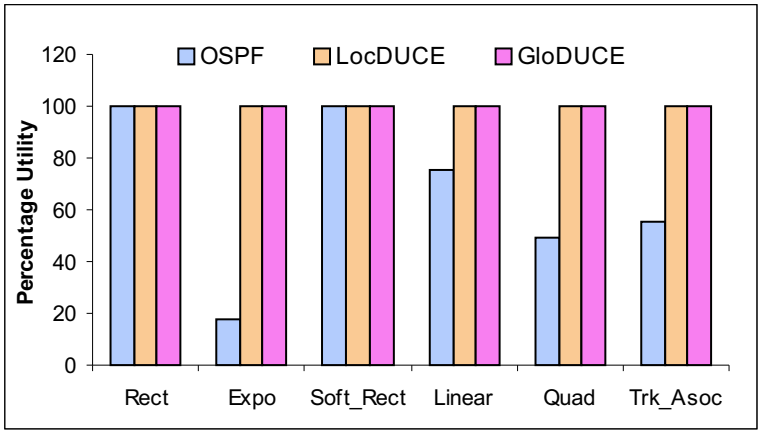


Figure 6.3: Performance in the Static Scenario: Percentage Utility

with forming routes and also does not form routes specific to DTs. The operating system forward messages using the "forwarding" function to the appropriate next hop based on this routing table. The sections below discuss the results that were obtained from the experiments.

6.2.1 Static Scenario

Table 6.1: DT Characteristics for Static Scenario

DTs	TUF Type	Maximum Utility	Deadline	Source Node
DT-1	`RECT`	100	4.0	S_2
DT-2	`EXPO`	100	4.0	S_1
DT-3	`SOFT_RECT`	100	4.0	S_2
DT-4	`LINEAR`	100	4.0	S_1
DT-5	`QUAD`	100	4.0	S_1
DT-6	`RAISED_LINEAR`	100	4.0	S_2

The characteristics of the DTs and their origin (i.e, the source node) is given in Table 6.1. All DTs are destined to a single destination connected to R_d. The DTs are distributed among two sources, S_1 and S_2 for all the experiments. The experiments were conducted to compare the performance of OSPF, LocDUCE and GloDUCE. They were repeated for 100 times and each time 500 packets were generated from the sources. Each point on the graph is an average of the 100 runs.

Figure 6.3 shows the performance in terms of percentage utility obtained in the case of

Table 6.2: Routes Taken by DTs Under OSPF, LocDUCE and GloDUCE

DTs	Source	OSPF	LocDUCE	GloDUCE
DT-1	S_2	R_2–R_3	R_2–R_3	R_2–R_3
DT-2	S_1	R_1–R_3	R_1–R_2–R_4–R_3	R_1–R_2–R_4–R_3
DT-3	S_2	R_2–R_3	R_2–R_3	R_2–R_4–R_3
DT-4	S_1	R_1–R_3	R_1–R_2–R_3	R_1–R_3
DT-5	S_1	R_1–R_3	R_1–R_3	R_1–R_3
DT-6	S_1	R_1–R_3	R_1–R_2–R_4–R_3	R_1–R_2–R_3

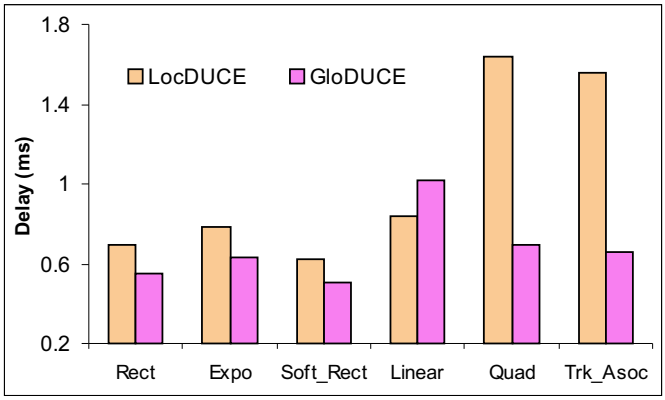


Figure 6.4: End-to-End Delay Comparison between LocDUCE and GloDUCE

OSPF, LocDUCE and GloDUCE. OSPF, by default chooses the shortest path available and this increases the load on the link between R_1–R_3 and affects the performance of all DTs using that link. Also, the scheduler used for the case of OSPF is FIFO and hence packets are forwarded in first-in-first-out order and hence all message flows are equally affected. Thus, the system-wide performance is bad. In the case of LocDUCE, each router maintains a *DTlist*, containing a list of DT flows that it forwards. This information helps the algorithm in balancing the load among all the possible interfaces leading to that destination. It starts assigning the links greedily, first for DTs that are likely to achieve a greater utility, then for the next lower one, and so on. GloDUCE, also works in a similar manner, except that it looks at the entire path from source to destination. Hence, performance of both LocDUCE and GloDUCE are identical for this case. This may not be true for all the cases, as will be illustrated in later sections of this Chapter. Since, GloDUCE maintains global information on the channels formed, it performs a better optimization in terms of system-wide utility achieved.

The Figure 6.4 compares the end-to-end delays for all DTs in the static scenario. The Table 6.2 shows the routes taken by the three approaches. OSPF "always" picks the lowest cost path (which in this case, is the lowest hop count). But, LocDUCE and GloDUCE select different routes and even choose unequal cost paths and balance DT traffic in all possible paths. It can be observed here that for almost all DTs, GloDUCE offers less delay than LocDUCE. The major difference, can be seen for DT-5 and DT-6. Since, DT-5 and DT-6 start at source S_1, LocDUCE performs route selection without the information that DT-1 and DT-2 are being routed through R_2. So, it routes the traffic towards R_2. But, this loads the links between R_2–R_3 and R_2–R_4–R_3. But, GloDUCE maintains information about the DTs that are being routed through R_2 and hence prefers routing DT-5 and DT-6 through the link R_1–R_3. Hence, better performance is achieved.

6.2.2 Dynamic Scenarios

In this section we discuss the various experiments that were conducted in a dynamic scenario. There were several dynamic scenarios that were run on the algorithms. They are:

1. Increasing Packet Arrival Rate
2. Importance Factor (Rectangular Functions)
3. Varying Message Latency

6.2.2.1 Increasing Packet Arrival Rate

In this set of experiments, there were six DTs that were used in the system. Four DTs were started from source S_1 and two from source S_2. DT-3, DT-4, DT-5 and DT-6 arrive at router R_1 and DT-1 and DT-2 arrive at R_2. The TUFs for these DTs are as listed in Table 6.3. The packet arrival rate of each DT was increased from 2 packets/sec. to 10 packets/sec in steps of 2. The performance of OSPF, LocDUCE and GloDUCE were measured and analyzed.

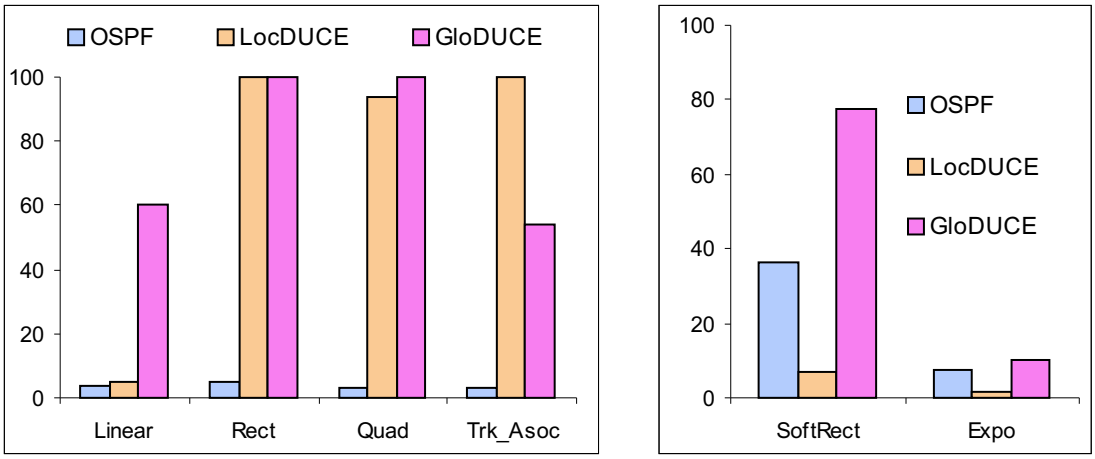
We have plotted the deadline misses and utility obtained for the last point (each DT traffic is at 80kbps, total rate at R_1 is 320 *kbps* and at R_2 is 160 *kbps*) so that the comparison is clearer. In Figure 6.5 we measure the performance in terms of utility obtained by each DT. Figure 6.5(a) and Figure 6.5(b) show the utility obtained for DTs at R_1 and, R_2 respectively. Each Figure shows the DTs that arrive at that router.

Table 6.3: DT Characteristics for Varying Arrival Rate Experiment

DTs	Utility Function Type	Maximum Utility	Deadline	Source Node
DT-1	`SOFT_RECT`	100	4.0	S_2
DT-2	`EXPO`	100	4.0	S_2
DT-3	`LINEAR`	100	4.0	S_1
DT-4	`RECT`	100	4.0	S_1
DT-5	`QUAD`	100	4.0	S_1
DT-6	`RAISED_LINEAR`	100	4.0	S_1

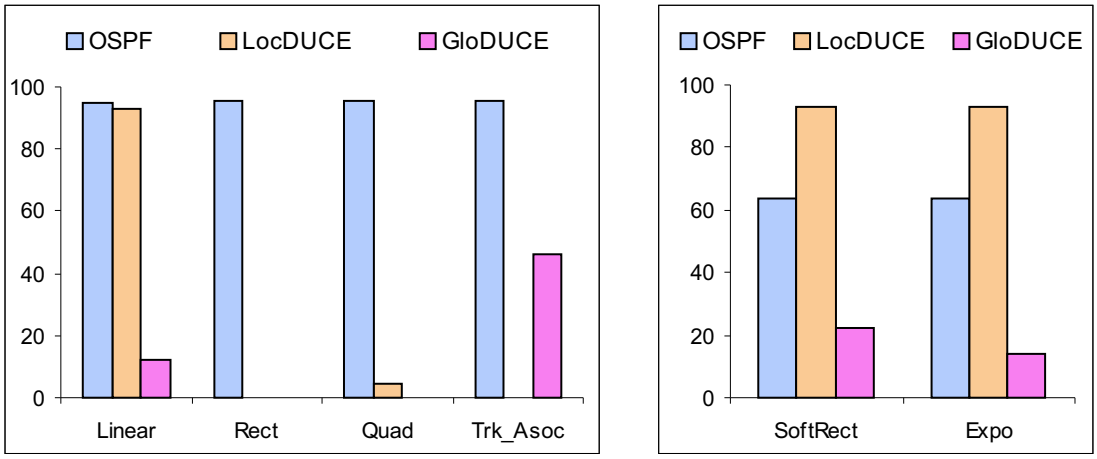
It is very clear from this plot that the system-wide utility obtained from GloDUCE is higher than that of both LocDUCE and OSPF. As, OSPF does not balance loads along unequal cost paths, the performance of all DTs identically degrade. Also, it does not prioritize any particular DT over another, at the interface level, as it uses a FIFO packet scheduler. DT-1 and DT-2 perform some what better, as the link which these DTs use is lightly loaded. But, they have TUFs as described in Figure 6.2(b) and Figure 6.2(f). Hence, the aggregate utility obtained under OSPF is reduced. In the case of LocDUCE, as it prioritizes DTs that achieve higher utility over the others, even though DT messages travel along a longer path, they achieve a higher system-wide utility. Similarly, GloDUCE also performs better and obtains a higher utility than OSPF. In addition, GloDUCE is "intelligent" in sharing the load, as it has global information regarding the channels established in other routers, and hence, is able to achieve even higher system-wide utility (Figures 6.5(a) and 6.5(b)) than LocDUCE. The plots in Figure 6.6(a) and Figure 6.6(b) show the percentage deadline misses for various DTs for the same scenario.

One important observation made from the performance measurement shown in Figure 6.7(a) is that OSPF performs better than LocDUCE. Under OSPF, DT-1 is routed along the shortest route i.e., link III from R_2 to R_3 and performs better than LocDUCE, as the link is lightly loaded. But, when routing is performed by LocDUCE, because of the load-



(a) Utility for DTs at R_1 (b) Utility for DTs at R_2

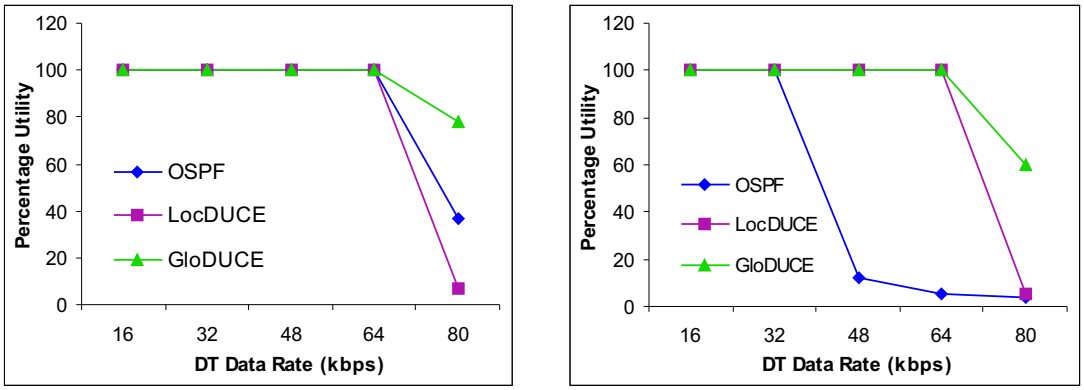
Figure 6.5: Varying Inter-Arrival Times: Utility Obtained Comparison



(a) Deadline Misses for DTs at R_1 (b) Deadline Misses for DTs at R_2

Figure 6.6: Varying Inter-Arrival Times: Deadline Misses Comparison

balancing at router R_1, some DT messages get routed to R_2, increasing the load on the link R_2–R_3. The load under LocDUCE being higher than that under OSPF, DT performance drops for LocDUCE. Figure 6.7(b) shows that OSPF performs very poorly as the link R_1–R_3 becomes heavily loaded. LocDUCE and GloDUCE tend to route away from this heavily loaded link and the DTs achieve a better performance.



(a) DT-1 on R_2 (b) DT-3 on R_1

Figure 6.7: Performance Comparison of DTs under Different Algorithms

6.2.2.2 Importance Factor (Rectangular TUFs)

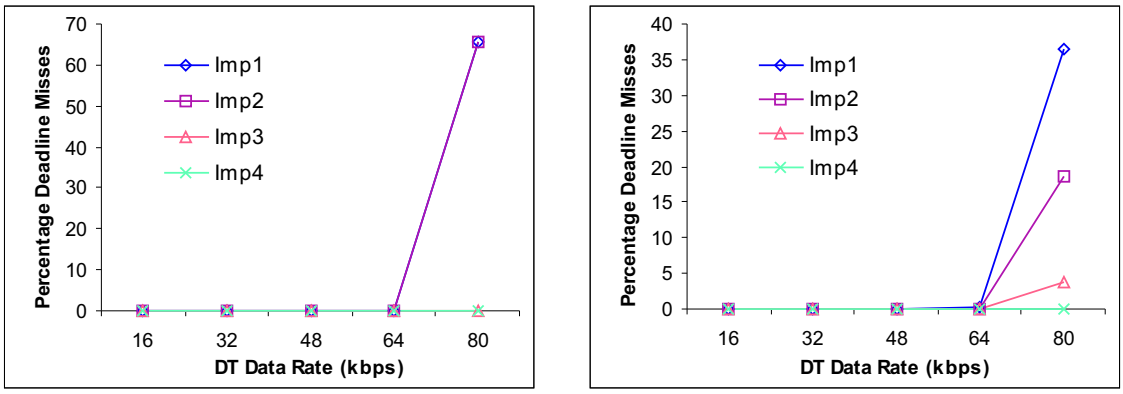
In the experiments that were conducted so far, we have considered dissimilar TUFs with same maximum utility values. But, in actual situations there would be DTs that have different maximum utility values and similar or dissimilar TUFs. In this section we evaluate the performance of the schemes by varying, what we call the "importance factor". Importance factor is basically a constant value that is used to scale the maximum utility of a DT's TUF. For the sake of simplicity we assume similar functions (i.e. all DTs have Rectangular TUFs). This makes it simpler to understand the performance that a particular algorithm would achieve.

Table 6.4 shows the DTs that were used for the experiments. In the table the most important task has the highest importance factor i.e. 4 ("Imp4"), the next lower is "Imp3" and so on. "Imp1" is the least important DT among all of them. All other characteristics of the TUFs are the same (i.e. all TUFs have same deadline and TUF type).

Figure 6.8(b) shows the percentage deadline misses for LocDUCE and Figure 6.8(a) shows

Table 6.4: DTs for Importance Factor Experiments

DTs	Maximum Utility	Deadline	Source Node
DT-1 (Imp1)	100	4.0	S_2
DT-2 (Imp2)	150	4.0	S_2
DT-3 (Imp3)	200	4.0	S_1
DT-4 (Imp4)	250	4.0	S_1



(a) LocDUCE (b) GloDUCE

Figure 6.8: Importance Factor: Percentage Deadline Misses Comparison

the percentage deadline misses for GloDUCE. It can be seen from these results that GloDUCE performs better than LocDUCE, if the overall performance is considered. LocDUCE suffers a larger percentage deadline misses. The system-wide utility obtained from LocDUCE and GloDUCE is shown in Figure 6.9.

6.2.2.3 Downstream Load

Downstream load is defined as the load on a link downstream, with respect to the router under consideration. In Figure 6.10, the downstream link is the link between R_2 and R_3. In this experiment, the load experienced by the link R_2–R_3 is varied by generating traffic flowing through this link and performance of GloDUCE and LocDUCE is measured. The Figure 6.10 shows the scenario in which the experiment was conducted.

In these experiments, the source S_1 generates traffic for 2 DTs each having a data rate of 64kbps. This remains the same throughout the experiments. The source S_2 generates the other DTs whose data rate is varied from 16kbps to 112kbps. This DT traffic is actually

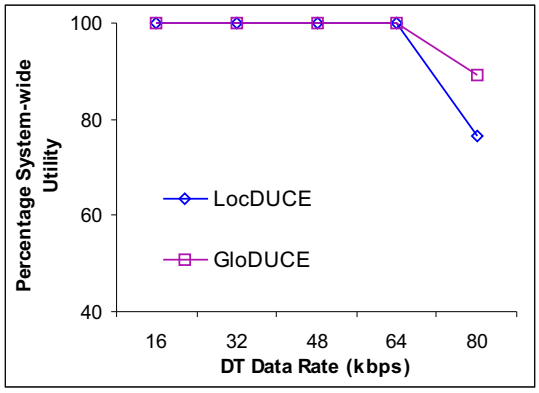


Figure 6.9: Importance Factor: System-Wide Utility Comparison

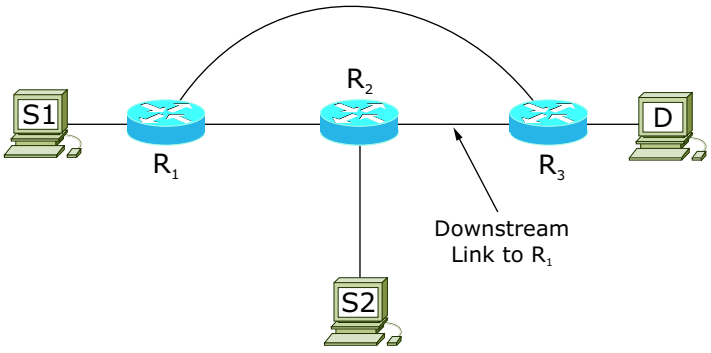
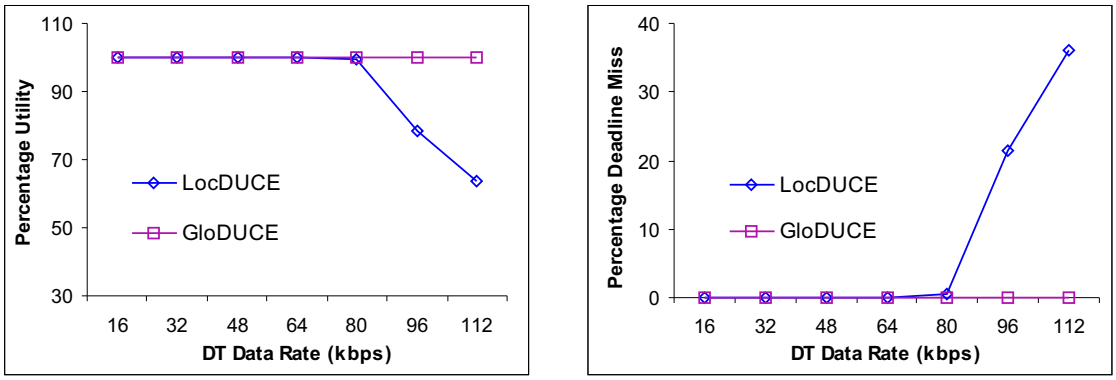


Figure 6.10: Scenario for Loading Downstream Links

the load on the downstream link, as it interferes with the DT traffic coming from router R_1. The Figure 6.11 shows the plot of the percentage utility and deadline misses for this experiment. The plot is with respect to the data rate in *kbps* of the DTs loading the downstream links.



(a) Percentage Utility (b) Percentage Deadline Misses

Figure 6.11: Load on Downstream Links: Performance Comparison for DT-1

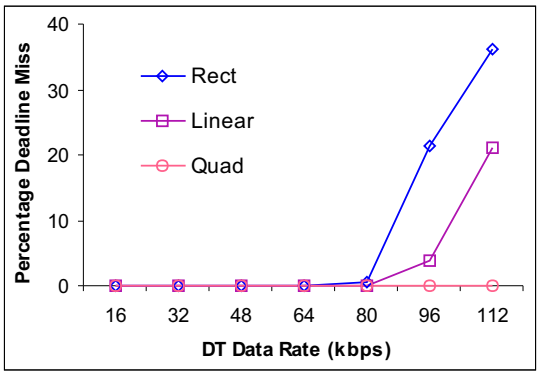
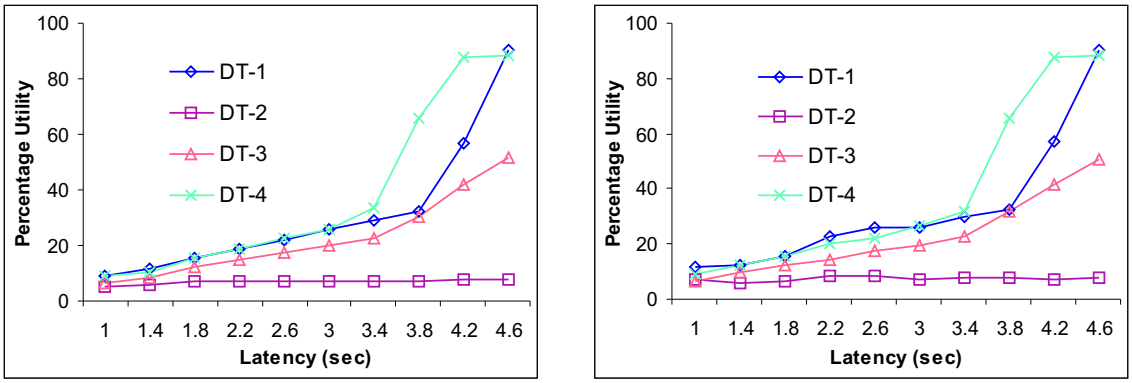


Figure 6.12: Percentage of Deadline Misses for DTs Under LocDUCE

6.2.2.4 Message Latency

Latency is defined as the difference between the actual time taken by a message to reach the destination and the deadline value configured for that message. By varying latency we are increasing/decreasing the sensitivity of the DTs to the variation in network delays. The network topology for this set of experiments was the same as shown in Figure 6.10. The DT messages were generated from sources S_1 and S_2, each generating traffic at a fixed rate of 10 *packets/sec.* (i.e data rate of 80*kbps*). The only variable factor in these

set of experiments is the value of relative deadline. The latency was varied from 1*sec.* to 4.6*sec.* and the performance in terms of application utility and deadline misses was measured. These experiments were conducted for both LocDUCE and GloDUCE.



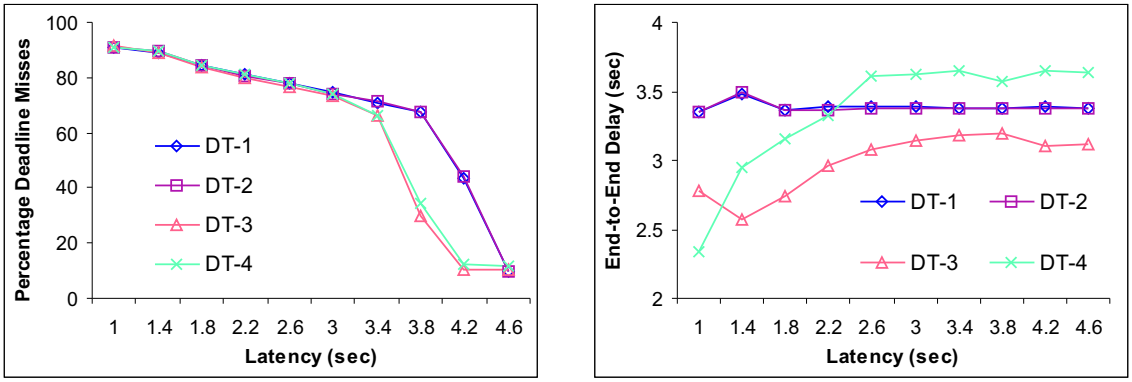
(a) GloDUCE (b) LocDUCE

Figure 6.13: Increasing Latency: Performance Comparison among DTs

The Figure 6.13 shows the plots for the DTs under GloDUCE(Figure 6.13(a)) and LocDUCE (Figure 6.13(b)). It can be seen that an increase in latency value improves the performance of the DTs. Both LocDUCE and GloDUCE show the same characteristics. From the Figure 6.13 we can see that the performance of DT-2 does not improve with increase in latency like other DTs. This can be attributed to the TUF type and contention it might face from other DTs. DT-2 has an exponential TUF, and suffers contention from DT-1 which has a rectangular TUF. Thus the scheduling algorithm favors the rectangular TUF. Also, it can be observed from Figure 6.14(a) that even when the deadline misses have reduced for DT-2, the utility obtained is still less. This is because of the fact that it has an exponential TUF and obtains very low utility when the delay is large (Figure 6.14(b)).

6.2.3 Overhead Measurement

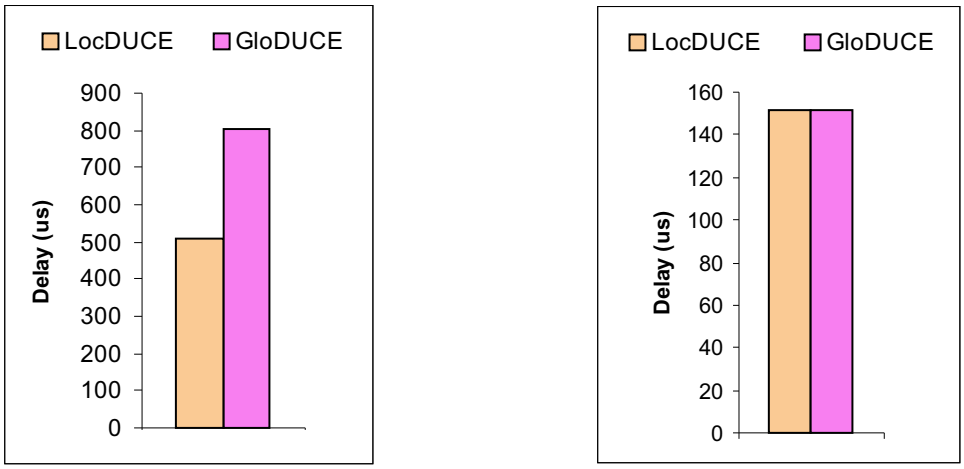
Overhead for LocDUCE and GloDUCE can be divided into (1) Channel Establishment Overhead and (2) Runtime Overhead. Channel Establishment overhead, is the time taken to establish a channel for a particular DT. Here the overhead is proportional to the number of DTs already registered with the router. Runtime Overhead is the overhead for



(a) Percentage Deadline Misses (b) End-to-End Delay

Figure 6.14: Comparisons under GloDUCE for Increasing Latency

determining whether a particular DT already exists or not and find the already selected route for that DT.



(a) Channel Establishment (b) Runtime

Figure 6.15: Overhead Comparison of Channel Establishment Algorithms

We ran some experiments in which only the registration process took place. We measured the overhead for the channel establishment and took an average of several such runs. Similarly, we conducted experiments for determining the runtime overhead and took the average value. Figure 6.15 shows the overhead for LocDUCE and GloDUCE, both during the channel establishment (Figure 6.15(a)) and runtime (Figure 6.15(b)).

6.3 Summary

We have provided a proof-of-concept style implementation and measured and compared the performance of it. The implementation architecture and assumptions made during the implementation make the overhead measurement very specific to this implementation. The performance and overhead can be reduced by following an efficient approach. For example, the routing module performs all the routing at the application layer. All messages received by the interface are forwarded to the application layer. But, the implementation provides us with valuable information regarding the feasibility of the suggested approach. Also, results obtained from this implementation follow the same trend as observed in the simulation experiments. We see that GloDUCE performs better than both LocDUCE and OSPF. There are some cases when LocDUCE performs worse in comparison to OSPF. We have also highlighted the reasons for this. GloDUCE eliminates these defects in LocDUCE and performs better than OSPF even for those cases. The Chapter 7 provides a derivation of the equations used by the algorithms to allocate channels and achieve a higher system-wide utility.

Chapter 7

Delay Analysis

In this chapter, we provide a discussion about the delay analysis and provide a derivation of the equations used by the routing modules to estimate the delay incurred by a message while traveling from a source, through the routers, to a destination. This analysis is based on the the Linux TCP/IP Stack implementation in Linux.

7.1 Life of a Packet through Operating System Stack

When any message arrives at the network interface of a node it triggers various hardware and software interrupts and the protocol stack implementation process the message. Here we discuss the life of a packet after it is received by the network interface.

7.1.1 Receiving a Packet

The receive process is triggered by the arrival of a packet at the wire, typically by a hardware interrupt, raised by the network device driver. The received packet is put from the hardware buffer into the IP Packet Queue. This triggers a software interrupt so that the higher layer can process the packet. The priority of the software interrupts is lower than that of a hardware interrupt as shown in the Figure 7.1 (derived from [29]).

The next higher layer is the IP Layer which picks up the packet and performs error checking and processes the packet. If the packet is destined to itself, it sends it to the higher layers by placing it on the socket queues. The IP Layer identifies the queues based

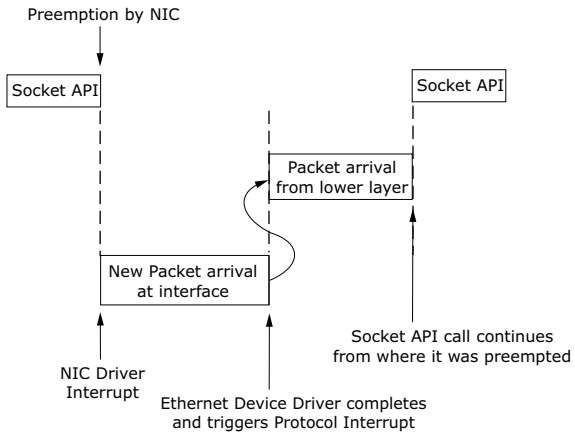


Figure 7.1: Interrupt Levels for scheduling of Network Tasks in the Operating System

on the socket descriptor number and places the packets on the corresponding socket queue. The receiving packet follows the left side of the Figure 7.2 (derived from [29]) with arrows pointing upwards. In this Figure, we use plural for the socket queues and singular for IP Packet Queue. This is because, there are multiple socket queues, one for each socket, but there is a single IP Queue for the whole operating system stack.

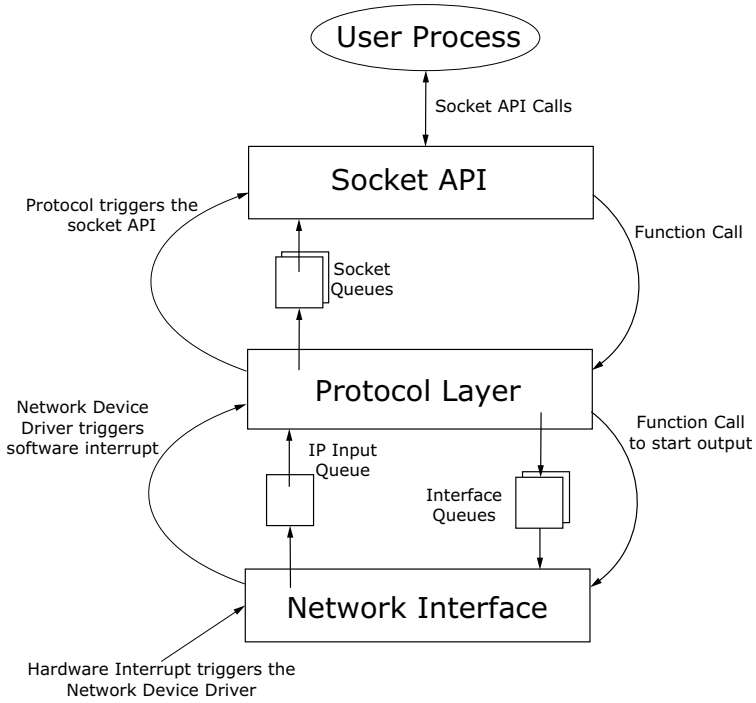


Figure 7.2: Communication between the Layers for Input and Output

7.1.2 Sending a Packet

The sending process is triggered by the application programs (user processes) executing system calls belonging to the socket API like `send()`, `sendto()` and `sendmsg()` or any other filesystem API calls like `write()`. The parameters of these function calls are passed to the transport layer protocols like UDP or TCP, which in turn trigger kernel state machines for passing the information. The protocol layers add the appropriate headers and finally place the packet on the interface queue for output by the device. The device is woken up and transmission of the packet begins.

7.2 Delay Analysis with Linux Implementation

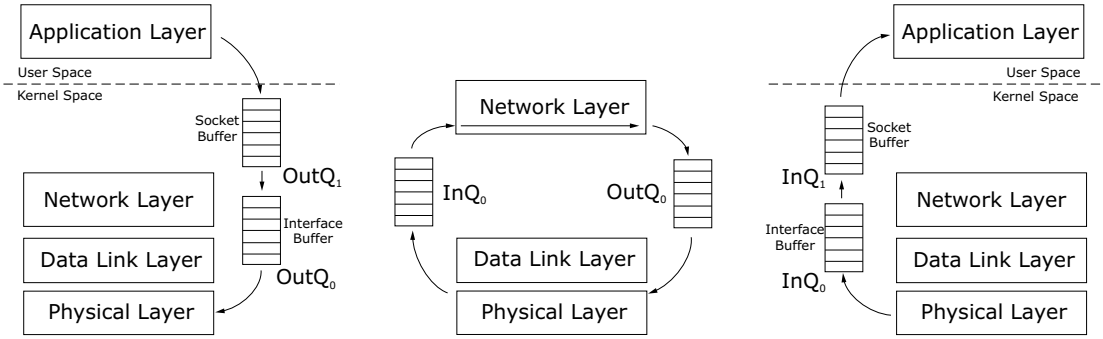
Section 7.1 shows the path a packet takes once inside the machine. This provides the basis for the delay analysis and to estimate the delay a packet might incur while traveling from the source to the destination in a WAN. The packets might not travel through all the layers shown in Figure 7.2. So, the delay incurred will be different in each node the packet passes through. In the sections that follow we shall analyze the delay a packet incurs at the source and, destination node, as well as the router in its path.

7.2.1 Single Hop Delay

Figure 7.3 shows the typical set of input and output queues through which message packets flow from the application-layer in a source node, via an intermediate-node such as a router, to the application-layer in a destination node. To estimate the end-to-end delay incurred by a message packet from the source application-layer to the destination application-layer, the total delay incurred by the packet in all the queues must be accounted for. Note that in our system model, all packet queues are scheduled by UPA.

7.2.1.1 Delay at First Output Queue

Consider an application packet p that arrives at the first output queue at a source node i (denoted $OutQ_1$ in Figure 7.3(a)). In [14], we show that the upper bound on the total



(a) Source Node (b) Router (c) DestinationNode

Figure 7.3: Node Input/Ouput Queues

delay incurred by p in this queue is given by:

$$O_1^i(p) = \sum_{q \in P_i} \left\lceil \frac{X(p) + X(q)}{w(q)} \right\rceil a(q)\,\delta_i + \left\lceil \frac{X(p)}{\theta} \right\rceil \delta_i \tag{7.1}$$

where P_i is the upper bound on the number of packets that can arrive at the output queue for outward transmission and δ_i is the aggregate worst-case, execution, dispatching, queue-transfer time of UPA (on source node i) for scheduling, dispatching, and transferring the packet from the output queue.

The upper bound $O_1^i(p)$ on packet delay derived in [14] is *sufficient, but not necessary.* This is because, the upper bound $O_1^i(p)$ is established in [14] by observing that any packet q will be scheduled by UPA before packet p, only if q arrives no sooner than $A(p) - X(q)$ and no later than $A(p) + X(p)$, where $A(p)$ denote p's arrival time at the output queue.

This interference interval for packet p, $I(p) = [A(p) - X(q), A(p) + X(p)]$, which has a length $X(p) + X(q)$, is only sufficient for a packet q to interfere with the transmission of packet p. Packet q can interfere with packet p only if q arrives during this interval. That does not imply that q will *always* interfere with p, if q arrives during this interval. It only implies that for q to interfere with p, it must arrive during this interval. But even if q arrives during this interval, it may very well not interfere with p. *This is precisely due to UPA's scheduling order.*

For example, if q were to arrive after $A(p) + X(p) - X(q)$ but before $A(p) + X(p)$, the absolute termination time of q will occur after that of p. Under EDF, q will then be scheduled after p, since q has a longer absolute deadline (or termination time) than that

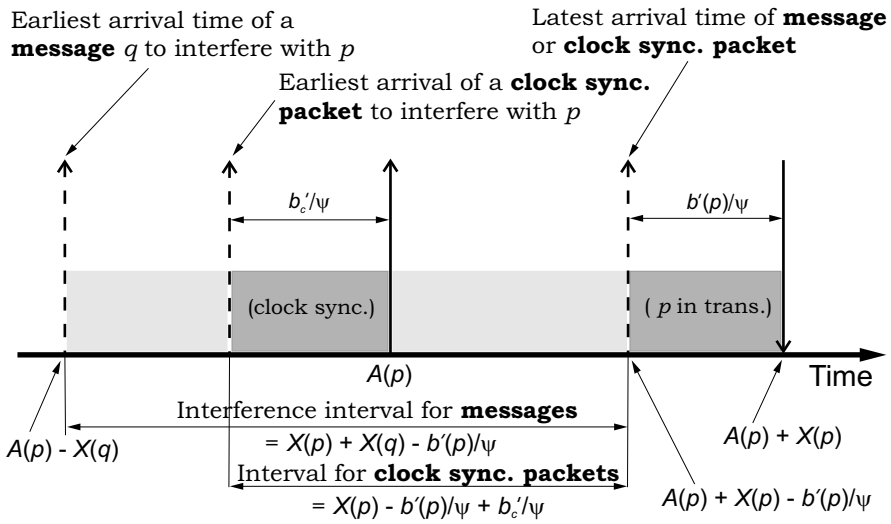


Figure 7.4: UPA's Packet Interference

of p. However, under UPA, it is quite possible that q can be scheduled before p, because q may have an extremely large utility than p and thereby q may appear earlier in UPA's schedule than p (due to UPA's re-rankings). Figure 7.4 illustrates the packet interference situation under UPA.

Developing a necessary and sufficient interference interval will require schedule construction. To avoid this and still obtain a "tighter" interference interval, we consider the notion of an "optimistic" interference interval during which a packet q may interfere with packet p with very likelihood.

Observe that UPA sorts packets by decreasing order of pseudo-slopes. Thus, we regard that if the pseudo-slope of packet q is larger than that of packet p, for all times during p's (sufficient) interference interval $I(p)$, then q has a very high likelihood for interfering with p's transmission. Thus, if $\Gamma(p,t)$ denotes the pseudo-slope of packet p at time t, then our first optimistic interference condition becomes:

$$\Gamma(q,t) > \Gamma(p,t), \forall t \in I(p) \tag{7.2}$$

Another key step in UPA's scheduling process is examining adjacent pairs of packets in the schedule, computing Δ, and swapping the packets, if the reverse order can lead to higher local aggregate utility. Thus, we regard that if $\Delta_{q,p}(t) \geq 0$ for all time instants t during the interval $I(p)$, then again packet q has a very high likelihood for interfering

with p's transmission. This becomes our second optimistic interference condition:

$$\Delta_{q,p}(t) \geq 0, \forall t \in I(p) \tag{7.3}$$

We thus determine an *optimistic* upper bound on packet p's delay in the first output queue at a source node i using Equation 7.1 by considering all packets $q \in P_i$, only if q satisfies Equations 7.2 and 7.3.

7.2.1.2 Delay at Second Output Queue

The delay incurred by a packet at the second output queue (denoted $OutQ_0$ in Figure 7.3(a)) can be similarly derived, except for two issues that will affect the delay. These include: 1. The input arrival rate of packets into the second output queue will now change, as it will depend upon the rate at which packets will be output from the first output queue; and 2. packet transmission times on the outgoing network link (from the source node to the next intermediate node) must be taken into account.

For a packet p that can arrive at the first output queue of a source node i for a maximum of $a(p)$ times during $w(p)$, the rate at which the packet will be output (from the first output queue) is given by:

$$R_1^i(p) = \left\lceil \frac{O_1^i(p)}{w(p)} \right\rceil a(p) \tag{7.4}$$

This will be the rate at which p will arrive at the second output queue.

For a clock synchronization packet c, the arrival rate at the second output queue is given by:

$$R_1^i(c) = \left\lceil \frac{O_1^i(c)}{\theta} \right\rceil \tag{7.5}$$

Thus, the (optimistic) upper bound on the delay incurred by a packet p to arrive at the next intermediate node, since its arrival at the source node's second output queue is given as:

$$\begin{aligned} O_0^i(p) = \sum_{q \in P_i} & \left[X(p) + X(q) - \frac{b'(p)}{\varphi} \right] R_1^i(q) \left[\frac{b'(q)}{\varphi} + \delta_i \right] \\ & + \left[X(p) - \frac{b'(p)}{\varphi} + \frac{b'_c}{\varphi} \right] R_1^i(c) \left[\frac{b'_c}{\varphi} + \delta_i \right] \end{aligned} \tag{7.6}$$

7.2.2 Total Delay at a Node

Thus, the (optimistic) upper bound on the total delay incurred by a packet p to arrive at the next intermediate node, since its arrival at a source node i's first output queue is given as:

$$N_i\left(p\right) = O_1^i\left(p\right) + O_0^i\left(p\right) \tag{7.7}$$

Observe that an (optimistic) upper bound on the total delay incurred by a packet p to arrive at a node j (which can either be an intermediate node or be the packet's destination node), since its arrival at *any* intermediate node i's first (input) queue (denoted InQ_0 in Figure 7.3(b)) is the same as that given by Equation 7.6. This is because the two queues at an intermediate node (denoted InQ_0 and $OutQ_0$ in Figure 7.3(b)) directly correspond to the two queues at a source node (denoted $OutQ_1$ and $OutQ_0$ in Figure 7.3(a)).

7.2.3 End-to-End Delay

A channel from source node i to destination node j is a sequence $C_i^j = \langle n_0, n_1, n_2, \cdots, n_m \rangle$ of m hops, where n_0 represents node i and n_m represents node j. To determine an upper bound on the end-to-end delay incurred by a packet on a channel (from source application-layer to destination application-layer), we need to aggregate the delay incurred by the packet at each node in the channel. Thus, an (optimistic) upper bound on the total delay incurred by a packet p on a channel C_i^j of m hops is given by:

$$R\left(p, C_i^j\right) = \left[\sum_{i=1}^{m} N_i\left(p\right)\right] + \hat{N}_j\left(p\right), \tag{7.8}$$

where $\hat{N}_j(p)$ denotes an optimistic upper bound on the total delay incurred by packet p to arrive at the application-layer at its destination node j, since its arrival at the first input queue of node j. $\hat{N}_j(p)$ is determined similar to Equation 7.7, except that the packet transmission times on the outgoing network link considered in Equation 7.6 is excluded.

7.3 Delay in Actual Implementation

In the sections described above, an important assumption was made; The queues are ordered in UPA order, i.e. in the decreasing order of the pseudo-slopes as described in Section 5.3. This would mean that the internal queues that the operating system kernel maintains at various levels (Section 7.1) have to be ordered in this fashion.

In the input direction, the packets queued at the IP Layer Queue are the only packets that face contention, as all incoming packets are queued here. Usually this queue size is around 50 packets [29]. Also, the priority of the IP Layer is higher than application processes, second only to the hardware interrupts raised by the network card driver 7.1. At the socket queue, only packets belonging to a particular application are queued, hence there is no contention. In this implementation, since the router sniffs packet off the interface and queues packets in the socket queue (all applications are sniffed by the same `PF_PACKET` socket), it is similar to the IP Layer Queue. Since, the implementation of the socket level functions has not been altered to de-queue the socket queue in UPA manner, the delay has been assumed to included in the delay from the time the routing module reads the packet to the time it places the packet on the link. In fact all measurements of the overhead of the routing module include this delay.

At the end hosts, i.e. source and destination machines these delays have been neglected. Since, the real-time packet generator is UDP based, there is no queuing at the socket queue. So, it is acceptable to assume that the only delay that a packet, leaving the source, incurs is the interference from other real-time application packets originating from the same node. In our implementation, this delay is negligible as the bandwidth between the source and the gateway is not constrained in any manner. Hence the delay from source to the router is neglected. Similarly, the delay at the destination node is also neglected.

Chapter 8

Past and Related Efforts

To guarantee timely delivery of real-time messages over a multi-hop network, the concept of *real-time channels* was developed. It was developed during the DASH project [30]. A Real-time channel (RTC) can be defined as *a unidirectional virtual circuit which once established, is guaranteed to meet application-specified performance requirements as long as the application does not violate its a priori specified traffic-generation characteristics* [15]. The channel establishment procedure can be divided into two distinct phases: off-line channel establishment and online packet scheduling. Though various channel establishment schemes have been proposed in literature [15, 31–33], very few of them explicitly address the issue of selecting a route from the source to the destination of a channel and satisfy end-to-end guarantees.

8.1 Real-time Channel Establishment

Real-time channel (RTC) was first defined in the work [15] of Ferrari and Verma. The scheme of providing real-time services by establishing real-time channels in wide area networks was proposed in this work. They also establish deterministic and statistical delay bounds for the real-time traffic in such networks. The work [33] develops a scheme for computing guarantees for delivery time of messages belonging to real-time channels. This paper contends that latency of packets travelling through a point-to-point link can be made more predictable using message scheduling and network flow control. The channel establishment consists of two distinct phases: channel establishment phase and run-time

message scheduling. Channel establishment is the selection of the appropriate route and run-time packet scheduling mechanism ensure that the properties that the channel was to guarantee are not violated. They also propose a channel establishment algorithm.

Developing on the scheduling mechanism described in [33], Shin et. al. in [34] propose a distributed route selection scheme to form RTCs. The only difference from [33], is that this scheme is taking into account the flows that are currently in the process of establishing a channel. The algorithm searches for a route in parallel by flooding connection requests through the network and prunes infeasible routes quickly. In their proposed scheme, each node maintains and exchanges with their neighbors certain information pertaining to the real-time applications going through them. They assume the existence of a multi-class Earliest Due Date (EDD) first scheduling algorithm, at the interface. The worst-case delay for the message flows, in the path from source to destination, is computed using fixed priority scheduling. Since, the algorithm also takes into account the channels that are in the process of being established, to determine the worst-case response times, the algorithm overestimates the response times of the channels. This might lead to denial of channel establishment requests in certain cases.

Another approach taken by Manimaran et. al. in [35] propose a parallel probe algorithm, but with different heuristics along different paths. They employ a two-pass scheme for the channel establishment phase; (1) *Reservation phase*, in the forward direction from source to destination to reserve resources; (2) *relaxation phase*, in the reverse direction, for releasing excess resources allocated during the previous phase. Admission control tests are performed during the reservation phase. If a call is rejected, resources reserved along the forward path are released. The probing takes place along k different paths simultaneously and to avoid excessive reservation or resources in all those paths, they propose a concept of *Intermediate Destinations* or IDs. The heuristic, based on which, the path has to be determined, is used to decide which of the neighbors is selected by a node.

8.2 Fault Tolerance for RTCs

In an internet-like multi-hop packet-switched network, messages experience random delay due to contention from other traffic flowing through the network. It is clear the messages belonging to a particular real-time channel/session would perform only when the path has reserved resources for it and, any change in the path, would cause a reduction in performance. In order to sustain the performance, or for graceful degradation, it might be necessary for some kind of fault tolerance scheme like, forward recovery or detect and recover approach. There has been several ideas that have been proposed for these schemes.

For providing fault tolerance the concept of "Dependable Real-time Communication" was conceived in [36], where a dependable communication channel consists of a primary channel and one or more backup channels. Real-time channels are established during the establishment phase by allocating sufficient resources for the application. But, in order to deal with network component failures, sparse resources are also reserved and, a backup channel is determined, during the establishment phase. In the event of a failure of the primary channel, the backup is activated. But, if the backup itself were to fail, another backup channel is determined. The authors of [36] also propose solutions for two key issues (a) backup route selection and (b) sparse resource calculation. In order to limit the sparse resources that are allocated, they propose schemes for resource multiplexing.

Forward recovery and detect approach for fault tolerance are best applicable for hard real-time systems, where-as the detect and recover approach for soft real-time applications. If there were both hard and soft requirements for a particular application, this is addressed by both these schemes. A mixture of these schemes is proposed in [37]. The proposed scheme uses bandwidth splitting and Forward Error Correction (FEC) advantages of dispersity based forward recovery approach. In addition it also employs the efficient sparse resource allocation used in detect and recovery approach of [36].

It is clear from the above discussion that proactive schemes like forward recovery has a huge resource overhead compared to the light-weight reactive schemes. The construction of backup channels, called "segmented backups" was proposed by [38]. A segmented backup consists of multiple backup channels each spanning contiguous portion of the

primary path. This is unlike an end-to-end backup spanning the entire length of the primary path. These segmented backups can also be generated optimized for different goals such as, better resource utilization versus better fault-recovery delay. From this approach, the researchers show improved network resource utilization, higher average call acceptance rate and better quality of service guarantees.

8.3 Related Work at Virginia Tech

The Real-time Systems lab at Virginia Tech conducts active research in the area of TUF scheduling. One of the earlier works by J. Wang proposes a TUF based packet scheduling algorithm called UPA in [14]. This algorithm was employed to design a real-time UA Ethernet switch to schedule messages that are characterized by TUFs. This work is limited only to the local area networks. In contrast to [14], this thesis proposes a channel establishment algorithm and extends the solution to larger networks like WANs.

Most of the past efforts on real-time channel establishment such as [15, 39–42], including the ones elaborated above, focus on deadline timing constraints and the hard timeliness optimality criteria that, *all deadlines must be satisfied*. They employ a two stepped process for channel establishment. One distinct phase for admission control, when the channel is "initialized" and resources allocated, and another phase in which the application packets use the channel for communication. Channel establishment algorithms for message flows that have arbitrary timeliness utility function constraints such as, linearly decreasing utility with respect to activity completion time[1], and consider soft timeliness optimality criteria such as maximizing accrued utility (MAU) have not been studied. This thesis is the first to address this problem.

[1]See Figure 1.2 for example non-rectangular timing constraints.

Chapter 9

Conclusions and Future Work

In this thesis we have presented utility accrual channel establishment algorithms for wide area networks. We have successfully implemented it in Linux operating system and evaluated its effectiveness by several performance measurement experiments. We have compared it to the widely used routing protocol for performing effective application characteristic specific route selection. The developed algorithms apply to the message model where applications are characterized by Jensen's time utility functions and in that "unimodal non-increasing functions". Both the proposed schemes aim to maximize the system-wide accrued utility and do very well in those aims. Since, computing a multi-constraint route in wide area networks is a $\mathcal{NP}$–hard problem, we have devised heuristic methods for performing the route establishment.

9.1 Advantages and Disadvantages of LocDUCE

LocDUCE performs a local optimization of the utility values and aims to achieve a higher system-wide utility. This makes the algorithm relatively simple (with respect to GloDUCE). The amount of information that needs to be stored in each router is limited to the number of interfaces there are on each router. When channels do not need to use common links in their path to the destinations, performance of LocDUCE is same as GloDUCE.

Since, LocDUCE performs a local decision, there are cases where the route is not optimum.

In cases, where the downstream links are loaded, its performance might be worse than OSPF.

9.2 Advantages and Disadvantages of GloDUCE

On the contrary, GloDUCE performs a global optimization, taking all routers in the system into account and thus achieves a even higher MAU. There are several conditions under which LocDUCE and GloDUCE perform the same. But, under some other conditions GloDUCE performs better. In most cases performs better than OSPF or LocDUCE. This scheme is best suited for mesh networks, with shorter path lengths, from source to destinations.

Global information of all routers in the system may amount to a lot of information. When a new channel is created, this information has to be broadcasted to all routers in the system. This can flood the network with channel update packets, if there are applications arriving at different times. Since, the complexity is directly proportional to the number of routers along the path to the destination (hops), more hops, more is the complexity for determining the route.

9.3 Future Work

There are several directions in which this research can take shape in the future. The delay estimation equations have to be made much tighter, thus reducing the difference between the estimated and actual delay value. The conditions that are derived in Chapter 7 are necessary only, and not sufficient. The performance of the channel establishment algorithms can be improved by implementing the algorithms in the operating system kernel. This will reduce the overhead an user-level program would experience. Integrating the algorithms with OSPF, specifically changing the message formats OSPF uses, to incorporate the exchange of DT characteristics can be a necessary modification to OSPF. Thus the functionality of the channel establishment can be implemented in OSPF like link state protocol. Incorporating fault tolerance in the channel establishment would

make the UA channel establishment algorithms robust to host and network failures and aid in recovering from faults and gracefully degrade the performance of the applications using these channels.

Maintaining state information about the real-time channels established, local in the case of LocDUCE and global in the case of GloDUCE, is not scalable. For large networks this information can be complex to maintain and use for performing channel establishment. New approaches to effectively reduce, or totally eliminate state information need to be devised. This will make the approach scalable and render it useful even for large networks. The same UA routing approach can be extended to wireless environments where frequent topology changes can occur.

Bibliography

[1] OMG, "Real-time CORBA 2.0: Dynamic Scheduling Specification," OMG Final Adopted Specification, Tech. Rep., September 2001.

[2] E. D. Jensen and J. D. Northcutt, "Alpha: A Non-Proprietary Operating System for Large, Complex, Distributed Real-time Systems," in *Proceedings of The IEEE Workshop on Experimental Distributed Systems*, 1990, pp. 35–41.

[3] T. O. G. R. I. R.-T. Group, *MK7.3a Release Notes*. Cambridge, Massachusetts: The Open Group Research Institute, October 1998.

[4] D. Wells, "A Trusted, Scalable, Real-Time Operating System," in *Proceedings of The Dual-Use Technologies and Applications Conference*, 1994, pp. 262–270.

[5] "Multi-Platform Radar Technology Insertion Program," http://www.globalsecurity.org/intell/systems/mp-rtip.htm/.

[6] "BMC3I Battle Management, Command, Control, Communications and Intelligence," http://www.globalsecurity.org/space/systems/bmc3i.htm/.

[7] E. D. Jensen, "Asynchronous Decentralized Real-time Computer Systems," in *Real-Time Computing*, W. A. Halang and A. D. Stoyenko, Eds. Springer Verlag, October 1992.

[8] P. Li, B. Ravindran, H. Wu, and E. D. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Transactions on Computers*, submitted August 2003 (under review). Available at: http://nile.ece.vt.edu/submissions/GUS-TOC03.zip.

[9] C. D. Locke, "Best-Effort Decision Making for Real-time Scheduling," Ph.D. dissertation, Carnegie Mellon University, 1986, CMU-CS-86-134.

[10] R. Clark, "Scheduling Dependent Real-time Activities," Ph.D. dissertation, Carnegie Mellon University, 1990, CMU-CS-90-155.

[11] G. Koren and D. Shasha, "D-Over: An Optimal On-line Scheduling Algorithm for Overloaded Real-Time Systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1992, pp. 290–299.

[12] K. Chen and P. Muhlethaler, "A Scheduling Algorithm for Tasks Described by Time Value Function," *Journal of Real-Time Systems*, vol. 10, no. 3, pp. 293–312, May 1996.

[13] D. Mosse, M. E. Pollack, and Y. Ronen, "Value-Density Algorithm to Handle Transient Overloads in Scheduling," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, June 1999, pp. 278–286.

[14] J. Wang and B. Ravindran, "Time-Utility Function-Driven Switched Ethernet: Packet Scheduling Algorithm, Implementation, and Feasibility Analysis," *IEEE Transactions on Parallel and Distributed Systems*, 2003, accepted August 2003, To appear, Available at http://nile.ece.vt.edu.

[15] D. Ferrari and D. C. Verma, "A Scheme for Real-time Channel Establishment in Wide Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368–379, April 1990.

[16] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley, "An Adaptive, Distributed Airborne Tracking System," in *Proceedings of The Seventh IEEE International Workshop on Parallel and Distributed Real-Time Systems*, ser. Lecture Notes in Computer Science, vol. 1586. Springer-Verlag, April 1999, pp. 353–362.

[17] D. Maynard, S. Shipman, R. Clark, J. Northcutt, R. Kegley, B. Zimmerman, and P. Keleher, "An Example Real-time Command, Control, and Battle Management

Application for Alpha," CMU Computer Science Dept., Tech. Rep., December 1988, Archons Project Technical Report 88121.

[18] G. L. Lann, "Proof-Based System Engineering and Embedded Systems," in *Lecture Notes in Computer Science*, G. Rozenberg and F. Vaandrager, Eds. Springer-Verlag, October 1998, vol. 1494, pp. 208–248.

[19] Z. Wang and J. Crowcroft, "Quality-of-Service Routing for Supporting Multimedia Applications," *IEEE Journal of Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[20] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for Next-Generation High-Speed Networks: Problems and Solutions," *IEEE Network*, vol. 12, pp. 64–79, November/December 1998.

[21] D. Eppstein, "Finding the k Shortest Paths," *SIAM J. Computing*, vol. 28, no. 2, pp. 652–673, 1998. [Online]. Available: http://www.ics.uci.edu/~eppstein/pubs/Epp-SJC-99.pdf

[22] V. Jimenez and A. Marzal, "Implementation of the Eppstein's k-Shortest Path Algorithm," http://terra.act.uji.es/REA/.

[23] G. Armitage, *Quality of Service in IP Networks: Foundations for a Multi-Service Network*. Macmillan Technical Publishing, 2000.

[24] B. Huffaker, E. Nemeth, and K. Claffy, "Otter: A General-purpose Network Visualization Tool," in *Proceedings of INET*, June 1999, Internet Society.

[25] A. Barábasi and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, pp. 509–512, October 1999.

[26] G. Insolvibile, "The Linux Socket Filter: Sniffing Bytes Over the Network," Linux Journal, Issue 86, June 2001. [Online]. Available: (http://oceanpark.com/webmuseum/linuxjournal/issue86-june2001/)

[27] M. Devera, "Hierarchieal Token Bucket (HTB) Packet Scheduler." [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/

[28] D. L. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks," *IEEE/ACM Transactions on Networks*, vol. 3, pp. 245–254, June 1995.

[29] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated Volume 2: The Implementation.* Addison Wesley Publishers, 1999.

[30] D. P. Anderson and D. Ferrari, "An Overview of the DASH Project," UCB/CSD University of California Berkeley, Tech. Rep., February 1988.

[31] D. D. Kandlur and K. G. Shin, "Design of a Communication Subsystem for HARTS," CSE Division, Dept. Of Electrical Eng. and Computer Science, Univ. of Michigan, Tech. Rep., December 1991, technical Report CSE-TR-109-91.

[32] K. G. Shin and C. C. Chou, "Design and Evaluation of Real-time Communication for Field Bus Based Manufacturing Systems," *Proceedings of the IEEE Local Computer Network Symphosium*, pp. 483–492, September 1992.

[33] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time Communication in Multi-hop Networks," *IEEE Transactions in Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, October 1994.

[34] K. G. Shin, C. Chou, and S. Kweon, "Distributed Route Selection for Establishing Real-time Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 3, pp. 318–335, March 2000.

[35] G. Manimaran, H. S. Rahul, and C. S. R. Murthy, "A New Distributed Route Selection Approach for Channel Establishment in Real-time Networks," *IEEE Transactions on Networking*, vol. 7, no. 5, pp. 698–709, October 1999.

[36] S. Han and K. G. Shin, "A Primary-backup Channel Approach to Dependable Real-time Communication in Multi-hop Networks," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 46–61, January 1998.

[37] S. Raghavan, G. Manimaran, and C. S. R. Murthy, "An Integrated Scheme for Establishing Dependable Real-time Channels in Multi-hop Networks," in *Proceedings of the*

8th International Conference on Computer Communications and Networks, October 1999.

[38] K. P. Gummadi, M. J. Pradeep, and C. S. R. Murthy, "An Efficient Primary-Segmented Backup Scheme for Dependable Real-time Communication in Multihop Networks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 81–94, February 2003.

[39] A. Banerjea, D. Ferrari, B. A. Mah, M. Moran, D. C. Verma, and H. Zhang, "The Tenet Real-time Protocol Suite: Design, Implementation, and Experiences," *IEEE/ACM Transactions on Networking*, vol. 4, no. 1, pp. 1–10, February 1996.

[40] K. G. Shin, C.-C. Chou, and S.-K. Kweon, "Distributed Route Selection for Establishing Real-time Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 3, pp. 318– 335, 2000.

[41] C.-J. Hou, "Routing Virtual Circuits with Temporal QoS Requirements in Virtual Path-based ATM Networks," *IEEE Transactions on Computers*, vol. 48, no. 11, pp. 1228– 1243, 1999.

[42] Q. Zheng and K. G. Shin, "Fault-tolerant Real-time Communication in Distributed Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 5, pp. 470–480, May 1998.

Appendix A

Simulation Topologies

A.1 Simulation Topologies

In this section we provide screen shots of the topologies that are used in the simulation study. Each of these topologies were randomly generated using the BRITE and visualized using Otter (Section 4.1.1, Page 31). The legend in the figures is for the node degree for a particular node. Each node in the figure corresponds to a router in the topology that is used for simulation. The source and destination nodes are added to these routers based on the random numbers generated for source-destination pairs.

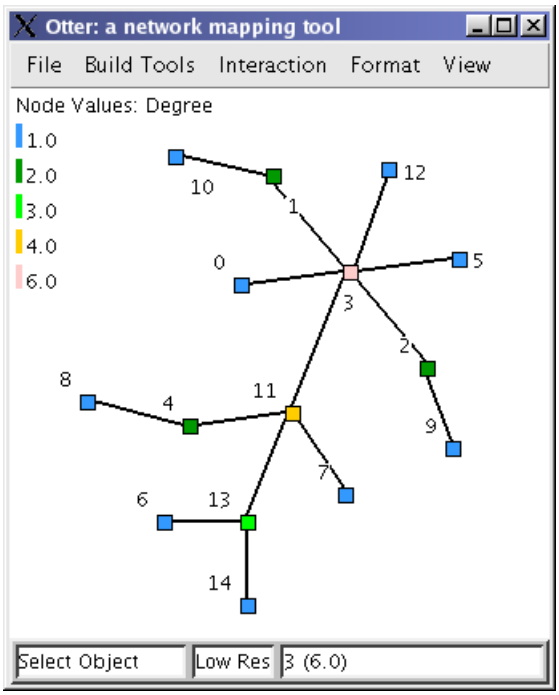


Figure A.1: Network Topology: Ratio-0

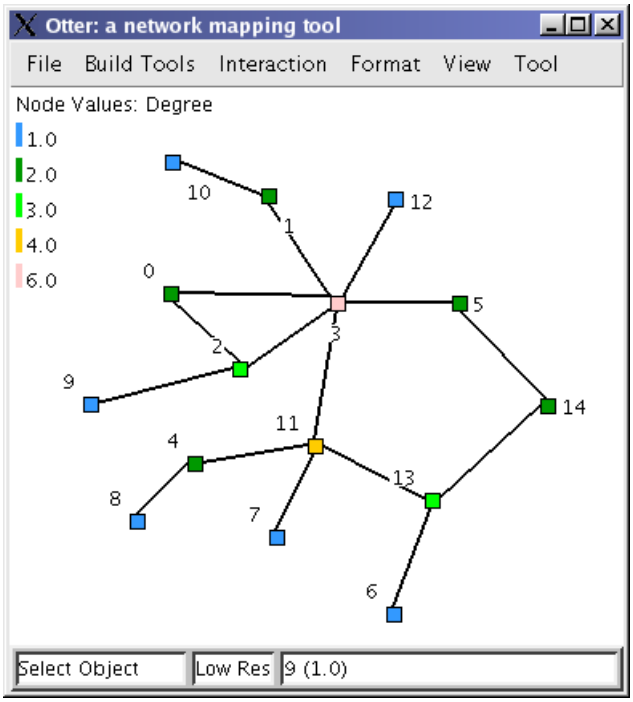


Figure A.2: Network Topology: Ratio-1

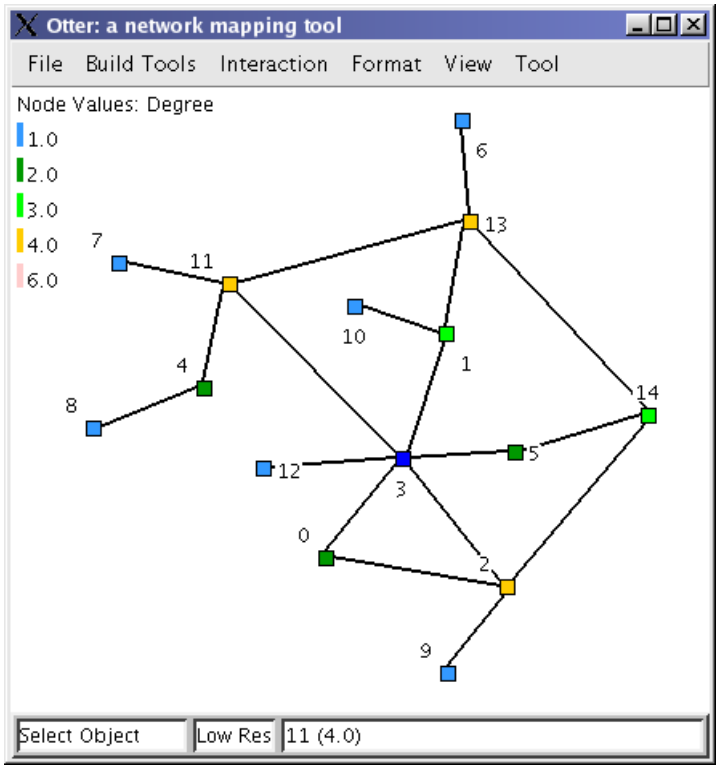


Figure A.3: Network Topology: Ratio-2

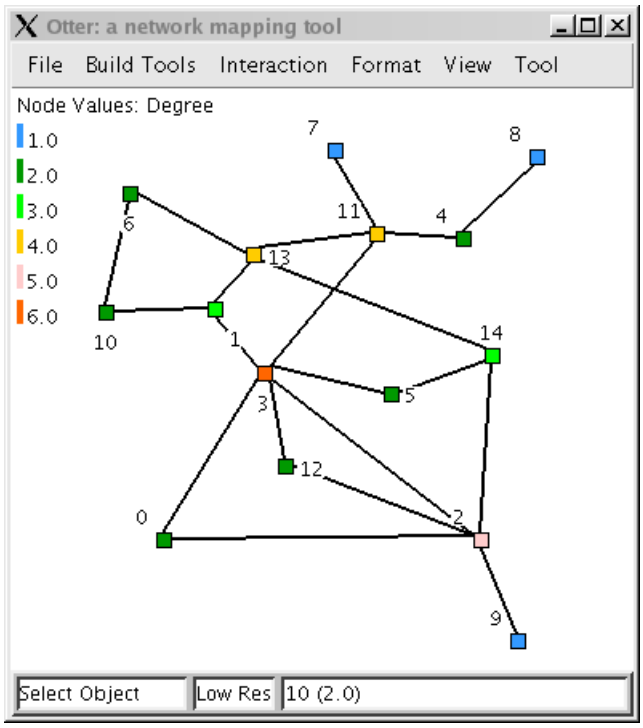


Figure A.4: Network Topology: Ratio-3

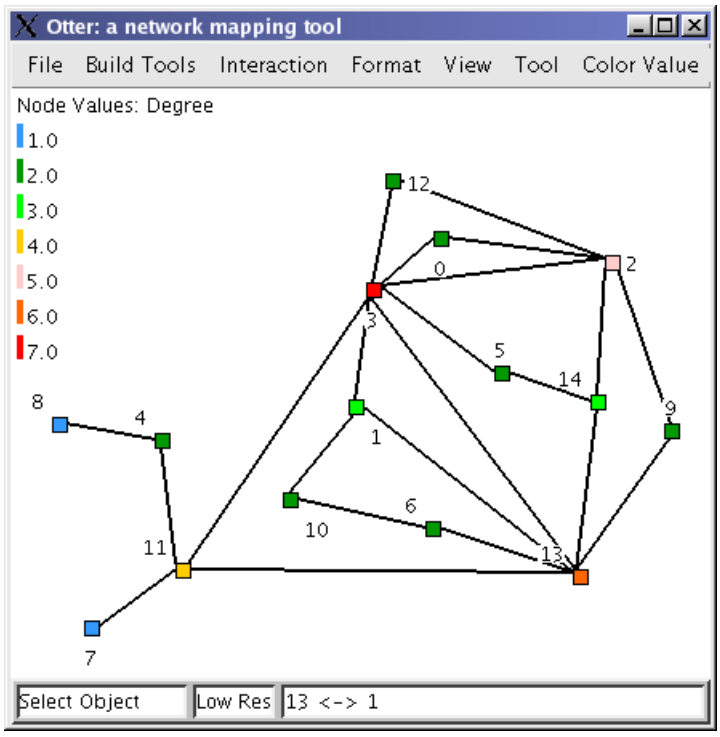


Figure A.5: Network Topology: Ratio-4

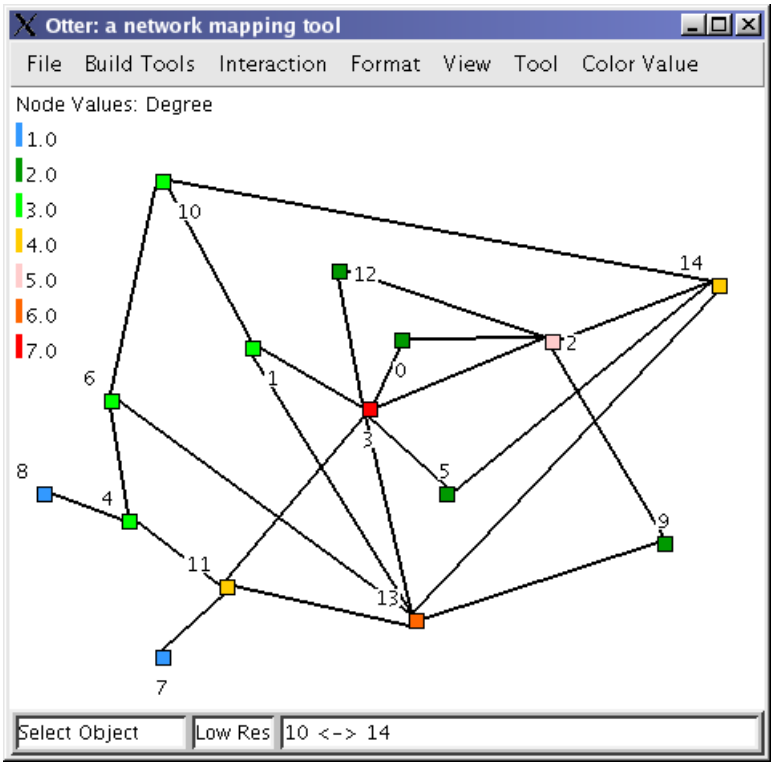


Figure A.6: Network Topology: Ratio-5

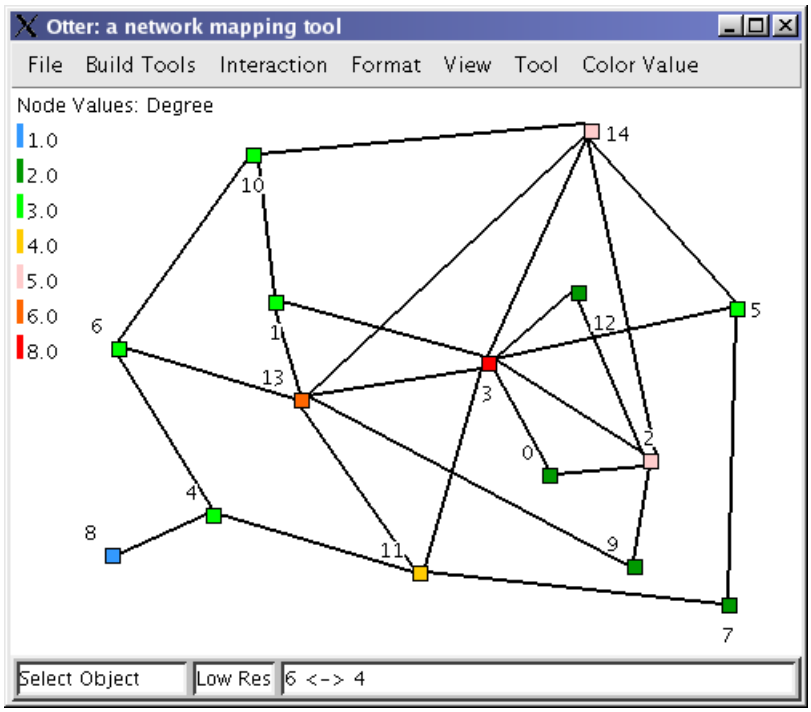


Figure A.7: Network Topology: Ratio-6

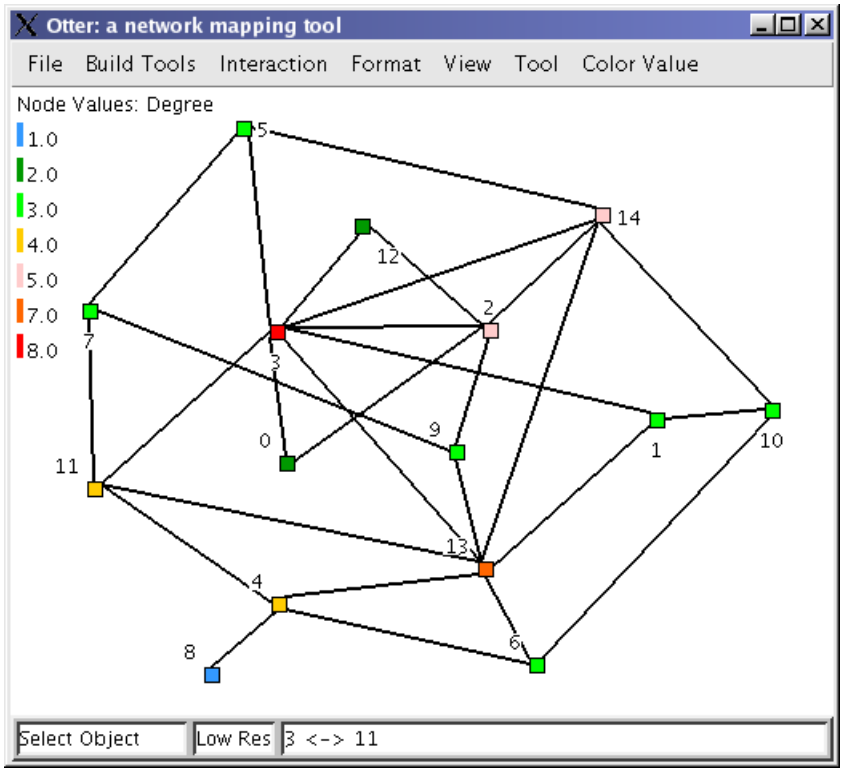
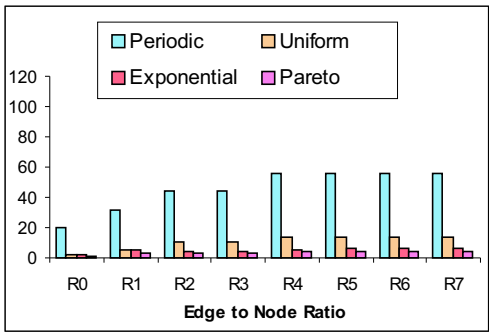


Figure A.8: Network Topology: Ratio-7

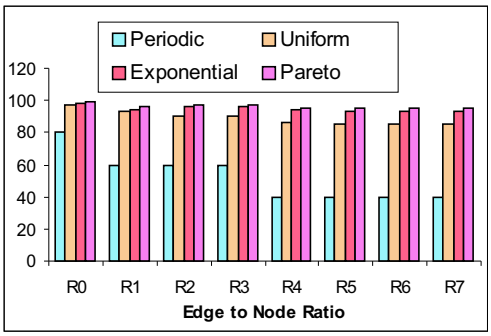
Appendix B

Simulation Results

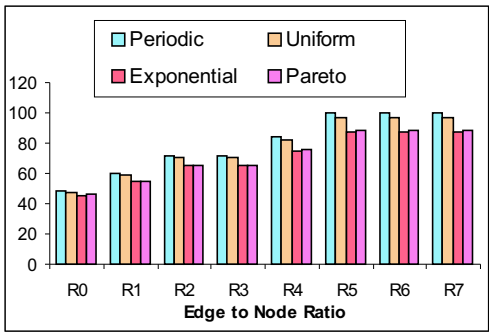
Figure 4.13 compare the performance of GloDUCE and LocDUCE in terms of utility obtained and deadline misses for the DTs used in the simulation experiments for heterogeneous TUFs.



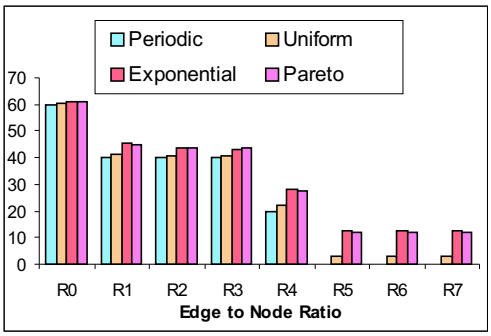
(a) OSPF: Percentage Utility



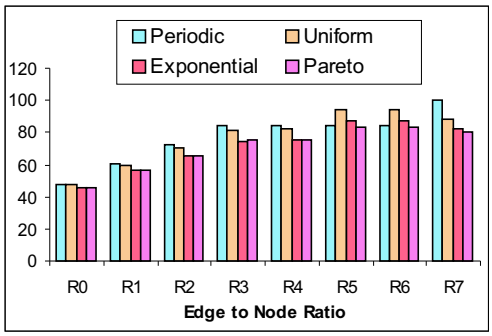
(b) OSPF: Percentage Deadline Miss



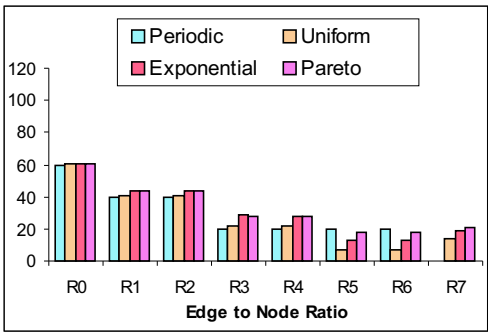
(c) LocDUCE: Percentage Utility



(d) LocDUCE: Percentage Deadline Miss

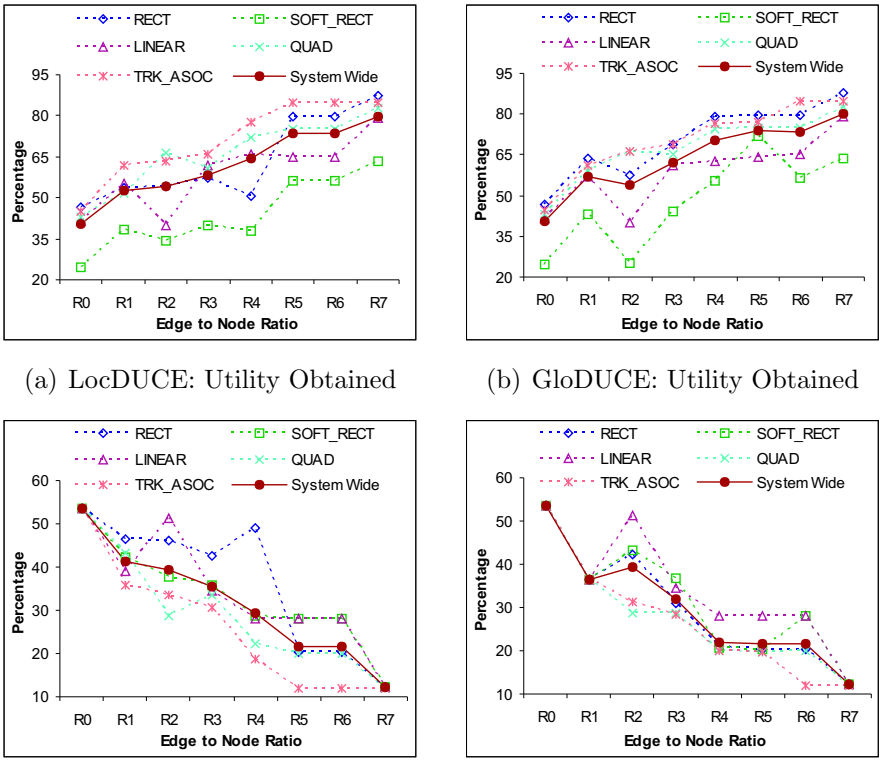


(e) GloDUCE: Percentage Utility



(f) GloDUCE: Percentage Deadline Miss

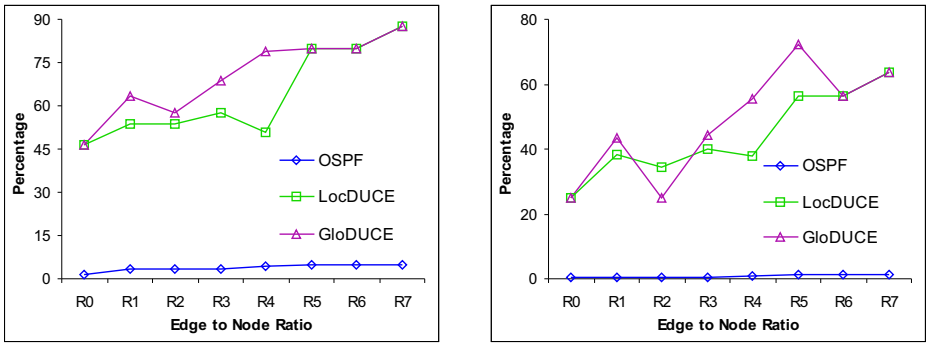
Figure B.1: Message Inter-Arrival Times: Distributions



(a) LocDUCE: Utility Obtained (b) GloDUCE: Utility Obtained

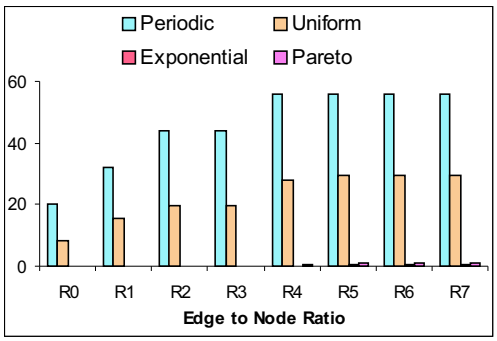
(c) LocDUCE: Deadline Misses (d) GloDUCE: Deadline Misses

Figure B.2: Heterogeneous TUFs Comparison: Message Inter-Arrival Times

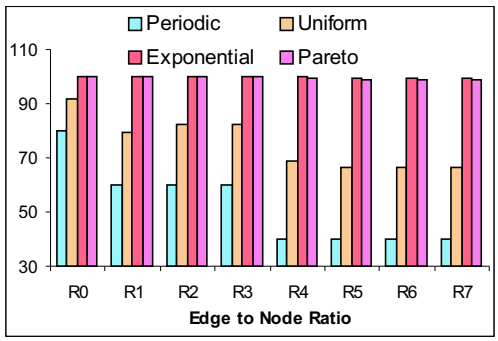


(a) `RECT` (b) `SOFT_RECT`

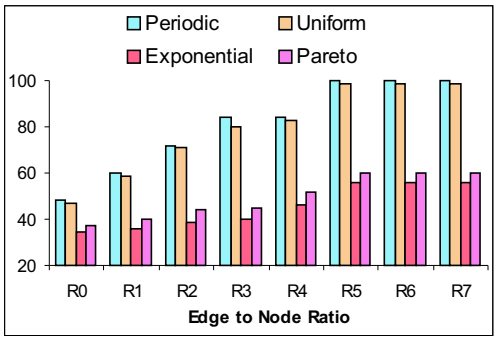
Figure B.3: Utility Obtained for Heterogeneous TUFs: Varying Message Inter-Arrival Times



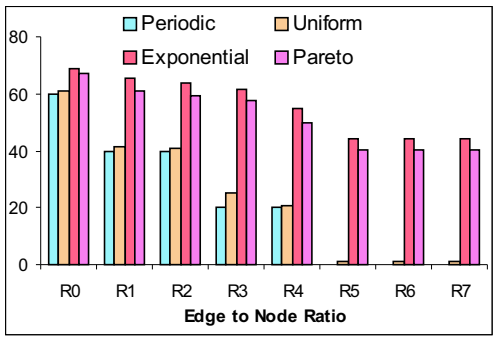
(a) OSPF: Percentage Utility



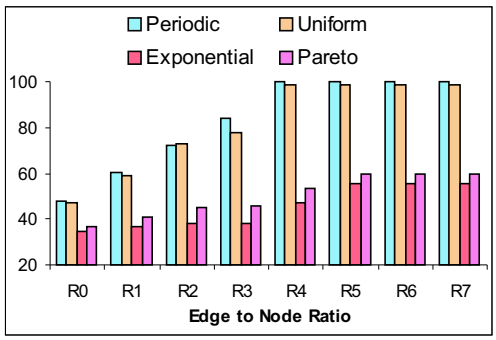
(b) OSPF: Percentage Deadline Miss



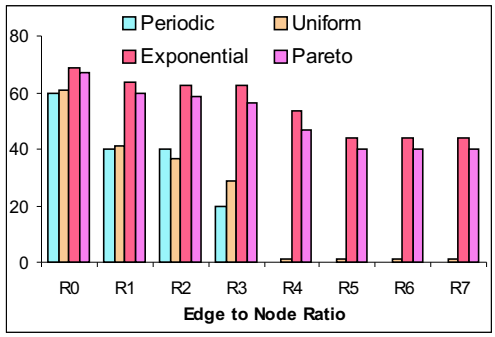
(c) LocDUCE: Percentage Utility



(d) LocDUCE: Percentage Deadline Miss

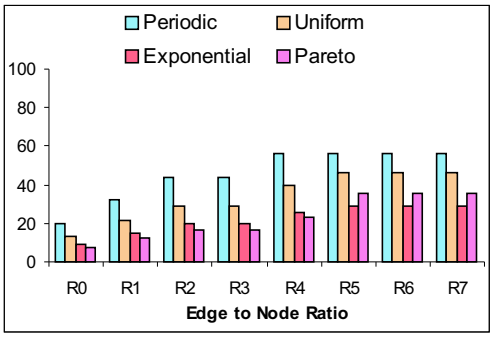


(e) GloDUCE: Percentage Utility

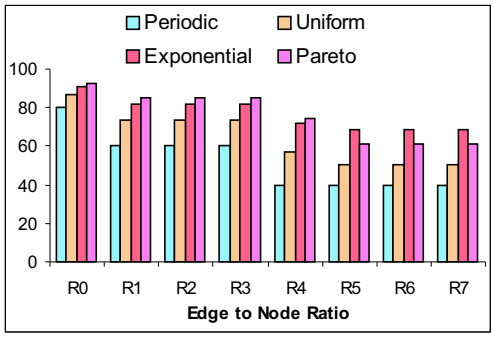


(f) GloDUCE: Percentage Deadline Miss

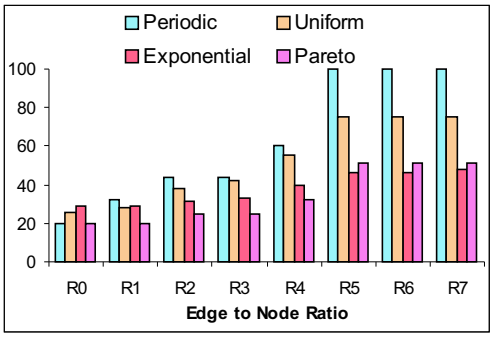
Figure B.4: Varying Message Sizes: Distributions



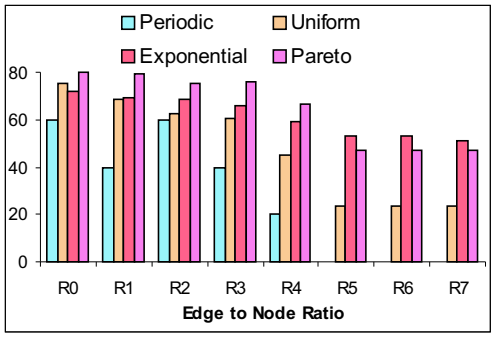
(a) OSPF: Percentage Utility



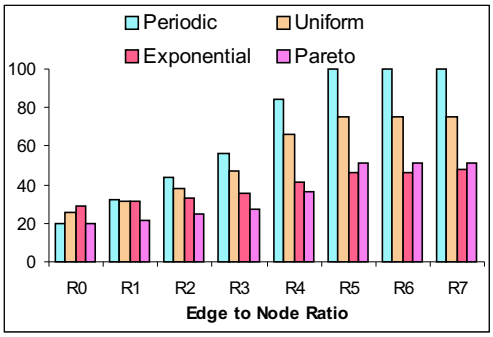
(b) OSPF: Percentage Deadline Miss



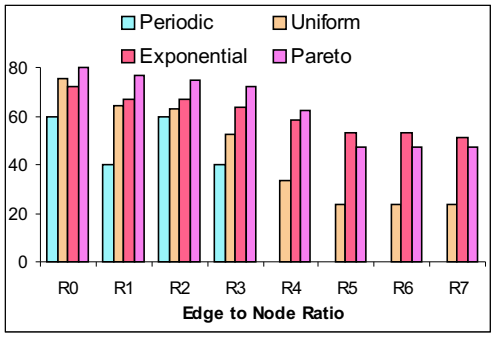
(c) LocDUCE: Percentage Utility



(d) LocDUCE: Percentage Deadline Miss

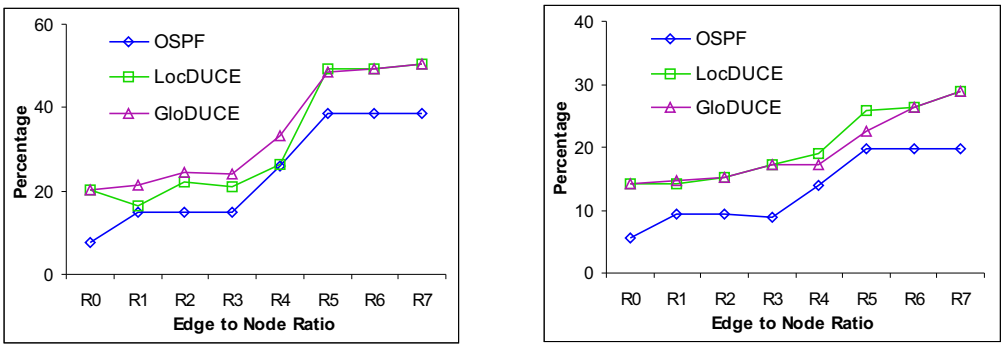


(e) GloDUCE: Percentage Utility



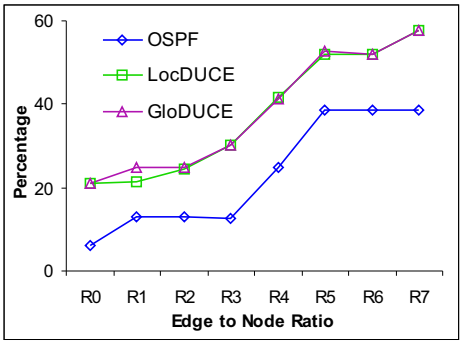
(f) GloDUCE: Percentage Deadline Miss

Figure B.5: Varying Message Deadline Values: Distributions

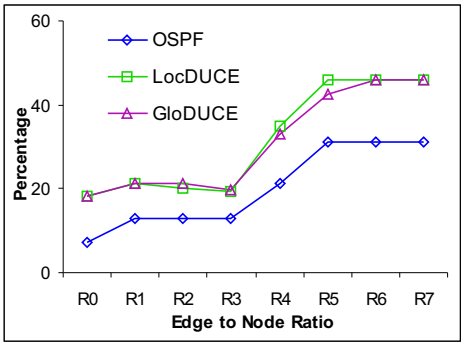


(a) `RECT`

(b) `LINEAR`



(c) `QUAD`



(d) `TRK_ASOC`

Figure B.6: Utility Comparison for Heterogeneous TUFs: Varying Message Deadlines

Appendix C

Confidence Interval Tables

The confidence intervals of the simulation and experimental evaluations are shown in tables here. This is because the values are very small to be viewed clearly in the actual plots shown in Chapter 4 (Page 30) and Chapter 6 (Page 59).

C.1 Simulation Experiments

Table C.1: Percentage Utility Comparison for Homogeneous TUFs: Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.7(a), Page 38)

Ratio	**OSPF**		**LocDUCE**		**GloDUCE**	
	Average	*Limits*	*Average*	*Limits*	*Average*	*Limits*
R0	1.238	±0.395	45.945	±0.027	45.945	±0.027
R1	2.861	±0.711	55.031	±0.065	56.336	±0.106
R2	3.137	±0.561	65.559	±0.072	65.559	±0.072
R3	3.181	±0.563	73.767	±0.058	74.877	±0.022
R4	3.997	±0.780	75.411	±0.145	87.770	±0.100
R5	4.315	±0.819	88.087	±0.110	88.087	±0.110
R6	4.315	±0.819	88.087	±0.110	88.087	±0.110
R7	4.315	±0.819	88.087	±0.110	88.087	±0.110

Table C.2: Percentage Deadline Misses Comparison for Homogeneous TUFs: Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.7(b), Page 38)

Ratio	**OSPF**		**LocDUCE**		**GloDUCE**	
	Average	*Limits*	*Average*	*Limits*	*Average*	*Limits*
R0	98.759	±0.390	60.824	±0.087	60.824	±0.087
R1	96.483	±1.117	45.088	±0.042	43.470	±0.085
R2	96.845	±0.372	43.404	±0.250	43.404	±0.250
R3	96.771	±0.410	29.939	±0.113	28.096	±0.117
R4	95.421	±1.096	27.634	±0.104	12.247	±0.098
R5	95.103	±1.166	11.903	±0.103	11.903	±0.103
R6	95.103	±1.166	11.903	±0.103	11.903	±0.103
R7	95.103	±1.166	11.903	±0.103	11.903	±0.103

Table C.3: Percentage Utility for Homogeneous TUFs (Ratio-4): Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.8(a), Page 39)

Threads	**LocDUCE**		**GloDUCE**	
	Average	*Limits*	*Average*	*Limits*
DT-1	88.332	±0.176	88.332	±0.176
DT-2	11.010	±0.795	86.982	±0.096
DT-3	87.620	±0.371	87.620	±0.371
DT-4	87.073	±0.151	87.631	±0.138
DT-5	88.209	±0.163	88.209	±0.163

Table C.4: Percentage Deadline Misses for Homogeneous TUFs (Ratio-4): Inter-Arrival Times Distributed by Pareto Distribution (Figure 4.8(b), Page 39)

Threads	**LocDUCE**		**GloDUCE**	
	Average	*Limits*	*Average*	*Limits*
DT-1	11.668	±0.176	11.668	±0.176
DT-2	88.990	±0.795	13.018	±0.096
DT-3	12.380	±0.371	12.380	±0.371
DT-4	12.927	±0.151	12.369	±0.138
DT-5	11.791	±0.163	11.791	±0.163

C.2 Implementation Results

Table C.5: Utility Obtained for Static Scenario (Heterogeneous TUFs): Confidence Intervals for Figure 6.3

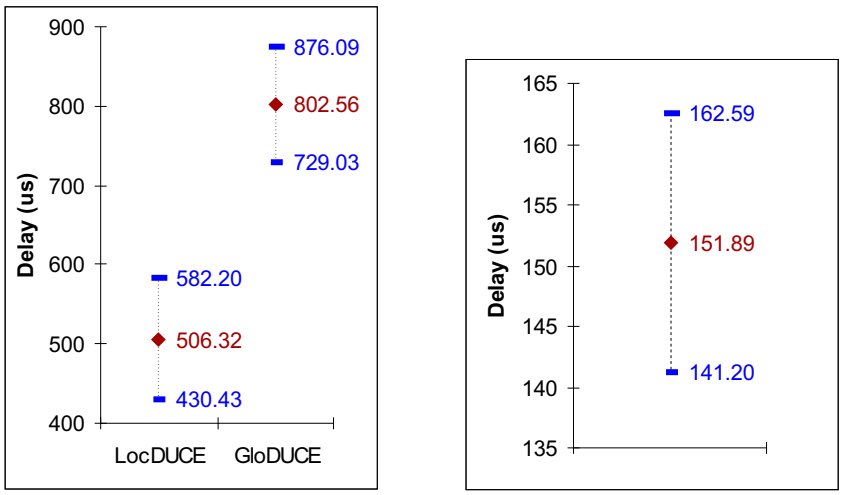
TUF Type	**OSPF**		**LocDUCE**		**GloDUCE**	
	Average	*Limits*	*Average*	*Limits*	*Average*	*Limits*
RECT	20000	±0	20000	±0	19998.837	±1.606
EXPO	3461.759	±14.749	3464.026	±13.85	19984.954	±0.585
SOFT_RECT	20000	±0	20000	±0	19998.823	±1.625
LINEAR	15101.056	±19.551	15095.992	±21.92	19997.191	±2.434
QUAD	9898.497	±15.837	9895.750	±15.04	19999.998	±0
TRK_ASOC	22218.124	±58.626	22210.492	±57.94	15000	±0

Table C.6: End-to-End Delay for Static Scenario (Heterogeneous TUFs): Confidence Intervals for Figure 6.3

TUF Type	**OSPF**		**LocDUCE**		**GloDUCE**	
	Average	*Limits ($\mu sec.$)*	*Average*	*Limits ($\mu sec.$)*	*Average*	*Limits ($\mu sec.$)*
RECT	275.730 $\mu sec.$	±11.688	286.304	±9.26	550.168μ sec	±12.03
EXPO	2.640 *sec.*	±3362.427	2.639 *sec.*	±3172.17	627.331 $\mu sec.$	±24.51
SOFT_RECT	326.568 $\mu sec.$	±11.288	323.921 $\mu sec.$	±12.35	509.770 $\mu sec.$	±11.10
LINEAR	2.627 *sec.*	±1066.969	2.627 *sec.*	±1025.84	1016.882 $\mu sec.$	±29.26
QUAD	2.705 *sec.*	±4242.159	2.706 *sec.*	±4060.14	696.706 $\mu sec.$	±27.24
TRK_ASOC	2.699 *sec.*	±6428.276	2.697 *sec.*	±5955.16	654.603 $\mu sec.$	±17.64

Table C.7: System-Wide Percentage Utility (Increasing Message Arrivals): Confidence Intervals for Section 6.2.2.1, Page 64

TUF Type	**OSPF**		**LocDUCE**		**GloDUCE**	
	Average	*Limits*	*Average*	*Limits*	*Average*	*Limits*
16	99.990	±0.001	99.966	±0.001	99.960	±0.001
32	99.959	±0.014	99.967	±0	99.966	±0.001
48	45.957	±0.019	99.960	±0.001	99.978	±0.002
64	42.203	±0.002	99.960	±0	99.969	±0.001
80	9.958	±0.556	51.273	±1.355	67.005	±1.666



(a) Channel Establishment (b) Runtime

Figure C.1: Confidence Intervals: Overhead Measurement

Vita

Mr. Channakeshava was born in Bagalkot, a small town in the Bijapur district of the state of Karnataka, India. In 1998, he got his Bachelor of Science degree in Electrical Engineering from Rashtriya Vidyalaya College of Engineering (RVCE), Bangalore, India. He then served as an engineer trainee in the Real-time Systems Group, at the Center for Development of Advanced Computing (CDAC) for a year. CDAC is a scientific society under the Ministry of Information Technology, Government of India and, is involved in design, development and deployment of advanced information technology based solutions. His responsibilities at CDAC included design, development, testing and installation of SCADA system software. This product has been installed in several regional load dispatch centers in India. His research interests are in real-time systems, distributed computing, operating systems and middleware, resource management, quality of service and network management. He loves to travel and enjoys photography in his spare time.