

# A RISK-AWARE WORKLOAD SCHEDULER TO SUPPORT EFFICIENT DATA TRANSFER FOR MOBILE COLLABORATIVE COMMUNITIES

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY  
IN THE FACULTY OF SCIENCE & ENGINEERING

2017

By  
Sadia Saleem  
School of Computer Science  
Faculty of Science & Engineering  
Supervisor: Dr. Ning Zhang  
Advisor: Dr. Kung-Kiu Lau

# Contents

<b>Abstract</b>	<b>14</b>
<b>Declaration</b>	<b>16</b>
<b>Copyright</b>	<b>17</b>
<b>Acknowledgements</b>	<b>19</b>
<b>1 Introduction</b>	<b>20</b>
1.1 Mobile Collaborative Community (MCC) . . . . .	20
1.2 Research Motivation and Challenges . . . . .	22
1.3 Research Aim and Objectives . . . . .	24
1.4 Research Hypothesis . . . . .	26
1.5 Research Method . . . . .	26
1.5.1 Literature Review . . . . .	26
1.5.2 Theoretical Work . . . . .	27
1.5.3 Evaluation . . . . .	28
1.6 Novel Contributions . . . . .	29
1.7 Publications . . . . .	29
1.8 Thesis Structure . . . . .	29
<b>2 Background and Literature Research</b>	<b>32</b>
2.1 Chapter Introduction . . . . .	32
2.2 Inverse Multiplexing . . . . .	33
2.3 Inverse Multiplexing of Wireless Channels . . . . .	34
2.3.1 Inverse Multiplexing of Multiple Channels attached to Same Device (MCSD) . . . . .	37
2.3.2 Inverse Multiplexing of Multiple Channels in MCC . . . . .	39

2.4	MCC in Detail . . . . .	39
2.5	Collaborative Data Transfer in MCC . . . . .	42
2.5.1	Potential Application Scenarios . . . . .	43
2.5.2	Common Observations from Application Scenarios . . . . .	45
2.5.3	Generic Data Transfer Models . . . . .	46
2.5.3.1	Collaborative Uploading . . . . .	46
2.5.3.2	Collaborative Downloading . . . . .	47
2.5.3.3	A Combined Model . . . . .	48
2.6	Achieving Efficient Collaborative Data Transfer in MCC . . . . .	49
2.6.1	Requirement Specifications . . . . .	49
2.6.2	Challenging Issues . . . . .	51
2.7	Existing Systems Supporting Collaborative Data Transfer in MCC . . . . .	53
2.7.1	Handheld Routers . . . . .	53
2.7.2	COMBINE . . . . .	53
2.7.3	PRISM . . . . .	54
2.7.4	MOPED . . . . .	54
2.7.5	Analysis Summary . . . . .	54
2.7.6	What is Still Missing? . . . . .	56
2.8	Our Research Goal . . . . .	58
2.9	Chapter Summary . . . . .	58
<b>3</b>	<b>Workload Scheduling Algorithms: A Literature Review</b>	<b>59</b>
3.1	Chapter Introduction . . . . .	59
3.2	Simple Scheduling Approaches . . . . .	60
3.2.1	Round Robin (RR) Scheduling . . . . .	60
3.2.2	Weighted Round Robin (WRR) Scheduling . . . . .	61
3.2.3	Randomized Round Robin (RandRR) Scheduling . . . . .	61
3.2.4	Limitations . . . . .	62
3.3	Application-Aware Scheduling Approaches . . . . .	62
3.3.1	Channel Pinned Scheduling . . . . .	62
3.3.2	Call-back Scheduling . . . . .	63
3.3.3	Limitations . . . . .	63
3.4	Channel-Conditions Adaptive Scheduling Approaches . . . . .	64
3.4.1	Link Quality Balancing (LQB) Scheduling . . . . .	64
3.4.2	Rate Based Scheduling . . . . .	64
3.4.3	ADaptive Scheduling (ADAS) . . . . .	65

3.4.4	Limitations . . . . .	65
3.5	Application-Aware and Channel-Conditions Adaptive Scheduling Approaches . . . . .	66
3.5.1	Adaptive Channel Pinned Scheduling . . . . .	66
3.5.1.1	Layer Priority Striping (LPS) . . . . .	66
3.5.1.2	Frame Priority Striping (FPS) . . . . .	67
3.5.1.3	Independent Path Striping (IPS) . . . . .	67
3.5.2	Adaptive Call-back Scheduling . . . . .	67
3.5.3	Work Queue (WQ) Scheduling . . . . .	68
3.5.4	Objective Driven Scheduling . . . . .	68
3.5.5	Limitations . . . . .	69
3.6	What is Still Missing? . . . . .	69
3.7	Best Way Forward: A Risk-Aware Workload (RAW) Scheduler . .	71
3.8	Chapter Summary . . . . .	71
<b>4</b>	<b>The RAW Scheduler: Architectural Design</b>	<b>72</b>
4.1	Chapter Introduction . . . . .	72
4.2	Design Requirements . . . . .	72
4.3	Design Assumptions . . . . .	73
4.4	Architecture Overview . . . . .	75
4.5	RAW Scheduler In Depth . . . . .	79
4.5.1	Data Splitting Module (DSM) . . . . .	79
4.5.2	Encryption and MAC Service . . . . .	82
4.5.3	Work Queues Manager (WQM) . . . . .	83
4.5.4	Trust Manager (TM) . . . . .	83
4.5.5	Access Control Module (ACM) . . . . .	87
4.5.6	Collaborator Monitoring Module (CMM) . . . . .	90
4.6	A Case Study: Scheduling MPEG Video Stream using RAW Scheduler . . . . .	90
4.6.1	Structure of MPEG Video Stream . . . . .	90
4.6.2	Adaptive Video Streaming . . . . .	92
4.6.3	Risk Aware MPEG Scheduling . . . . .	93
4.7	Evaluation Methodology . . . . .	96
4.7.1	Modelling . . . . .	97
4.7.2	Test-bed Experiments . . . . .	97
4.7.3	Emulation . . . . .	97

4.7.4	Simulation . . . . .	98
4.7.5	Our Hybrid Approach . . . . .	98
4.8	Chapter Summary . . . . .	99
<b>5</b>	<b>The RAW Scheduler: Evaluation</b>	<b>100</b>
5.1	Chapter Introduction . . . . .	100
5.2	MWQ Algorithm Design . . . . .	101
5.3	MWQ Algorithm Implementation . . . . .	102
5.3.1	Round Robin (RR) Algorithm . . . . .	102
5.3.2	Channel Pinned (CP) Algorithm . . . . .	102
5.3.3	Single Work Queue (SWQ) Algorithm . . . . .	103
5.3.4	Multi-level Work Queue (MWQ) Algorithm . . . . .	103
5.4	MWQ Algorithm Performance Evaluation using Simulation . . . .	104
5.4.1	National Chiao Tung University network simulator (NC-TUns) . . . . .	104
5.4.2	Simulator Capabilities and Features . . . . .	106
5.4.2.1	Kernel Re-entering Methodology . . . . .	106
5.4.2.2	Reusing All Real-Life Application Programs . . .	108
5.4.2.3	High Simulation Speeds and Repeatable Simulation Results . . . . .	108
5.4.2.4	Support for Various Important Networks . . . . .	108
5.4.2.5	Multi-interface Mobile Node . . . . .	108
5.4.3	Components and Architecture of NCTUns . . . . .	110
5.4.4	Simulation Environment . . . . .	112
5.4.4.1	Scenario . . . . .	113
5.4.4.2	Configurations . . . . .	114
5.4.5	Performance Metrics . . . . .	118
5.4.6	The Abbreviations Used . . . . .	119
5.4.7	QoS Evaluation of MWQ Algorithm . . . . .	120
5.4.7.1	Simulation Experiment 1 . . . . .	120
5.4.7.2	Simulation Experiment 2 . . . . .	121
5.4.7.3	Simulation Experiment 3 . . . . .	123
5.4.8	Major Findings . . . . .	124
5.5	Performance Evaluation using Test-bed Experiments . . . . .	125
5.5.1	Experimental Setup . . . . .	125
5.5.2	Toolset and Language . . . . .	126

5.5.2.1	Java . . . . .	126
5.5.2.2	Bluez . . . . .	127
5.5.2.3	HCI Tool . . . . .	127
5.5.2.4	IP Route . . . . .	127
5.5.3	MCC Network Formation . . . . .	127
5.5.3.1	Hand Shaking Process . . . . .	129
5.5.3.2	Adding Multiple PPP Connections . . . . .	129
5.5.3.3	PPP Configuration and Chat Scripts . . . . .	129
5.5.4	RAW Scheduler Implementation: An HTTP Downloader .	129
5.5.4.1	Device Scanner/Manager . . . . .	131
5.5.4.2	Partition Scheduler . . . . .	133
5.5.4.3	Partition Downloader . . . . .	139
5.5.4.4	File Merger . . . . .	140
5.5.4.5	Operating System (OS) Interactions . . . . .	140
5.5.4.6	GUI . . . . .	141
5.5.5	QoS claims to be evaluated . . . . .	141
5.5.6	Test-bed Experiments and Results . . . . .	142
5.5.6.1	Experiment 1 . . . . .	142
5.5.6.2	Experiment 2 . . . . .	144
5.5.6.3	Experiment 3 . . . . .	145
5.5.6.4	Experiment 4 . . . . .	147
5.5.6.5	Experiment 5 . . . . .	149
5.5.6.6	Experiment 6 . . . . .	150
5.5.7	Major Findings . . . . .	151
5.6	Chapter Summary . . . . .	151
<b>6</b>	<b>Conclusion and Future Work</b>	<b>153</b>
6.1	Conclusion . . . . .	153
6.2	Suggestions for Future Research . . . . .	154
	<b>Bibliography</b>	<b>156</b>
<b>A</b>	<b>WLANs Vs. WWANs: Throughput Comparison</b>	<b>164</b>
A.1	Typical Throughput Rate of WLAN Technologies . . . . .	164
A.2	Typical Throughput Rate of WWAN Technologies . . . . .	165

<b>B</b>	<b>NCTUns Simulation Details</b>	<b>166</b>
B.1	NCTUns Network Scalability Problem . . . . .	166
B.2	Simulation Code . . . . .	167
B.2.1	Round Robin Algorithm Code . . . . .	167
B.2.2	Channel Pinned Algorithm Code . . . . .	172
B.2.3	Single Work Queue Algorithm Code . . . . .	177
B.2.4	Multi-level Work Queue Algorithm Code . . . . .	183
B.2.5	Receiver Code . . . . .	190
<b>C</b>	<b>HTTP Downloader Implementation Details</b>	<b>193</b>
C.1	Bluetooth Pairing Process . . . . .	193
C.2	Chat Script Example and Explanation . . . . .	194
C.2.1	Example Configuration for a PPP Connection Authentication	194
C.2.2	Chat Script to Connect to Default Service Provider . . . .	195
C.2.3	Configuration for a PPP Connection Authenticated with Lyca . . . . .	196
C.2.4	Chat Script to Connect to Lyca . . . . .	197
C.2.5	Log File . . . . .	198
C.3	HTTP Downloader Code . . . . .	199
C.3.1	DownloadRequest.java . . . . .	199
C.3.2	DownloadRequestPartition.java . . . . .	207
C.3.3	HTTPRequestUtil.java . . . . .	210
C.3.4	PartitionScheduler.java . . . . .	217
C.3.5	SinglePartitionThread.java . . . . .	223

# List of Tables

2.1	Analysis Summary of Existing Systems Supporting Collaborative Data Transfer in MCC. . . . .	56
2.2	Comparison of Existing Systems Supporting Collaborative Data Transfer in MCC against Scenario Requirements. . . . .	57
3.1	Comparison of Existing Workload Scheduling Approaches against Data Scheduling Requirements. . . . .	69
4.1	Access Control Policy for CAPAC. . . . .	89
4.2	Access Control Policies for Risk Aware MPEG Scheduling to Maximise (i) Security and (ii) QoS. . . . .	95
5.1	Abbreviations for Simulation Results . . . . .	119
5.2	Total Transfer Time for 50 MB . . . . .	120
5.3	Improvement Percentage of MWQ for 50 MB . . . . .	120
5.4	Revised Total Transfer Time for 50 MB . . . . .	121
5.5	Revised Improvement Percentage of MWQ for 50 MB . . . . .	121
5.6	Total Transfer Time for 100 MB . . . . .	123
5.7	Improvement Percentage of MWQ for 100 MB . . . . .	124
A.1	Typical Throughput Rate of WLAN Technologies. . . . .	164
A.2	Typical Throughput Rate of WWAN Technologies. . . . .	165



# List of Figures

1.1	MCC Infrastructure. Reproduced from [1]. . . . .	21
1.2	Thesis Structure . . . . .	31
2.1	Inverse Multiplexing of Multiple Channels. . . . .	33
2.2	(MCC) Infrastructure: The environment includes various multi-homed mobile devices equipped with both WWAN and WLAN interfaces. Mobile devices form a mobile community using WLAN and collaborate to simultaneously use multiple WWANs. Reproduced from [1]. . . . .	40
2.3	Model of Collaborative Upstream Data Transfer . . . . .	47
2.4	Model of Collaborative Downstream Data Transfer . . . . .	48
2.5	A Combine Model of Collaborative Data Transfer . . . . .	49
4.1	Risk and Trust. Reproduced from [2] . . . . .	75
4.2	CAPability-aware Access Control (CAPAC) model . . . . .	77
4.3	RAW Scheduler Architecture . . . . .	78
4.4	CAPability-aware Access Control (CAPAC) Components . . . . .	79
4.5	Data Tagging and Splitting . . . . .	82
4.6	Process of Capability Level Assignment . . . . .	88
4.7	Typical Structure of a Group-of-Pictures (GoP) in a MPEG Video Stream. Reproduced from [3]. . . . .	91
4.8	MPEG Video's <i>B</i> -frames in Presence (left) and Absence (right) of Base-frames. Reproduced from [3]. . . . .	92
4.9	Flowchart of MPEG Video Splitting Algorithm . . . . .	94
5.1	The Basic MCC Simulation Topology . . . . .	105
5.2	The Kernel Re-entering Simulation Methodology. Reproduced from [4] . . . . .	107

5.3	The Relationship between the Multi-interface Node and the Device Node. Reproduced from [4] . . . . .	109
5.4	An example of a multi-interface node structure. Reproduced from [4] . . . . .	109
5.5	The architecture of NCTUns. Reproduced from [5] . . . . .	110
5.6	The Simulation Setup - Static Conditions . . . . .	113
5.7	The Topology Editor . . . . .	114
5.8	The Multi-interface Node. . . . .	115
5.9	The Node Editor . . . . .	116
5.10	The Node Editor - Protocol Module Settings . . . . .	116
5.11	The Link Editor . . . . .	117
5.12	The Log File . . . . .	118
5.13	Graph: Total Transfer Time for 50 MB . . . . .	122
5.14	Graph: Throughput (kbps) for 50MB Data Transfer . . . . .	123
5.15	Graph: Throughput (kbps) for 100MB Data Transfer . . . . .	124
5.16	The Test-Bed Setup for Experiments . . . . .	126
5.17	HCI Tool Script to Get Available Bluetooth Connections . . . . .	128
5.18	Script to Add Alternate IP Tables . . . . .	128
5.19	Configuration for a Dialup Connection Authenticated with Virgin . . . . .	130
5.20	Chat Script to Dial Out to Virgin Mobile Data Services . . . . .	130
5.21	Device Manager Flow . . . . .	132
5.22	Partition Scheduler - SWQ (a) . . . . .	134
5.23	Partition Scheduler - SWQ (b) . . . . .	135
5.24	Partition Scheduler - MWQ (a) . . . . .	137
5.25	Partition Scheduler - MWQ (b) . . . . .	138
5.26	File Merger Code . . . . .	140
5.27	HTTP Downloader GUI . . . . .	141
5.28	Ex1: 2 MB Download over a Single Virgin Channel (a) . . . . .	143
5.29	Ex1: 2 MB Download over a Single Virgin Channel (b) . . . . .	143
5.30	Ex2: 2 MB Download over Single Lyca Channel (a) . . . . .	144
5.31	Ex2: 2 MB Download over Single Lyca Channel (b) . . . . .	145
5.32	Ex3: 2 MB Download over Two Lyca Channels (a) . . . . .	146
5.33	Ex3: 2 MB Download over Two Lyca Channels (b) . . . . .	146
5.34	Ex4: 2 MB Download over one Virgin and Two Lyca Channels (a) . . . . .	148
5.35	Ex4: 2 MB Download over one Virgin and Two Lyca Channels (b) . . . . .	148

5.36	Ex5: 2 MB Download over two Virgin and Two Lyca Channels (a)	149
5.37	Ex5: 2 MB Download over two Virgin and Two Lyca Channels (b)	150
5.38	Comparison: 2 MB Download over four Channels using four Scheduling Techniques . . . . .	151
C.1	Example Configuration for a PPP Connection Authenticated with PAP or CHAP . . . . .	195
C.2	Chat Script to Connect to Default Service Provider . . . . .	196
C.3	Configuration for a PPP Connection Authentication with Lyca . . . . .	197
C.4	Chat Script to Connect to Lyca Mobile Service Provider . . . . .	198
C.5	Log File from a Test Run . . . . .	199

## Acronyms

**ACM** Access Control Module

**ADAS** ADaptive Scheduler

**BAG** Bandwidth Aggregation

**CMM** Collaborators Monitoring Module

**CP** Channel Pinned

**DSM** Data Splitting Module

**DC** Direct Contribution

**DT** Direct Trust

**EDGE** Enhanced Data rates for GSM Evolution

**FPS** Frame Priority Striping

**GB** Giga Byte

**GSM** Global System for Mobile communications

**GPRS** General Packet Radio Service

**ICT** Information and Communications Technology

**GUI** Graphical User Interface

**IMUX** Inverse Multiplexer

**IPS** Independent Path Striping

**IC** Indirect Contribution

**IT** Indirect Trust

**LPS** Layer Priority Striping

**LQB** Link Quality Balancing

**LTE** Long Term Evolution

**MAC** Message Authentication Code

**MB** Mega Byte

**MCC** Mobile Collaborative Community

**MCSD** Multiple Channels attached to Same Device

**MOPED** MObile grouPEd Device

**MT** Mobile Terminal

**MTU** Maximum Transmission Unit

**NCTUns** National Chiao Tung University network simulator

**NMT** Nordic Mobile Telephone

**QoS** Quality of Service

**RAW** Risk-Aware Workload

**RPC** Reverse Path Controller

**TM** Trust Manager

**UMTS** Universal Mobile Telephony System

**WLAN** Wireless Local Area Network

**WCDMA** Wideband Code Division Multiple Access

**WQ** Work Queue

**WQM** Work Queue Manager

**WWAN** Wireless Wide Area Network

# Abstract

Ad-hoc Mobile Collaborative Community (MCC) enables two or more low bandwidth mobile phone/PDA network channels to achieve a virtual high-bandwidth channel for collaborative data transfer using inverse multiplexing techniques. This research looks at how to achieve best effort QoS for MCC communities and proposes a novel Risk-Aware Workload (RAW) scheduler, to support efficient collaborative data transfer in MCC. To this end, the thesis has made the following four novel contributions. Firstly, it presents a comprehensive requirements analysis of the MCC scenarios and literature review of the existing solutions proposed to address these requirements. The analysis and literature review have led to the identification of our research aim, i.e. to design the RAW scheduler, an energy and QoS performance efficient workload scheduler for MCC. Secondly, it presents a novel architecture for the RAW scheduler. The architecture is designed for battery and memory constrained mobile devices and supports best effort QoS for multi-stream applications. The design has made use of a modular approach so that any of the architectural modules can be replaced as technologies advance without affecting other modules. Thirdly, it presents the design and evaluation of an energy-efficient Multi-level Work Queue (MWQ) scheduling algorithm, a core component of the RAW scheduler. The algorithm is application requirement aware and is adaptive to the changes in the underlying channel conditions. We have evaluated the QoS performance of the MWQ scheduling algorithm by comparing it against the performances of three other state-of-the-art scheduling algorithms using the NCTUNs simulator. The simulation setup aggregates the bandwidth of two independent channels and results show an improvement of 45% in total data transfer time for MWQ as compared to the Single Work Queue (SWQ) algorithm and up to 62% as compared to the Round Robin (RR) algorithm. In comparison with the Channel Pinned (CP) algorithm, the improvement ranges between 28% to 95%, depending on the distribution of data to the two

channels. Fourthly, we have implemented an "HTTP Downloader" application with the novel RAW scheduling design built-in. This test-bed implementation is done using real world mobile devices and network communication technologies. We performed a detailed evaluation of the QoS performance of our "HTTP Downloader". The evaluation results show that the RAW scheduler delivers better QoS results in terms of average throughput and total data transfer times by adapting to the changes in the channel conditions. It has been found that, among the four alternatives, the MWQ algorithm delivers the best results when we plugged it into our test-bed application. To download a 2MB file from the Internet using four collaborators, MWQ showed an improvement of 46% in total download time as compared to the RR algorithm. The percentage improvement values in the similar scenario for the CP algorithm is 35% and the SWQ algorithm is 17%.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.



# Copyright

Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the head of School of Computer Science.

*Dedicated to My Beloved Grand Parents*

# Acknowledgements

I thank God Almighty for giving me enough powers, physical and mental, to pursue this first research endeavour. I acknowledge the contributions of my supervisor Dr. Ning Zhang as marvelous. She has helped me taking the first step towards self-actualization and has always guided me in my research. I thank my family, all the friends, university staff and other people who helped me towards getting my research done. Last but not least, I want to thank Islamic International University, Pakistan for financially funding this study.

# Chapter 1

## Introduction

### 1.1 Mobile Collaborative Community (MCC)

Wireless mobile networks and devices have become extremely popular as they provide access to information and communication anytime and anywhere. Mobile devices such as laptops, mobile phones, smart phones and PDAs that come equipped with multiple wireless network channels are called multi-homed devices. These channels include at-least one Wireless Wide Area Network (WWAN) channel to connect to the Internet and at-least one Wireless Local Area Network (WLAN) channel to connect the device to the neighbouring wireless network devices. WWAN (e.g. GPRS [6], EGDE [7]) channels are comparatively limited in bandwidth and loss-prone in packet delivery. WLAN (e.g. Wi-fi [8], Bluetooth [9]) channels in contrast provide higher bandwidth. Though WLANs can provide high-speed Internet connectivity to mobile users, their coverage is limited to a relatively small geographical area. The users have to rely on WWAN channels with limited bandwidth to connect to the Internet when wired Internet access point is not available. The bandwidth limit of many WWANs is often insufficient for data demanding multi-stream applications, such as telemedicine and disaster relief management systems requiring simultaneous transfer of the audio, video and other data streams.

High speed Internet access is one of the major hurdles in implementing applications discussed above in emergency scenarios. Despite 3G/4G hype, it is still difficult for an individual WWAN channel to effectively support high bandwidth applications. One possible solution to this problem is to use inverse multiplexing

[10] [11]. Inverse multiplexing is a process by which multiple relatively low bandwidth channels are aggregated to form one high bandwidth virtual channel (See section 2.2 for details). A device with multiple WWAN connections can inverse multiplex these low bandwidth channels together. Existing systems, e.g. Mobile Access Router (MAR) [12], Pluribus [13], MobiStream [14], NATALIE [15] and Horde [16] support such bandwidth aggregation.

Moreover, multiple multi-homed devices in close vicinity can discover each other through a high speed WLAN to form an ad-hoc community to logically combine and share the bandwidth of each other's WWAN channels using inverse multiplexing. This ad-hoc community of multi-homed devices is called Mobile Collaborative Community (MCC).

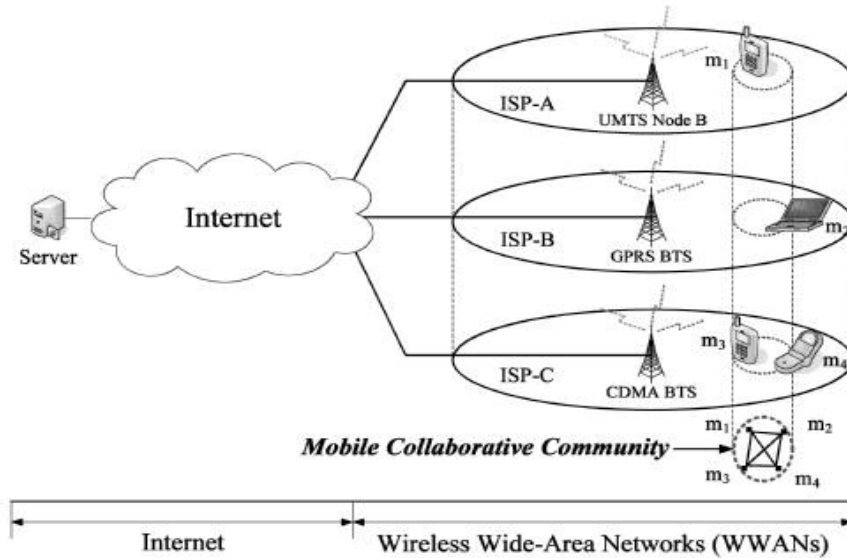


Figure 1.1: MCC Infrastructure. Reproduced from [1].

Figure 1.1 illustrates the infrastructure of MCC, where a few multi-homed devices in WLAN range form a mobile community and collaborate to simultaneously use multiple WWANs. PRISM [1], COMBINE [17], Handheld Routers [18] and MOPED [19] have proposed collaborative data transfer mechanisms for MCC. The Inverse Multiplexer (IMUX) running on the source node schedules data across the multiple collaborators in the MCC. A source node uses high speed WLAN channels to transfer data to its collaborators. Every collaborator then transfers allocated data using individual WWAN channel to the destination. Hence, the data reaches the destination at an aggregated rate. This collaboration

can increase the effective WWAN bandwidth available to every node of the MCC (See section 2.4 for details).

## 1.2 Research Motivation and Challenges

Achieving a high speed virtual broadband channel for the members is the main objective of MCC. A number of solutions have been proposed to support the efficient data transfer between the MCC nodes and a remote host across the Internet. However, existing MCC solutions have concentrated on collaborative data downloading, specially multimedia streaming [1] [17] [18], but have ignored the data uploading. Some assumed that the same solution can work for the upstream data transfer as well [1]. This assumption is not valid, as in the case of collaborative downloading, an IMUX/ workload scheduler runs on a sever or a proxy in the Internet, but in the case of uploading, it resides on a resource (i.e. battery power, computation power and memory) constrained mobile node. Hence, the solutions designed for the collaborative data downloading may not be most efficient when being applied to the collaborative data uploading

Consider the scenario of battle field reporting, where soldiers are equipped with multi-homed mobile devices and they may want to use these devices to communicate their location, physical geography, video of battle field and other important observations, etc. to their commanders located at a distant head quarter. Same is the case for the emergency responders in a disaster relief scenario or reporters covering the wars, they may require a high-speed Internet connection to effectively communicate with their headquarters, or to efficiently access resources available on the Internet. Similarly, deploying a telemedicine or an emergency service in rural/remote areas are other potential applications of MCC. All these applications clearly require the support for upstream data transfer. Some applications may require transfer of he sensitive data, e.g. military reporting, telemedicine. Furthermore, applications may require to support multiple streams of data with distinct Quality of Service (QoS) requirements.

Existing systems have largely focused on the community formation protocols, efficient data scheduling algorithms, and congestion control mechanisms. Providing security for the sensitive data has been side-stepped in almost all of the existing systems. They either assumed that the same user owns all mobile devices [19], or that the improved throughput alone is the main requirement [18].

UCAN [20] have considered the secure crediting in mobile ad-hoc collaborative environment, but it is not a MCC solution, and more importantly, it ignores a key resource, viz. energy. COMBINE [17] discusses an energy efficient and secure accounting scheme to keep track of the credits earned or debits incurred as nodes buy and sell bandwidth. The systems in [17] [1] have briefly discussed the data security issues and suggested to use the encryption techniques or reputation system to protect against the malicious behaviour of collaborating nodes. However, they haven't designed or implemented any solution to cater data security.

As mentioned earlier, existing systems are largely motivated by the multimedia streaming, hence, they do not support the simultaneous transfer of multiple data streams. Only, Horde [16] supports the multiple data streams and allows an application to specify its QoS goals for each data stream. Supporting multiple data streams each with distinct and possibly conflicting QoS acquirements is a major issue that this research is set to address and it is addressed in a way that is energy efficient.

Collaborative data transfer in the MCC offers a significant performance improvement, but it also raises a number of challenging issues:

1. How to adapt to the dynamically changing set of available WWAN channels and their QoS conditions, for example varying bandwidth, loss rate and latency of underlying WWAN channels.
2. How to support the multiple applications/data streams, which might have diverse QoS and security requirements.
3. In the case of collaborative data uploading, an IMUX runs on a resource (i.e. battery power, computation power and memory) constrained mobile node, which raises a significant challenge of finding the solution with minimal resource consumption.
4. How to address data security issues, as we can not assume that all the collaborators are trustworthy. Particularly, in the ad-hoc collaborative environment of the MCC, members may not have previous interactions, so may be totally strangers to each other. Moreover, the data that needs to be transferred can have multiple levels of sensitivity.

Previously, a majority of the related work done in workload scheduling is under the assumption that the underlying channels are homogeneous and have

stable QoS conditions. Hence, these static scheduling algorithms assign a fixed amount of data to each channel in a given time period. For example, the most commonly used scheduling algorithm is Round Robin (RR) (or its variants) that allocates a fixed percentage of data on each channel. Also if this assumption is true then the applications need not to know any information related to the underlying network conditions. In our MCC scenario, however, the biggest challenge is that the underlying channels are using WWAN technologies which are prone to disconnections and QoS fluctuations resulting in varying levels of bandwidth, packet latencies and loss rates. Furthermore, the applications we are considering are heterogeneous in terms of the network QoS aspects they are sensitive to. For example, some applications are sensitive to latency, some are not. Hence, a scheduling algorithm should be able to schedule the transmission of application packets with heterogeneous QoS requirements on the channels with varying channel conditions. This leads us to employ an intelligent scheduling algorithms that can continuously change the data allocation to different channels according to the application requirements and updated channel conditions. Such scheduling algorithms are called the dynamic scheduling algorithms and some state-of-the-art inverse multiplexing solutions [16] [17] [1] [18] have used these algorithms.

### 1.3 Research Aim and Objectives

The aim of this research is to develop a novel solution to support efficient collaborative data transfer in MCC. The core component of this solution is a novel scheduler, a Risk-Aware Workload (RAW) scheduler. Our scheduler minimizes the risk of selecting malicious collaborating nodes by ranking them according to their QoS trust levels and ensuring that the high priority data are scheduled over the most trusted nodes. Our proposed scheduler exhibits the following features:

- Awareness of the application specific QoS requirements.
- Adaptive to the changes in underlying WWAN channel conditions, e.g. available bandwidth, loss rate, and latency.
- Adaptive to the QoS trust levels of the collaborators.
- Energy efficient in terms of communication and processing cost.

The above aim is supported by the following six objectives.



1. To investigate and identify the scenarios of collaborative data uploading in MCC.
2. To investigate and specify the functional requirements for supporting efficient collaborative data transfer based upon the scenarios identified in 1.
3. To investigate and critically analyse the related work against the requirements specified in 2, so as to identify research gaps and challenging issues. In analysing the related works, we focus on:
  - their assumptions and implementation methods,
  - the strengths and limitations in their methods,
  - the areas for improvement and technology advance.

Our initial research led us to the identification of a research gap, i.e. the need for a novel workload scheduler to support efficient collaborative data uploading in MCC.

4. To design a novel workload scheduler to address this identified gap. To achieve this design, the most challenging tasks were :
  - (a) to design an algorithm to rank the potential collaborators according to their QoS trust levels;
  - (b) to design a multi-criteria scheduling algorithm. This algorithm should take into account the following factors:
    - i. application specific QoS requirements,
    - ii. channel conditions,
    - iii. QoS sensitivity levels of data, and
    - iv. collaborator's trust levels.
  - (c) It is worth emphasising that while achieving (a) and (b), the scheduler should also be made as energy efficient as possible, and we achieve this by using computationally efficient operations in the design of the associated algorithms and by reducing the number of interactions and inter-operations among the architectural building blocks as much as possible, such that the scheduler will consume as less battery power as possible on the hosting mobile node.

5. To implement and evaluate the designed solution and compare the results with those from the related work.
6. Publish papers to benchmark our efforts and get reviews.

## **1.4 Research Hypothesis**

The hypothesis of this research is that dynamic scheduling of priority data packets over best-effort QoS channels may better support the end-to-end QoS than the static scheduling.

## **1.5 Research Method**

The research methodology used for this project consists of three key components: a literature survey and a critical analysis of the related work, theoretical work, and an evaluation of the proposed solutions using simulation modeling and test-bed experiments.

The research for this project is conducted in a three phase process. First of all, a literature review is conducted. All the literature regarding the related work done in the past is thoroughly surveyed with an eye on any current development. During the literature review, the areas that have potential for enhancement and have gaps for improvement are identified which leads to the second phase of the research. In the second phase, theoretical work on the enhancement areas identified is conducted. The theoretical work is guided by the findings of the literature survey to develop a novel solution. Finally, in the third phase an evaluation of the solution is performed. We performed simulation as well as test-bed experiments.

### **1.5.1 Literature Review**

The project is started with a detailed study of the literature on Inverse Multiplexing, MCCs and QoS in mobile ad-hoc environment. A full understanding was developed about the MCC application scenarios, characteristics of an MCC set-up and its QoS requirements. It was evident from the beginning of the research that all the existing works mainly focused on improving the QoS for the downloading of homogeneous data streams, especially for video streaming. On further

thorough investigation and analysis it was established that the data uploading and multi-stream data transmission was not worked on as there was very little existing work in the area. Performing critical analysis and review of the previous work satisfies objectives (1) and (3). After a careful and detailed analysis of the most advanced and related solutions of scheduling, access control and QoS for MCC, an enhancement hypothesis was developed. The useful data and information were extracted and compiled for the reference during the design of solution. The guide lines for the solution were drawn by establishing the requirements for the solution to support QoS in the presence of multiple streams of data upload. This satisfies the objective (2) of our research. The formulation of an enhancement hypothesis and a design requirements specification leads to the next step of our work, i.e. the theoretical work.

### **1.5.2 Theoretical Work**

During the initial literature review, a potential enhancement area was identified, i.e. to support secure and efficient multi-stream data transfer (uploading as well as downloading) for the MCC. The survey, analysis and review of the existing relevant work continued in this phase as well. In the literature review phase, the focus of research was more general and wider. However, the research focus in this phase is dictated by the enhancement hypothesis and requirements specifications drawn in phase one.

In this phase, an extensive research and analysis of the current work continued. However, the scope and direction of the research was determined by our enhancement hypothesis and the requirements specifications goals. As a result of this work, we concluded three novel solutions. First, the design of a novel RAW scheduler architecture which supports the dynamic adjustments of scheduling strategies in response to the dynamic MCC channel conditions and QoS requirements provided by the multi-stream application. Second is the design and evaluation of a Multi-level Work Queue (MWQ) scheduling algorithm. Third novel solution is the design of a CAPability-aware Access Control (CAPAC) model. The MWQ algorithm is one of the major components of the CAPAC. The detailed design of these three solutions satisfies the objective (4).

### 1.5.3 Evaluation

In the third and final phase of the project the proposed solutions are evaluated and backed by the evaluation result statistics. Evaluation could be performed by any of the following four methodologies i.e. simulation, emulation, experimental or mathematical. Out of these four, we choose simulation and experimental methodology as they suited our evaluation goals the most. The simulation methodology was chosen to evaluate the suitability and performance of our proposed MWQ algorithm in a controlled environment. Later, a detailed evaluation was conducted by doing the test-bed experiments, to proof the efficiency of our proposed RAW scheduler by implementing an example application and testing it in a real world scenario. On the other hand, mathematical methodology is not feasible and applicable in our scenario due to the complex nature of network. The emulation require specialised hardware devices to connect with the emulation software.

A hybrid of simulation and experimental methodologies is used for the RAW scheduler performance evaluation. NCTUns [21] simulation tool is used for the initial evaluation of the work. Before conducting the evaluation, simulation model was designed carefully. To validate the simulation model we used validation scripts that come with the simulation tool i.e. NCTUns. The simulator output was then compared with the reference output to validate if the simulator has been correctly configured. After being certain that the simulation model represents our solution specification exactly and accurately, the simulations were run and results were obtained. Later, we implemented a test-bed application. We implemented an "HTTP Downloader" application that implements the RAW scheduler design to download data from the Internet using multiple mobile phones. We performed a detailed experimental evaluation of the QoS performance of our proposed scheduling solution as compared to other state-of-the-art scheduling algorithms. A detailed analysis of the proposed solutions was performed on the basis of the simulation as well as experimental results which satisfied our research objective (5).

Finally the simulation as well as experimental results were analysed and a comparison was drawn between the existing state-of-the-art scheduling solutions and the efficiency of our proposed solution was established. After final evaluation and analysis of the results the thesis culminates with the conclusions and identification of the future research directions.

## 1.6 Novel Contributions

The work presented in this thesis has led to four novel contributions:

1. An architecture for a novel RAW scheduler that can run on a resource (memory, processing and battery power) constrained mobile device and supports best effort QoS for multi-stream applications.
2. The implementation, simulation and performance evaluation of a novel MWQ scheduling algorithm that can schedule multiple data streams according to the application level QoS requirements.
3. The test-bed implementation of a RAW scheduler application, in MCC using real world mobile devices and network channels. All implementation was done at the application layer using suitable tools and technologies.
4. The design of a novel CAPAC model along with its components to support the novel RAW scheduler.

## 1.7 Publications

Parts of the research contained in this thesis have led to the following conference publication, which has been peer-reviewed.

- Sadia Saleem & Ning Zhang, "A Risk-Aware Workload scheduler to support secure and efficient collaborative data transfer in mobile communities", 9th Annual Conference on Wireless On-demand Network Systems and Services (WONS), Courmayeur, 9-11 January 2012.

## 1.8 Thesis Structure

The remainder of this thesis is organised as follows:

- Chapter 2 introduces the background concepts and presents the motivating application scenarios and a survey of the related literature. A number of observations are made on the applications scenarios. The challenges posed by the MCC environment are discussed. This chapter also reviews state-of-the art solutions to collaborative data transfer in MCC and outlines the missing bits.

- Chapter 3 focuses on the literature review of workload scheduling algorithms, it classifies and compares the existing scheduling approaches in MCC and proposes a RAW scheduler for efficient and secure collaborative data transfer.
- Chapter 4 specifies the design requirements, assumptions and architecture overview of the proposed RAW scheduler. Architectural components are discussed at the block level. The main components include the MWQ algorithm, the QoS tagging framework, the data encryption module, and a CAPAC model. This chapter also outlines the evaluation methodology used in this research. Parts of this chapter have been published in the following conference paper: 'A Risk-Aware Workload scheduler to support secure and efficient collaborative data transfer in mobile communities' [22].
- Chapter 5 presents the detailed evaluation of our proposed scheduling solutions. It presents a simulation study which compares the QoS performance of our novel MWQ algorithm with three state-of-the-art scheduling algorithms. The results of this simulation study is discussed to identify the best way to achieve better QoS in a MCC environment. Later, the test-bed implementation of a "HTTP Downloader" application is built using the RAW scheduler design is discussed in detail. Justification and use of all the tools and technologies is given. The lengthy details of implementation are moved to the appendices for reference. The QoS performance claims are listed and experimental scenarios are discussed. Finally, the results obtained to compare QoS performance and they are analysed against our research claims and conclusions are made.
- Chapter 6 concludes this thesis and suggests directions for the future research.

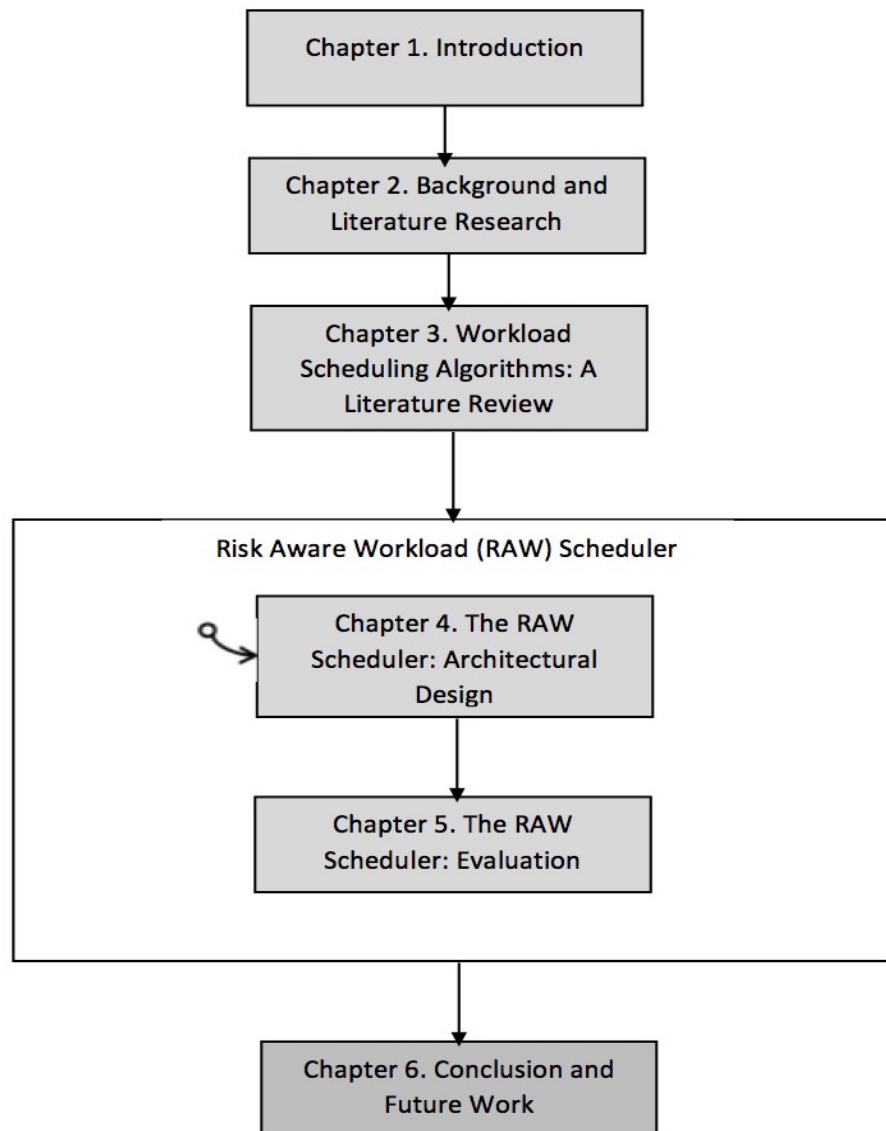


Figure 1.2: Thesis Structure

# Chapter 2

## Background and Literature Research

### 2.1 Chapter Introduction

This chapter presents an overview of the basic concepts that will be subsequently used in the design of our solution for efficient data transfer in a MCC. It also presents the motivation for this research, a survey of the related literature in the area of MCC, and identifies a way forward to address the issue of QoS provisioning for multi-stream collaborative data transfer in MCC. The building blocks of this scheduler will be covered in Chapter 4.

The organisation of the chapter is as follows: Section 2.2 introduces the concept of inverse multiplexing. Section 2.3 presents the two classes of wireless networks and compare them against their throughput and coverage capabilities. Later, it discusses how the concepts of inverse multiplexing and MCC can be used to overcome the throughput and coverage limitations of these networks. Section 2.4 describes the MCC architecture in detail. Section 2.5 analyse potential application scenarios of collaborative data transfer in MCC. This section also identifies three generic models of collaborative data transfer in MCC. Section 2.6 identifies the functional requirements and challenges associated with data transfer in MCC. Section 2.7 discusses some related efforts to collaborative data transfer in MCC and identifies what is missing in the related work. Section 2.8 outlines the way forward to achieve best effort QoS provisioning in MCC. Finally, Section 2.9 summarises the chapter.



## 2.2 Inverse Multiplexing

Inverse multiplexing traditionally used in analogue dial-up installations, is a process of aggregating several low-bandwidth information channels in a network to form an effective high-bandwidth virtual channel. Inverse Multiplexer (IMUX) stripes the data from a sender over multiple channels, each of these channels then forwards the data to the receiver. At the final destination, the forwarded packets are reassembled at an aggregated rate. For example, if six different independent 56-kb/s data channels are established between points A and B, inverse multiplexing can be used to combine these channels to create a single 336-kb/s ( $6 \times 56 = 336$ ) virtual channel. Figure 2.1 illustrates the scenario of inverse multiplexing of multiple low bandwidth channels to get a high bandwidth virtual channel.

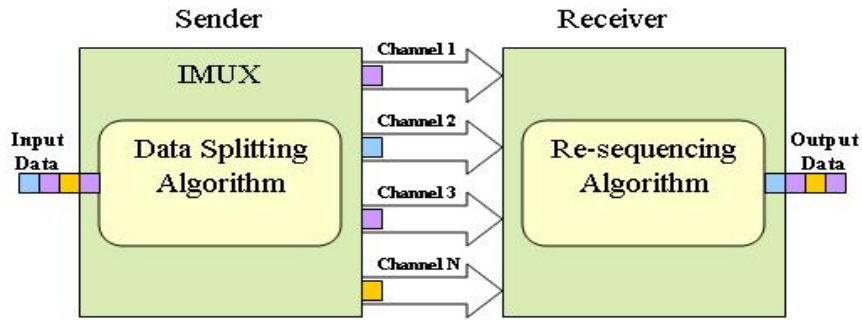


Figure 2.1: Inverse Multiplexing of Multiple Channels.

The task of the IMUX is to perform this channel aggregation (Figure 2.1). Specifically, the IMUX assures that all the channels are present. It establishes dialled channels as required and verifies the integrity of any existing leased channels. The IMUX then segments the transmission data stream and sends it out over the individual channels. At the receiving end, the IMUX accepts the data from these channels, reorders the segments, and compensates for variances in channel's transit times. Depending upon the inverse-multiplexing protocol in use, the IMUX may monitor the integrity of the aggregated connection. If transmission problems occur, the IMUX can take diagnostic action by replacing failed or failing channels with the functional channels to maintain the integrity of the connection. The IMUX can also add or remove channels from the aggregated connection without terminating the connection. This allows the total amount of bandwidth between the two sites to vary according to the real-time bandwidth requirements or for

the economies of operation. This feature is sometimes referred to as dynamic bandwidth allocation, or rubber bandwidth.

How is dynamic bandwidth allocation helpful? In video conferencing, the quality of video increases as the bandwidth between sites increases. A video conferencing session might use a 384 kb/s inverse-multiplexed connection at the beginning of a course lecture, when the lecturer is on camera and moving on the stage. The bandwidth can be reduced when the camera focuses for an extended period on a static screen or board, when the relative lack of motion requires less bandwidth and therefore less expense. In LAN inter networking, an inverse-multiplexed connection between two remote LANs might be established at 64 or 128 kb/s, which would be adequate for most simple interactive activities. If that connection becomes saturated for a predetermined period of time, for example, by a large file transfer, the bandwidth of the connection can be adjusted to accommodate the increase in traffic. The file transfer would take place much more quickly. At the end of the file transfer, the bandwidth could be reduced to the original value. The benefit is efficient utilisation of available bandwidth, hence, reduced overall bandwidth costs.

Inverse multiplexing has been used in a variety of applications with the details of implementation are dependent upon the usage model. In any case, a complete solution must include [23]:

- A means for determining when to impose inverse multiplexing.
- A mechanism for the segmentation of data.
- A mode for scheduling the data over multiple links.
- A fast and accurate reassembly method.

## 2.3 Inverse Multiplexing of Wireless Channels

Wireless networks have become incredibly popular recently. Although they may not become a substitute for traditional wired networks, due to security issues, network performance and cost; they are becoming an increasingly popular second option. Wireless networks can largely be classified into two categories based upon the size of geographic coverage areas, WLANs and WWANs.

**WLANs:** WLANs are designed for the wireless connectivity in the home and

office environments. The typical range of geographical coverage is in the order of tens of meters in diameter [9][8]. To increase the coverage, repeaters or additional access points can be deployed. A WLAN has two modes of working, an infrastructure mode and an ad-hoc mode. In the infrastructure mode, an access point is used, and nodes communicate with each other through the access point. In the ad-hoc mode, wireless nodes communicate with each other directly, or via other communication nodes. Power consumption is fairly high for the WLAN communication as compared to the WWAN, making battery life and heat a concern. Bluetooth [9] and Wi-Fi [8] are two important WLAN technologies.

- Bluetooth is an open communication standard for exchanging data over short distances, using short length radio waves. Bluetooth can connect several devices, creating Wireless Personal Area Networks (WPANs) at distances up to 10 meters [9]. As devices in the WPAN broadcast radio waves, they do not have to be in the line of sight of each other.
- Wi-Fi sometimes called "Wireless Ethernet", denotes a set of WLAN standards commonly referred as IEEE 802.11x standards. Wi-Fi uses the same radio frequencies as Bluetooth, but with higher power, resulting in a stronger connection and better range from the base station. The Wi-Fi router using a 802.11b or 802.11g standard typically have a coverage area in the range of 32 meters indoors and 95 meters outdoors [8]. The new 802.11n standard however, can exceed that range by more than two times [8].

**WWANs:** WWANs connect different devices over a large geographical area to allow long range communication and resource sharing. The typical range of geographical coverage is in the order of tens of kilometres in diameter. WWANs are connected via radio, satellite and mobile phone technologies. WWANs can be broadly classified into (1) cellular networks and (2) mobile data networks, based on the types of network switching technologies being used. Cellular networks, being primarily voice oriented, generally utilise a circuit switching technology; whereas a mobile data network uses a packet switching technology. However, new cellular network technologies support the packet switching technology. Cellular technologies fall broadly into four categories:

- First Generation (1G) Technologies support the analogue cellular networks that use circuit switched connections for the data transport. In general, the analogue cellular infrastructure systems are not an efficient means of

sending data due to limited available capacities, limitations of data recovery, low security, and the high cost of use for many applications. One of the widely used 1G standards is Nordic Mobile Telephone (NMT) [24].

- Second Generation (2G) Technologies use the digital transmission technologies. Digital technologies provide many advantages for both the voice and data transmission. These advantages include increased system capacity, increased security against casual eavesdropping, superior cell hand-off, and better recovery of radio signal under different conditions. In addition to speech, these technologies support services such as fax, short messaging, and roaming of mobile end-stations. The existing standards in use worldwide include Global System for Mobile communications (GSM) [25], General Packet Radio Service (GPRS) [6] and Enhanced Data rates for GSM Evolution (EDGE) [7].
- Third Generation (3G) Technologies use the high-tech infrastructure networks, handsets, base stations, switches and other equipment, that enable mobile service providers to offer high-speed Internet access, data, and multimedia services. An example of 3G technology is Wideband Code Division Multiple Access (WCDMA) also known as Universal Mobile Telephony System (UMTS) [26].
- Fourth Generation (4G) Technologies support all IP packet-switched networks, mobile ultra-broadband (gigabit speed) access and multi-carrier transmission. They enable facilities such as IP telephony, ultra-broadband Internet access, gaming services and streamed multimedia for mobile users. Pre-4G technologies such as mobile WiMAX [27] and Long Term Evolution (LTE) [28] available in market are already being labeled "4G".

**WLANs vs. WWANs** WLANs generally provide higher bandwidth capabilities than the WWANs. They are typically privately owned, e.g. wireless systems that are deployed in a corporation, hospital, educational campus, etc. Wi-Fi compliant WLANs are used more by the home users and as public WLANs. WWANs use various devices, e.g. telephone lines, satellite dishes, and radio waves to service an area broader than a WLAN can cover. The typical coverage range of GSM is 35 kilometres [25] and 5-15 kilometres for WiMAX [27], as compared to 32-95 meters range for Wi-fi [8] and 10 meters for Bluetooth [9]. However,

the bandwidth/data transfer rate offered by the WWAN channels are up to 3 times lower than that offered by the WLANs [18]. (See Appendix A for detailed throughput comparison).

To overcome the bandwidth limitations of a single WWAN channel, Snoeren et al. [29] proposed to use inverse multiplexing. However, frequent variations in the participating channel conditions, e.g. available bandwidth, packet drop rate etc. adds extra challenge to the IMUX performance. He proposed an inverse multiplexing scheme, termed Link Quality Balancing (LQB), adaptive to the changes in the QoS conditions of bundled channels. LQB uses relative performance metrics to adjust traffic scheduling across the bundled channels. Prior work in the domain of inverse multiplexing of WWAN channels broadly follows two paths: (1) the bandwidth aggregation of all the WWAN channels attached to a single device and (2) multiple devices collaborate to aggregate the bandwidth of their individual WWAN channels.

### 2.3.1 Inverse Multiplexing of Multiple Channels attached to Same Device (MCSD)

This section briefly discusses the existing works supporting collaborative data transfer using inverse multiplexing of multiple channels attached to one device. Following is the brief summary of a few existing MCSD solutions:

- **Horde:** Horde[2] is a middleware that facilitates the flexible data striping for a diverse range of applications. However, its publications discuss only telemedicine applications. Horde supports inverse multiplexing of multiple WWAN channels attached to a special device. It claims to separate the data striping policy from the scheduling mechanism by providing support for applications to specify their network QoS objectives dynamically, using an objective specification language. Each objective simply defines the network QoS requirements of an individual data stream. Horde employs a random-walk scheduler that uses the QoS objectives specified by the application to schedule packets across multiple channels. The QoS variations in underlying WWAN are observed and scheduler make dynamic scheduling decision. We will discuss the scheduling algorithm employed by Horde in section 3.5.4.
- **MAR:** The Mobile Access Router (MAR) [12] makes use of the multiple WWAN channels available to a single mobile device. It studies how to

exploit different levels of diversity (e.g. channel diversity, network diversity, and technology diversity) in wireless data networks to support the high-speed mobile data applications. The MAR router is a multi-homed wireless device that can include multiple wireless interfaces from different wireless networks (e.g. GPRS, 3G, WLAN etc.) and operators (e.g., Vodafone, Orange etc.) simultaneously. MAR aggregates the available bandwidth in all available wireless interfaces. The MAR router can be placed in a moving vehicles (e.g. car, bus, train) to enable high-speed data access to multiple end users. Based on the application's QoS requirements, it dynamically instantiate new channels, aggregates the bandwidth and shifts load from low quality to better quality channels. MAR thus provides a faster, more stable, and reliable communication channel to the mobile users.

- **NATALIE:** NATALIE [15] is a network-aware traffic distributor, which combines multiple heterogeneous network channels and schedules IP packet transmissions over those channels dynamically. To achieve the maximum possible throughput, NATALIE measures the communication characteristics of the underlying links and dynamically adjusts its scheduler to assign packets to each link in proportion to its available capacity.
- **PluriBus:** PluriBus [13] is a system to provide high performance Internet access on-board in moving vehicles. Like other systems described above, its aim is to seamlessly combine multiple lossy WWAN channels into a reliable communication channel. PluriBus employs a novel technique called opportunistic erasure coding to achieve its aim.
- **MobiStream:** MobiStream [14] is a video streaming system that exploits the perceptual value in video content and the characteristics of the link layer and physical layer channels to enable error-resilient video streaming over WWAN. MobiStream achieves this by partitioning the video frames into a number of small, independently decode-able, blocks of data called "slices" and assigns priority based on its perceptual (visual) usefulness. Its priority-based video-data scheduling over multiple WWAN links enables error-resilient video streaming.
- **BAG:** Chebrolu et al. proposed a network layer architecture that enables a diverse multi-access service. Bandwidth Aggregation (BAG) is one of

these services [30]. This architecture supports simultaneous use of multiple network interfaces attached to one mobile terminal. An important aspect of the BAG service is the Earliest Delivery Path First (EDPF) scheduling algorithm that partitions the traffic onto different interfaces such that the QoS requirements of the application are met. It ensures that the packets meet their playback deadlines by scheduling them based on the estimated delivery time of the packets.

These works have proposed solutions mainly for video streaming and telemedicine applications. Horde [16], pTCP [31], and Snoeren et al. [29] have focused on the mechanism of striping packets across multiple WWAN channels attached to the same device. Since these systems assume that the multiple WWAN channels are attached to the same device, the issues of how to accomplish local communication, nodes privacy and data security are not considered.

### 2.3.2 Inverse Multiplexing of Multiple Channels in MCC

Other systems have followed the second path, and considered an ad-hoc group of multiple mobile computing devices interconnected through their compatible high-speed LAN interfaces. Sharma et al. [18] has named this ad-hoc group a *MCC*. Each MCC member uses its WAN interface independently, and optionally offers to transfer data for other members.

The set of participating channels connecting the MCC members to the IMUX can be logically combined to yield a higher-speed aggregated channel that is available for any of the individual MCC members. Distinct features associated with the MCC are local communication, presence of stranger nodes, node mobility and heterogeneity, etc. Next section discusses the MCC architecture and its features in detail.

## 2.4 MCC in Detail

As discussed in previous section, multi-homed mobile devices can form collaborative community to share the bandwidth of each other's WWAN channels using inverse multiplexing [18][1][17]. One of the promises of MCC is efficient data transfer across the Internet. The nodes in MCC are interconnected through their compatible high-speed WLAN channels. Each MCC member node independently

uses its WWAN interface to connect to the Internet via cellular base station. MCC member nodes share their WWAN channel (fully or partially) with other community members. The set of participating WWAN channels can be logically combined using an inverse multiplexing protocol. The IMUX running on the source node strips/schedules data across multiple collaborators in the MCC. Every collaborator then transfers allocated data using individual WWAN channel to the common destination. Figure 2.2 illustrates the infrastructure of MCC, where multi-homed devices in WLAN range form a mobile community and collaborate to simultaneously use multiple WWANs [1]. Such collaboration can increase the effective WWAN bandwidth available to every node of the MCC.

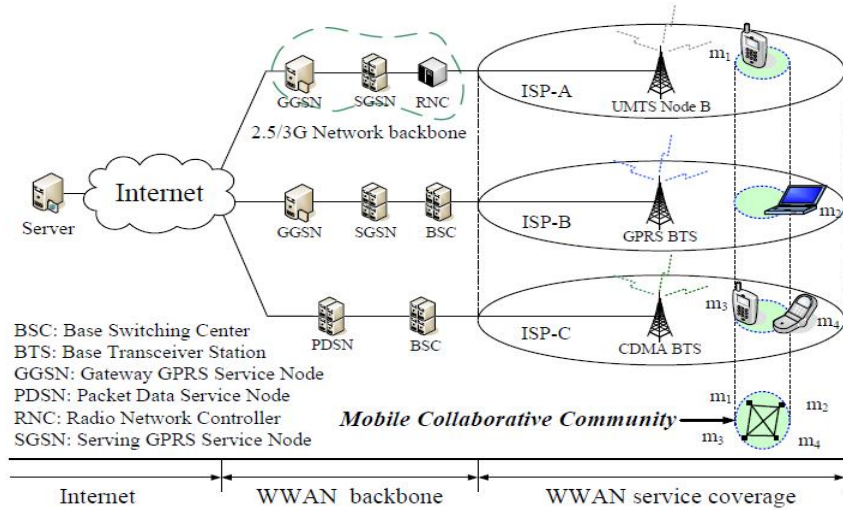


Figure 2.2: (MCC) Infrastructure: The environment includes various multi-homed mobile devices equipped with both WWAN and WLAN interfaces. Mobile devices form a mobile community using WLAN and collaborate to simultaneously use multiple WWANs. Reproduced from [1].

Existing systems supporting collaborative data transfer in MCC largely focus on the group (community) formation protocols, efficient data scheduling algorithms and congestion control mechanisms. PRISM [1] proposes a proxy based network layer implementation of an inverse multiplexer (PRISM-IMUX) and a congestion control mechanism (TCP-PRISM) at the sender side. MOBILE grouPEd Device (MOPED)[19] is a framework that enables a user's set of personal devices to appear as a single mobile entity connected to the Internet. This framework includes a transport layer protocol to aggregate the bandwidths of multiple



channels. It also provides a lightweight network layer MOPED Routing Architecture (MRA) to support multi-paths between the home agent and a receiver. Sharma et al. in Handheld Routers [18] proposes an application requirements aware and channel conditions adaptive architecture for data scheduling over multiple channels. COMBINE [17] is a system for collaborative downloading which proposes energy efficient algorithms for group formation and workload scheduling. It also proposes a framework for the collaboration incentives. It also models and estimates the energy cost required for the accounting system to support proposed incentive mechanism.

It is important to mention that most of the existing works have concentrated on collaborative data downloading, specially multimedia streaming [18][17], but have ignored the data uploading. Two major reasons for ignoring the support for upstream data transfer could be:

1. Existing systems assume that same solutions can work for upstream data transfer, e.g. PRISM [1] claims that it can support upstream data transfer by placing PRISM-IMUX at a mobile node in the community. Later, we will demonstrate that the scenarios of collaborative uploading have particular requirements (section 2.6.1) and challenges (section 2.6.2) associated with them. These factors should be kept in mind while designing an IMUX to support data uploading.
2. Mobile devices (specially mobile phone handsets) were not capable of generating and storing large amount of data. Even less than ten years ago mobile phone handsets and similar devices had internal memory/storage in tens of MegaBytes (MBs). These days the storage capability of many mobile devices is in tens of GigaBytes (GBs). Moreover, there is a recent trend in improving multimedia capturing support in mobile phones. Nowadays, mobile phones are usually equipped with high resolution digital cameras and video capturing support. All these factors add to the possibilities of creating and storing large files on mobile phones. Now mobile users may want to upload and share these files or broadcast live video streams. See Appendix B for trend graphs in mobile phones storage and camera resolution capabilities.

The majority of the existing works ignore security and privacy issues, which are vital in such a highly ad-hoc setting. COMBINE [17] and PRISM [1] briefly

discuss security issues and ask for a reputation and punishment system to discourage malicious behaviour of collaborating nodes, and propose to use encryption techniques to protect data. However, they do not address these issues in their system design, implementation and evaluation.

Moreover, most of the existing works ignore application specific QoS requirements. They are aimed at providing improved data scheduling algorithms under the assumption that the underlying channels are relatively stable and homogeneous. If this assumption holds, there is little reason to give an application control over how the scheduling is done. In an MCC environment, however, the underlying links are neither stable nor homogeneous. Therefore, the manner in which the IMUX decides to schedule the transmission of application packets can have a large influence on the observed bandwidth, packet latencies and stream loss rates. Furthermore, the application streams may be heterogeneous with respect to which aspects of the network service they are sensitive to: some applications care about average latency, some not; some care about the data loss more than the others; and some care more about the variance of the latency than they do about the average latency.

Although, there is not any particular application that motivated this work, understanding some aspects of the potential application scenarios is useful in understanding many of the design decisions underlying our proposed solution. Next section describes some of the scenarios of collaborative data transfer. It also lists some common observations from the application scenarios. At the end, three models of collaborative data transfer are presented.

## 2.5 Collaborative Data Transfer in MCC

The idea of Mobile Collaborative Communication (MCC) is not new and there is some existing work in this area as well. In the recent few years there has been immense development and progress in the area of mobile network technologies and device capabilities which can result in progress in the area of collaborative communication as well. This is the motivation of this project; to explore the possibility of an idea of achieving the efficient collaborative data transfer in a variety of scenarios.

The idea of MCC is very thrilling in the sense that it can revolutionised the way communication is done in many scenarios. If there is a solution that is a

bit more generic and provides enough security or reliability (which currently no system does) then it can take connectivity and usability of existing systems to a whole new level and open a door to a different realm. MCC uses low cost, highly, widely available and accessible mobile data connection. These connections can be owned or even hired. MCC can be used for myriad of purposes and it can open a whole new range of possibilities.

For some clarifications, let's consider the following MCC application scenarios.

### 2.5.1 Potential Application Scenarios

- **Disaster Relief:** I personally participated in the relief efforts when Pakistan was hit by a massive earthquake in October 2005 that caused widespread destruction in the country's Northern regions. Responding to the urgent need of coordination and management of the relief efforts, an information-sharing web portal named Relief and Information Systems for Earthquakes in Pakistan (RISE-PAK) <sup>1</sup> was set up. As a member of the RISE-PAK team, I witnessed the challenges in acquiring data from remote, earthquake affected villages. The traditional telephone wiring system had collapsed due to the earthquake. We had to rely on mobile phone services, which started working in most of the affected areas within 2-3 days of the disaster. Though the mobile service providers were offering Internet connectivity using GSM and GPRS technologies, the bandwidth was not sufficient to upload multimedia data on our portal efficiently. Another web portal was developed for locating the missing persons. In order to enter their details and pictures, the teams had to collect the required information from relief camps and hospitals and later upload it from the nearby offices with high speed Internet connections. Again, the data collection team had mobile phones but since the high bandwidth Internet connectivity was unavailable, a lot of time was consumed in collecting, transferring and entering data from the remote locations.

Later in 2011, a case study by Gisli Olafsson on how Information and Communications Technology (ICT) played a role in the 2011 Pakistan flood response, and how information was managed during the response, concludes that nothing much has evolved since 2005. The Internet connectivity had

---

<sup>1</sup><http://en.wikipedia.org/wiki/RISE-PAK>

improved but only slightly [32] [33].

- **Battlefield Reporting:** As discussed earlier, in battlefield reporting the soldiers equipped with multi-homed wireless devices may require to send data efficiently and securely to their commanders, in case of damage to their specialized communication equipment. This data may include geographical information like maps, video of battle field, pictures and voice messages. In this case, high reliability and security are two important requirements along with the efficiency of communication channel. This is also the case for many other emergency services, e.g. fire brigade, police and law enforcement agencies etc.
- **Telemedicine:** Consider the scenario of a person injured in a road accident that needs immediate medical care. If the patient's vital statistics can be relayed to the doctors at the hospital while still in the ambulance, doctors at hospital can be better prepared and patient can get treatment immediately upon its arrival. Such telemedicine application may have different data streams for pulse rate, blood pressure etc, and may require high speed, reliable Internet connectivity.

Another application of telemedicine is extremely relevant in developing countries since the major percentage of the population lives in remote, rural areas whereas the best-equipped hospitals and medical experts are concentrated in the urban areas. Access to health care by the majority of the population is limited given the low ratio of doctors to patients and the general economic deprivation of the people. Telemedicine is a cost effective and efficient way to bridge this gap between demand and supply of health care facilities in remote areas.

- **Tourisms:** Now consider a scenario of a group of tourists travelling or staying together. They may want to stream videos, share some large images or video files over the social media, or backup important data. They may also want to access other high bandwidth web services using their mobile devices. However, geographical or monetary limitations can limit their access to high speed Internet. Same scenario is valid for wild life photographers, mountaineers, or news reporters, etc.

### 2.5.2 Common Observations from Application Scenarios

Important observations that are common in all above mentioned scenarios are listed below.

- **Lack of Wired Infrastructure:** Wired communication channel for Internet connectivity is either not available or access to it is not feasible.
- **High Bandwidth Communication Channel:** High bandwidth communication channel is required, as many scenarios require the transfer of large data files or multiple data streams.
- **Heterogeneous Data Streams:** Most of the applications require the simultaneous transmission of multiple data streams, and these streams could be heterogeneous, e.g. video, audio etc.
- **Data Security:** Data that needs to be transferred may contain sensitive information, e.g. patient's personal information in telemedicine application and geographical location in military reporting application.
- **Availability & Reliability:** Communication channel should be highly available and reliable; i.e. ideally, there should not be any connection drop out. Secondly, there should not be a single point of failure in the network.

All scenarios require high speed, secure and reliable wireless Internet connectivity. WLANs can not be used because of their limited geographical coverage. WWAN channels can be used for the Internet connectivity in such situations. Despite 3G hype, individual WWAN channel may not be able to support high bandwidth applications efficiently. High speed WWAN technologies (e.g. EvDO, 4G) will take years to become widely available, especially in third world countries. These High speed WWAN connections can also incur significant monetary cost for the mobile users. Even in the presence of high speed WWAN connections, we may require back up communication channels that can be formed on demand. Ad-hoc networks can operate without the presence of any pre installed network infrastructure, while being able to handle node mobility and dynamic network topologies. Due to these properties, ad-hoc networks have been employed in many critical applications, such as military, law enforcement, or disaster-relief operations [34]. Hence, one possible solution is to form an ad-hoc MCC that aggregates a few low

bandwidth WWAN channels to form a high bandwidth virtual channel. Transferring data through this collaborative community promises an efficient data transfer even if few connections are lossy or unavailable. This solution is cost effective as the MCC is formed on demand using mobile devices and channels already available to the members. Moreover, it is flexible, scalable and easy to deploy as it uses existing networking infrastructure and its components (multi-homed mobile devices) are readily available. It has the potential to act as a primary as well as backup communication channel.

### 2.5.3 Generic Data Transfer Models

In this section, we present three generic models of collaborative data transfer in MCC. They are models for collaborative uploading and downloading in MCC, respectively, and a combined model which supports both. The main entities in these models include:

- Initiator (I)
- Collaborators (C1, C2, C3)
- IMUX / Data Scheduler (DS)
- MCC
- WWAN and WLAN channels
- Proxy
- Web/Application Server
- Internet Cloud

#### 2.5.3.1 Collaborative Uploading

The collaborative uploading model supports upstream data transfer from the collaborators to a web/application server on the Internet. Consider a setting where a mobile node (named *Initiator*) wants to send a large data file to a web/application server. Due to power, time, bandwidth or monetary constraints, the initiator seeks to utilize the WWAN channels of multiple neighbouring mobile devices, i.e. *Collaborators*. We assume that both the initiator and the collaborators are equipped with multi-homed mobile devices. Steps they need to follow for collaborative uploading are:

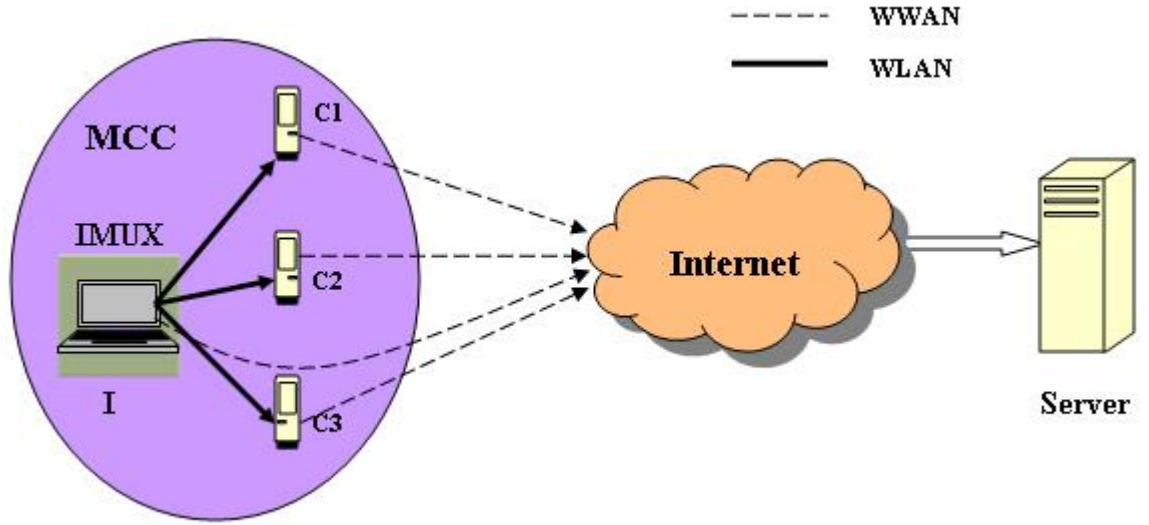


Figure 2.3: Model of Collaborative Upstream Data Transfer

Step 1. Initiator uses its WLAN channels to build a community of mobile collaborators who agree to share their WWAN channels to send data for the initiator.

Step 2. Once the group of collaborators is formed, the initiator starts dividing the file into data chunks. An IMUX/DS running on the initiator does this slicing and sends the scheduled data to the multiple collaborators through fast WLAN channels.

Step 3. Collaborators use their respective WWAN channels to send allocated data to the server across the Internet cloud.

Step 4. Aggregation program running on the server reassembles the data packets received from the multiple collaborators.

### 2.5.3.2 Collaborative Downloading

Now, we present the typical model of collaborative data transfer for the downstream traffic, which has been used in many existing systems [17][18][1]. Here, an initiator wants to receive a large data file or a data stream from a server through multiple collaborators. The motivation and assumptions are same as that for collaborative uploading, but steps to accomplish collaborative downloading are different.

Step 1. Initiator first builds a community of mobile collaborators.

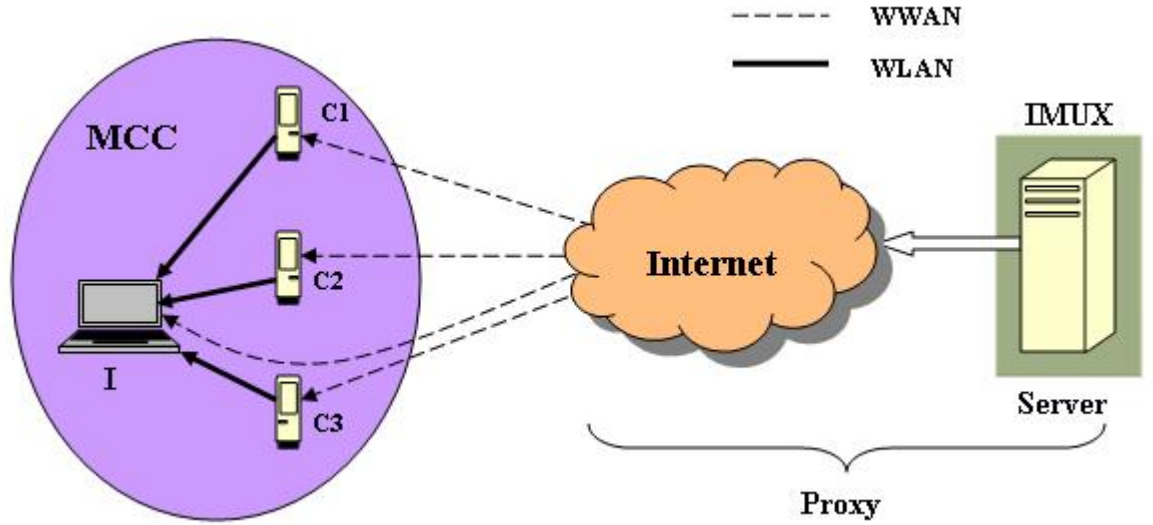


Figure 2.4: Model of Collaborative Downstream Data Transfer

Step 2. Once the group of collaborators is formed, the initiator sends a list of collaborators and their WWAN channel conditions to the IMUX/DS. In the case of downstream data transfer, the IMUX/DS runs on the server itself or on a dedicated proxy. This proxy can be located anywhere in the Internet cloud.

Step 3. IMUX/DS divides the file into appropriate data chunks and sends them to multiple collaborators through their respective WWAN channels.

Step 4. Collaborators receive data chunks and send them to the initiator through their high speed WLAN channels.

Step 5. Aggregation program running on the initiator reassembles the data chunks.

### 2.5.3.3 A Combined Model

The combined model of collaborative data transfer supports both upstream and downstream traffic. This model aggregates the components and functionality of previously described models. It can be seen from the Figure 2.5 that, to support the collaborative data transfer in both directions, the solution must have a dual implementation of IMUX/DS. As discussed earlier, most of the existing solutions in the MCC support downstream data transfer only, with IMUX/DS running on the proxy/server. To support data uploading in MCC, an IMUX/DS runs on the MCC initiator node. These nodes could be resource (e.g. battery,



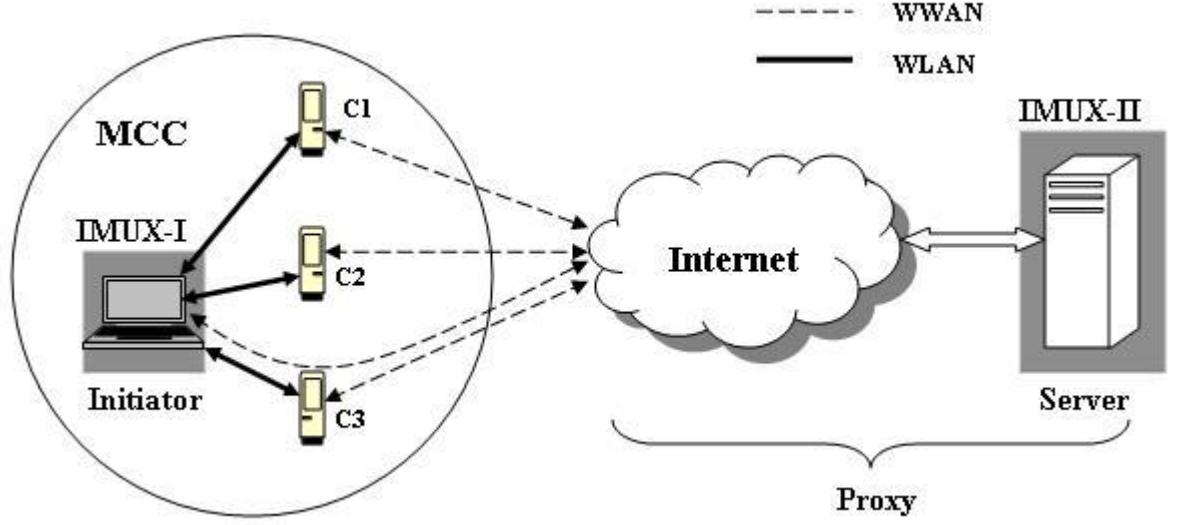


Figure 2.5: A Combine Model of Collaborative Data Transfer

processing power, and memory) constrained. Moreover, in the case of uploading, the IMUX/DS running on the initiator and collaborators communicate through WLAN instead of WWAN channels. Hence, the design considerations for the IMUX/DS running on the initiator are different from the one running on the server.

## 2.6 Achieving Efficient Collaborative Data Transfer in MCC

### 2.6.1 Requirement Specifications

In this section, we present the requirement specifications for supporting collaborative data transfer (both uploading and downloading) in the MCC, based on the potential application scenarios and the data transfer models. These requirements motivated our design of a novel and efficient data scheduler. We also demonstrate later, in section 2.7.5 that none of the existing solutions satisfies these requirements completely.

1. **Real Time Dynamic Channel Selection:** The proposed solution should support on demand group formation. Group formation should be initiated

by the source node, i.e. initiator, who wants to transfer a data file. Collaborators should respond to the collaboration request and exchange necessary information, e.g. remaining battery power and WWAN channel conditions like available bandwidth, loss rate, and latency etc. Initiator should be able to choose the most appropriate channels dynamically from a set of available channels. It should also be able to capture and adapt to the changes in underlying channel conditions.

2. **Application Requirements Aware:** The proposed solution should be aware of the application specific QoS and security requirements. Different applications may have different QoS requirements e.g. some applications can not tolerate data loss, where as, few are sensitive to jitter, latency, etc. [16]. Some applications may also have multi-level security requirements e.g. in telemedicine application, a patient's personal information might require higher security than the pulse rate stream. Our proposed solution should adapt to these requirements, during group formation and workload scheduling.
3. **Support for Applications with Heterogeneous Data Streams:** Some scenarios of collaborative data transfer may require transmission of multiple and possibly heterogeneous data streams. These data streams may have dissimilar QoS and security requirements. Our system should be able to handle these multiple and possibly heterogeneous data streams originated from the same or multiple applications.
4. **Protection for Sensitive Data:** Some scenarios of collaborative uploading may require the transfer of sensitive data, e.g. military reporting, telemedicine. This data may have multiple levels of sensitivity, hence our proposed solution should satisfy security requirements and be able to adapt to different sensitivity levels. MCC is a highly ad-hoc collaborative environment, where collaborating nodes may have no prior interactions with each other. Hence, our proposed solution should be able to deal with strangers and be able to operate securely with malicious collaborators.
5. **Battery Power and Processing Efficient:** As we have illustrated in the model for collaborative uploading that an IMUX/DS runs on a mobile node. This mobile node could be resource (e.g. memory, processing power,

battery power) constrained. Hence our solution should incur:

- **Minimal Computational Cost**

Proposed system architecture and constituent algorithms should be designed such that, they incur minimum processing cost. Less processing will save the battery power of the host node.

- **Minimal Communication Cost**

Also the proposed solution should be designed to minimize the inter-node and intra-node communication. Minimal communication will save the battery power consumed during the information transfer and the computational power used during the processing of transmitted information.

## 2.6.2 Challenging Issues

Challenging issues in designing a solution for MCC based on the above mentioned requirements are:

1. **Multiple Criteria Optimal Scheduling:** Our proposed solution must take into account the following factors and schedule data optimally, to utilize the collaborator's shared resources. Support for data security must be integrated into this optimal scheduling criteria.
  - **Application Specific QoS Requirements:** Application may have strict constraints on network QoS. How to specify and interpret these requirements, and adapt the solution according to them is one of most challenging task.
  - **Dynamically Varying Set of Available WWAN Channels:** As discussed earlier, in MCC nodes mobility is high, i.e. nodes can leave and join the community rapidly. These departures could be announced or unannounced, e.g. because of the channel failure. Furthermore we need to differentiate between the channel failure and temporary drop-out.
  - **Dynamically Changing WWAN Channel Conditions:** WWAN channels are characterised with frequent QoS fluctuations [16], e.g.

short term changes in available bandwidth, loss rate and jitter. Capturing and adapting to these changes is challenging and usually incur considerable processing and reporting cost.

2. **Support Data Security and Privacy:** In an ad-hoc collaborative environment, data security is one of the major concerns. The possibility of eavesdropping, spoofing, denial-of-service, and impersonation attacks increases, due to anonymity, high mobility, limited range and potentially unreliable wireless links of collaborating nodes. Security approaches used for the fixed networks are not feasible here due to the salient characteristics of the ad-hoc networks. In our MCC scenario, the following factors are most important:

- **Unknown Collaborators:** Due to high mobility, the collaborators in MCC may have no prior interactions with the initiator, which raises a serious issue i.e. lack of trust between the peers. Our system should be able to handle multiple levels of collaborator's trust.
- **Multi-level Data Sensitivity:** Different applications may have different security requirements, depending on the level of data sensitivity, e.g. a battle field video is more sensitive than a music video uploaded to web. Furthermore, different data streams or data segments originated from the same application may have different sensitivity levels, e.g. for military reporting application, highlighted important observations or geographical information may have high security requirements than the video or audio streams.

3. **Finding a Solution with Minimal Battery Power Consumption:** The most challenging task is to make the solution energy efficient such that its associated algorithms and the interactions and inter-operations among the architectural building blocks, will consume the least level of battery and processing power on the hosting mobile node. Although achieving energy efficiency is not the primary goal of our solution, still it is a major consideration during the design phase.

## 2.7 Existing Systems Supporting Collaborative Data Transfer in MCC

This section presents an overview of the existing works supporting collaborative data transfer in MCC and in the end critically analyses these existing works against the MCC scenario requirements specified in section 2.6.1.

### 2.7.1 Handheld Routers

Sharma et al. in Handheld Routers [18] focus on the delivery of high-bandwidth streaming media. They have proposed an application-aware and channel-adaptive architecture for data assignment over multiple low bandwidth channels. They have proposed three application aware data scheduling algorithms:

1. Layer Priority Striping (LPS)
2. Frame Priority Striping (FPS)
3. Independent Path Striping (IPS)

These algorithms schedule layers/frames/sub flows of data to the collaborators according to their WWAN channel reliability. They use loss rate as a metric for channel reliability. We will look into the details of these algorithms in section 3.5.1.

### 2.7.2 COMBINE

COMBINE [17] supports collaborative downloading in the MCC. It mainly focuses on the protocols of fast and energy-efficient group formation and data scheduling. It has proposed an energy efficient data scheduling algorithm named as Work Queue (WQ) algorithm. We will discuss this algorithm in detail in section 3.5.3. In the WQ algorithm, the initiator forms a WQ with fixed, equal sized data chunks of the file to be downloaded. Each collaborator queries the initiator for the next data chunk when it is finished with previous download and is ready for more work. The collaborators keep getting the download work until the queue is empty. Hence, the work performed by each collaborator is proportional to its WWAN speed. The initiator does not need to allocate work explicitly. COMBINE also discusses the incentives for collaboration, and security

and privacy issues in relation to their proposed accounting scheme. COMBINE is implemented at the application layer which makes it easy to get application requirements, and also to deploy existing networking infrastructure and mobile devices without any changes in hardware or operating systems.

### **2.7.3 PRISM**

PRISM [1] focuses on the data scheduling and congestion control mechanism to reduce the out of order packet deliveries. It proposes a proxy based network-layer implementation of an inverse multiplexer (PRISM-IMUX) and a transport layer implementation of a congestion-control mechanism (TCP-PRISM) at the sender side. PRISM-IMUX handles both the data forwarding and TCP acknowledgments (ACKs). It captures the data traffic at the network layer and schedules it to the best WWAN channel using an ADaptive Scheduler (ADAS). We will discuss ADAS in detail in section 3.4.3. The TCP-PRISM uses negative ACK information shipped by the Reverse Path Controller (RPC) at the proxy to reduce loss recovery time. It adjusts the congestion window size based on the WWAN-channels-state information.

### **2.7.4 MOPED**

MOBILE grouPEd Device (MOPED) [19] is a network model that deals with the group mobility. This framework treats a user's set of personal devices as a single, virtual device and supports the collaboration of these devices to appear as a single entity in the internet. MOPED includes a transport layer protocol to dynamically aggregate the available bandwidths of multiple channels to provide the user with the best possible network service. It also provides a lightweight network layer MOPED Routing Architecture (MRA) to support the multi-paths between the home agent and a receiver. MRA can take advantage of the additional bandwidth by routing different flows through different connections.

### **2.7.5 Analysis Summary**

Table 2.1 presents an overview of the assumptions, main contributions and future directions of the existing works described in previous sections.

Sate of the Art	Assumptions	Main Contributions	Future Directions
Handheld Routers	Used for delivery of high-bandwidth streaming media.	Channel adaptive scheduling for video transfer.	Estimation of WWAN sharing cost and accounting system.
	Application's QoS requirements are known	Implement application aware LPS algorithm.	support for information privacy and security.
			Identify homogeneous links for better aggregation results.
COMBINE	Multi-homed devices willing to share resources for a cost.	Cost model for collaboration incentives and accounting system.	Need for reputation system to blacklist cheaters.
	Mainly used for collaborative downloading.	Energy efficient and network conditions adaptive protocol for group formation.	Certification of transferred contentblocks by source (like BitTorrent).
	Presence of a trusted accounting server.	Energy efficient and network adaptive workload scheduling by implementing WQs.	
		Lightweight and secure accounting scheme.	
		Briefly discuss data/nodes privacy and security issues.	
		Application level implementation.	
PRISM		User interface.	
	Mainly used for music/video file download.	A network layer implementation of inverse multiplexer (PRISM-IMUX) on proxy.	Use proxy as trusted party and encryption to secure packet header information.
	Multi-homed devices willing to form collaborative community for mutual benefit.	Propose ADAS algorithm that is adaptive to WWAN channel conditions and incur least reporting overhead.	Use reputation and punishment system to discourage malicious collaborators.
	Mobile community is formed via Service Location Protocol.	A transport layer congestion control mechanism TCP-PRISM at sender side.	
	IMUX is located at Performance Enhancement Proxies (PEPs) of each 3G's access network.		
	Generic Routing Encapsulation (GRE) is enabled at each mobile host.		
	Both sender and receiver support TCP-SACK.		

Sate of the Art	Assumptions	Main Contributions	Future Directions
MOPED	Single user owns all the devices in collaborative group.	Transport layer implementation of aggregation protocol.	Support for sharing with non-MOPED-enable devices.
	Mobile community is formed via Service Location Protocol.	Lightweight network layer routing architecture to support multi-paths between home agent and a receiver.	MOPED specific transport layer protocol.
	Presence of a fixed proxy within the internet.	Support multiple components of locally connected set of devices.	User interaction model to configure and manage MOPED.
		Support movement of devices from one component to another.	

Table 2.1: Analysis Summary of Existing Systems Supporting Collaborative Data Transfer in MCC.

### 2.7.6 What is Still Missing?

We have already seen a brief overview of the existing works supporting collaborative data transfer. We are comparing the state-of-the-art in terms of addressing all the functional requirements stated earlier in section 2.6.1. We have added *Support for Data Uploading* as an additional requirement (justified by data transfer models in section 2.5.3.3). Table 2.2 summarises the comparisons that leads us to identify the knowledge gaps in existing efforts and finally, to an open issue, which we address in this research.

It can be seen from the table that there is not a single system that addresses all of the basic requirements of collaborative data transfer. Providing security for the sensitive data has been side-stepped in all of the above mentioned systems. They either assume that the same user owns all the devices (e.g., MOPED [19]) or that the improved performance alone is the key requirement (e.g., Handheld Routers [18], PRISM [1]). COMBINE [17] briefly discusses few data security issues and the candidate solutions. It discusses an energy efficient and secure accounting system in detail, to support its proposed collaboration incentives framework. Similarly PRISM [1] also mentions few data security issues and suggests to use encryption techniques or reputation system to protect against malicious behaviour of collaborating nodes. However, they do not include data security support in their proposed solutions. We are not aware of any prior work that addresses data



<i>Requirements</i>	<i>State of the Art</i>			
	Handheld Routers	COMBINE	PRISM	MOPED
On Demand Group Formation	✓	✓	✓	✓
Dynamic Channel Selection	✓	✓	✓	✓
Application Requirements Aware	✓	✓	X	X
Heterogenous Data Streams	X	X	X	X
Protection for Sensitive Data	X	<sup>1</sup>	<sup>2</sup>	X
Energy and Processing Efficient	X	✓	X	X
Support for Data Uploading	X	X	<sup>3</sup>	X

Table 2.2: Comparison of Existing Systems Supporting Collaborative Data Transfer in MCC against Scenario Requirements.

security issues in the context of collaborative data transfer in mobile collaborative communities.

Another important observation is that almost all systems support collaborative downloading, specially multimedia streaming, (e.g. COMBINE, PRISM). As illustrated in *Model for Collaborative Downloading* (see section 2.5.3.2) in case of downstream data transfer, IMUX runs on the proxy or server. This could be the reason that most of the existing systems ignored energy efficiency while designing their data schedulers. PRISM [1] claims that it can support upstream traffic by placing PRISM-IMUX at a mobile node in the community. In terms of energy efficiency they only mention the reduction in power consumption achieved by fast data transmission. There is no evidence that they consider energy efficiency as a key requirement in their system architecture and components design.

Considering the resource limitations of the MCC member devices, we can not simply use existing implementations of IMUX for data uploading. We need to

<sup>1</sup>Briefly discusses few data security issues and recommends to use authentication and reputation system.

<sup>2</sup>Listed few data security issue in future work and recommends to use encryption and reputation system for data security.

<sup>3</sup>Claims that their solution for collaborative downloading can work for uploading, but no implementation and evaluation.

consider the requirements and challenges associated with data uploading scenarios while designing an IMUX for mobile nodes. Energy efficiency and data security are issues that have largely been ignored and we are discussing and considering these open issues in our research.

Furthermore, there is not a single MCC solution that supports the simultaneous transfer of multiple data streams according to their individual QoS and security requirements. Designing a scheduling solution that can support best effort QoS for multiple data streams is going to be the focus of our research. Energy efficiency and data security would be part of our design consideration, however, they both are complete studies in their own. We have therefore included the detailed study, enhancement and evaluation, of energy efficiency and security of RAW scheduler in future work.

## **2.8 Our Research Goal**

Analysis of existing systems has led us to the open issue of how to design a data scheduler that is application requirements aware, adaptive to the changes in underlying channel conditions and supports the multiple levels of QoS and security in an energy efficient manner. Hence, our research goal is to design a performance efficient workload scheduler to support collaborative data transfer in dynamic MCC environment. Chapter 4 describes the design of a novel workload scheduler that meets this goal.

## **2.9 Chapter Summary**

The contributions of the chapter are three-fold: first, it has provided background information on inverse multiplexing and MCC; second, the applicability of MCC to a range of scenarios was analysed and requirements and challenges associated with collaborative data transfer in MCC were identified; third, the related literature was surveyed for the state-of-the-art approaches to support collaborative data transfer. Based on this survey, a research gap has been identified that may help to achieve better QoS in a MCC environment. The next chapter presents the literature survey of the state-of-the-art approaches to data scheduling employed by the IMUXs.

## Chapter 3

# Workload Scheduling Algorithms: A Literature Review

### 3.1 Chapter Introduction

This chapter provides a survey of workload scheduling schemes employed by Inverse Multiplexers (IMUXs). As we have discussed in previous chapters, IMUX schedules data segments, i.e. packets or fragments, across multiple outgoing channels. Scheduling techniques employed by IMUX varies from simple approaches like RR scheduling to more complex ones like application requirements aware, channel conditions adaptive scheduling. We have classified these scheduling schemes into four broad classes based on the approach used, These classes are (1) Simple Scheduling, (2) Application-Aware Scheduling, (3) Channel-Conditions Adaptive and (4) Application-Aware and Channel-Conditions Adaptive. This chapter critically analyses these approaches and evaluates their suitability. In conducting the survey, the good design principles of the approaches are examined for possible inclusion in our proposed solution. This chapter also briefly highlights various implementations of the scheduling algorithms such as the RR, CP and SWQ algorithm. We will be using these algorithms as a reference during the performance evaluation of our novel scheduling algorithm.

The organisation of the chapter is as follows: Section 3.2 discusses the working, merits and limitations of simple scheduling approaches. Section 3.3 discusses the working, merits and limitations of application requirements aware scheduling algorithms. Section 3.4 analyse the scheduling algorithms that are adaptive to the changes in underlying channel conditions. Section 3.5 discusses the most

intelligent class of scheduling algorithms which are both application requirements aware as well as adaptable to the network changes. Section 3.6 summarises the comparison between different scheduling approaches. Section 3.7 outlines the proposed approach we are going to employ in a novel RAW scheduler. Finally, Section 3.8 summarises the chapter.

## 3.2 Simple Scheduling Approaches

Broadly speaking simple scheduling schemes distribute data over multiple channels using some static criteria. These criteria may require information about i) the data to be scheduled, e.g. file size, packet size, and ii) channel conditions, e.g. bandwidth, loss rate, latency etc. These scheduling algorithms assume that the channel conditions are static, i.e. these conditions do not change over the period of time.

### 3.2.1 Round Robin (RR) Scheduling

The simplest scheduling approach is RR. It divides the data into segments of equal size. The RR scheduler keeps a static list of available channels. It cycles through all available network channels and sends the same number of data segments on each channel. In this way all the channels get the same amount of workload.

RR scheduling method is simple, which makes it easy to implement, and computation/energy efficient. It works reasonably well and provides fair workload distribution when the underlying channels are stable and homogeneous and data segments are of the same size [29].

However, this method can not make optimum channel usage when underlying channels are heterogeneous in terms of channel's QoS. Slow channel(s) may limit the overall throughput of inverse multiplexing system, by causing delay in data reassembly at the receiver [29]. This situation could be worse in the case of varying size of data segments, e.g. if scheduler alternates between long and smaller data segments and schedules them over two heterogeneous channels. In this case, all the longest data segments may get scheduled on the slow channels. Hence, arrival order of data segments at the receiver may differ from their order at the source. RR scheduling can be adjusted by adding sequence numbers to the data segments, which can be used to re-sequence data at the receiver [35].

### 3.2.2 Weighted Round Robin (WRR) Scheduling

WRR scheduling is designed to achieve a fair data distribution across the heterogeneous channels. It computes a weight for each network channel. These weights can be fixed or estimated based on the channel conditions, e.g. bandwidth, loss rate, etc. This scheduling algorithm cycles through the network channels and sends as many data segments as indicated by the weight associated with that channel [29] [16]. For example, if there are only two channels, and weight of channel  $c1$  is 1 and channel  $c2$  is 3, then  $c1$  will get  $(1/(1+3))$ , i.e. one forth of the total workload. Hence a channel with the higher weight gets more workload.

Weighted Random (WR) scheduling is a special case of WRR where weights are computed using weighted probability. Weighted probability indicates the probability of selecting a particular channel for the next transmission. This allows a proportionality that is not limited to the ratios of integers as in WRR scheduling [23].

WRR scheduling improves the overall throughput of the system by assigning less workload on slow channels that can reduce delay in data reassembly at the receiver. However varying size packets are still problematic for fair load distribution and data reassembly. Moreover, computing weights based on the channel's QoS is computationally expensive, specially in the case of MCC which is characterised with frequent QoS changes due to the mobility of the collaborating nodes. Furthermore, RR and WRR scheduling schemes assume that all packets are from one data stream, and for multi-stream applications may result in unfair distribution [16]. For example, consider an application with three input data streams  $s1, s2$  and  $s3$  and have three or a multiple of three network channels available. Suppose, there are three channels  $c1, c2$  and  $c3$ . RR or WRR scheduler will map  $s1$  to  $c1, s2$  to  $c2$  and  $s3$  to  $c3$ . This is a fair distribution if underlying channels are homogeneous. However, for heterogeneous channels if  $c3$  is lossier than the other two, then stream  $s3$  will experience much higher average loss rate than the other two data streams.

### 3.2.3 Randomized Round Robin (RandRR) Scheduling

To overcome the limitations of RR and WRR for supporting multi-streams, Zhau [36] proposed Randomized Round Robin (RandRR) scheduling. In this case, the IMUX keeps a static list of the input data streams and output transmission

channels. It first selects an output channel using a random function then applies any RR variant scheduling scheme for selecting the input data stream and vice-versa [16] [36].

Although this scheduling scheme tries to deal with the multiple data streams fairly, it inherits a few limitations from RR and WRR scheduling, e.g. varying sized data segments may delay some packets.

### **3.2.4 Limitations**

The simple scheduling approaches described above have a few common limitations, one major limitation is that they all use static criteria for the data scheduling. These schemes assume that the number of underlying physical channels will remain the same, i.e. no channel will leave or join, and that the channel conditions will stay static during the scheduling.

Secondly, scheduling schemes that fall in this class do not consider the application level scheduling requirements. Different applications may have diverse QoS and security requirements, e.g. some applications are sensitive to data loss few are to jitter, latency, etc. [16]. Cheung [37] analysed special scheduling requirements associated with the delay sensitive media, and developed an algorithm for smart scheduling of streaming media along with error correction over multiple burst-loss channels. Making the scheduler aware of these requirements can improve the overall QoS performance.

## **3.3 Application-Aware Scheduling Approaches**

Different applications may have diverse QoS and security requirements, which can not be handled by the simple scheduling algorithms. Furthermore, multi-stream applications may have different requirements for different data streams. Hence scheduling algorithms should be more flexible to accommodate these diverse requirements. Existing application aware scheduling algorithms include Channel Pinned scheduling and Call-back scheduling.

### **3.3.1 Channel Pinned Scheduling**

Channel Pinned (CP) scheduling allows multi-stream applications to pin each data stream with a pre-defined network channel, i.e. each data stream will be

scheduled on at least one fixed network channel. This scheduling technique is simple and easy to implement as it doesn't make dynamic decisions about the channel allocation. It is a popular solution for scheduling multi-path multimedia streams [11]. Channappayya [11] uses this scheduling scheme by dividing an image into four coded bit streams, then transmitting them over the four low-rate noisy channels. At the receiver, the image is reconstructed by assembling the data received from each channel.

This scheme doesn't support reallocation of data streams in the case of channel failure. Moreover, it discards all the packets of a data stream that its associated channels can not hold, even if other channels have capacity. To overcome these limitations, splitting scheme should be able to facilitates interpolation, i.e. the estimation of lost data from available data. For example, Channappayya [11] presents a multiple description image coding scheme which facilitates the interpolation process in the case of one or more channel failures.

### **3.3.2 Call-back Scheduling**

Call-back scheduling exposes available network channels to the application, asking it to assign data across the network channels itself. The application's policy is hard-wired in the call back code. Since the application itself is scheduling the data across the channels, it can provide specialized code to meet its requirements [12][38].

In principle, a call-back scheduler implemented for a specific application has the potential to be more computationally efficient. However, it is difficult to implement for even moderately complex applications [16]. This is specially true when there are multiple applications and multiple streams each having diverse requirements.

### **3.3.3 Limitations**

This class of scheduling algorithms try to cater for the applications with specific requirements and schedule the data over the appropriate network channels. However, application-level implementation of these scheduling algorithms usually incur packet processing overhead. Most importantly, these algorithms do not adapt to the QoS changes in the underlying channel conditions. This adoption is very crucial specially when underlying channels are WWAN channels, as they

are characterised with frequent QoS fluctuations and drop-offs [16].

## 3.4 Channel-Conditions Adaptive Scheduling Approaches

In our scenario of inverse multiplexed WWAN channels, we need to consider frequent changes in the underlying channel conditions, e.g. failure, announced departures or fluctuations in QoS. In case of these changes, scheduler should respond and reallocate data accordingly. Following scheduling algorithms adapt in response to the changes in channel conditions.

### 3.4.1 Link Quality Balancing (LQB) Scheduling

LQB scheduling proposed by Snoeren [29] extends the WRR scheme by dynamically adjusting the Maximum Transmission Unit (MTU) size for each link<sup>1</sup>. In this scheme the packets are split into fragments of MTU size, proportional to the observed bandwidth of the underlying channels. LQB uses the loss rate as link quality metric to compute a relative short-term available bandwidth for each channel. Here, the goal is to create variable length fragments such that each fragment can be transmitted in roughly the same amount of time. In this way reassembly can proceed with much less delay. Although this scheme reduces the chances of packet reordering, slow channels can limit overall throughput of the system. Furthermore, fragmentation incurs reasonable processing overhead.

### 3.4.2 Rate Based Scheduling

Magalhaes et al. [39] addresses data scheduling over multiple channels of the same mobile host. The rate based transmission technique is used for the bandwidth tracking and congestion control. They employ the packet pair probing technique [40] to measure the available bandwidth of a channel. This technique entails sending two packets back-to-back on a channel, and measuring the inter arrival time of those packets at the receiver. The inter arrival time is a snapshot of the bandwidth of that channel when packet size is equal to the channel's

---

<sup>1</sup>Terms links and channels can be used interchangeably



MTU. Inverse of this time defines the optimal rate to send packets on this channel. Hence by sending packets at regular intervals, and tracking the inter arrival times at the receiver, changes in channel bandwidth can be tracked without using packet pair method. The Multimedia Multiplexing Transport Protocol (MMTP) [38] is a rate based multiplexing protocol designed to carry packets with strict deadlines. Another rate based protocol is Reliable Multiplexing Transport Protocol (R-MTP) [41], designed for the reliable transmission of bulk data to the mobile systems that have access to multiple link-layer technologies. R-MTP uses selective acknowledgements for the reliability and bandwidth estimation for flow and congestion control.

### 3.4.3 ADaptive Scheduling (ADAS)

Optimal scheduling of packets over heterogeneous wireless channels require up-to-date channel-state information. Obtaining this information is expensive, specially in mobile environments, due to the fluctuating data traffic rate and the wireless channel dynamics [1]. Although it is possible to measure a channel's condition and report it to the proxy, but frequent changes will incur significant report processing overhead and transmission power consumption to resource limited mobile hosts. ADAS proposed in PRISM [1] is a light weight packet scheduling algorithm. It maintains up-to-date channel-state information by using an ACK-control mechanism i.e. Reverse Path Controller (RPC). The RPC exploits TCP's control information which is carried by ACKs and infers each channel's state information e.g. packet loss, delay and data rate from the ACKs. It provides this information to ADAS without incurring any reporting overhead to mobile nodes. Moreover, it uses packet's expected arrival times over each channel to reduce out-of-order packet deliveries, and to increase overall system throughput. It is important to mention that PRISM is a solution for collaborative downloading, in that case RPC and ADAS runs on a proxy. PRISM takes off reporting overhead from the mobile hosts but RPC is still a processing overhead in the case of collaborative uploading, where RPC needs to run on a mobile device.

### 3.4.4 Limitations

This class of scheduling algorithms adapt to the changes in underlying channel conditions. Maintaining channel conditions information is expensive in terms of

processing and communication overhead. Most of the algorithms in this class are implemented at transport or network layer, which require end-system kernel or protocol stack modification. Additional disadvantage of the network layer implementation is that, it is generally not aware of the application characteristics and requirements.

## 3.5 Application-Aware and Channel-Conditions Adaptive Scheduling Approaches

The scheduling algorithms belonging to this class are both application requirements aware and adaptive to the changes in channel conditions. Several efforts in this class meet most of our scenario requirements. (see Table 3.1)

### 3.5.1 Adaptive Channel Pinned Scheduling

This scheduling approach is based on the concept of channel pinned scheduling described earlier in section 3.4.1. However this *Adaptive* version responds in adaptation to the underlying channel conditions by redistributing the data streams as the channel conditions change. Redistribution is usually not frequent, and short term variations are ignored. Sharma et al. in Handheld Routers [18] has presented following three algorithms for video streams.

#### 3.5.1.1 Layer Priority Striping (LPS)

This algorithm can schedule video streams that are hierarchically layer-coded. This encoding process generates a base layer  $l_0$  containing the most important information required for decoding, and one or more enhancement layers ( $l_i : i = 1, \dots, n$ ). Enhancement layer  $l_k$  can only be constructed if all sub-layers  $l_i : i = 0, \dots, k - 1$  are available. Thus, the layer index  $i$  corresponds to the layer priority. The LPS algorithm assigns base layer to the most reliable channels, and enhancement layers to the channels of corresponding reliability level. It uses the channel loss rate as the metric for reliability. For each layer, packets are distributed across the allocated channels using WRR scheduling. Allocation of layers to the channels *Shift Up* if a new channel of higher reliability joins and *Shift Down* if it becomes unavailable.

### 3.5.1.2 Frame Priority Striping (FPS)

The FPS algorithm splits data based on the frame type. Sharma et al. [18] have used this algorithm for scheduling MPEG video stream, due to its inherent frame types. MPEG video [42] in general consists of three types of frames, 1) intra-coded pictures i.e. *I*-frames, 2) predictive coded pictures i.e. *P*-frames, and 3) bi-directionally coded pictures i.e. *B*-frames. The priority order of the frames is  $I > P > B$ , based on their dependency on each other for decoding. Hence, MPEG video stream can be separated into three sub-streams ( $sf_I, sf_P, sf_B$ ) based on these frame types. Similar to LPS, in FPS channels are allocated according to the sub-stream priority. The *I*-frame sub-flow ( $sf_I$ ) is sent over the most reliable channels, and so on. This allocation also adapts to the changes in channel conditions, which in this case is channel's loss rate.

### 3.5.1.3 Independent Path Striping (IPS)

This algorithm is well suited to the multiple state video coding [43], where a stream is encoded into multiple *independently* decode-able sub-flows. However, information from one sub-flow can be used to correct the errors in other sub-flows. The IPS tries to achieve path diversity by allocating a separate channel for each description. Video can be reconstructed back (although at lower quality) even if one or more sub-flows are lost. This scheduling scheme is helpful in the scenarios where unannounced channel drop-off are frequent. This scheme adapts to the long term drop offs by re-allocating corresponding sub-flows to other available channels.

## 3.5.2 Adaptive Call-back Scheduling

*Adaptive* call-back scheduling allows applications to schedule data across the available channels itself and adapts to the changes in channel conditions. See section 3.4.2 for details on call-back scheduling. It works like the congestion manager framework of the operating systems [44]. This framework uses a module called congestion manager, which provides the network flow management and allows applications to be notified and adapt to the changing network conditions. However, such algorithm is difficult to implement for multiple applications and for multiple streams.

### 3.5.3 Work Queue (WQ) Scheduling

COMBINE [17] proposes this energy efficient algorithm that supports collaborative downloading. According to this algorithm, the initiator splits the file to be downloaded into fixed equal sized work items. These data chunks (byte ranges) constitute the WQ at the initiator, i.e. the sender. Collaborators query the initiator and pick up the next available work item from the WQ. Each collaborator downloads the amount of data specified in its work item and transfers it to the initiator. It then picks up more work by querying WQ and keep working until WQ is empty. In this way the work performed by each collaborator is proportional to its WWAN speed, without the initiator having to allocate work explicitly. Hence, there is no need to maintain up-to-date channel state information. This inherent feature of WQ algorithm makes it energy efficient and suitable to run on a mobile device.

However, the work-item size needs to be picked appropriately. Too large work-item size will make this algorithm less agile to the changes in channel conditions. It also increases the amount of data that needs to be resent in case of any channel failure. On the other hand, too small a work-item size will place considerable processing overhead on the scheduler.

### 3.5.4 Objective Driven Scheduling

Horde [16] introduces an objective driver scheduling technique in which an application can specify its QoS objectives. Horde also provides a specification language that allows the applications to express their scheduling goals as network-QoS objectives. *"An Objective defines a QoS goal and describes how the achievement of that goal adds to, or subtract from, overall application utility"* [16].

Objectives can be defined for single or multiple streams. When an application does not specify an objective for some QoS aspect of a stream, it is assumed that the application does not care about that aspect of QoS for that particular stream. Hence, better channel transmission slots are first assigned to the streams who specify that QoS aspect in their objective and rest of transmission slots are allocated to those stream who have not.

### 3.5.5 Limitations

This class of scheduling algorithms makes most intelligent schedules, but it incurs significant processing overhead, as obtaining the updated channel conditions and application requirements is expensive in terms of processing and reporting. Moreover, such multi criteria scheduling algorithm is difficult to implement.

## 3.6 What is Still Missing?

We have already discussed the working, merits and limitations of each scheduling algorithm and have classified them into four scheduling classes. Table 3.1 summarizes the comparison of these scheduling approaches against our scenario requirements stated in section 2.6.1. This comparison leads us to identify knowledge gaps in the existing efforts and finally to our choice of a novel RAW scheduling approach.

<i><b>Scheduling Approaches</b></i>	<i><b>Scheduling Requirements</b></i>				
	Dynamic Channel Selection	Application Requirements Aware	Support for Heterogenous Data Streams	Protection for Sensitive Data	Minimal Computation and Communication Cost
Simple	X	X	X	X	✓
Application-Aware	X	✓	X	X	X
Channel-Conditions Adaptive	✓	X	X	X	<sup>1</sup>
Application-Aware and Channel-Conditions Adaptive	✓	✓	<sup>2</sup>	X	<sup>3</sup>

Table 3.1: Comparison of Existing Workload Scheduling Approaches against Data Scheduling Requirements.

Table 3.1 shows that none of the scheduling approaches fulfils all the MCC scenario requirements. However, the application-aware and channel-conditions adaptive scheduling approach is most related to our scenario requirements. This approach successfully addresses the two basic requirements, i.e. dynamic channel selection and application requirements awareness. There are three very important MCC requirements that are overlooked by almost all related scheduling algorithms. These requirements are:

<sup>1</sup>ADAS claims to be energy efficient.

<sup>2</sup>Objective driven scheduling supports multiple heterogeneous data streams.

<sup>3</sup>WQ scheduling is energy efficient by design.

1. **Support for Multiple Heterogeneous Data Streams:** Some scenarios of collaborative data transfer may require the transmission of multiple and possibly heterogeneous data streams. Almost all existing scheduling approaches can not handle these multiple and possibly heterogeneous data streams originated from one or multiple application(s). Most of the application requirements aware algorithms focused on one application with only one data stream except Horde [16]. They consider multimedia streaming [1] [17] as a motivating application. Only the objective driven scheduling approach proposed by Horde [16] allows an application to specify its QoS goals for the multiple streams.
2. **Data Security:** Some scenarios of collaborative data transfer may require transfer of sensitive data, e.g. military reporting, telemedicine. This data may have multiple levels of sensitivity, hence the scheduling algorithm should be able to satisfy security requirements and, be able to adapt to different sensitivity levels. Providing security for the sensitive data has been side-stepped in all of the above mentioned scheduling approaches. They assume that the improved performance alone is the key requirement [18] [1]. We are not aware of any prior work that addresses the data security issues in the context of collaborative data transfer in MCC.
3. **Energy Efficiency:** As mentioned earlier, almost all existing systems support collaborative downloading, where IMUX runs on a proxy/server. This could be the reason that most of the existing systems ignore energy efficiency while designing their workload schedulers. PRISM [1] claims that it can support upstream traffic by placing PRISM-IMUX at a mobile node in the community. In terms of energy efficiency they only mention the reduction in power consumption achieved by fast data transmission. There is no evidence that they consider energy efficiency as a key requirement in their system architecture and components design. Considering the resource limitations of MCC member devices, we can not simply use the existing implementations of IMUX for supporting data uploading.

### 3.7 Best Way Forward: A Risk-Aware Workload (RAW) Scheduler

Analysis of existing scheduling schemes against the requirements has led us to a design decision that our proposed solution must be adaptive to the application requirements and changes in the channel conditions. Furthermore, it is realised that energy efficiency should be an important design consideration, hence, the proposed scheduler needs to be designed such that the QoS performance is optimal and the energy consumption is minimum. By providing the scheduler with the application specific QoS and security requirements and channel conditions information, we are making it aware of the risk associated with each scheduling decision.

Few existing works in data scheduling are most related to our proposed scheduling approach. For example, the WQ algorithm proposed by COMBINE [17] is adaptive to the channel conditions in an energy efficient manner. We propose to use a novel multi-level WQ algorithm to cater the multiple levels of application's QoS and security requirements (see chapter 5 for details). The application-aware scheduling algorithms presented by Sharma et al. in Handheld Routers [18], are of our interest as they schedule data according to the priority of frames, layers or sub-flows.

### 3.8 Chapter Summary

This chapter has surveyed some of the scheduling algorithms that are proposed for the inverse multiplexed environments. It has classified them into four classes and critically analysed each class. It then compared the state-of-the-art scheduling approaches in terms of addressing all the MCC scenario requirements. It has been identified that the application-aware and channel-conditions adaptive scheduling approach is most related to our MCC scenario requirements. However, several limitations have been identified in most of the related work. For example, limited support for multiple heterogeneous data streams, lack of data security and energy efficiency. At the end, we have outlined our vision to design a RAW scheduler that overcomes the limitations of existing solutions. We will discuss our vision of risk-aware scheduling and a detailed design of the proposed RAW scheduler in the next chapter.

# Chapter 4

## The RAW Scheduler: Architectural Design

### 4.1 Chapter Introduction

This chapter describes the architectural design of the RAW scheduler, and the methodology used to evaluate its performance. It describes the working of each of its architectural component in details. These components are the Data Splitting Module (DSM), the Encryption and MAC (Message Authentication Code) Service, the Work Queue Manager (WQM), the Trust Manager (TM), the Access Control Module (ACM) and the Collaborators Monitoring Module (CMM).

The organisation of this chapter is as follows. Section 4.2 and 4.3 outlines the requirements and assumptions for a novel RAW scheduler. Section 4.4 presents the high-level architectural overview. Section 4.5 presents the detailed architecture design. It describes the functionality of each component of the proposed scheduler. Section 4.6. discusses a case study of MPEG video scheduling using the RAW scheduler. Section 4.7 discusses different evaluation methodologies and justifies the choice of using a hybrid of simulation and test-bed experiments as the evaluation methodology for this thesis.

### 4.2 Design Requirements

The RAW Scheduler enables efficient and secure scheduling of data in the MCC, in a processing efficient manner. The architectural design of this scheduler is based on the requirements discussed in section 2.6.1. These requirements include:



- Support for multiple applications where each may have multiple data streams.
- Best-effort support of QoS and security requirements associated with each application.
- Adaptive to the changes in channel conditions measured in terms of available bandwidth and data loss rate, etc.
- Adaptive to the QoS capability levels of the collaborators,.
- Energy efficient.
- In addition to the functional requirements mentioned above, the RAW scheduler architecture should be *modular* to enable flexibility and extension of its components.

In satisfying the above requirements, we need to:

1. Design a framework to capture the application level QoS and security requirements for multiple streams.
2. Design a data splitting algorithm which takes into account the application level requirements.
3. Design an algorithm to evaluate the capability levels of the potential collaborators.
4. Design an access control method that takes into account both application's QoS requirements and collaborator's QoS capability levels while granting the access permissions to multiple collaborators.

The challenging issue here is how to accomplish (1)-(4) with minimum costs. In other words, design decisions should be taken at the architectural, components and algorithmic levels to minimise the communication and processing costs on the hosting mobile node.

### 4.3 Design Assumptions

The following assumptions have been made in the design of the RAW scheduler:

- The community formation is managed outside the scheduler by a module called the Community Manager. The task of community formation can be accomplished by using a community formation protocol such as those proposed in [17] [18]. the Community Manager has the option of adding more collaborators in cases where existing transmission channels are not capable of supporting the required level of QoS and if there are more (ideally better quality) channels available.
- The congestion control mechanism has not been considered in this scheduler design. This function is considered as a separate module from the scheduler. There are a few existing proposals to control the transport layer congestions [1][38].
- The RAW scheduler can support multiple QoS priority levels, but, for the proof of concept, in this thesis, we use three QoS priority levels. The data with highest QoS priority, ( i.e. highest bandwidth and/or loss tolerance requirements <sup>1</sup>) is hosted in highest level WQ, i.e.  $WQ_1$ . The  $WQ_2$  hosts data with lower QoS requirements and so on. It is worth noting that the RAW scheduler design is dynamic, hence, can support WQs with more than three levels.
- We assume that the RAW scheduler runs only one scheduling session at a time, and during each session it schedules data for one application only. However, an application can have multiple streams with different, possibly conflicting, QoS and security requirements.
- We also assume that applications specify their QoS and security requirements by attaching a corresponding tag to the data. Implementing a 'Data Tagger' is outside the scope of our project, however, we have defined the specifications for these QoS and security tags. For application level QoS, we consider end-to-end channel characteristics like available bandwidth and loss tolerance. Three security requirements namely confidentiality, integrity and availability are considered in the design. It is important to mention that the design is flexible enough to support additional QoS and security characteristics.

---

<sup>1</sup>we have reversed the data loss rate to get a loss tolerance value to make it a measure of goodness

## 4.4 Architecture Overview

This section presents a high level view of the RAW scheduler's architecture. *"Risk is the combination of the probability of an event and its consequences"* [45]. Whereas, *"trust is the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible"* [46]. Generally, a trust relationship involves two parties: a trustor and a trustee. Yan [47] defines *"trustor as a person or entity who holds confidence, belief, faith, hope, expectation, dependence, and reliance on the goodness, strength, reliability, integrity, ability, or character of another person or thing, which is the object of trust - the trustee"*. As shown in Figure 4.1, there is a correlation between the risk and trust, i.e. more trust implies less risk and vice versa [48]. Hence, trusting infers the willingness of a trustor to accept perceived risk, if any, from the trustee.

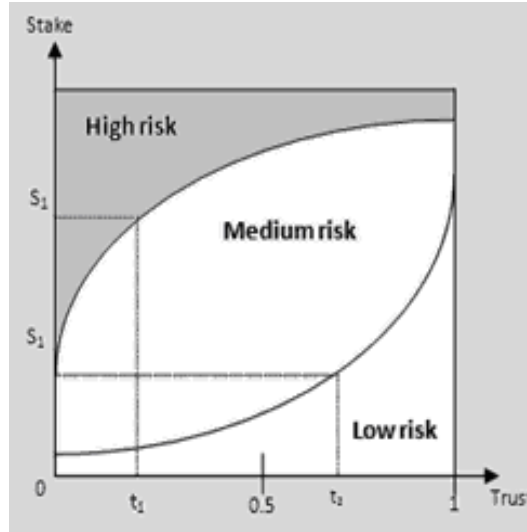


Figure 4.1: Risk and Trust. Reproduced from [2]

In the MCC, perceived risk is higher mainly because of the node anonymity, high mobility, limited range and potentially unreliable wireless links of the collaborating nodes, etc. The possibility of eavesdropping, spoofing, denial-of-service attacks increases. The QoS and security approaches used for the fixed networks are not feasible in MCC due its ad-hoc infrastructure and resource constrained nature of the member devices.

**Our Vision:** The RAW scheduler employs the correlation between the values at

stake and the values of trust (as shown in Figure 4.1 ), to minimize the risk of poor QoS performance associated with the scheduling decisions. To do so, it splits a data stream into multiple sub-streams according to the QoS requirements of the application. The application requirements are specified for each data stream and are notified using tags that are sent along with the associated data stream. The RAW scheduler splits a data stream  $S$  into multiple sub-streams  $S_i : i = 1, \dots, n$  based on the QoS tags. Here, the sub-stream index  $i$  corresponds to the sub-stream QoS priority, where the first index refers to the data with the highest bandwidth and loss tolerance requirement. In other words,  $S_1$  would be a data stream with strict QoS requirements and  $S_2$  with less strict QoS requirements and so on. The multi-level WQs are also assigned with the priority levels. The sub-streams are allocated to the multi-level WQs in such a manner that the highest priority data is buffered in the WQ with the highest priority level. Hence, there is a one-to-one relation between the sub-stream QoS priority level and the WQ priority level (here after called index). Equation (4.1) represents the assignment of sub-streams on the WQs.

$$WQ_i \Leftarrow S_i : i = 1, \dots, n \quad (4.1)$$

The RAW scheduler also ranks collaborators according to their QoS (bandwidth/loss tolerance) capability levels,  $C_i : i = 1, \dots, n$ . The capability level of a collaborating node is evaluated based on its QoS offered value and QoS trust value. The QoS offered value is provided by each collaborator during the community formation phase. The QoS trust value is evaluated by the initiator and it is based on the collaborator's QoS performance during previous collaborations. The trust evaluation process considers previous collaborations of a collaborator with the initiator as well as with other MCC member nodes.

The scheduler schedules a sub-stream over the nodes with a capability level that is not lower than the priority level of the sub-stream. In other words, assuming that a sub-stream has a priority level of  $S_k$ . The nodes that are allowed to serve this sub-stream must have a minimum capability level of  $C_i \geq S_k$ , where  $i = k, k + 1, \dots, n$ . To define these access permissions, a novel CAPability-aware Access Control (CAPAC) model is proposed. In this model, access permissions are defined based on the collaborator's (i.e. subject's) QoS capability level. The objects to be protected are WQs. In this way, a RAW scheduler makes risk-aware scheduling decisions to achieve best effort QoS. Figure 4.2 represents our vision

of capability-aware access control model.

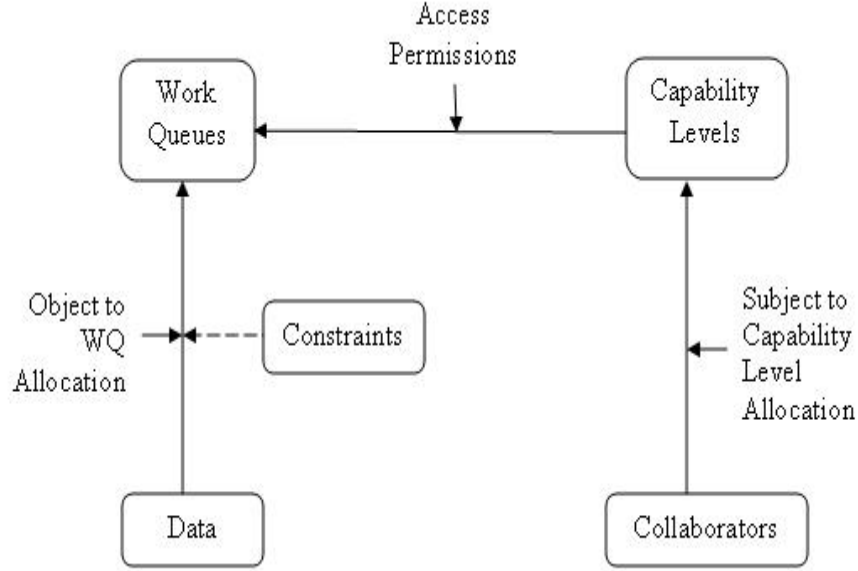


Figure 4.2: CAPability-aware Access Control (CAPAC) model

Figure 4.3 sketches the high-level architecture of the RAW scheduler. It can be seen that scheduler has the following architectural components:

- Data Splitting Module (DSM)
- Encryption and MAC (Message Authentication Code) Service
- Work Queue Manager (WQM)
- Trust Manager (TM)
- Access Control Module (ACM)
- Collaborator Monitoring Module (CMM).

The security requirements of an application are also specified in the form of tags, and these tags are generated and attached to the data stream(s) by a module called the 'Data Tagger'. Upon the receipt of a data stream, the DSM first interprets the tags attached to the data stream to obtain its QoS and security requirements. Depending on the QoS requirements carried in the data tags, the DSM splits incoming data into sub-streams. The QoS requirements of a sub-stream define the priority rank of that sub-stream. A data stream with the highest

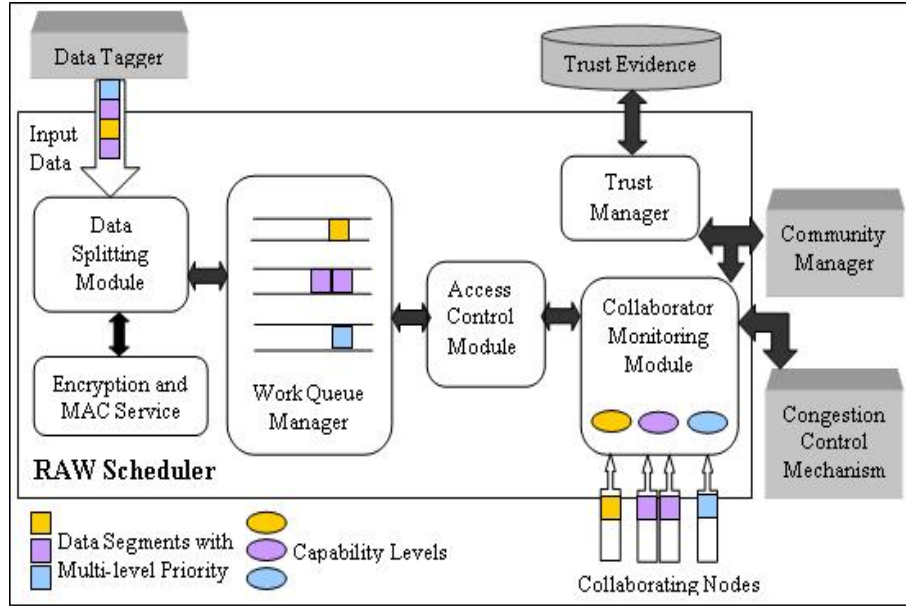


Figure 4.3: RAW Scheduler Architecture

bandwidth and loss tolerance requirements will get the highest priority. Moreover, the DSM uses security tags to identify any sensitive data that requires encryption and/or MAC to ensure its confidentiality and/or integrity, respectively. For any such sensitive data, the DSM passes the data to the Encryption and MAC Service where the data is encrypted and/or MACed. The DSM then sends these QoS priority ranked, partially encrypted, sub-streams to the multi-level WQs. These WQs buffer the highest priority data in the highest level WQ and the lowest priority data in the lowest WQ, and so on. WQM monitors the state of the WQs, e.g. the length of each queue. If a WQ has its buffer full, then WQM tries to send its associated data to the next available (lower level) WQ. In this way, the chances of congesting a particular WQ can be minimised. In other words, WQM applies some extra constraints on the data allocation to the WQs. The TM evaluates the trust values of the collaborators based on their previous QoS performances. The CMM uses these trust values to compute the QoS capabilities of the collaborators. It then classifies them into QoS capability levels, each corresponding to one WQ.

The assessment of the QoS requirements of an application and calculation of the QoS capability levels of collaborators are carried out independently. The ACM defines access permissions that matches the data priority levels with the

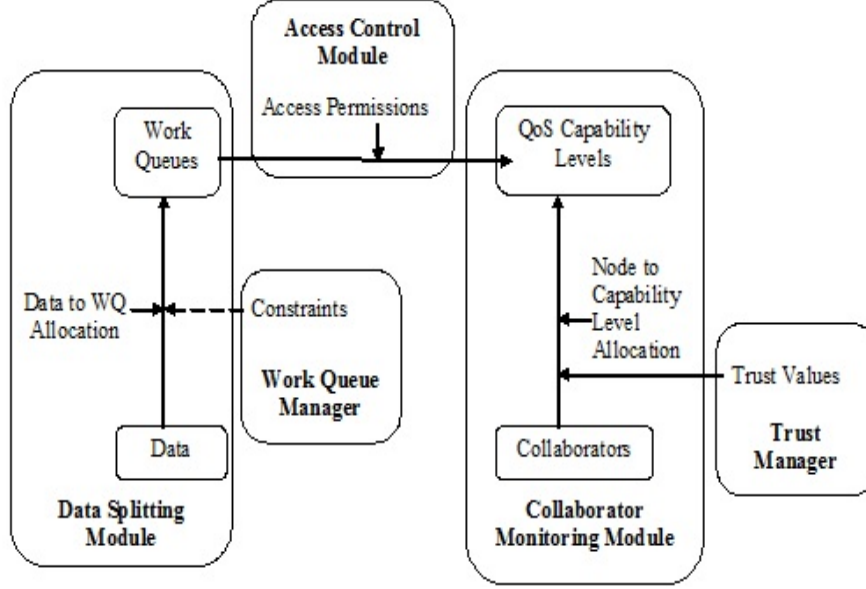


Figure 4.4: CAPability-aware Access Control (CAPAC) Components

collaborator's QoS capability levels. Figure 4.4 presents the contribution of different modules in the CAPAC model employed by the RAW Scheduler. The CMM coordinates the communication between the RAW scheduler and the Community Manager, and monitors the long-term changes in the channel conditions, i.e. node departure and channel failure. It also helps TM in assigning the trust certificates/ratings to the collaborators after each collaborative session. The next section presents the detailed working of each architectural component.

## 4.5 RAW Scheduler In Depth

This section describes the workings of the RAW scheduler's architectural components in detail.

### 4.5.1 Data Splitting Module (DSM)

The DSM splits an application's data stream into multiple sub-streams, according to the application's QoS and security requirements. The RAW scheduler uses two conventional metrics to capture the application level QoS requirements, these are bandwidth and loss tolerance. It recognises three application level data security requirements: confidentiality, integrity and availability. The RAW scheduler

supports the availability requirement by integrating it with the conventional QoS parameters described above. Hence, an application may only specify confidentiality and integrity as security parameters. An application might have multiple streams, each having different, possibly conflicting requirements. Consider the scenario of an emergency health care application. The application may require to secure patients personal identification data but may not be concerned about the security of a data stream carrying the pulse information or a video stream used for correspondence with doctors at a distant location. However, the stream carrying pulse rate may impose a lower data loss rate than the video stream.

We propose to use tags to allow an application to attach its requirements to its data streams. Our proposed tags are similar to the tick-tags [49]. These tags are stream centric, i.e. they are defined for the data streams and are interleaved with the data inside the streams. In other words, these tags are not maintained separately from the data they refer to. A tag uniquely identifies the streaming data objects to which it refers. In the data model used for our tagging framework, a data stream  $S$  is considered as a sequence of packets that arrive over the time. An application can attach a tag  $t$  to a streaming object  $O$ , where  $O$  can be an entire data stream  $S$  or an individual packet  $P$  or a group of packets  $G$ . Moreover, an object  $O$  can have multiple tags attached to it. A tag has the following format:

$t : \{tid, sid, applicability, type, content\}$

- Here,  $tid$  uniquely identifies the tag, and  $sid$  is the stream identifier that allows our scheduler to support multiple streams simultaneously.
- *Applicability* defines the granularity of a streaming object to which the tag refers. It has three values: 0 to represent an entire stream, 1 to represent one packet and a number  $n > 1$  to represent a group of  $n$  packets. The more fine-grained tagging generates more tags for a stream, hence, incurs higher tag processing cost.
- The *Type* attribute of this tagging schema classifies a tag as QoS or security tag. At the moment, we specify two values for the *type* field: 0 means it is a security tag and 1 means it is a QoS tag. This design is flexible to accommodate new tag types.
- The value of *content* depends on the *type* of the tag. For example, if it is a security tag then the value of *content* is an array: {confidentiality,



integrity}, where each element can have value either 1 or 0. This is to represent whether that particular security aspect is required or not. However, if it is a QoS tag the value of *content* is an array: {bandwidth, loss tolerance} where each element can have value either H, M or L to represent High, Medium or Low requirement for that QoS aspect, respectively. It is worth noting that the *content* can be a string, number, or an array and *content* array can be of any length.

This tagging method has a number of advantages.

- The tags can be shared by several streaming objects. This can be done by specifying an appropriate *applicability* value. It can reduce the memory and processing overhead.
- The tags interleave with streaming data, facilitating a faster search for the objects they refer to. The data splitting algorithm proceeds in a sequential manner when processing a tag and its associated data.
- These tags are as dynamic as the streaming data. They are infinite, arrive on-time, stay in the system for a finite time and later get discarded.
- Furthermore, the frequency and structure of these tags are independent of the data they refer to. If an application does not tag its data streams then the data streams will not be processed for any specific QoS and security support. However, the RAW scheduler tries to support the best-effort QoS by maximising the utilisation of the high bandwidth and loss tolerant channels.

As shown in Figure 4.5, the Data Tagger embeds tags into the original data streams. These tags precede the data they refer to. The DSM processes the stream in a sequential order. It first processes a tag and then its associated data and so on. Based on the values of security tags for confidentiality and integrity it identifies the data that requires security processing. The DSM then uses the Encryption and MAC Service to encrypt and/or append MAC to any data identified.

The DSM also splits data into sub-streams based on the QoS requirements, namely the required bandwidth and data loss rate. We have reversed the data loss rate to get a loss tolerance value to make it a measure of goodness. It makes

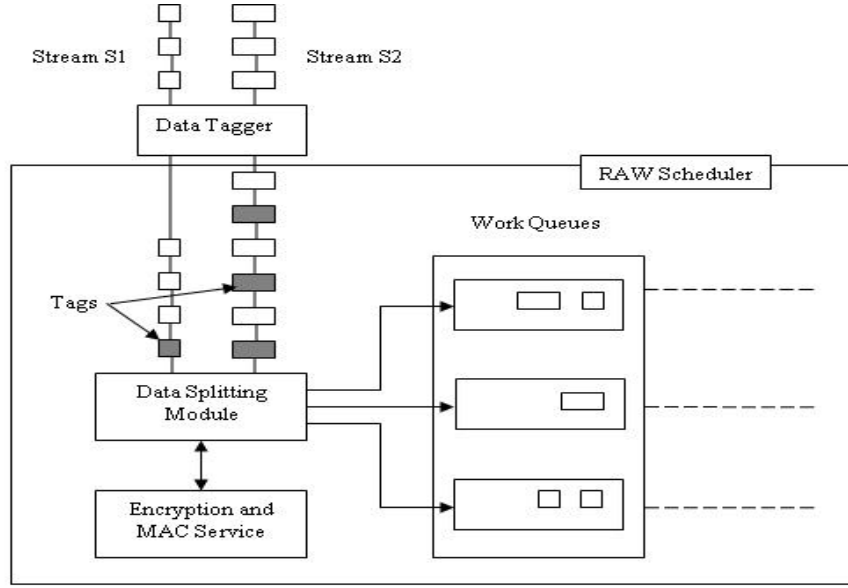


Figure 4.5: Data Tagging and Splitting

the tag design consistent. Each of the two parameters can have three possible values, i.e. High, Medium and Low. The DSM ranks these sub-streams according to the QoS parameter values. The sub-stream with the highest QoS requirements will be assigned to the highest QoS priority level. The DSM then feeds these partially encrypted and MACed, QoS priority ranked sub-streams into the WQs of corresponding levels.

#### 4.5.2 Encryption and MAC Service

The Encryption and MAC Service is a library of light-weight encryption algorithms, e.g. SRMT [50] and MAC algorithms. Any encryption and MAC algorithm can be supported, plugged in or deleted from the Encryption and MAC Service. If an incoming streaming object has its security tag value(s) set, the DSM calls this service for security processing (encryption and/or MACing). Instead of applying the computationally expensive data security techniques to the entire data stream, the RAW scheduler supports selective encryption and MACing of the data at several granularity levels: a single or a group of packet(s). The selective security processing of data makes the solution light-weight [51] [52] [53] [54]. Applications can always specify a requirement for applying these security techniques on the entire stream(s).

### 4.5.3 Work Queues Manager (WQM)

The WQM executes the novel MWQ algorithm. In proposed multi-level WQs, each WQ has a distinctive index, corresponding to the priority levels of multiple sub-streams generated by the DSM. The placement of the sub-streams onto the WQs is pretty straight forward and is expressed by the equation (4.1). This equation places the sub-stream with the highest priority level (most strict QoS requirements) in the WQ with the highest QoS index.

To further optimize scheduling, the WQM considers other factors as well, e.g. lengths of queues, the number of collaborating nodes serving a WQ, etc. WQM monitors the status of each WQ. If a WQ is full then it tries to schedule the associated data onto a WQ at a lower index. The WQ index determines the QoS capability levels of the collaborators that can access it. The CMM uses the trust values to determine the trustworthiness of the QoS capability claims made by the collaborators. The Trust Manager evaluates these trust values of the collaborators.

### 4.5.4 Trust Manager (TM)

The ad-hoc and open environment of the MCC and resource constraints in mobile devices make the design of TM a challenging task as compared to other Internet related solutions. For example, the Internet relies on a central, fixed and always available infrastructure of certification-authority and directory servers for trust management [55]. However, such a fixed trust infrastructure is not feasible in MCC, that suggests a decentralized trust infrastructure. Furthermore, Internet trust relations are generally long-lived and stable, therefore they do not need to be evaluated frequently [55]. In contrast, the MCC trust relations are typically short lived due to the node mobility. Therefore, the evaluation of trust in MCC should be on the fly, and this process must be short and fast. Most importantly, trust establishment may need to be performed with a subset of trust relations, i.e. with incomplete trust evidence, since there may be times where very few nodes are reachable in a MCC.

Based on the design considerations discussed above, we propose a decentralized and reputation based design of the TM. The TM running on an initiator  $i$  (i.e. trustor) uses the following four factors to estimate a trust value  $T_{ij}$  for a collaborator  $j$  (i.e. trustee).

1. Direct Trust (DT): i's previous direct experiences with the trustee ( $DT_{ij}$ )
2. Indirect Trust (IT): j's reputation in the MCC ( $IT_{ij}$ )
3. Direct Contribution (DC): j's direct contribution for the truster ( $DC_{ij}$ )
4. Indirect Contribution (IC): j's contribution for other MCC members ( $IC_{ij}$ )

Hence, the value of  $T_{ij}$  is calculated using the following equation:

$$T_{ij} = f(DT_{ij}, IT_{ij}, DC_{ij}, IC_{ij}) \quad (4.2)$$

TM uses weights to govern the relative significance of the four factors. It is important to note that the TM evaluates the trust values for bandwidth and loss tolerance of a collaborator separately. We are not representing the trust equations for both QoS trust ratings separately in this section, as formulas are essentially the same.

**Direct Trust (DT):** The DT represents the initiator's belief on the collaborator's QoS capacities based on the past collaborations. Given  $\alpha$  and  $\beta$ , the DT can be expressed as [56]:

$$DT_{ij} = \frac{1 - \alpha^n}{\beta} \quad (4.3)$$

Where,  $n$  is the number of node i's satisfied transactions with the node j (the meaning of satisfied transaction is defined later in this section).  $\alpha$  is the learning rate - a real number in the interval  $[0,1]$ . The smaller the  $\alpha$ , the faster the trust value grows.  $\beta$  is the punishment coefficient of an unsatisfied transaction. For a satisfied transaction, the  $\beta$  value is equal to 1 and obviously for an unsatisfied transaction  $\beta > 1$ . This coefficient makes the descending rate faster than the ascending rate for the trust value [57]. This reflects human behaviour of distrust, where it takes much more time/transactions to gain trust than to lose it. The value of  $DT_{ij}$  is in the range of  $[0,1]$ . The value of  $n$  starts from 0 to reflect zero level of trust when there is no prior interaction between the pair of nodes. As the number of satisfied transactions increase, the DT value approaches to 1.

**Indirect Trust (IT):** In the dynamic and open environment of MCC, an initiator may encounter an unknown collaborator. In such cases, the direct trust between the two nodes is zero. The initiator has to estimate the collaborator's

trustworthiness and one way of doing this is through the recommendations from other nodes. Trust built on the recommendations is termed as indirect trust and it can be calculated using Equation(4.4) [56]:

$$IT_{ij} = \frac{\sum_{t=1}^k (DT_{it} * DT_{tj})}{k} \quad (4.4)$$

where,  $k$  is the minimum number of recommendations required to calculate a significant IT value, and this number is determined by the initiator. If recommendations are more than  $k$ , then initiator only uses the top  $k$  recommendations. If the number of recommendations are less than  $k$ , then the division is still by  $k$ , resulting in a lower indirect trust level.  $DT_{it}$  denotes the direct trust value node  $i$  gives to the node  $t$  (recommending node), and  $DT_{tj}$  direct trust value node  $t$  gives to the node  $j$ . Since the direct trust values  $DT_{it}$  and  $DT_{tj}$  are in the range of  $[0,1]$ , the indirect trust of node  $i$  in node  $j$  (based on the node  $t$ 's recommendations) is also in the range of  $[0,1]$  and is always less than  $DT_{it}$  or  $DT_{tj}$ .

It is important to note that the recommendations involve only a single level of indirection, i.e. they must come from the nodes who have direct interactions with both node  $i$  and node  $j$ . The indirect trust value can be derived via a recommendation path, involving two or more nodes [58]. However, such recommendation path is not favoured in our TM design as its implementation is expensive and it is complex to derive a very insignificant trust value.

**Direct Contribution (DC):**  $DC_{ij}$  denotes the total amount of data (in megabytes) that node  $j$  has transferred for node  $i$ . For the derivation of an overall trust value in the range to  $[0,1]$ , we need to normalize  $DC_{ij}$  by scaling it between 0 and 1. To do so, assume that there are  $DC$  values stored in  $i$ 's trust database. Here,  $DC = (DC_{ij}, \dots, DC_{in})$  and  $n$  is the total number of collaborators who had direct experience with  $i$ . The normalized value of  $DC_{ij}$  for a collaborator  $j$  can be calculated using Equation(4.5):

$$Normalized(DC_{ij}) = \frac{DC_{ij} - \min(DC)}{\max(DC) - \min(DC)} \quad (4.5)$$

where,

$\min(DC)$  = the minimum value for  $DC$

$\max(DC)$  = the maximum value for  $DC$

If  $\max(DC)$  is equal to  $\min(DC)$  then  $Normalized(DC_{ij})$  is set to 0.5.

**Indirect Contribution (IC):** IC indicates the contribution that a collaborator has made for other nodes in the MCC. The formula to evaluate the IC score of a node  $j$  from node  $i$ 's point of view is as follows [56]:

$$IC_{ij} = \sum_{t=1}^k (DT_{it} * DC_{tj}) \quad (4.6)$$

Where,

$DT_{it}$  is the direct trust value node  $i$  gives to the node  $t$

$DC_{tj}$  is the direct contribution value of node  $j$  towards the node  $t$

$k$  is the number of recommending nodes.

Instead of calculating the contribution that a node has made to all the MCC members, the above formula reflects the view that is realistic for the initiator. Initiator is interested in knowing the contribution of the collaborator to the nodes the it knows and trusts. Weighing the recommendation score from a node with its DT value helps to neutralize the effect of colluding parties, where a group of nodes lie about their contributions against each other.

**Trust Evidence Management:** The TM evaluates the direct experience and direct contribution of a node  $j$ , i.e.  $DT_{ij}$  and  $DC_{ij}$ , from the trust evidence stored on the trustor,  $i$ , itself in its local trust database. To compute the reputation values, i.e.  $IT_{ij}$  and  $IC_{ij}$ , we suggest that each node carries its *portfolio of credentials* [59] earned from the past interactions with any of the MCC members. A collaborator provides this trust evidence (certificates) to the initiator prior to a collaborative session. In this way, the collaborators can prove their trustworthiness, without the presence of a recommending node. Hence, there is no need to propagate reputation information across the community. This reduces the communication costs introduced by the trust evaluation process. To further reduce the communication and processing overheads, we suggest that an initiator publishes a list of recommenders trusted by it, during the community formation process. The collaborators are required to send only trust certificates signed by the recommenders in the list. Although, it may reduce the quantity of trust evidence, the overall quality of evidence is improved.

**Transaction Rating:** After the completion of a collaborative session, the initiator has to issue a new trust rating certificate to each of the participating collaborators. Initiator also updates its DT and DC scores in its local trust database based on the transaction satisfaction level. In our design, there are three transaction satisfaction levels: satisfied, fair and unsatisfied. The two threshold values (for bandwidth and loss tolerance separately) are configured for each collaborative session and are used to define the transaction satisfaction levels. These threshold values are:  $th(H)$  and  $th(L)$ , representing the threshold for high QoS and low QoS, respectively. A transaction is regarded as satisfactory for a certain QoS aspect if that QoS aspect's average performance is greater than the  $th(H)$  value (configured for that particular QoS aspect). In other words, a transaction is regarded as satisfactory in terms of bandwidth if the average data transfer speed has reached the  $th(H)$  level configured for bandwidth and the same is true for the data loss rate. The transaction is treated as fair where the QoS performance falls below the  $th(H)$  level but stays above the  $th(L)$ . Below  $th(L)$  level, the transaction is regarded as unsatisfactory for that QoS aspect. As mentioned earlier, we are not representing trust equations for both QoS aspects separately, as formulas are essentially the same. Hence, the formulas to update DT based on the equation (4.3) are:

- Satisfied:  $DT_{ij} = \frac{1-\alpha^{n+1}}{1}$
- Fair:  $DT_{ij}$  unchanged
- Unsatisfied:  $DT_{ij} = \frac{1-\alpha^{n-1}}{\beta}$

For a satisfied transaction, the  $\beta$  value is equal to 1 while for an unsatisfied transaction the  $\beta$  value is  $> 1$ . This coefficient makes the descending rate faster than the ascending rate for the trust value [57]. Moreover, the malicious nodes are added to a black list. In this way, nodes making false claims about their QoS and/or showing malicious behaviour, get their trust rating reduced.

#### 4.5.5 Access Control Module (ACM)

As we have discussed earlier, the TM evaluates the QoS trust values of each collaborators. These trust values are used as weights to compute the more realistic QoS values, from the QoS values offered by the collaborator. Based on these

realistic QoS value of each collaborator, it is placed in a QoS capability level. Each capability level is treated as an access right group. The ACM defines access permissions for each of the right groups, instead of defining them for each collaborator. Figure 4.6 represents the process of capability level assignment of a collaborator, carried out by the CMM and TM during the community formation phase. The capability level of a member remains the same during a collaborative session.

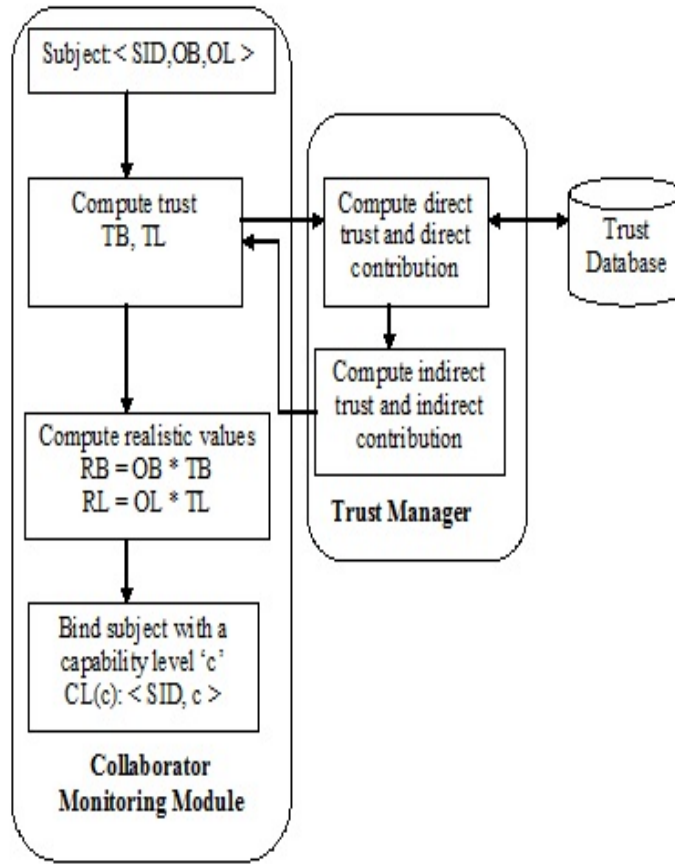


Figure 4.6: Process of Capability Level Assignment

Figure 4.6 shows that a collaborator, i.e. subject, provides CMM with offered bandwidth and loss tolerance values, represented by the OB and OL in figure 4.6. The CMM then asks TM to compute trust values for that subject (for each QoS parameters, i.e. bandwidth and loss tolerance) using its QoS performance history. Based on the returned trust values, i.e. TB and TL, and QoS values offered by the subject itself, i.e. OB and OL, CMM calculates the realistic values for the



bandwidth and loss tolerance, i.e. RB and RL, for that node. Using the QoS threshold values  $th(H)$  and  $th(L)$ , defined in previous section, node with RB and RL will be assigned to a QoS capability level. As mentioned earlier, each capability level represents a right group, hence CMM binds that collaborator with a right group with QoS capability level index 'C'. In this thesis, three QoS capability levels are defined as proof.

- High represented by index 1
- Medium represented by index 2
- Low represented by index 3

The collaborators with a capability index 1 offer the highest level of QoS. ACM considers WQ's as objects that need to be protected and collaborator groups are subjects who should be granted proper access rights. ACM defines the access scope for each group in terms of the highest level of WQ that can be accessed by the nodes in that group. A group needs to have its capability level value (C) equal to or greater than the corresponding index of the WQ. Hence, the nodes that are allowed to serve a WQ must have a minimum capability level of  $C_i \geq WQ_k$ , where  $i = k, k + 1, \dots, n$ . The ACM also defines the priority order in which different WQs should be accessed.

Capability Level	Access Privileges
<b>High</b>	$WQ_1$
	$WQ_2$
	$WQ_3$
<b>Medium</b>	$WQ_2$
	$WQ_3$
<b>Low</b>	$WQ_3$

Table 4.1: Access Control Policy for CAPAC.

Table 4.2 shows the proposed access control policy for a MWQ with three levels. As shown in the table, access control rules restrict the low capability nodes to get data from the  $WQ_3$  only. The medium capability nodes can access both  $WQ_3$  and  $WQ_2$  while the high capability nodes can access all three WQs. The order of access is from top to bottom, i.e. a node will first serve the WQ of highest index that is accessible. If this WQ is empty then it will serve the WQ at a lower level and so on. This access control policy reduces the chances of allocation of the data with strict QoS requirements over the least capable (low QoS) nodes.

### 4.5.6 Collaborator Monitoring Module (CMM)

the CMM provides an interface between the RAW scheduler and the collaborators. It coordinates with the TM to calculate the capability levels of the collaborators. At the end of each session, it assigns trust certificates to the collaborators based on their QoS performance during a collaborative session. Most importantly it monitors the long-term changes in the channel conditions, e.g. unannounced node departure and channel failure, etc. and notify Community Manager, which inturn can add new nodes in the MCC, if required.

## 4.6 A Case Study: Scheduling MPEG Video Stream using RAW Scheduler

This section explains the scenario of risk aware scheduling of MPEG video [42] in the MCC. We have employed a modified version of the FPS algorithm (see details in section 3.5.1.2) with some preliminary assumptions. We assume that all participating nodes involved in the MPEG video transfer have already been classified into QoS capability levels by the CMM. For this example, we assume three QoS capability levels namely High (H), Medium (M), Low (L). We also assume that the collaborative group is formed such that the aggregated bandwidth of bundled WWAN channels is sufficient for the desired MPEG video stream transmission.

Our proposed algorithm splits a MPEG video stream into multiple sub-streams and assigns them to the MWQs based on the importance of the frames. Access privileges are defined to restrict the scheduling of important frames over the most trusted nodes.

### 4.6.1 Structure of MPEG Video Stream

The MPEG-1 standard [60] is a coding format for the audio and video streams. We are considering the video streams [42] for this case study. A MPEG video frame typically is composed of 20 to 60 packets. The loss of one packet leads to the loss of a complete frame. Depending on the type of frame, this loss leads to the perceptible artifacts. For example, some coloured squares (blocking) might appear in the video, different scenes overlap, videos may become completely unrecognisable, or the movement of objects in the video is irregular, etc. The main

feature of the MPEG video is that frames are no longer independent of each other as is the case with Motion-JPEG, which is a series of single JPEG [61] images.

A MPEG video sequence consists of three different frame types:

1. Intra-coded pictures i.e. *I*-frames
2. Predictive coded pictures i.e. *P*-frames
3. Bi-directionally coded pictures i.e. *B*-frames

The frames from one *I*-frame up to the frame before the next *I*-frame form a so called the Group of Pictures (GoP). Since, the *I*-frames are encoded similarly to the JPEG images, their decoding is independent of other frames. The decoding of the *P*-frames depends on the preceding *I* or *P* frames of the same GoP. The decoding of the *B*-frames depends on both the preceding and the succeeding *I* and *P* frames [62].



Figure 4.7: Typical Structure of a Group-of-Pictures (GoP) in a MPEG Video Stream. Reproduced from [3].

Figure 4.7 shows a sequence of MPEG frames and their inter-dependencies which are relevant for decoding the stream. The arrows show the dependencies between the frames. For example, correct decoding of  $B_2$  is not possible without the presence of  $I_0$  and  $P_3$ . Similarly, the whole GOP cannot be decoded properly if the corresponding I-frame is not present.

In terms of user-perceived quality, the loss of an *I* or *P* frame causes visible artifacts to appear during the rendering of the dependent frames (see Figure 4.8) while the loss of a *B*-frame does not influence the quality of any other frames much. This makes the loss of *B*-frames preferable to the loss of *P* and *I* frames [3].

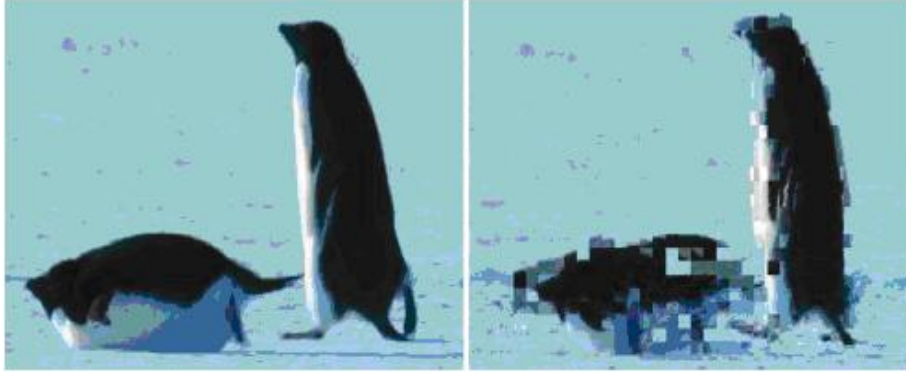


Figure 4.8: MPEG Video’s *B*-frames in Presence (left) and Absence (right) of Base-frames. Reproduced from [3].

#### 4.6.2 Adaptive Video Streaming

The multimedia streaming over the wireless networks experience high channel error rate and variable channel bandwidth. Hence, it is very important to adapt the transmission rate to the variable network conditions. There are two different approaches to perform transmission rate adaptation. The first approach is to adapt the source rate to match the channel rate. This approach often incurs a large delay on the adaptation as the channel state information needs to be sent back to the multimedia server and multimedia server has to wait for the next clear point (e.g. an *I*-frame in case of MPEG) in order to switch to a different rate. The second approach is to prioritize the packet transmission and selectively drop packets at the wireless last hop [3][63]. Packets that are the least important in terms of reducing overall distortion should be dropped first. In practical network deployments, these two approaches can complement each other. Our work favours the second approach.

Kozlov et al. [62] built a scheduling mechanism for the MPEG video streaming over a single WWAN channel. Their algorithm favours the transmission of more important MPEG frames at the expense of the less important ones. They use Medium Access Control (MAC) of the 802.11 standard to do instant detection of the bandwidth fluctuations by observing the transmission rate from the sending buffer. The decision on whether to drop and how many frames need to be dropped is based on the deadline, size, and importance [63]. Sharma et al. [18] have used FPS algorithm to split the MPEG video data based on the frame type. It separates a MPEG video stream into three sub-streams ( $sf_I, sf_P, sf_B$ ) based

on the three frame types. The wireless transmission channels are then allocated according to the sub-stream priority, i.e. I-frame sub-flow ( $sf_I$ ) is sent over the most reliable channels, and so on. This allocation also adapts to the changes in the channels conditions, which in this case is channel's loss rate. We have utilized the same concept of FPS for MPEG video splitting with the difference that we utilized our light weight MWQ algorithm to schedule these sub-streams onto most reliable channels. Our proposed CAPAC ensures that the most important frames are scheduled over the channels offering best QoS.

### 4.6.3 Risk Aware MPEG Scheduling

As discussed earlier, the MPEG video consists of three different frame types,  $I$ ,  $B$ , and  $P$  frames. The  $I$  frames are independent of other frames. The decoding of  $P$  frames depends on the preceding  $I$  or  $P$  frames, and correct decoding of  $B$  frames is not possible without presence of base frames i.e.  $I$  and  $P$  frames. The whole GoP cannot be decoded properly if the corresponding  $I$  frame is not present. Hence, the order of frame importance is  $I > P > B$ .

We are using three level WQs corresponding to each frame type (i.e.  $I$ -Queue,  $P$ -Queue and  $B$ -Queue). The DSM have employed an algorithm that schedules MPEG frames to the corresponding work queues, i.e.  $I$  frames to the  $I$ -Queue,  $P$  frames to the  $P$ -Queue and  $B$  frames to the  $B$ -Queue. In this way, the WQ at the highest level, i.e.  $I$ -queue accommodates the most important data, i.e. the  $I$ -frames. Figure 4.9 presents the flow chart of the MPEG frame splitting algorithm. Algorithm 1 presents the pseudo code of the algorithm.

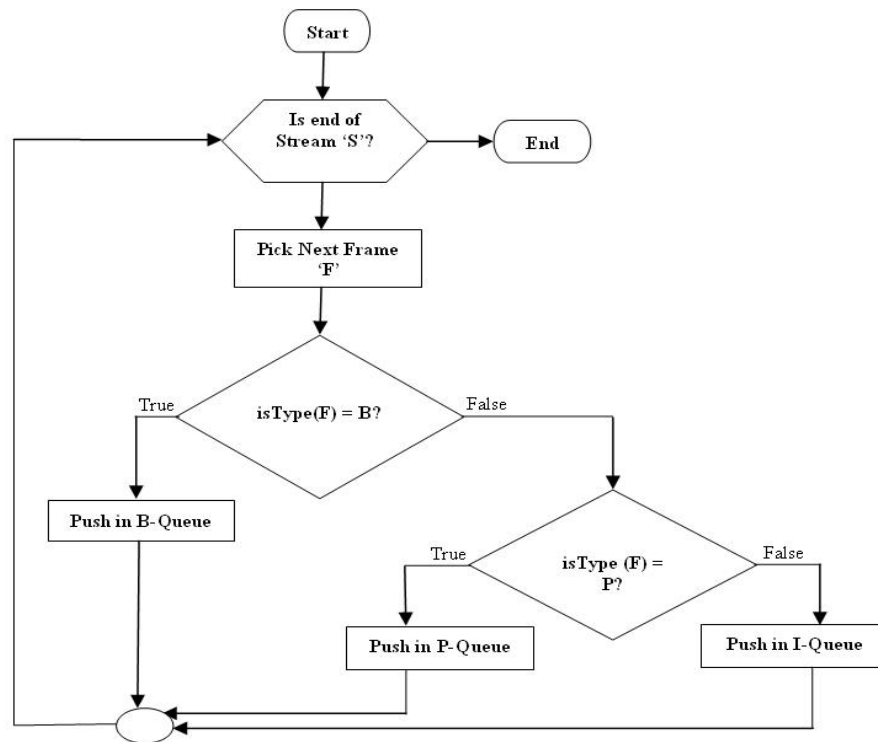


Figure 4.9: Flowchart of MPEG Video Splitting Algorithm

---

<b>Algorithm 1:</b> Assign $S \Rightarrow WQ$
<b>Require:</b> $S \wedge WQ$
<b>Ensure:</b> $S \Rightarrow WQ$
$WQ.initialize(3)$ {initialize a 3 level WQ}
<b>while</b> $S.nextFrameAvailable()$ <b>do</b>
$F \Leftarrow S.nextFrame()$
<b>if</b> $F.isType = B$ <b>then</b>
$WQ_B.push(F)$ {push frame in B-queue}
<b>else if</b> $F.isType = P$ <b>then</b>
$WQ_P.push(F)$ {push frame in P-queue}
<b>else</b>
$WQ_I.push(F)$ {push frame in I-queue}
<b>end if</b>
<b>end while</b>

---

The CMM evaluates the QoS capability value of each collaborating node and classify it as a High (H), Medium (M) or Low (L) capability node. The ACM defines the access scope for each capability level.

(i) Trust Level	(ii) QoS Capability Level	Work Queues
Trusted	High	I-Queue P-Queue B-Queue
Semi Trusted	Medium	P-Queue B-Queue
Untrusted	Low	B-Queue

Table 4.2: Access Control Policies for Risk Aware MPEG Scheduling to Maximise (i) Security and (ii) QoS.

Table 4.2 shows the proposed access control policy for the WQs hosting MPEG video frames. It restricts the Low capability nodes to get data from the

*B*-Queues. The nodes belonging to the Medium capability level can access both the *B* and *P* queues while nodes belonging to the High capability level can access all WQs. The priority order of the WQ access is from top to bottom, i.e. a node will serve the WQ of highest accessible level first, if it is empty then it can serve the node at the lower level and so on. In this way, the chances of base frames (*I* and *P*) allocation over the low QoS nodes are reduced.

Interestingly same scheduling technique is useful for the security purpose. If CMM ranks the collaborators into ‘Trusted’, ‘Semi Trusted’ or ‘UnTrusted’ (in terms of security) then the Untrusted nodes are not able to spoof videos as we are not sending the base frames on them. The WQ access rules are the same (shown in Table 4.2) and can be used to ensure that the base frames get scheduled over the Trusted nodes. Hence, if least trusted nodes try to render dependent frames transferred through them, visible artifacts appear in the video (see Figure 4.8). We can apply data encryption techniques on the dependent frames to make video completely unrecognisable.

## 4.7 Evaluation Methodology

This section discusses four popular evaluation methodologies for the QoS performance evaluation in the network research. These methodologies are (i) modelling, (ii) test-bed experiments, (iii) emulation and (iv) simulation. A hybrid approach of using simulation and test-bed experiment methodology is found to best suit our evaluation requirements, and justifications for this choice are provided.

The purpose of our QoS evaluation is to determine the ability of our proposed solutions to provide the QoS functions in the MCC. However, experiments must be repeatable and assessable to guarantee the reproducibility of the results [64]. Vitek et al. [65] in their research highlighted a few structural factors inhibiting quality research. They suggest that *“Important results in systems research should be repeatable, they should be reproduced, and their evaluation should be carried with adequate rigour”* [65]. For the purpose of the following discussion, we make a distinction between the repeatability and reproducibility. Repeatability is the ability to repeat experiments in a consistent manner. Reproducibility is the ability to reproduce the results derived from the previous experiments, typically performed by other researchers [64].



### 4.7.1 Modelling

Evaluation using modelling is valuable when a controlled test-bed experiment is not feasible. Mathematical models can capture many aspects of a network and network performance. However, creating a mathematical model for a large-scale dynamic network like MCC is complex. This is because of the difficulty to model the network characteristics, such as node mobility, wireless channel characteristics, etc. We can develop mathematical models to represent an appropriate level of complexity, however, it will lead to a smaller set of findings. This limitation reduces the usefulness of the findings. Furthermore, the degree of rigour in model development and assessment can vary greatly, and existing beliefs might influence the model assumptions and results. Hence, evaluation via mathematical modelling is ruled-out for use in this thesis.

### 4.7.2 Test-bed Experiments

The test-bed experiments evaluate the performance of new network protocols and architectures on full-scale physical networks. The major benefit of generating experimental data from a test-bed is that these data are based on the real-world conditions. Hence, results are more closer to those which are likely to be experienced during a general real-world deployment.

However, the real-world characteristic of test-beds introduces a number of drawbacks. First, they are affected by stochastic factors e.g., radio environment and node mobility that make them hard to reproduce results [66]. To reproduce the results, these factors need to be controlled but this can limit the design of test-beds [64]. Second, test-beds are expensive to construct. The costs include the cost of equipment, and the time cost of the participating people. The experimental evaluation methodology, no doubt is the most accurate way of performance evaluation, that is why we used test-bed experiments for the detailed evaluation of our proposed scheduler despite of time, cost and equipment constraints.

### 4.7.3 Emulation

While modelling and simulation simplify some parts of a real environment to understand the impact of other factors. Emulation go half way between the simulation and test-beds, by modelling some parts and running other parts in

the real world. [64] They try to address the problems of scaling and repeatability by emulating the wireless channel and the mobility.

However, many emulator test-beds require special hardware components. Therefore it is difficult for other researchers to reproduce and verify results [64]. Emulator test-beds can not always be a substitute for real world experiments [64].

#### **4.7.4 Simulation**

Simulation is one of the most widely used evaluation methodologies in the area of computer networks. It allows to model a computer network by specifying the behaviour of network nodes and communication channels [67].

Simulation is economical because it can carry out experiments without the actual hardware. Also, it is easy to control stochastic factors (e.g., radio environment and node mobility) and scalability. Hence it is easy to repeat experiments and reproduce results [68] [64]. Moreover, simulation is flexible that enables the evaluation of new communication protocols and network architectures under various conditions [68]. Using simulation-based evaluation as a first-step can significantly contribute to the success of experimental testing later [66]. Hence, simulation is chosen as the first-step evaluation methodology for this thesis.

#### **4.7.5 Our Hybrid Approach**

A hybrid approach of using the simulation and test-bed methodology is found to best suit our evaluation requirements. As simulation is an economical and easy way for making initial assessment of new network protocols and algorithms. Firstly, the simulation was used to evaluate the QoS performance gain of our proposed MWQ scheduling algorithm as compared to existing state-of-the-art scheduling solutions in a controlled environment. The NCTUns [21] simulation tool was used for the initial evaluation of the work given its built-in support for the multi-interface node (details in Chapter 5). Before conducting the evaluation, simulation model was designed carefully. To validate the simulation model we used validation scripts that come with the simulation tool, i.e. NCTUns. The simulator output was then compared with the reference output to validate if the simulator has been correctly configured. After being certain that the simulation model represents our solution specification exactly and accurately, the simulations were run and results were obtained. The results of this initial evaluation helped

us to identify the best way to achieve better QoS in a MCC environment, and conduct more detailed evaluation of our RAW scheduler. However, a software bug in the simulator restricted the expansion of the MCC network (see Appendix B.1). We could only evaluate limited scenarios by running simulation experiments.

In stage two evaluation, we implemented a test-bed application, an "HTTP Downloader", that is built using the RAW scheduler design to conduct experiments in a real life environment (networks and devices). The "HTTP Downloader" application downloads data from the Internet by using WWAN channels of multiple mobile phones simultaneously. The implementation details are discussed (for details, refer to Chapter 5), and the selections of, and justifications for, the selections of the tools and technologies are given. Core components of RAW scheduler were implemented and tested to see if the MWQ algorithm still performs better. The QoS performance of the RAW scheduler was tested for long-term as well as short term fluctuations in channel conditions. The results obtained were analysed against our research claims and conclusions are made. A detailed analysis of the proposed solutions was performed on the basis of the simulation as well as experimental results.

## 4.8 Chapter Summary

This chapter has presented the building blocks of RAW scheduler and has discussed how it is evaluated. The building blocks of RAW scheduler are the DSM, Encryption and MAC Service, WQM, TM, ACM and CMM. A case study of MPEG video scheduling has been used to demonstrate the working of proposed RAW Scheduler. A hybrid approach of using the simulation and test-bed experiments is introduced in last section. Simulation was selected as the first-step investigation methodology, and the NCTUns was chosen as the simulator. For detailed performance evaluation, test-bed experiments were selected, details of which can be found in the next chapter.

# Chapter 5

## The RAW Scheduler: Evaluation

### 5.1 Chapter Introduction

This chapter presents the evaluation of the novel RAW scheduler and its core components namely the MWQ scheduling algorithm, the CAPAC module and the CMM. The main focus of this novel scheduler is to support the best-effort QoS provisioning for multi-stream data in a processing efficient way. The algorithm achieves this by (1) ranking the data into multiple priority queues based on the application-level requirements, (2) create and maintain an ad-hoc community of collaborating mobile devices, (3) rank collaborators according to their QoS capability levels, (4) allowing collaborators to request for a work-item only when ready to transfer, instead of assigning them the data by the scheduler, (5) making sure that only the collaborators with high QoS capability can access the data in high priority queues.

The chapter is organised as follows. Section 5.2 presents the design of a MWQ scheduling algorithm. Section 5.3 presents the MWQ implementation in detail using pseudo-code. The three state-of-the art scheduling algorithms are implemented to compare the QoS performance of our proposed algorithm. These algorithms belongs to different classes of scheduling algorithms already being discussed in chapter 3. The section 5.4 presents a simulation-based performance evaluation of the MWQ algorithm. This section also discusses the major findings from the simulation. These findings helped us to conduct thorough test-bed experiments to evaluated the QoS performance of the RAW scheduler. Section 5.5 provides test-bed implementation details and the design of an example application. It also provides the details of experiments performed to measure the QoS

performance gains. Finally, Section 5.6 presents a summary of the chapter.

## 5.2 MWQ Algorithm Design

The design of MWQ algorithm consists of three parts: a sender entity, a receiver entity, and a data queue entity. The sender maintains a WQ of the fixed sized work-items. It splits the data to be transferred into work-items and place them into multiple WQs. These WQs are then accessed by the collaborators who are willing to fetch data for the receiver. The distinctive features of the MWQ algorithm are explained below.

1. **Priority Data Queues:** Multiple data queues are maintained, where each queue has a priority index/level. Data from multiple streams can be split and assigned to these priority data queues according to their QoS or security requirements. This feature will later help scheduling the most important data on the better QoS path or trusted nodes. This priority ranking supports the access control model of the RAW scheduler.
2. **Request-Assign Model:** The MWQ algorithm is energy efficient by design as it supports a Request-Assign model. In this model, the scheduler is not assigning the data on the available channels instead the collaborators are requesting for work-items. It reduces the communication and processing cost at the sender node hosting the workload scheduler. This model enables the fast nodes/channels to get more work automatically and also adapts to the short terms changes in channel conditions by design.
3. **Queue Access Order:** MWQ algorithm supports an access order to the WQs where all collaborators first request data from the highest priority WQ. If that WQ is empty only then they request for the data in WQ of a lower level and so on. This access order ensures that the highest priority data gets scheduled first. If aggregative bandwidth of the collaborating channels are not able to hold the incoming data stream then the data from the lowest queue gets delayed or dropped.

The following section describes the implementation details of the MWQ algorithm and our reference scheduling algorithms.

## 5.3 MWQ Algorithm Implementation

This section describes the implementation of RR, CP, SWQ and MWQ algorithms. As we have discussed earlier, inverse multiplexing is a simple concept used in data communication when a data stream is too large for a single transmission channel. The data is split into smaller fragments and the fragments are transmitted over separate transmission channels and are reassembled at the receiving end. These channels may have different QoS characteristics like bandwidth, latency, and loss rates. Once data have been fragmented, a scheduler on the multiplexer decides when and in what order to transmit these fragment on to the available links. To improve the performance scheduler strives to balance the fragments distribution. We present four of these scheduling algorithms here.

### 5.3.1 Round Robin (RR) Algorithm

The first is the RR approach that allocates the fragments amongst all the available links in an ordered fashion. Our implemented application creates multiple TCP connections with the receiver on each of its IP addresses (Interfaces) and sends data in a fixed order. The application creates dummy data packets of 1kb size and sends them to the receiver alternatively on each interface in RR fashion. For example it reads 1kb and sends it on interface-1 and then again reads 1kb and sends it to interface-2 and so on.

---

**Algorithm 2:** RR Scheduling Algorithm

---

**Input:** number of collaborates, Single Queue Q

**Output:** Average throughput, transfer time

**while**  $data\_sent \leq total\_data$  **do**

**for**  $i \leftarrow 1$  **to**  $number\_of\_collaborators$  **do**

$fragment \leftarrow$  read next chunk from Q;

$collaborator[i] \Rightarrow$  sends fragment;

$data\_sent += fragment$

---

### 5.3.2 Channel Pinned (CP) Algorithm

The CP scheduling approach pins a data stream to a selected channel. This scheduler sends fragments from a particular stream to the pinned channel only

and discarding all the fragments that the channel can not hold, even if other channels have spare capacity.

Our implemented application creates multiple TCP connections with the server. It creates multiple independent threads in it, one thread for each connection. Each thread use an independent queue to send the data.

---

**Algorithm 3:** CP Scheduling Algorithm

---

**Input:** number of collaborates  $N$ , multiple Streams  $\langle S_1, S_2, \dots, S_n \rangle$

**Output:** Average throughput, transfer time

```

foreach thread  $i \leftarrow 1$  to  $N$  do
    while  $data\_sent[i] \leq S[j]_{size}$  do
         $fragment \leftarrow$  read next chunk from  $S[j]$ ;
         $collaborator[i] \leftarrow$  sends fragment;
         $data\_sent[i] \leftarrow data\_sent[i] + fragment$ 

```

---

### 5.3.3 Single Work Queue (SWQ) Algorithm

In this approach, all collaborators gets their data fragments from a single data queue. Each collaborator checks the queue for data fragment (also termed as work-item) as soon as it successfully transferred the previous fragment.

---

**Algorithm 4:** SWQ Scheduling Algorithm

---

**Input:** number of collaborates  $N$ , single queue  $Q$

**Output:** Average throughput, transfer time

```

foreach thread  $i \leftarrow 1$  to  $N$  do
    while  $data\_sent \leq Q_{size}$  do
        if (mutual exclusion condition) then
             $fragment \leftarrow$  read next chunk from  $Q$ ;
             $collaborator[i] \leftarrow$  sends fragment;
             $data\_sent \leftarrow data\_sent + fragment$ 

```

---

### 5.3.4 Multi-level Work Queue (MWQ) Algorithm

The MWQ scheduling algorithm uses multiple data queues and prioritizes queues so that most important data can be assigned to the high priority queues and any free collaborators can read the data from the top priority queue first.

---

**Algorithm 5:** MWQ Scheduling Algorithm

---

**Input:** number of collaborates  $N$ , Multiple queues  $mQ_1, mQ_2, \dots, mQ_n$

**Output:** Average throughput, transfer time

```
for  $j \leftarrow 1$  to  $total\_queues$  do
    foreach thread  $i \leftarrow 1$  to  $N$  do
        while  $data\_sent[j] \leq mQ_{size}[j]$  do
            if (mutual exclusion condition) then
                 $fragment \leftarrow$  read next chunk from  $mQ_j$ ;
                collaborator[i]  $\leftarrow$  sends fragment;
                 $data\_sent[j] \leftarrow data\_sent[j] + fragment$ 
```

---

## 5.4 MWQ Algorithm Performance Evaluation using Simulation

This section analyses four different scheduling algorithms (RR, CP, SWQ and MWQ) to compare and contrast their QoS performance in a simulated environment and evaluate their strengths and weaknesses. The aim is to measure the performance gains of the MWQ algorithm against other algorithms and to find directions of improvement before conducting the detailed evaluation using real world test-bed experiments.

### 5.4.1 National Chiao Tung University network simulator (NCTUns)

The ns2 (network simulator) [69] is most widely used in the MCC research. The literature review phase revealed that most of the-state-of-art have used this simulator. More than 80% of the research papers cited in chapter 2 and 3 have used ns2 for quantitative evaluation. Hence, it was our first choice as a simulator. We installed, configured and programed it to simulate the underlying MCC network (shown in Figure 5.1).

However, we decided to switch to another simulator, i.e. NCTUns (National Chiao Tung University network simulator) [5] after a few months, due to following reasons:

- The ns-2 simulator does not support network tunnelling by design and its implementation involves changes in transport and network level protocols.



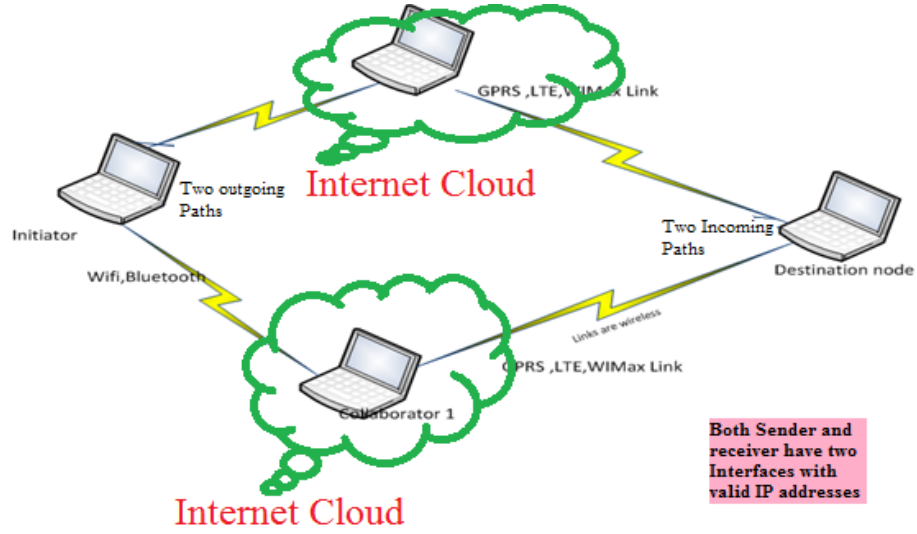


Figure 5.1: The Basic MCC Simulation Topology

On the other hand, NCTUns supports a multi-interface node that is ideal for simulating a receiver node in the MCC.

- The ns-2 cannot dynamically change its destination node at run time like a normal real-life application program does. The traffic sink agent on the destination node must be paired with the traffic source agent on the source node at the beginning of a simulation.
- Ns-2 network simulator is a user-level program, it does not let another user-level application program to ‘run’ on top of it. Hence, a real-life application program cannot run normally on a network simulated by ns-2. However, NCTUns allows an application/algorithm to run on a simulated node.
- NCTUns uses the operating systems (Linux kernel) TCP/IP (or UDP/IP) protocol stack to conduct simulations and emulations. To enable the use of Linux’s TCP/IP stack, kernel re-entering methodology is used. This feature enables it to test any real world network application.
- NCTUns provides a highly-integrated and easy-to-use GUI environment for constructing the topologies and traffic models. After simulation is completed, with playback option data flow could be seen.
- The NCTUns simulation engine adopts an open-system architecture and is open source.

- NCTUns supports remote, concurrent, and distributed simulations.
- NCTUns can be easily used as an emulator.

## 5.4.2 Simulator Capabilities and Features

A key advantage that NCTUns gains over other network simulators is that it allows for the use of real TCP/IP protocol stack, which in turns enables the use of real life applications as traffic generators for the simulation purposes. The NCTUNS achieves this ability by employing a novel kernel re-entering simulation methodology.

NCTUns has a built-in support for a multi-interface node that can conveniently simulate a receiver node in the MCC. In the following, sections, we describe its capabilities and features that are relevant to our simulation scenario.

### 5.4.2.1 Kernel Re-entering Methodology

NCTUns takes advantage of an abstraction of Linux that allows to create pseudo network interfaces (tunnel interface) which may or may not have corresponding physical network device. From the kernel's point of view, a network interface is a software object that can process packets hence a pseudo network interface functionality is exactly the same as that of a normal network interface. If an application program writes a packet to this virtual device it will enter the kernel and for the kernel the packet would appear to have come from real network and will be passed on to the TCP/IP stack for further processing just like real Ethernet packet. Similarly, the receiving application program will read the packet and for kernel the packet would appear to have transmitted over the network. Figure 5.2 depicts this concept.

Figure 5.2, illustrates how to simulate a TCP sender application program running on host 1 to send its TCP packets to a TCP receiver application program running on host 2 by using tunnel (virtual) network. The network interfaces here are pseudo network interfaces and does not have associated physical network. The TCP/IP stack being used is the real stack run the by the kernel and two TCP/IP protocols depicted here actually are the same.

The setting up of virtual simulation network involves following two operations. First, is to configure kernel routing table of the simulation machine in such a way that any TCP packets sent from host 1 to host 2 should be sent through tunnel

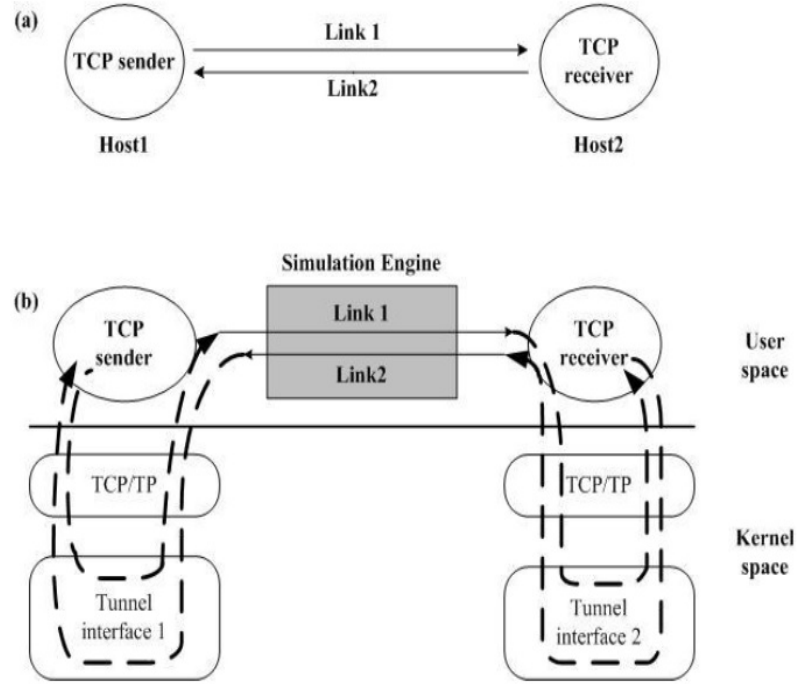


Figure 5.2: The Kernel Re-entering Simulation Methodology. Reproduced from [4]

(pseudo) network interface 1 and similarly tunnel (pseudo) network interface 2 must be chosen for the packets sent from host 2 to host1. Second, for the two links to be simulated, a simulation engine process is run to simulate them. it simulates a packet's transmission on the link from host  $i$  to host  $j$  ( $i = 1$  or  $2$  and  $j = 3 - i$ ) by reading a packet from the special file of tunnel interface  $i$ , waiting for the link's propagation delay time plus the packet's transmission time on the link (in virtual time) to elapse, and then writing this packet to the special file of tunnel interface  $j$ .

While the NCTUNs simulation engine is running, this simulated "virtual network" is alive. Figure 5.2(b) depicts this simulation scheme. Since simulation of links happens outside the kernel, the kernel on both hosts do not know that their packets actually are exchanged on a virtual simulated network. This makes it possible for all TCP/IP based real-life network application programs to run on the simulation network. The kernels and TCP/IP stacks on both the sending and receiving hosts are the same hence the name "kernel re-entering methodology".

#### **5.4.2.2 Reusing All Real-Life Application Programs**

The ability of NCTUNs to run real life applications (unmodified code) on simulated environment as well as on real UNIX machines not only save time and effort but also provides several other unique advantages. A key advantage is the generation of realistic network traffic, which leads to more useful and accurate results than using the simple synthetic benchmarks. Also it allows to evaluate performance of these real-life applications under various network conditions and then improved before releasing to the public.

#### **5.4.2.3 High Simulation Speeds and Repeatable Simulation Results**

NCTUNs combines the kernel re-entering simulation technique with its discrete event simulation methodology which results in the execution of speedy simulations. NCTUNs has modified the process scheduler of the Linux kernel to control the execution order all processes involved in the simulation more accurately. These processes include simulation engine process and all other real-life application processes. If we keep the random number seed same across our simulation runs then results are repeatable.

#### **5.4.2.4 Support for Various Important Networks**

NCTUNs simulates Ethernet-based IP networks with fixed nodes and point-to-point links. It simulates IEEE 802.11 (a)(b) wireless LAN networks, GPRS cellular networks, advanced optical burst switching (OBS) etc. Moreover, it simulates multi-interface mobile nodes multiple heterogeneous wireless interfaces.

#### **5.4.2.5 Multi-interface Mobile Node**

Multi-interface network node is a mobile node equipped with multiple radio devices. NCTUNs implements four types of radios namely IEEE 802.11(b) infrastructure mode, IEEE 802.11(b) ad-hoc mode, GPRS, and DVBS2/RCST. Availability of these radios enable a multi-interface node to access wireless LAN, mobile ad-hoc networks, telecommunication networks, and satellite networks at the same time.

As shown in Figure 5.3, at the time of creating a multi-interface node, type and number of radios the multiple-interface node contains must be specified. This multi-interface node is called a device node because the simulation engine

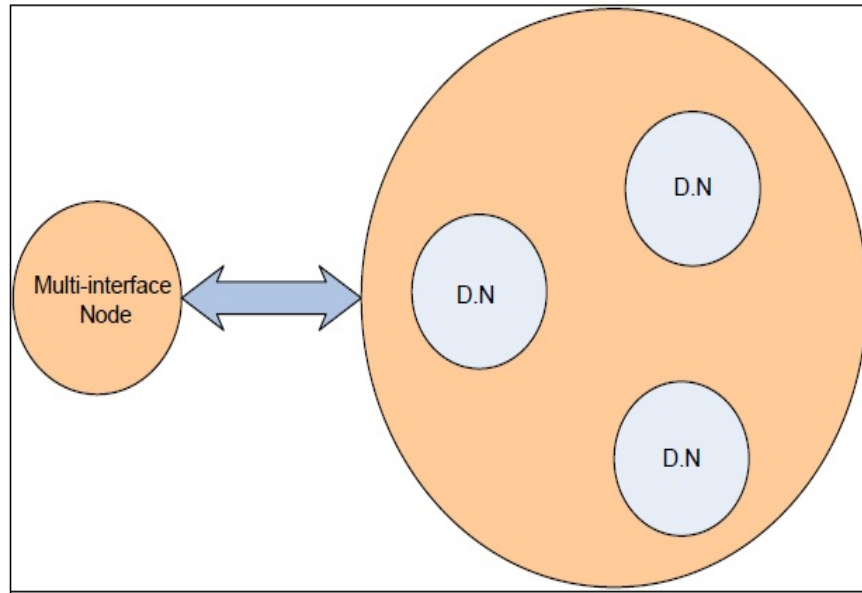


Figure 5.3: The Relationship between the Multi-interface Node and the Device Node. Reproduced from [4]

simulates this radio using a single-interface network node. A GUI program is used to help in creating node structure for a specific selected radio.

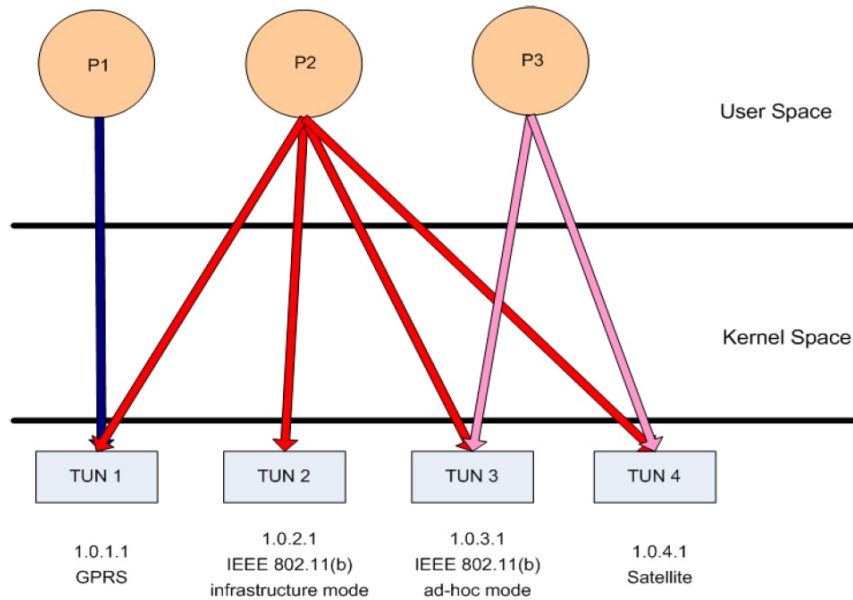


Figure 5.4: An example of a multi-interface node structure. Reproduced from [4]

The application programs exploit pseudo network interface (tunnel device

driver) to utilize the four heterogeneous network radios on a multi-interface node without modification. They achieve this by creating sockets for each interface as normal and use these sockets in an ordinary manner.

### 5.4.3 Components and Architecture of NCTUns

In this section, we provide an overview of the architecture of NCTUns. As shown in Figure 5.5, NCTUns is mainly composed of six components: 1) Graphical User Interface (GUI); 2) Dispatcher; 3) Coordinator; 4) simulation engine; 5) application programs; and 6) patches to the kernel TCP/UDP/IP protocol stacks. The main functions of these components are explained below.

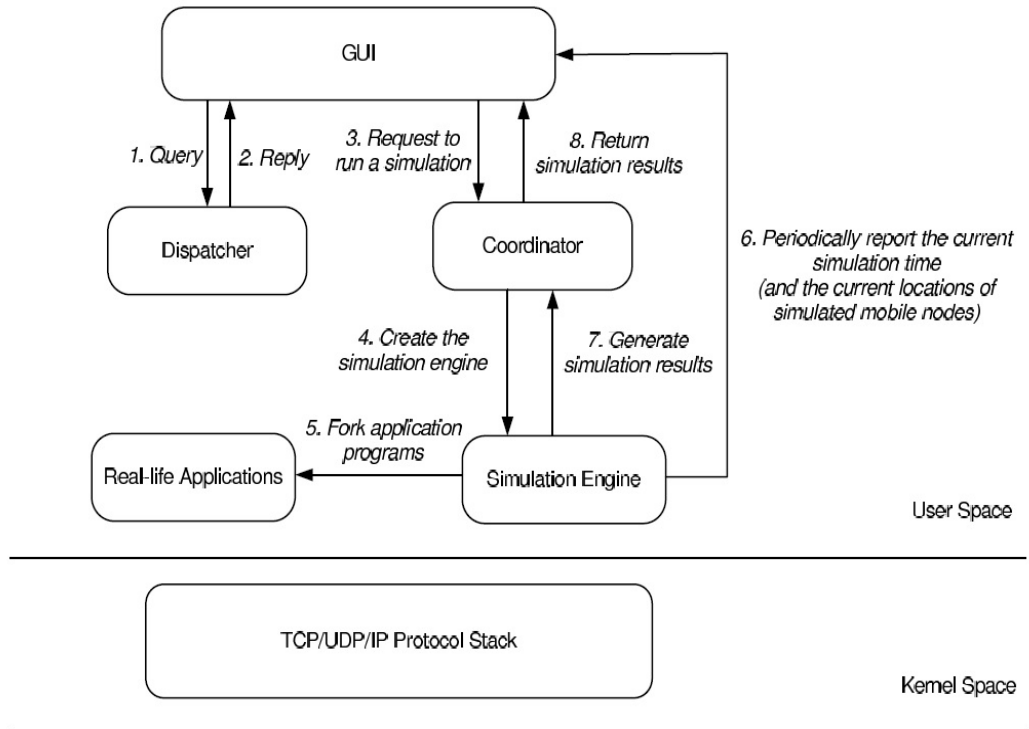


Figure 5.5: The architecture of NCTUns. Reproduced from [5]

1. **GUI:** A front-end GUI ("nctunsclient") in NCTUns provides useful facilities for users to efficiently create simulation and emulation Cases.
2. **Dispatcher:** In NCTUNS, a simulation engine and GUI can reside on different machines, so the Dispatcher program is sent an inquiry message by

GUI to find out which simulation server is available. The Dispatcher program, apart from keeping a register of simulation server also monitors their status and selects an available simulation server upon receiving a request issued by the GUI program.

3. **Coordinator:** The Coordinator program is the first program to be run before starting a simulation server as it forks (creates) a simulation engine to perform simulation. Also it processes the commands received from Dispatcher and reports back to the Dispatcher with the status of the created simulation engine process. The Coordinator also collects the simulation results and sends them to the GUI program.
4. **Simulation Engine:** The core responsibilities of the simulation engine is simulating various protocol behaviours and implementation of an event scheduler. In addition, during simulation the simulation engine process will periodically report the current simulation time to the GUI program.
5. **Application Program:** Application program are responsible for generating network traffic in a simulated network. As stated earlier that the way the network is simulated enables most real-life application program to run directly on a node simulated by NCTUns to generate realistic network traffic.
6. **Kernel Patches:** NCTUns uses Linux patches to modify real TCP/IP stack running on the simulated machine. These modifications however are restricted to the Kernel timers so that each simulated node can advance their times based on the NCTUns simulated clock rather than the real-world clock.

In the following, we explain how NCTUns performs a simulation in detail. Suppose that one has finished specifying his simulation case. By clicking the "Run" command (on the GUI control panel), one can trigger the GUI program to start a simulation. The GUI program first inquiries the Dispatcher program whether any simulation server is now available. If not, the Dispatcher program returns a "No servers are available" message to the GUI program. Otherwise, the Dispatcher program picks an available simulation server and then sends the GUI program the IP address of the chosen simulation server. After receiving the IP address of the chosen simulation server, the GUI program sends a simulation

request (with the files describing the simulation case to be run) to the Coordinator program running on the chosen simulation server. After receiving such a request, the Coordinator program forks a simulation engine process to run the received simulation case.

The simulation engine first constructs the topology of the simulated network, sets up global data structures used for the simulation, and creates/initializes the protocol stack of each simulated node. It then initializes the simulation clock to zero, inserts "Create Application" events (explained below) into the event scheduler, and finally starts the simulation. A "Create Application" event is used to notify the simulation engine of when and which application should be created (forked) during simulation. When the simulation clock advances to the time specified by a "Create Application" event, the simulation engine will fork the application program specified by that event (to generate traffic). Note that the timers of an application process forked by the simulation engine are controlled by the virtual clock of NCTUns. That is, when an application process invokes library calls, such as `gettimeofday()`, `sleep()`, and `alarm()`, these library calls will be triggered based on the virtual clock of NCTUns. This is achieved by modifying time-related system calls in the kernel. During simulation, the simulation engine periodically reports the current simulation time to the GUI program to show the progress of the simulation. After the simulation is done, the Coordinator program will collect and pack all the log files generated during simulation and then transmit them back to the GUI program for further processing and display.

#### **5.4.4 Simulation Environment**

Simulations are performed using the NCTUns (version 6.0). The data is generated and scheduled using applications written in C language. To further facilitate the application, configuration files are used to setup network paths and set data queue(s) size. Simulations are conducted on a laptop computer (on MacBook Pro 2.66 GHz (Intel Core 2 Duo) with 4 GB RAM) running the Fedora 12 operating system running in a Virtual Machine (VM). There are other applications running in parallel to VM on MacBook laptop.



#### 5.4.4.1 Scenario

The considered scenario consists of a MCC with a sender node (multi-interface) connected to the two collaborators (mobile access points). These collaborators are also connected to a common destination node, i.e. a receiver. Figure 5.6 shows the simulated network topology.

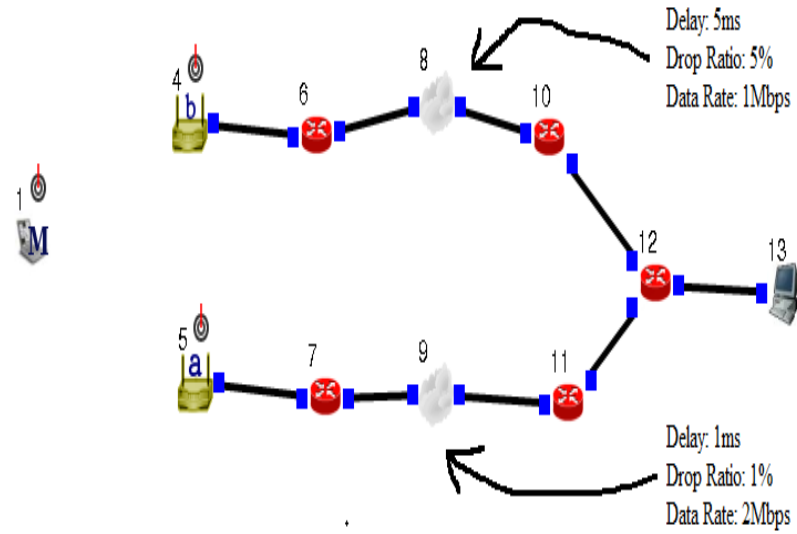


Figure 5.6: The Simulation Setup - Static Conditions

In this simulated MCC network topology, we are using two distinct paths from a single sender (node1) to a receiver (node13). These two paths are distinct Internet channels via two collaborators (node4 and node5) to a receiver. The dynamic channel conditions on both paths are simulated by adding a delay variance on both paths. The path 2 provides higher QoS in terms of data rate, drop ratio and delay. The scenario chosen for our simulations has the following path characteristics:

- Path 1:
  - Bandwidth: 1 Mbps
  - Delay: 5ms
  - Drop Ratio: 5%
  - Delay Variance: minimum 1% to maximum 5% with mean 3%

- Path 2:  
 Bandwidth: 2 Mbps  
 Delay: 1ms  
 Drop Ratio: 1%  
 Delay Variance: minimum 1% to maximum 10% with mean 5%.

During the simulation, all four data scheduling algorithms (RR, CP, SWQ, MWQ) were run one by one on the sender node. These algorithms schedule data to the collaborators for simultaneous data transfer.

#### 5.4.4.2 Configurations

To facilitate the process of reproducing the simulation results, below are screen shots of how to configure the simulation setup described above.

First, the topology editor provides a convenient and intuitive way to graphically construct a network topology. A constructed network can be a fixed wired network or a mobile wireless network. Due to the user-friendly design, all GUI operations can be performed easily and intuitively.

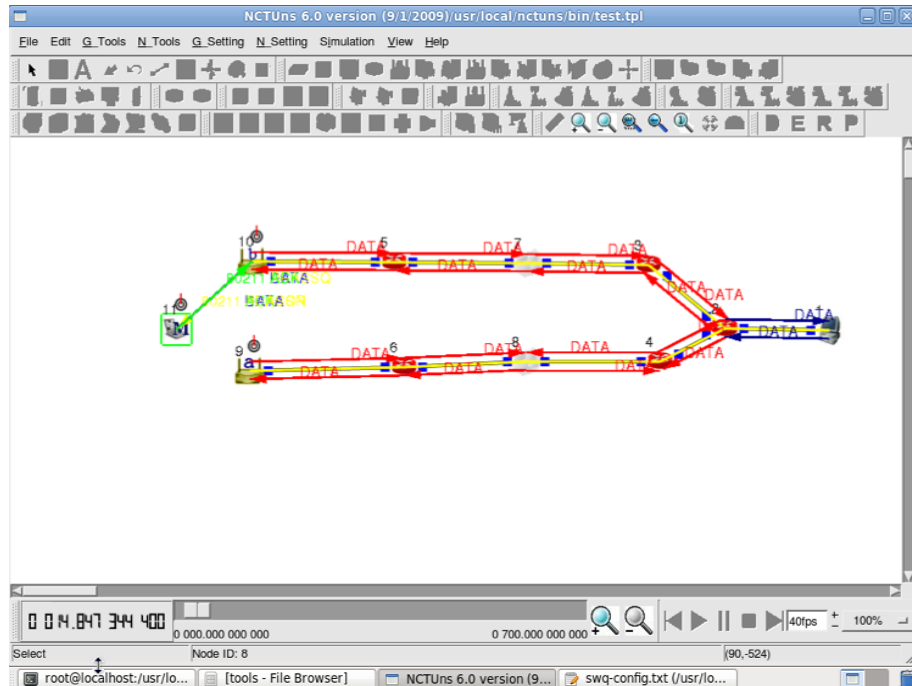


Figure 5.7: The Topology Editor

A major component of our simulation scenario is multi-interface node, that

can be added as a mobile node with multiple network interfaces ready to configure and run. (See Figure 5.8 )

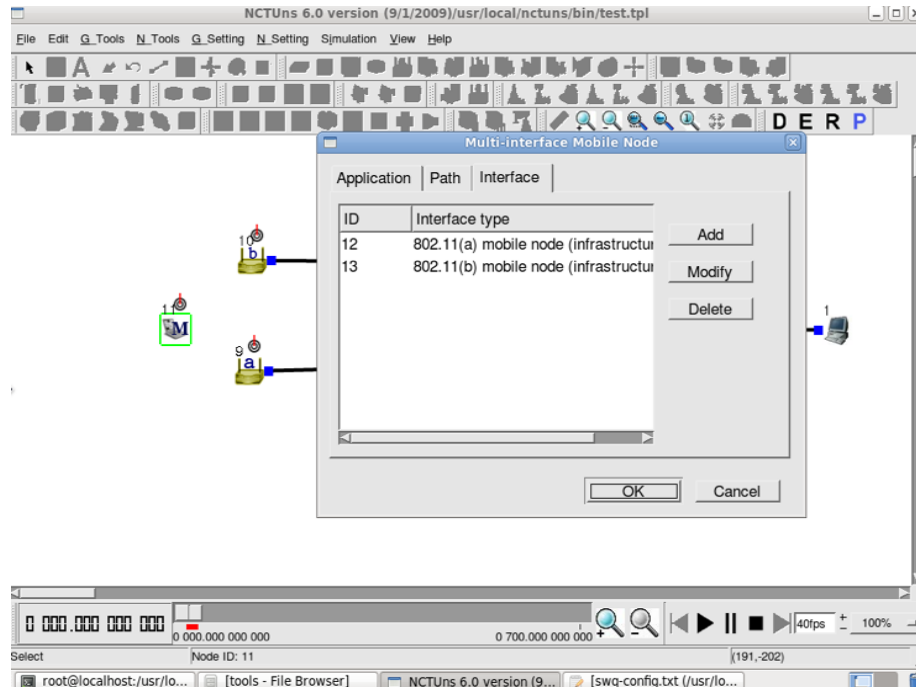


Figure 5.8: The Multi-interface Node.

A network device (node) may have many attributes. Setting and modifying the attributes of a network node can be easily done. Just double-clicking the icon of a network node. An attribute dialog box pertaining to this node will pop up. A user can then set the device's attributes in the dialog box as shown in Figure 5.9.

The node editor provides a convenient environment in which a user can flexibly configure the protocol modules used inside a network node. By using this tool, a user can easily add, delete, or replace a module with his (her) own module. This capability enables a user to easily test the performance of a new protocol. Using the node editor, a user can also conveniently set the parameter values used by a specific protocol module. Each box in the node editor represents a protocol module. A user can double-click a protocol module box to pop up its parameter dialog box. (see Figure 5.10).

Similarly for network links, settings can be done by double-clicking the path. A dialog box will appear where different values of different attributes can be specified (see Figure 5.11).

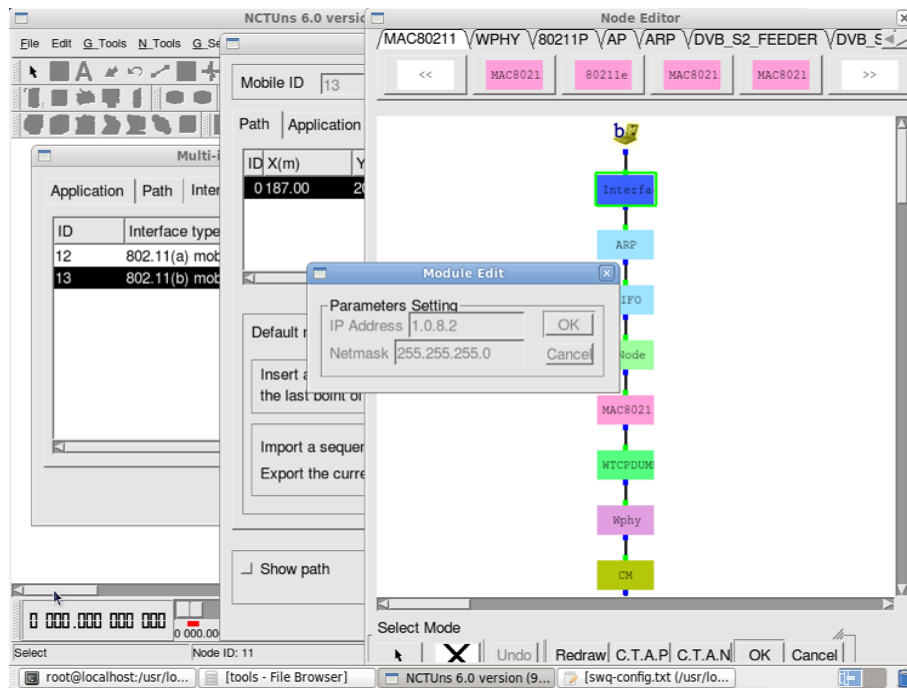


Figure 5.9: The Node Editor

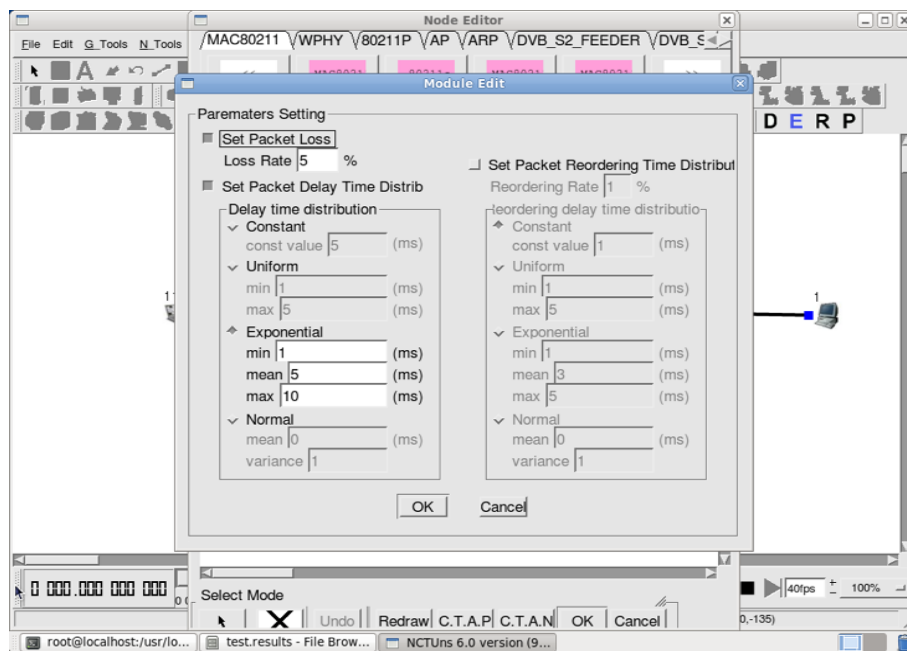


Figure 5.10: The Node Editor - Protocol Module Settings

By using the packet animation player, a packet transfer trace logged during a simulation can be replayed at a specified speed. Both wired and wireless networks

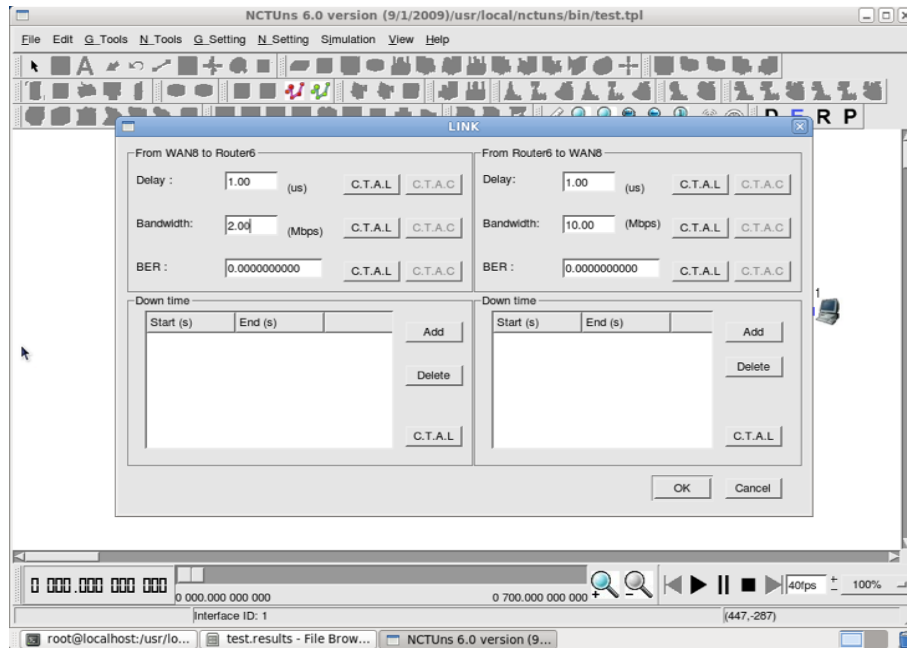


Figure 5.11: The Link Editor

are supported. This capability is very useful because it helps a researcher visually see and debug the behavior of a network protocol. It is very useful for educational purposes because students can see how a protocol behaves.

After finishing a simulation, a binary trace file, `SimulationName.ptr`, is generated by the simulator (see Figure 5.12). We used `printPtr` application to decode the binary file into a plain text file. The traces have the following scheme:

```
802.11 RX 1503414258 2030 ACK <0 0> <133 149 149> 13248917 14 0 NONE
3
```

1. The first entry is the type of protocol
2. The second entry is the type of event. This can be RX (response), TX (transmission), RTX (retransmission) and DROP.
3. The third column is the time of the event shown by number of ticks.
4. The fourth column is the duration of the event.
5. The fifth column is the type of message. This can be DATA (802.3/802.11 Data packet), RTS (802.11 RequestToSend packet), ACK (802.11 Acknowledgement packet), etc.

```

802.11 RX 10016799 2030 ACK <0> <11 10 10> 130 14 0 NONE 3
802.11 TX 10020426 2670 DATA <0> <11 10 10> 106 102 0 NONE 3
802.11 RX 10020429 2670 DATA <0> <11 10 10> 106 102 0 NONE 3
802.3 TX 10023089 665 DATA <11 1> <10 5> 106 78 0 NONE
802.3 RX 10023109 665 DATA <11 1> <10 5> 106 78 0 NONE
802.11 TX 10023199 2030 ACK <0> <10 11 11> 131 14 0 NONE 3
802.11 RX 10023202 2030 ACK <0> <10 11 11> 131 14 0 NONE 3
802.3 BTX 10023774 513 DATA <0> <5 7> 133 64 0 NONE
802.3 BRX 10023784 513 DATA <0> <5 7> 133 64 0 NONE
802.3 BTX 10024383 2788 DATA <0> <8 4> 129 64 0 NONE
802.3 BRX 10024383 2788 DATA <0> <8 4> 129 64 0 NONE
802.3 TX 10027181 525 DATA <0> <4 8> 134 64 0 NONE
802.3 RX 10027191 525 DATA <0> <4 8> 134 64 0 NONE
802.11 BTX 10030729 2430 DATA <0> <10 0 0> 107 70 0 NONE 3
802.11 BRX 10030732 2430 DATA <0> <10 11 0> 107 70 0 NONE 3
802.3 TX 10040639 2802 DATA <0> <8 6> 134 64 0 NONE
802.3 RX 10040649 2802 DATA <0> <8 6> 134 64 0 NONE
802.3 TX 10043451 674 DATA <11 1> <6 8> 128 78 0 NONE
802.3 RX 10043451 674 DATA <11 1> <6 8> 128 78 0 NONE
802.3 BTX 10045175 5145 DATA <0> <7 3> 133 64 0 NONE
802.3 BRX 10045185 5145 DATA <0> <7 3> 133 64 0 NONE
802.3 TX 10050330 538 DATA <0> <3 7> 135 64 0 NONE
802.3 RX 10050340 538 DATA <0> <3 7> 135 64 0 NONE
802.3 TX 10057058 3342 DATA <11 1> <8 4> 128 78 0 NONE
802.3 RX 10057068 3342 DATA <11 1> <8 4> 128 78 0 NONE
802.3 BTX 10060410 542 DATA <0> <4 2> 137 64 0 NONE
802.3 BRX 10060420 542 DATA <0> <4 2> 137 64 0 NONE
802.3 TX 10060962 529 DATA <0> <2 4> 138 64 0 NONE
802.3 RX 10060972 529 DATA <0> <2 4> 138 64 0 NONE
802.3 TX 10061501 663 DATA <11 1> <4 2> 136 78 0 NONE
802.3 RX 10061511 663 DATA <11 1> <4 2> 136 78 0 NONE
802.3 BTX 10062174 541 DATA <0> <2 1> 140 64 0 NONE
802.3 BRX 10062184 541 DATA <0> <2 1> 140 64 0 NONE
802.3 TX 10062725 593 DATA <0> <1 2> 141 64 0 NONE
802.3 RX 10062735 593 DATA <0> <1 2> 141 64 0 NONE
802.3 TX 10063288 673 DATA <11 1> <2 1> 139 78 0 NONE
802.3 RX 10063298 673 DATA <11 1> <2 1> 139 78 0 NONE
802.3 TX 10063371 658 DATA <11 1> <1 2> 142 78 0 NONE
802.3 RX 10063381 658 DATA <11 1> <1 2> 142 78 0 NONE
802.3 TX 10064839 628 DATA <11 1> <2 4> 143 78 0 NONE
802.3 RX 10064849 628 DATA <11 1> <2 4> 143 78 0 NONE
802.3 TX 10065277 693 DATA <11 1> <4 8> 144 78 0 NONE
802.3 RX 10065287 693 DATA <11 1> <4 8> 144 78 0 NONE
802.3 TX 10088879 3400 DATA <11 1> <8 6> 144 78 0 NONE
802.3 RX 10088889 3400 DATA <11 1> <8 6> 144 78 0 NONE
802.3 TX 10093179 5265 DATA <0> <7 5> 135 64 0 NONE
802.3 RX 10093176 5265 DATA <0> <7 5> 135 64 0 NONE
802.3 TX 10092289 642 DATA <11 1> <6 9> 145 78 0 NONE
802.3 RX 10092299 642 DATA <11 1> <6 9> 145 78 0 NONE
802.11a TX 10094001 1550 DATA <0> <9 11 11> 145 102 0 NONE 6
802.11a RX 10094004 1550 DATA <0> <9 11 11> 147 102 0 NONE 6
802.11a TX 10095724 400 ACK <0> <11 9 9> 148 14 0 NONE 6

```

Figure 5.12: The Log File

6. The sixth column is the node IDs based on the IP address.
7. The seventh column is the node IDs based on the MAC address.
8. The eighth column is the ID of the packet.
9. The ninth column is the size of the packet.
10. The tenth column is about the number of RTX for a package.
11. The eleventh column is the cause of DROP, e.g. COLL (collision), CAP (capture), BER (bit error), etc.
12. The last column is the frequency channel.

All useful information from the trace files are extracted using AWK scripts.

### 5.4.5 Performance Metrics

The performance evaluations of the different scheduling algorithms are carried out by using the following metrics. The following metrics have been analysed:

throughput, total data transfer time and percentage improvement. In the following, we describe how we computed every metric. We made 10 independent simulation runs for each result and the averages of the results are presented below.

- **Throughput:** is calculated dividing the total amount of data received by the time of simulation.

$$throughput = \frac{receivedpackets * packetsize}{simulationtime} \quad (5.1)$$

- **Total data transfer time** is measured in seconds for each end-to-end file transfer.
- **Improvement percentage** is the ratio of positive change from one number to a higher number as expressed by a percentage. The total data transfer time for any algorithm  $X$  is the first base number against which we calculate the improvement of base number2 which is total data transfer time of our proposed MWQ algorithm.

$$\text{Improvement Percentage} = (\text{number1} - \text{number2}/\text{number2}) * 100$$

Where,

number1 = total data transfer time for algorithm  $X$

number2 = total data transfer time for MWQ algorithm

#### 5.4.6 The Abbreviations Used

The abbreviations that are used in the evaluation section are given in following table.

Abbreviations	Descriptions
RR	RR algorithm
CP 20:30	CP algorithm with 20MB data allocated to path 1 and 30MB to path 2
SWQ	SWQ algorithm
MWQ 20:30	MWQ Queue algorithm with 20MB data placed in Queue 1 and 30MB in Queue 2

Table 5.1: Abbreviations for Simulation Results

Here CP 20:30 means the data is divided such that 20MB is scheduled to path 1 and 30MB to path 2. Similarly, CP 60:40 means 60MB is scheduled to path 1

and 40MB through path 2. Similarly, MWQ 20:30 means that queue 1 has 20MB of data whereas queue 2 has 30MB of data assigned to it.

In next section, we perform evaluation by running different experiments using the scenario and the simulator that we have described earlier.

### 5.4.7 QoS Evaluation of MWQ Algorithm

#### 5.4.7.1 Simulation Experiment 1

We run all four scheduling algorithms listed above and measured the average end-to-end throughput in Kbps. We then calculated the total transfer time taken by each algorithm to transfer 50MB data. We also calculated the improvement gained by the MWQ algorithm and derived the improvement percentage. Results are shown in the following tables:

Algorithm	Time (sec)
RR	243
CP 20:30	194
SWQ	220
MWQ 20:30	152

Table 5.2: Total Transfer Time for 50 MB

Algorithm	Improvement (sec)	Improvement (%)
RR	92	60.92
CP 20:30	43	28.47
SWQ	69	45.69

Table 5.3: Improvement Percentage of MWQ for 50 MB

#### Improvement Percentage Calculation:

From the results presented in Table 5.4, we can see the baseline for the comparison of MWQ total data transfer time improvement with RR for 50MB data is 243 seconds.

The improvement gained by MWQ is  $(243-151) = 92$  seconds.

Hence, Improvement Percentage is  $(92/151)*100 \approx 61\%$

#### Analysis:

1. Table 5.4 shows that RR performed worst due to its static scheduling approach. MWQ transferred data quicker than all other algorithms.



2. Table 5.5 shows that MWQ is showing Improvement percentage ranging from 28.5% to 61% to other scheduling algorithms.
3. An interesting observation from Table 5.4 shows that the SWQ algorithm is taking more time than the CP algorithm, i.e. CP is performing better than the SWQ algorithm. To further investigate the CP performance, we have conducted another experiment (Simulation Experiment 2) explained in next section by altering the channel allocation for the data.

#### 5.4.7.2 Simulation Experiment 2

One possible reason of the CP's better performance could be that both individual channels had sufficient bandwidth to support their data load. To confirm this, we repeated the above experiment and changed the channel allocation such that more data (30MB) is now allocated to the path 1 and 20MB to the path 2. We also altered the data allocation for MWQ algorithm such that more data, i.e. 30MB is kept in queue 1 and 20MB in queue 2. Following are the combined results from the previous experiments:

Algorithm	Time (sec)
RR	243
CP 20:30	194
CP 30:20	290
SWQ	220
MWQ 20:30	152
MWQ 30:20	151

Table 5.4: Revised Total Transfer Time for 50 MB

Algorithm	Improvement (sec)	Improvement (%)
RR	92	60.92
CP 20:30	43	28.47
CP 30:20	139	92.05
SWQ	69	45.69

Table 5.5: Revised Improvement Percentage of MWQ for 50 MB

#### Analysis:

1. Above results confirm that changing the data allocation to queues does not affect the performance of the MWQ algorithm, however, it has increased

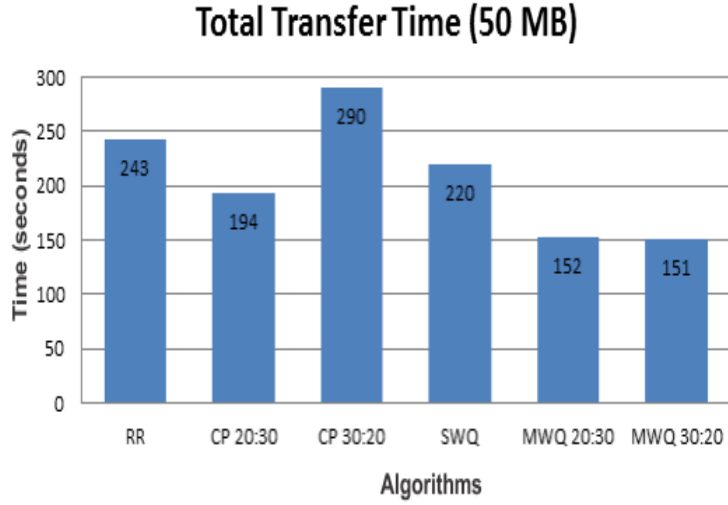


Figure 5.13: Graph: Total Transfer Time for 50 MB

the total data transfer time of the CP algorithm from 194 seconds to 290 seconds.

2. Improvement percentage of MWQ against CP increased from 28.5% to 92% after altering the channel allocations. To further analyse the above results, we calculated the average throughput of all the algorithms, See Figure 5.14 to see the throughput plotted against the time scale.

#### Analysis:

1. RR throughput is constantly less than all other algorithms; unless CP throughput suddenly drops lower than the RR algorithm.
2. SWQ and MWQ stayed consistent in terms of throughput and their throughput is high as well.
3. MWQ shows slightly better throughput rates than the SWQ, hence, it finished the data transfer earlier.
4. In the case of CP, the throughput has dropped by more than half when one of the channel finishes its data transfer and second channel is still sending. The reason is that the CP algorithm does not support dynamic re-allocate of data to the idle channel.

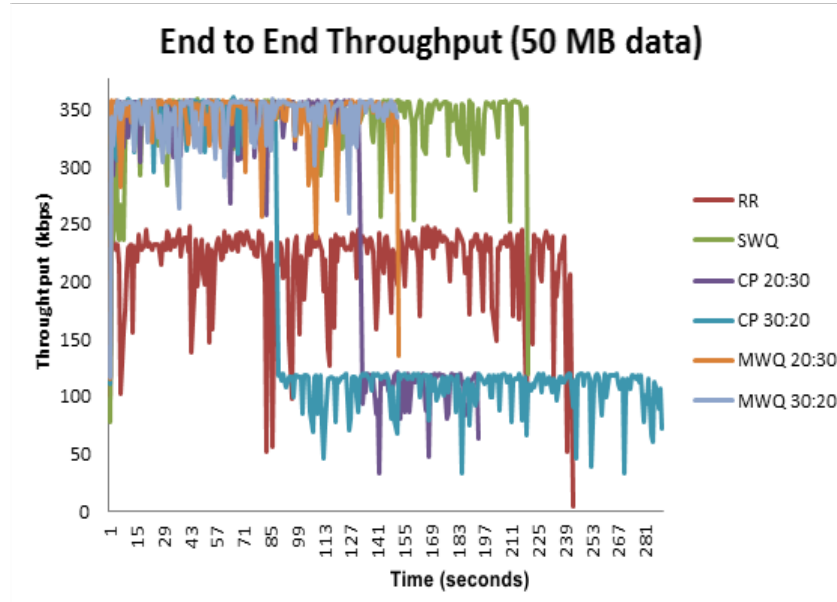


Figure 5.14: Graph: Throughput (kbps) for 50MB Data Transfer

5. In the case of CP 30:20 less data was schedule through the fast channel, hence it finished its job early and stayed idle during the rest of simulation.

#### 5.4.7.3 Simulation Experiment 3

To verify the results from simulation experiments 1 and 2, we have repeated both of them for 100MB data size. Results confirmed that MWQ behaved consistently and maintained its improvement percentage. Results can be seen in the following tables and graphs.

Algorithm	Time (sec)
RR	491
CP 20:30	391
CP 30:20	591
SWQ	439
MWQ 20:30	302
MWQ 30:30	303

Table 5.6: Total Transfer Time for 100 MB

Algorithm	Improvement (sec)	Improvement (%)
RR	189	62.58
CP 40:60	89	29.47
CP 60:40	289	95.69
SWQ	137	45.36

Table 5.7: Improvement Percentage of MWQ for 100 MB

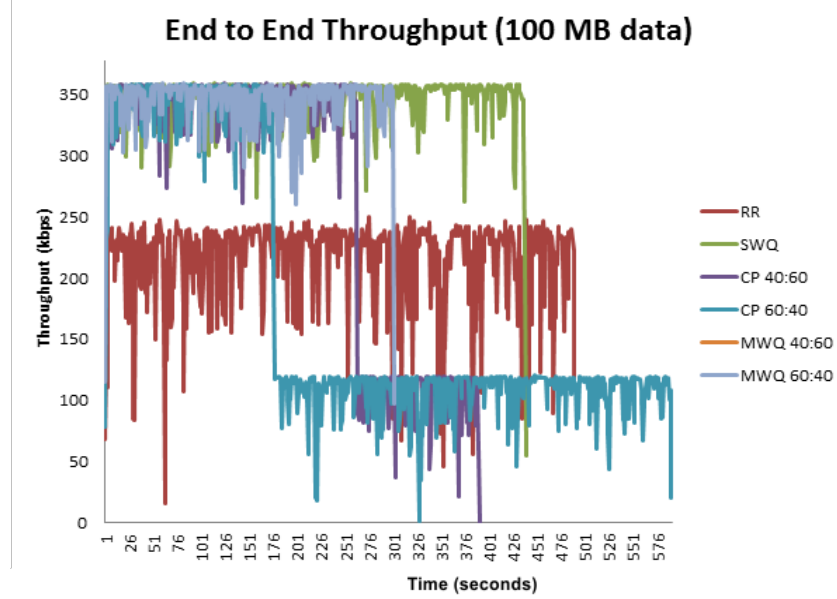


Figure 5.15: Graph: Throughput (kbps) for 100MB Data Transfer

#### 5.4.8 Major Findings

The simulation study of the four scheduling algorithms has led us to the following two major findings.

1. MWQ supports better QoS in terms of throughput than other three algorithms. This improvement in QoS is achieved by adapting dynamically to the QoS changes in the short term network conditions.
2. The SWQ algorithm can lead to similar QoS as the MWQ algorithm. However, MWQ is more flexible in terms of storing data according to their priority order. To achieve this, the MWQ algorithms should be integrated with an access control model discussed in Chapter 4.

From the findings presented in this section, we can conclude that MWQ supports better QoS than the reference scheduling algorithms under short-term QoS

changes in network conditions. However, we need to consider the QoS capabilities of the collaborators before giving them access to a WQ. It enables the scheduling of most important data on trusted or high QoS channels only. Hence, it is suggested to integrate this MWQ implementation with an access model presented in chapter 4 and perform more experiments. These experiments must be able to produce long-term changes in QoS of underlying network. After careful consideration, we chose to conduct test-bed experiments to evaluate RAW scheduler in real life scenario.

## 5.5 Performance Evaluation using Test-bed Experiments

### 5.5.1 Experimental Setup

In our experimental setup, we used a Mobile Terminal (MT), i.e. a laptop as an HTTP data receiver. The laptop we used is a Lenovo Intel Core i3 (2.40 GHz) with 4 GB RAM. MT was running Windows 7 operating system with service pack 1 installed. The MT was connected to the WWAN network through three collaborating mobile devices, i.e. mobile phones. We used three Nokia 206 mobile phones as collaborating mobile devices. The MT downloaded the web content from the web host by using the WWAN links of participating MCC mobile phones simultaneously. To establish an underlying MCC network, the MT first connected itself to these mobile phones through bluetooth and established a Point-to-Point Protocol (PPP) connection with each of them. We used a bluetooth device plugged into the USB port of the laptop that helps connecting to the multiple Nokia phones simultaneously. For this purpose, the phones were placed in close proximity to one another and they did not use any specialized antennas. The experimental setup is shown in the Figure 5.16 below.

These mobile phones were using the mobile data services of two leading cellular service providers, namely, Virgin and Lyca, through their GPRS interfaces. The QoS effects of the bluetooth links were negligible as compared to the GPRS effects, which dominate the changes in QoS performance during these experiments. It is an established fact that the round trip time and variance of a GPRS link is much greater than a bluetooth link and it supports lower bandwidth as well. We did not do any special hardware enhancements in the phone or at the network.

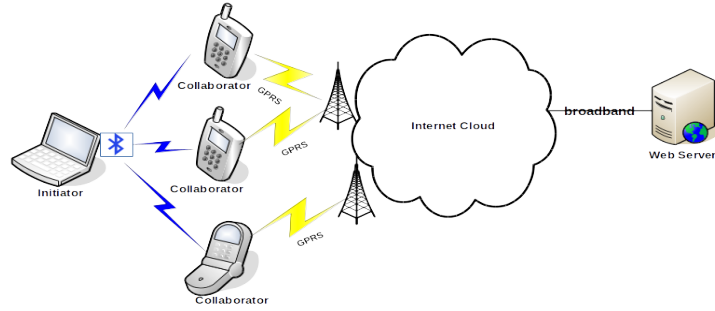


Figure 5.16: The Test-Bed Setup for Experiments

The only change that had to be done in the collaborating device is to change its name to a set convention. To become part of our implemented MCC, the device name should start with the service provider name like lyca, virgin etc. In this way, only those devices that had agreed to be a part of the collaborative community by changing their names were used for the bandwidth aggregation. There is, however, a regular procedure of bluetooth handshake and acceptance from the user is required (see section 5.5.3). Before proceeding to further details of the MCC formation, the next section briefly lists the tools and libraries used to implement our test-bed network and example application.

## 5.5.2 Toolset and Language

### 5.5.2.1 Java

We uses Java as the programming language for our application development. Java is useful when programs require lower-level network communication. Furthermore, it has a library with built-in support for high-level application protocols (such as HTTP). Several other tools and libraries have been used for implementing different modules of our application. The main tools and libraries used are listed below.

### 5.5.2.2 Bluez

We installed bluez package to pair the bluetooth devices from the command line with the bluez-simple-agent using following command.

```
bluez-simple-agent hci# xx:xx:xx:xx:xx:xx
```

Replace # with the Bluetooth adapter number (e.g. hci0) and xx:xx:xx:xx:xx:xx with the MAC of our Bluetooth device. If we already had paired a device and need to remove it from the database (e.g. for re-pairing), following command can be used

```
bluez-simple-agent hci# xx:xx:xx:xx:xx:xx remove
```

### 5.5.2.3 HCI Tool

"hcidtool" is a helpful utility which scans for available bluetooth devices in the surroundings and tries to connect to them. It has been used in this application to find new mobile devices and connect to them over bluetooth.

hcidtool is also used to configure Bluetooth connections and send special commands to Bluetooth devices. If no command is given, or "option -h" is used, hcidtool prints some usage information and exits. Below is the script that uses the hcidtool to discover all available bluetooth connections.

### 5.5.2.4 IP Route

Linux routes all network traffic with the help of routing tables. There is a default routing table that redirects all traffic to a default connection, i.e. to an optimal route. In order to utilize the paths from all connections, the "ip route" utility is used to add a new routing table against each connection and to add its entry into the default routing table. These alternate tables enable data traffic to flow through all network connections simultaneously. An example of "ip route" usage can be seen in the Figure5.18.

## 5.5.3 MCC Network Formation

To test any application for the scenario of collaborative data transfer in MCC, our application should be able to form an underlying network of the collaborating mobile devices. Following are the main steps and scripts to configure our test-bed MCC network.

```

#!/bin/sh

hcitool scan > ./hcitoolscan.out

FLAG=0
while read line
do
    if [ "$FLAG" = "1" ]
    then
        #echo $line
        NAME=`echo $line|awk '{print $1}'`
        MAC=`echo $line|awk '{print $2}'`
        echo "$NAME $MAC"
    fi

    INPUT=`echo $line|cut -c10-12`
    if [ "$INPUT" = "..." ]
    then
        #echo "$INPUT"
        FLAG=1
    fi
done< ./hcitoolscan.out

#echo "$NAME $MAC"

```

Figure 5.17: HCI Tool Script to Get Available Bluetooth Connections

---

```

#!/bin/bash
if [ "$#" != "3" ]
then
    echo "Usage : $0 interface_name ip_address gateway"
    exit
fi
IF=$1          #eth0
IP=$2          #IP
P=$3           #IP2
P_NET=$3       #IP3

TABLE_NO=`echo $IF|cut -c4-`
TABLE_NO=`expr $TABLE_NO + 2` #Alternate routes start from table 2 and ppp0

sudo ip route flush table $TABLE_NO
sudo ip route add $P_NET dev $IF src $IP table $TABLE_NO
sudo ip route add default dev $IF via $P table $TABLE_NO
sudo ip route add $P_NET dev $IF src $IP
sudo ip rule add from $IP table $TABLE_NO|

```

Figure 5.18: Script to Add Alternate IP Tables



### **5.5.3.1 Hand Shaking Process**

Handshaking sometimes referred to as "pairing" is a process where two devices negotiate on how they will be communicating with each other. This is especially important in the case of multiple wireless devices so that they don't interfere with each other. Bluetooth pairing is generally initiated manually by a device user. The bluetooth link of a device is made visible to the other devices. The standard steps of bluetooth pairing that we used to connect our MT with the collaborating mobile phones are explained in the Appendix C.1.

### **5.5.3.2 Adding Multiple PPP Connections**

As stated earlier, one of the biggest challenges presented by the MCC network topology is the simultaneous transfers of data from one sender, to one destination, but via multiple network routes. Operating systems use a routing tables that by default pick an optimal route to the destination and redirect all the data to it. To override this default behaviour, we have written a script (see Figure 5.18) that adds a new routing table for each connection and then adds its entry into the main routing table. All data traffic having source IP of the newly creating connection should use this new routing table called alternate routing table. These alternate routing tables helped us in achieving the desired functionality of routing data through a specific network channel/route.

### **5.5.3.3 PPP Configuration and Chat Scripts**

Figure 5.19 shows the configuration we used for the PPP connection authentication with the Virgin (also called VirginMedia) mobile service provider. Figure 5.20 is the chat script used to connect to the Virgin's mobile data service.

## **5.5.4 RAW Scheduler Implementation: An HTTP Downloader**

In this section, we present the details of a prototype application that we developed for proof of concept. This application is an "HTTP Downloader" that downloads the HTTP data in MCC from the Internet to realize the core functionality of novel RAW scheduler.

The HTTP Downloader is a software that enables HTTP data download while



```

# This optionfile was generated by pppconfig 2.3.17.
# but then humans changed it so it would work :D
#
hide-password
noauth
connect "/usr/sbin/chat -v -f /etc/chatscripts/virgin"
debug
/dev/rfcomm
115200
defaultroute
noipdefault
user "virginmedia"
remotename virginmedia
ipparam virginmedia

usepeerdns

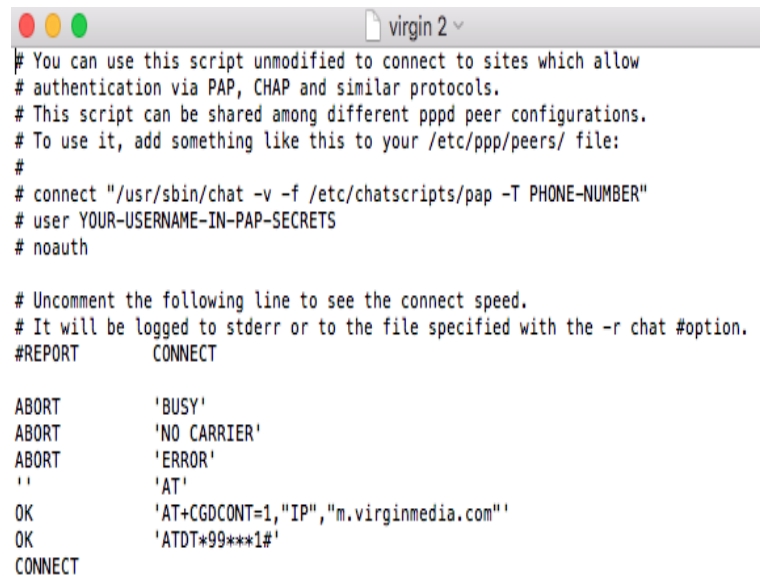
crtscts
lcp-echo-failure 0

##

persist

```

Figure 5.19: Configuration for a Dialup Connection Authenticated with Virgin



```

# You can use this script unmodified to connect to sites which allow
# authentication via PAP, CHAP and similar protocols.
# This script can be shared among different pppd peer configurations.
# To use it, add something like this to your /etc/ppp/peers/ file:
#
# connect "/usr/sbin/chat -v -f /etc/chatscripts/pap -T PHONE-NUMBER"
# user YOUR-USERNAME-IN-PAP-SECRETS
# noauth

# Uncomment the following line to see the connect speed.
# It will be logged to stderr or to the file specified with the -r chat #option.
#REPORT      CONNECT

ABORT        'BUSY'
ABORT        'NO CARRIER'
ABORT        'ERROR'
''           'AT'
OK           'AT+CGDCONT=1,"IP","m.virginmedia.com"'
OK           'ATDT*99***1#'
CONNECT

```

Figure 5.20: Chat Script to Dial Out to Virgin Mobile Data Services

utilizing the multiple WWAN connections (if available) through inverse multiplexing. To keep it expendable and easy to maintain, we have divided our implementation into different modules where each handle different functionality. Our HTTP Downloader modules are closely aligned with the RAW Scheduler modular design. Following sub-sections explain the main components of our example application in details.

#### **5.5.4.1 Device Scanner/Manager**

The device scanner, also referred as the device manager, is responsible for the process of adding mobile device into MCC. It starts off by scanning for all the available devices that can be connected to the application. Once a device is found, it checks if it is already connected to the application. Once the device scanner establishes that the device is not already connected, the mobile data carrier on the device is identified through the device name (pre-configured by the user) and script to connect to the Internet service provider is triggered. As adding a new connection alone into MCC does not allow multiple connections to be multiplexed, hence, this module calls a script to add a new routing table for this connection and to add an entry in the main routing table. In this way, all the traffic having source IP of the newly created connection could use this new routing table. These multiple routing tables later help HTTP Downloader to route a data chunk through a specific channel. Figure 5.21 lists all classes belong to this module and shows the interaction between different methods of these classes.

The main actor shown in Figure 5.21 is a GUI class. As soon as a user opens the application, the GUI initiates the device scanner and performs the following steps:

1. The GUI starts a thread in which the device scanner runs a method that searches for any new devices and adds the new device channels to the system. It also checks if any existing device channel has dropped and updates this information in the application state.
2. It also logs the state of devices for the analysis.
3. Run() then calls ConnectNewDevice which in turn calls multiple scripts to perform various tasks at the operating system level. The following describes the steps of this process.

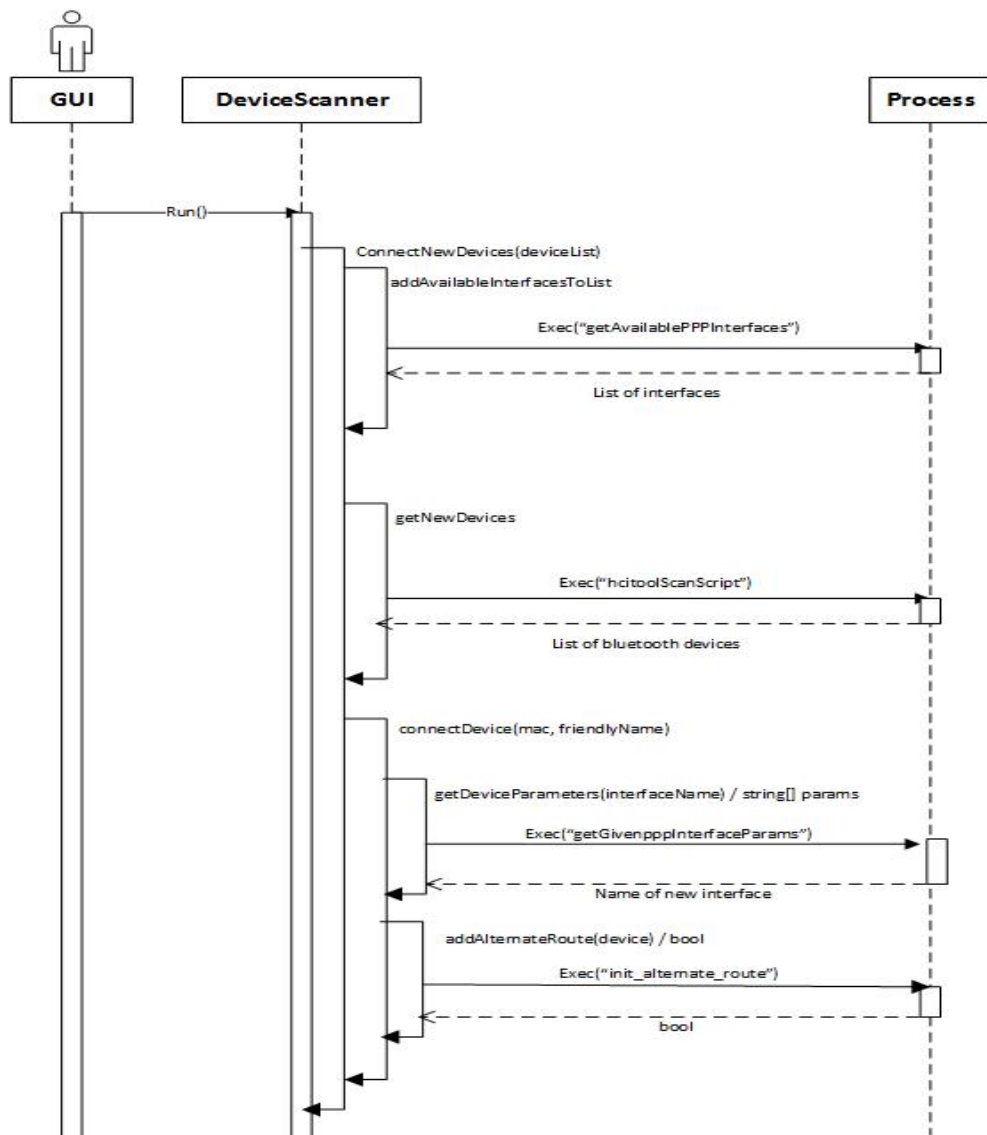


Figure 5.21: Device Manager Flow

- (a) In the first step, information of all available interfaces is saved into a list to be accessed and compared in the following steps.
  - (b) `getNewDevices` returns a list of devices that are reachable via blue-tooth.
  - (c) Then the `connectDevice` method is called with the MAC and friendly name of the device. It then runs the connection script using a process and if successful, gets the IP address and other details of the device by calling `getDeviceParameters`.
  - (d) Once all the details of the device are retrieved, an alternate route to the Internet is updated in the routing tables using `addAlternateRoute` script explained earlier in section 5.5.3.2.
4. This process repeats itself throughout the execution of our application on a fixed interval of time.

#### 5.5.4.2 Partition Scheduler

The core aim of the application is to enable receiver to download HTTP data from Internet through multiple channels simultaneously. The HTTP request is broken down into smaller HTTP requests that can be downloaded in parallel. The size of each request has been fixed after studying the characteristics of channels to improve the probability of completing each request in a single attempt. Each partition request is handled by a thread and it can be downloaded in parallel. The partition manager also referred as the thread manager, manages these threads. Its main responsibility includes assigning partition threads to different collaborating channels and keeping track of the progress. Once the thread manager detects that all the threads have finished downloading requests, it triggers another module that is described later, namely "file manager". The file manager is responsible for the re-construction of the original file from the partition data. Figure 5.22 shows the flow of Single Queue implementation in partition scheduler module.

Figures 5.22 and 5.23 show how the partition scheduler works at a macro level, starting from the point when a downloading request is initiated till it is completely finished and the file manager is triggered. The following describes the steps of the process.

1. The user enters the URL of the file (media/non-media) to be downloaded

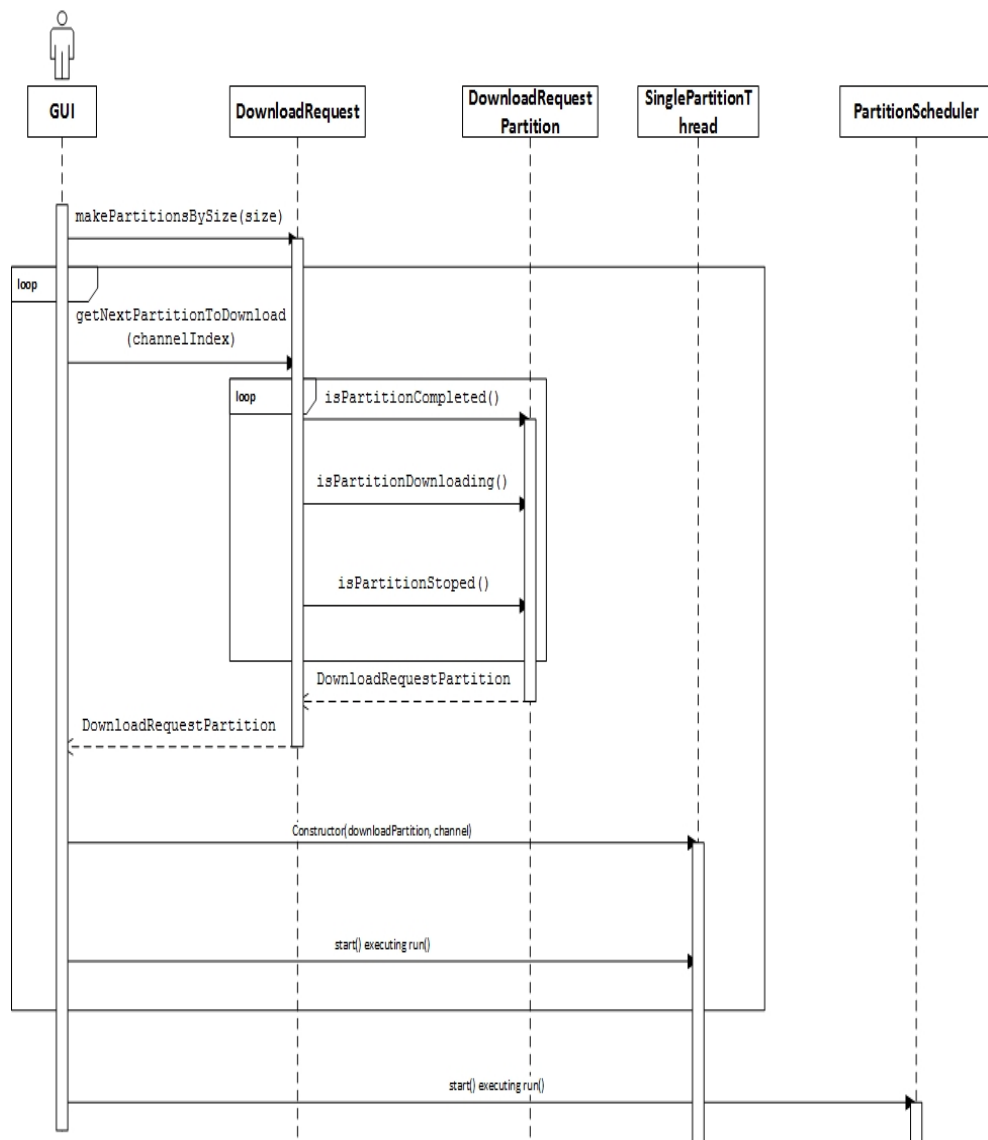


Figure 5.22: Partition Scheduler - SWQ (a)

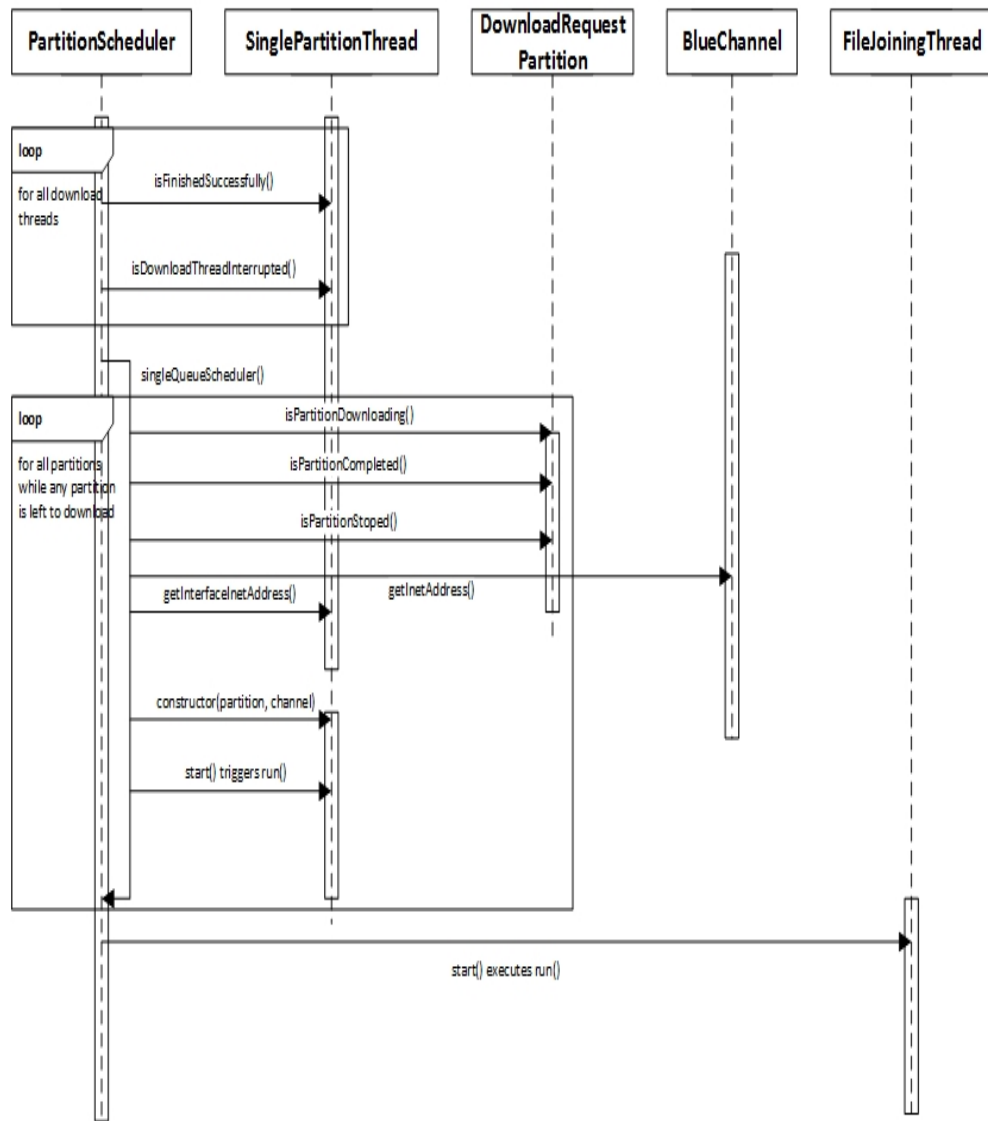


Figure 5.23: Partition Scheduler - SWQ (b)

and hits the download button. All inputs are validated and the total download size is calculated.

2. The application calculates all the partitions based on the download size. Note that it also tags each partition with a priority level but for SWQ based scheduler, it does not have much effect on the the overall download process.
3. Later, all the partitions are created as separate download requests (partitions). Each channel can then request to process a download partition. The partition scheduler also takes into account the quality of the channel but it is not applicable in the case of SWQ implementation. In Figure 5.22 you can see a bootstrap partition assignment. This assignment is followed by the allocation that is done according to the WQ policy. However, in the bootstrap assignment it is done outside the scheduler as threads might get contention while being run between the download threads, device threads and UI thread. This strategy helps to start the download process right away.
4. Before scheduling, each partition state is checked. The download should not be in progress already, completed or stopped.
5. At this stage all available channels are assigned a download partition and the list of all partitions along with the currently running thread list is passed on to the partition scheduler. The partition scheduler is also given the original download request information so it can forward a request to the file manager after the simultaneous download is completed.
6. In Figure, 5.23 the partition scheduler flow starts off by checking for any finished threads. A thread is considered finished if it has downloaded its assigned partition successfully or it has failed to download the partition completely. Note that if a thread fails to download a complete download partition assigned to it, the next thread would resume downloading it rather than restarting it in order to save power and cost. If the thread has not failed yet then the partition scheduler does not re-assign the work and waits for it to resume downloading or go to the fail state.
7. This implementation checks repeatedly after a fixed number of seconds if



any download request is still waiting to be downloaded. If so, it is assigned to a free channel.

8. In this way, the scheduler continues to manage the partitions and makes sure all of them are downloaded. Once all the HTTP data is downloaded, it calls the file manager through a file joining thread which re-construct the required file from the data downloaded by multiple partition threads.

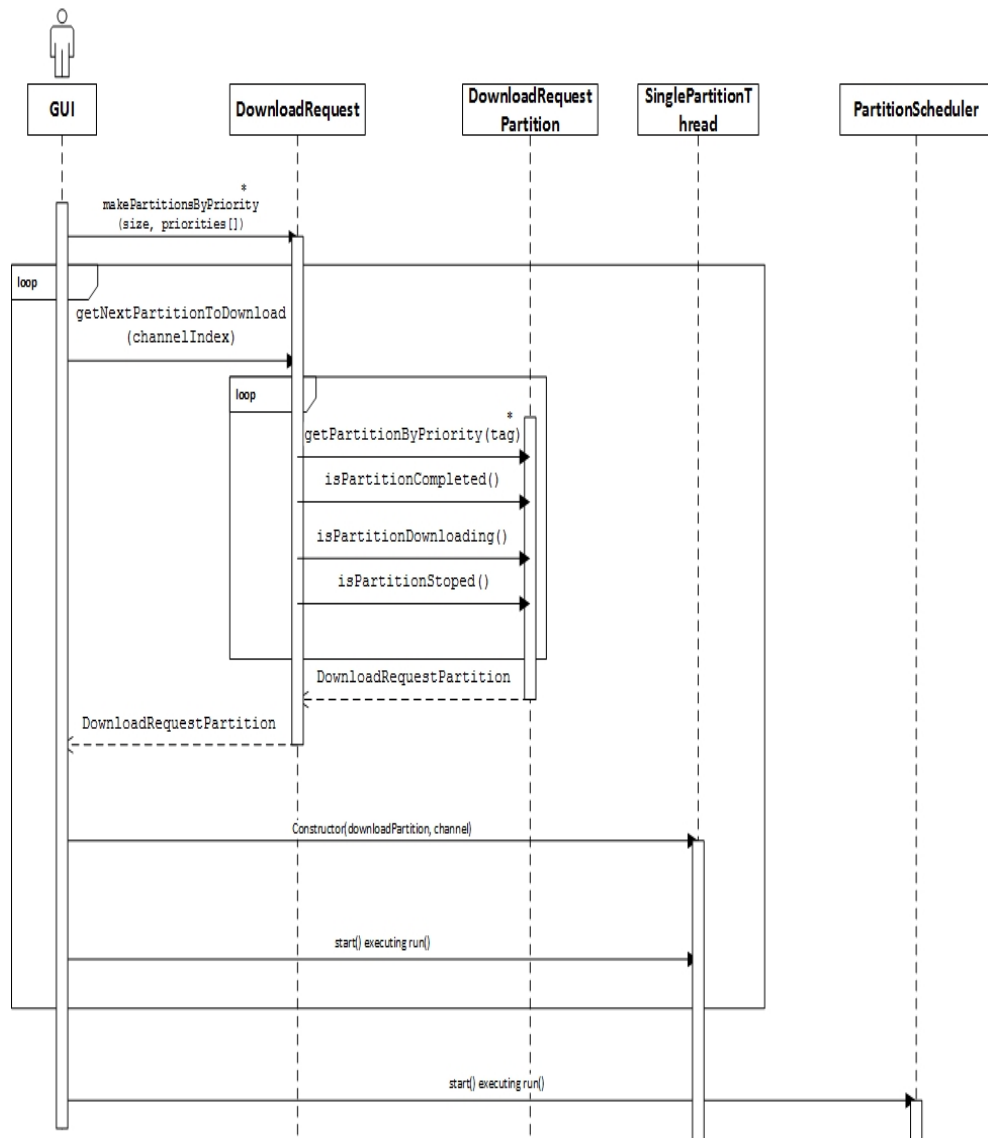


Figure 5.24: Partition Scheduler - MWQ (a)

In the above Figures 5.24 and 5.25 the differences between SWQ and MWQ implementation are marked with an asterisk (\*). The main difference is in the

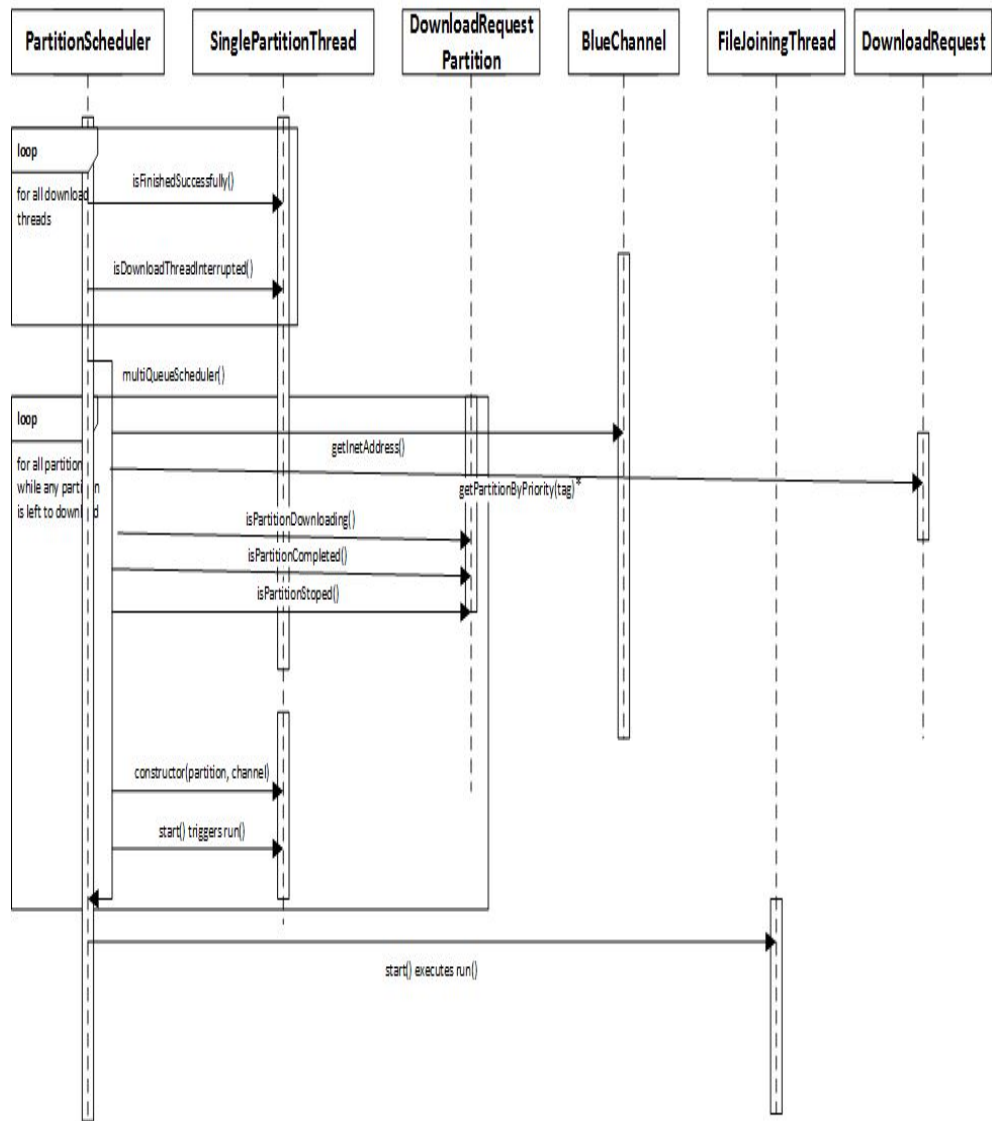


Figure 5.25: Partition Scheduler - MWQ (b)

process of how a partition of work is assigned to a channel. In MWQ implementation each channel is assigned a tag according to its QoS capability. You can see in Figure 5.24 that rather than calling `makePartitionsBySize()` method, the partition scheduler call `makePartitionsByPriority()` method. This creates data partitions and assigns a specific level of quality tag out of the given priority array. Next, the `DownloadRequest` gets `DownloadRequestPartition` by passing along a tag that represents the quality of channel it wants to assign work to. The scheduler makes sure that correct channel is assigned according to the QoS requirement of the partition to be downloaded. If a channel of same level of QoS is not available, the scheduler looks up for channels at a lower level of QoS to serve the request.

The only scenario when a partition tagged with a better QoS requirement is assigned to a channel with lower quality is when there is no better QoS channel available and there is a danger of queue starvation.

#### **5.5.4.3 Partition Downloader**

The partition downloader is a thread created by a partition manager to download a partition request. Each partition downloader also referred as a partition downloading thread itself queries the manager to check out a partition request. The manager returns with the next work item/partition (HTTP bytes to be downloaded) if it has not been downloaded by any other partition downloading thread. When the partition downloading thread completes a download, it marks the request as downloaded and requests the manager for the next work item.

As a result, better performing thread (hence the channel associated with the thread) will automatically download more partitions. In this way, our application is not maintaining any QoS information of the participating MCC channels but still all connections/channels download data according to their QoS capabilities.

In case of a blackout, a timeout would occur. The downloaded part of the request would be saved and the partition would be marked incomplete. The partition manager would assign it to threads/channels requesting for more work. The newly assigned partition downloading thread will continue downloading from the point where the last thread stopped. This ensures that no duplicate data is downloaded, to save time and bandwidth.

#### 5.5.4.4 File Merger

Once all the partitions have been downloaded, the file merger comes into play. It is responsible for joining data of all the partitions that have been downloaded by the partition downloader threads into one single file in correct sequence so that it can be used by the users.

```
public static File mergeFile(ArrayList<File>argFilesList, String argNewFileName, String argParentPath)
throws FileNotFoundException, IOException{

    File resultantFile = new File(ApplicationConstants.DOWNLOAD_FOLDER_PATH, argNewFileName);
    FileInputStream from = null;
    FileOutputStream to = null;
    to = new FileOutputStream(resultantFile);
    for( int i = 0 ; i < argFilesList.size() ; i++){
        try {
            from = new FileInputStream(argFilesList.get(i));
            byte[] buffer = new byte[4096];
            int bytesRead;

            while ((bytesRead = from.read(buffer)) != -1)
                to.write(buffer, 0, bytesRead); // write
        } finally {
            to.flush();
            if (from != null)
                try {
                    from.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
    }
    if (to != null)
        try {
            to.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    return resultantFile;
}
```

Figure 5.26: File Merger Code

#### 5.5.4.5 Operating System (OS) Interactions

There are many operations that are carried out at the OS level with the help of scripts being executed from the application code itself. These include scripts for detecting all available devices over bluetooth that can be connected to this system, dialing the script of a device that hasn't been connected, the alteration of the main routing table along with the creation of connection specific routing table and cleaning of "rfcomm" ports once their corresponding connections depart.

#### 5.5.4.6 GUI

The Graphical User Interface (GUI) of our HTTP Downloader is minimalistic yet effective. It allows the user to enter the URL of the file to be downloaded and buttons to start or cancel the downloading. It also shows progress of the download, number of connections/channels connected along with the overall download speed per second.

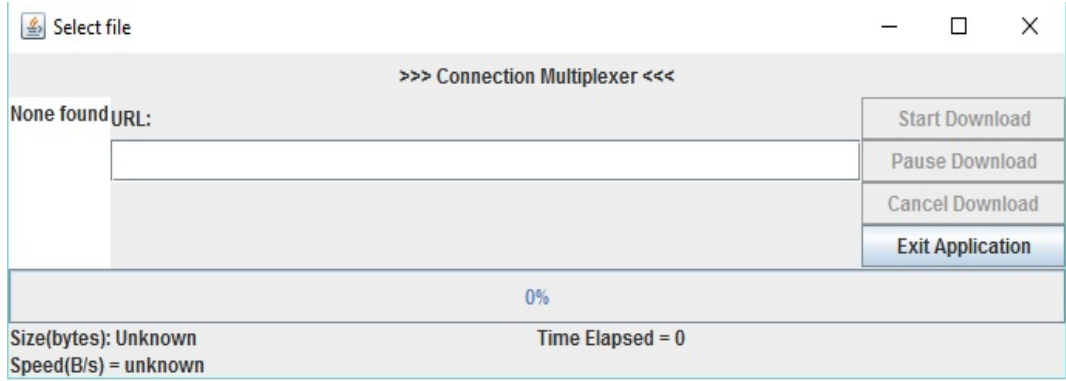


Figure 5.27: HTTP Downloader GUI

#### 5.5.5 QoS claims to be evaluated

This section lists major claims that we make related to improved QoS performance during collaborative data transfer for an "HTTP Downloader" application, implemented as an example application of RAW Scheduler. We claim that our application is:

1. Adaptive to the following QoS fluctuations/changes in channel conditions.
  - (a) Short term QoS changes/fluctuations are addressed by using the MWQ algorithm which by design is adaptive to these frequent short term QoS changes. We have already established that MWQ algorithm adapts well to QoS fluctuations using simulation. Now, we are evaluating an example application of RAW scheduler in a real life scenario to measure its QoS performance gains.
  - (b) Long-term QoS changes due to node arrival/departure/failure are handled in the RAW Scheduler using CMM and community manager. Our test bed implementation is using a device manager and a partition

scheduler to adapt to these node membership changes. The device manager periodically scans for all available collaborators in the blue-tooth range and also detects the collaborators lost during that interval. It notifies the departures to the partition scheduler which reschedules the work of departed collaborators to other available collaborators. These modules can add new collaborators and assign them work if required.

2. Degrade QoS gracefully in case of QoS fluctuation/degradation such that the data with the highest priority is served first. And if the channel conditions are such that the scheduler needs to drop or delay some data then it would be the data with the least priority.

Though energy efficiency is not our major claim, different measures are taken at the architectural and algorithmic levels to minimise the processing and communication costs. To prove each claim, we carefully designed the following experiment sets. The next section provides the details of the experiments that evaluate our claims.

## **5.5.6 Test-bed Experiments and Results**

### **5.5.6.1 Experiment 1**

This experiment is designed to serve as a reference to evaluate the QoS performance of inverse multiplexed channels. In this experiment, we downloaded a file of size 2 MegaBytes (MBs) which is equivalent to 16 MegaBits (mbs) from a live web hosting server. The download was using HTTP downloader application with only one participating mobile phone. The mobile phone was using a Virgin GPRS channel.

Observations:

1. We observed that the throughput observed during the download was in the range of 41.6 - 49.6 kilo bits per second (kbps).
2. Total time taken from the start of the download till the file reconstruction at receiver was 330 seconds.
3. Although throughput of GPRS channel was fluctuating but there was no connection drop out during the download.

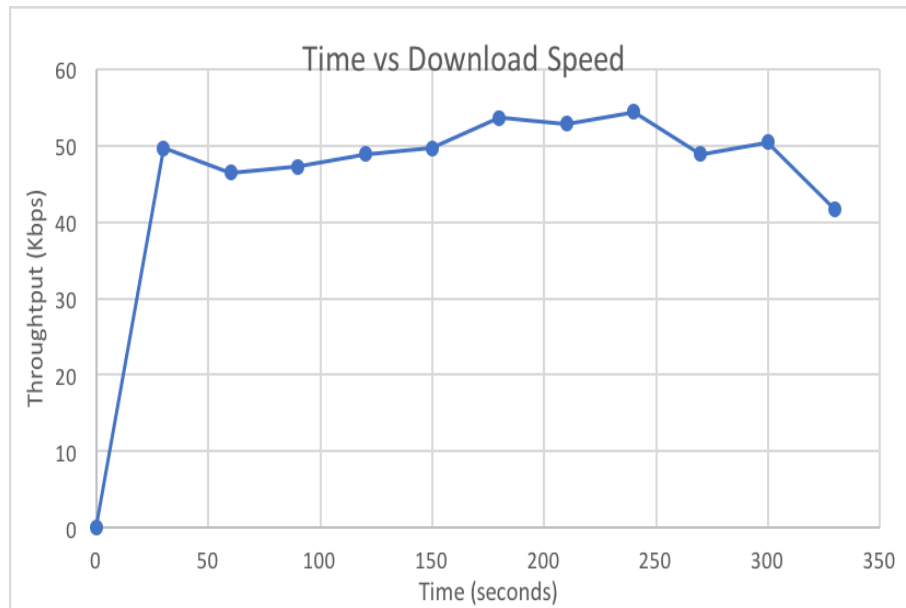


Figure 5.28: Ex1: 2 MB Download over a Single Virgin Channel (a)

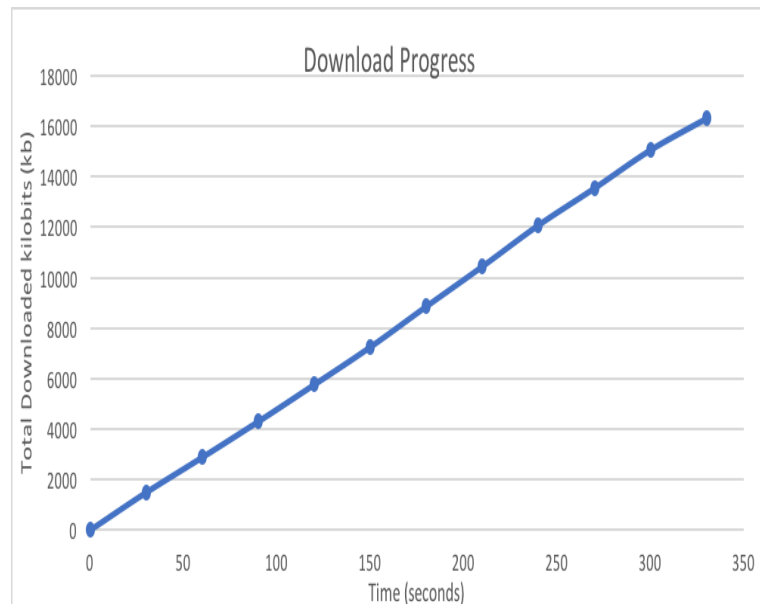


Figure 5.29: Ex1: 2 MB Download over a Single Virgin Channel (b)

### 5.5.6.2 Experiment 2

This experiment is the base/reference experiment to evaluate the QoS capability of a Lyca GPRS channel. A 2 MB data file was downloaded from the Internet over a single Lyca channel.

Observations:

1. We observed that the throughput observed during the download was in the range of 24.88 - 36 kilo bits per second (kbps). This throughput is considerable lower than the data rates achieved by the corresponding Virgin channel.
2. Download was completed in 540 seconds which is longer than the total download time of a single Virgin channel.
3. There were quite a few spikes in the throughput but there was no connection drop out.
4. The over all QoS observed is poor as compared to an individual Virgin channel. Hence, we classified Lyca as low QoS channel and Virgin as a high QoS channel for the following experiments.

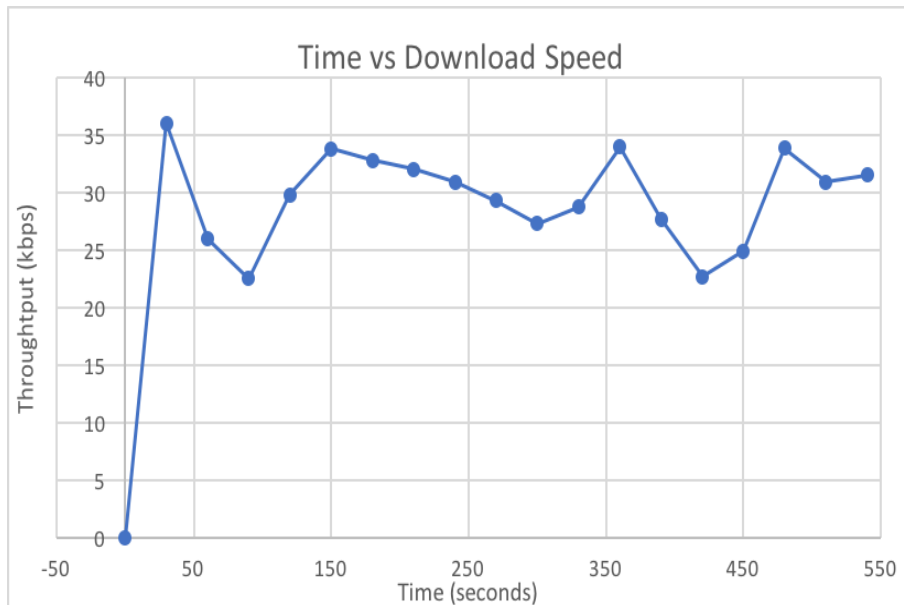


Figure 5.30: Ex2: 2 MB Download over Single Lyca Channel (a)



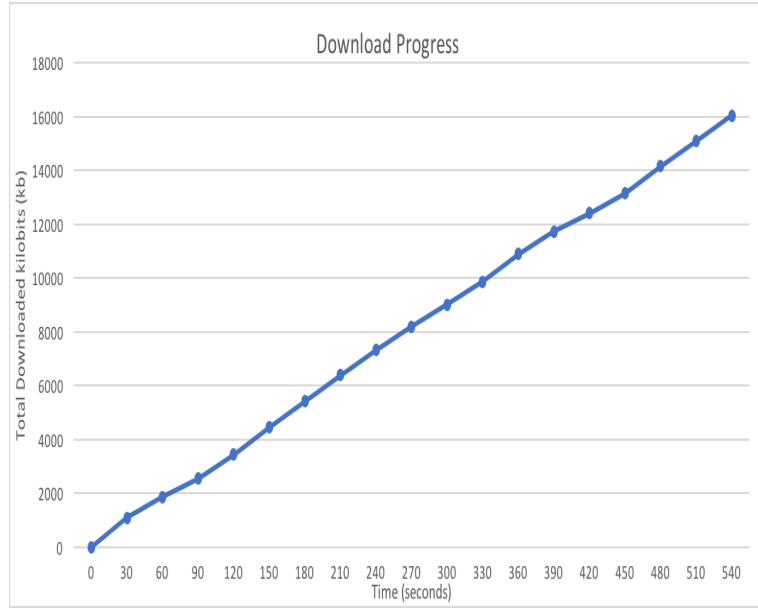


Figure 5.31: Ex2: 2 MB Download over Single Lyca Channel (b)

### 5.5.6.3 Experiment 3

This experiment is designed to compare the QoS performance of inverse multiplexed channels as compared to the individual channels and to evaluate the performance gains. Here, we observed the performance of Lyca channels while simultaneously downloading a 2 MB file from the Internet.

Observations:

1. The aggregated speed achieved was in the range of 33.6 - 60 kbps. This throughput improvement is huge as compared to the individual channel.
2. File download was completed in 301 seconds only. Hence, there is a reduction in the total download time, which saves battery of individual mobile device.
3. There were quite a few spikes in throughput but again there was no connection drop outs.

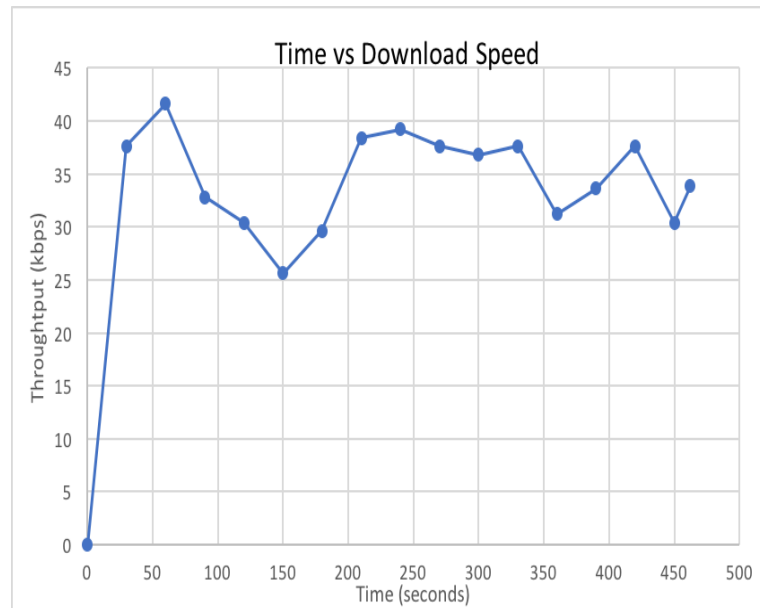


Figure 5.32: Ex3: 2 MB Download over Two Lyca Channels (a)

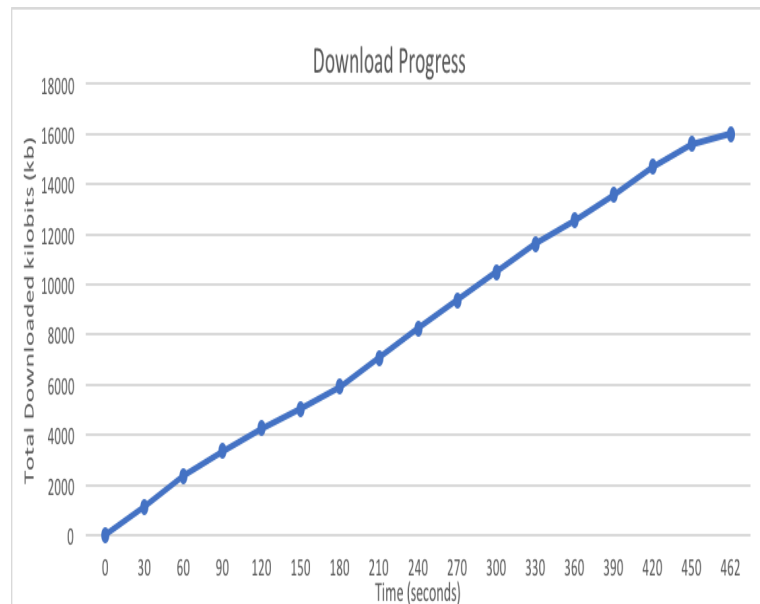


Figure 5.33: Ex3: 2 MB Download over Two Lyca Channels (b)

#### 5.5.6.4 Experiment 4

This experiment is designed to observe the QoS performance of the inverse multiplexing of heterogeneous channel. Here, we evaluated the performance of one virgin and two Lyca channels while simultaneously downloading a 2 MB file from the Internet. As we have mentioned earlier that we have already classified Virgin channels as high QoS capability and Lyca channels as low QoS capability based on results of experiment 1 and 2.

Observations:

1. The aggregated throughput achieved was between 67.2 - 122.4 kbps.
2. Total time for downloading was just 159 seconds, which is approximately half of total time taken by one Virgin channel and one third of the time taken by one Lyca channel to download the same file.
3. After 1 minute and 3 seconds, one of the networks stopped transferring data and the channel got disconnected. However, it was reconnected by the device manager in the next scan within a few seconds. You can see in the graph below that the throughput recovered after channel was included back in the MCC by our application.
4. Obviously aggregated throughput of three channels is far better than the aggregated throughput of two channels. It is important to note that spikes in throughput are not very frequent because aggregated channels compensate each other for short term changes to some extent.
5. A reduction in total download time means less battery power consumed by individual mobile device during the download.

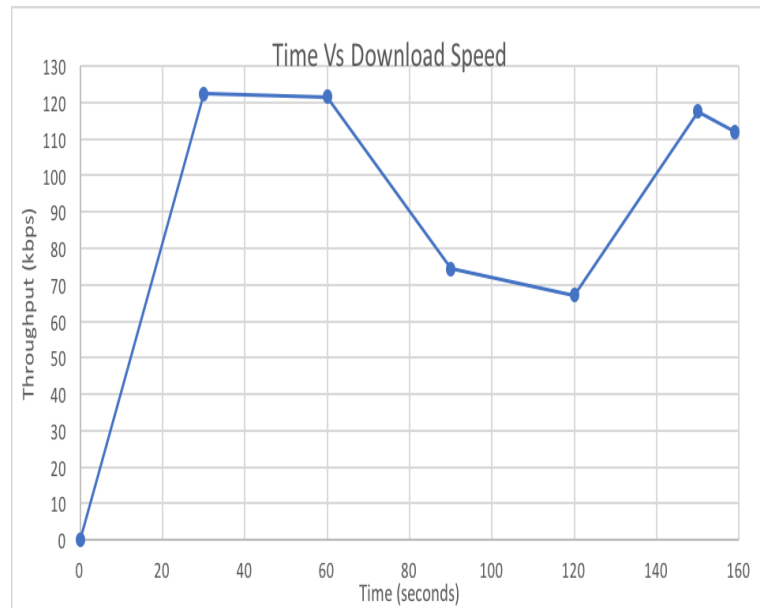


Figure 5.34: Ex4: 2 MB Download over one Virgin and Two Lyca Channels (a)

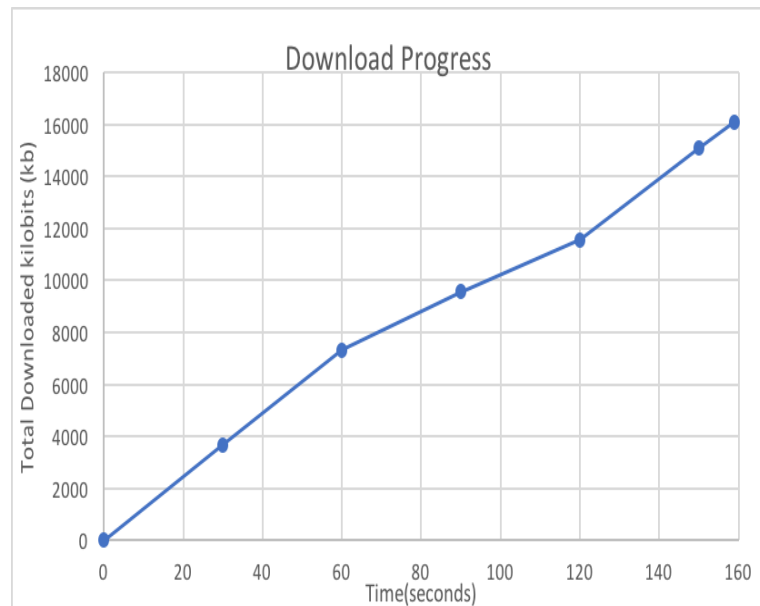


Figure 5.35: Ex4: 2 MB Download over one Virgin and Two Lyca Channels (b)

#### 5.5.6.5 Experiment 5

Next, we observed the QoS performance of 4 inverse multiplexed connections (2 Lyca and 2 Virgin) to download a 2 MB file. It is important to note that all connections were present throughout during the download process.

Observations:

1. The aggregated throughput achieved was in the range of 123.2- 158.4 kbps. Here minimum speed achieved was better than previous experiment where one node departed for some time.
2. The total download time for this experiment was 111 seconds. It is the best performance recorded by our MCC network running a MWQ algorithm to send more data on better quality Virgin channels.

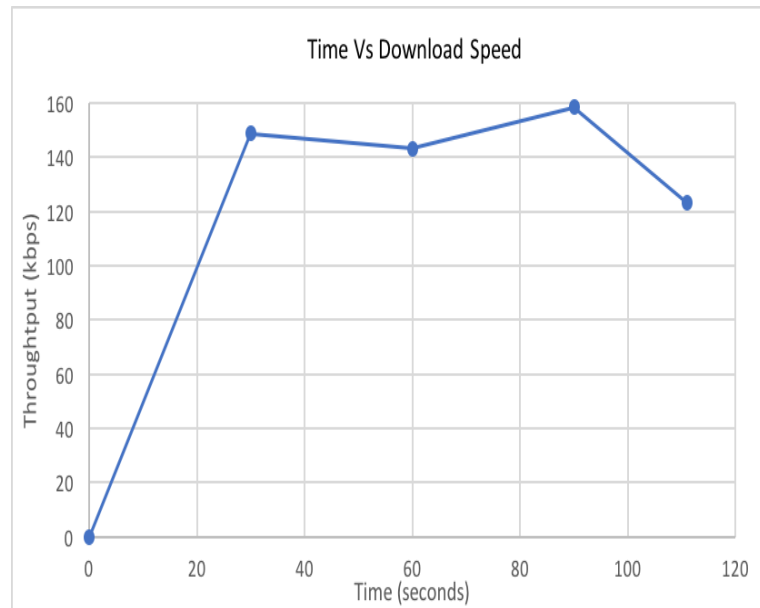


Figure 5.36: Ex5: 2 MB Download over two Virgin and Two Lyca Channels (a)

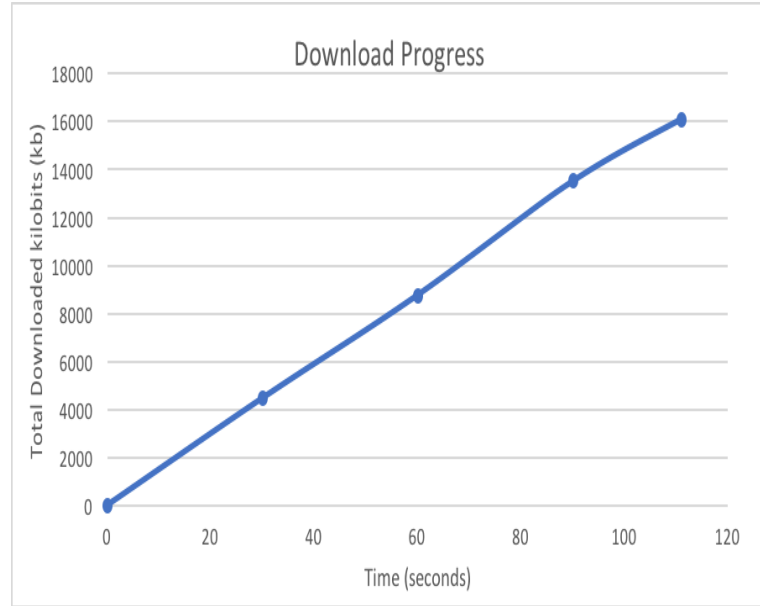


Figure 5.37: Ex5: 2 MB Download over two Virgin and Two Lyca Channels (b)

#### 5.5.6.6 Experiment 6

We repeated all experiments with RR, CP and SWQ algorithms and below is a comparison of total download time taken to finish a 2 MB download with four channels (same as in experiment 5).

Observations:

1. The MWQ performed better than other algorithms by allocating data intelligently over all channels.
2. The data was divided in the ratio of 60:40 where 60% data has high throughput as a requirement. RR algorithm tried to allocate the data equally on all channels, hence slow channels proved a bottleneck and it took longer to complete download as compared to smart scheduling of MWQ algorithm.
3. As data was divided in the ratio of 60:40, in case of CP algorithm, the throughput of the pinned channels dictated the overall throughput achieved, which was better than RR as we tied Virgin channels with 60% of the data.
4. The percentage improvement of MWQ in terms of total download time is 46% as compared to RR algorithm, 35% for CP algorithm and 17% for SWQ algorithm in the presence of four collaborating nodes.

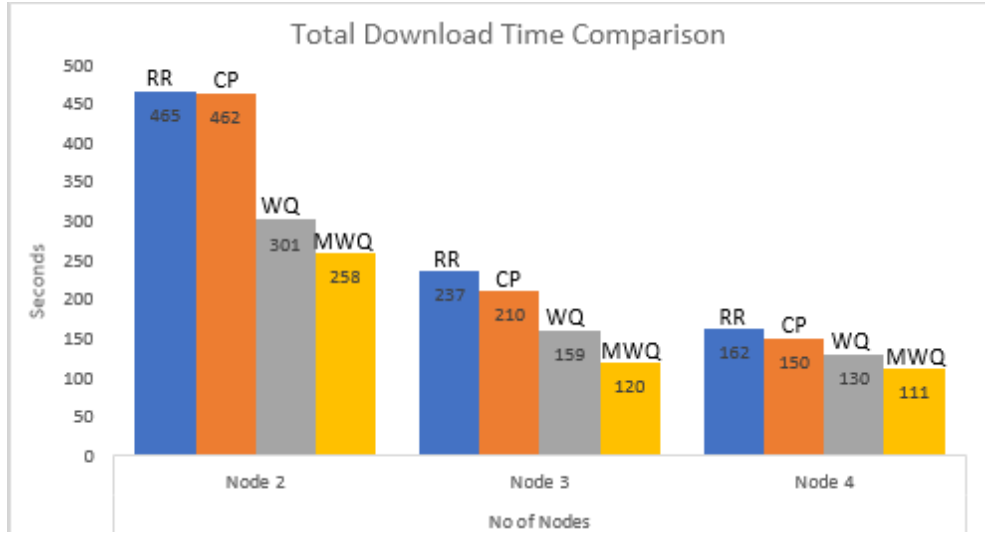


Figure 5.38: Comparison: 2 MB Download over four Channels using four Scheduling Techniques

### 5.5.7 Major Findings

Based on the above experimental results we can safely say that HTTP Downloader is able to utilise the bandwidth aggregation of collaborative channels intelligently. All the above experiments were using two levels of work-queues and majority of the data was tagged with better throughput requirement. Short terms changes as well as long term changes were handled by the system and they are reflected in the throughput graphs and download completion time graphs.

## 5.6 Chapter Summary

This chapter presented the QoS evaluation of our novel RAW scheduler. Firstly, the design of dynamic MWQ scheduling algorithm and three alternatives of MWQ algorithm have been presented. A trace driven simulator was used for the initial QoS assessments of the MWQ algorithm in controlled environment. We simulated short term QoS fluctuations in the channel conditions to observe the behavior and measure the performance gain of all four algorithms. Later, test-bed experiments were conducted for detailed study of RAW scheduler performance and comparison. An example application as well as other scripts of network formation were implemented to realise an example application. This application downloaded

HTTP data from the Internet by using WWAN channels of multiple mobile devices simultaneously. It has been found that RAW scheduler delivers better QoS results in terms of average throughput and total data transfer times by adapting to the changes in channel conditions. It has been found that, among the four alternatives the MWQ algorithm delivers the best results when we plugged it into our test-bed application.



# Chapter 6

## Conclusion and Future Work

The focus of this thesis is on achieving the best effort QoS in the MCCs during collaborative data transfer. This chapter summaries the work in this thesis, presents the conclusions drawn from the findings of this research, and gives recommendations for the future work.

### 6.1 Conclusion

A good knowledge about the area of collaborative data transfer in mobile communities, especially about workload scheduling algorithms, has been acquired by a thorough literature review. Several scenarios of collaborative data transfer have been analysed in order to specify the requirements and the challenges associated with them. Related work has been critically analysed against these requirements to find the missing bits.

Data transfer models employed by existing MCC solutions [17][18] have been found to be incomplete. These solutions have concentrated on the data downloading, specially multimedia streaming [1][17], but have ignored data uploading. The main cause was the inability of mobile phone handsets to generate and store large amounts of data, and also because they assumed that the same solution can work for the upstream data transfer [1]. This assumption is not valid, as in the case of collaborative downloading, a workload scheduler runs on a sever or a proxy in the Internet, but in the case of uploading, it resides on the resource constrained mobile source node. Hence, the same solution can not work for both. In addition to this, the existing systems have ignored data security and privacy issues. They either assumed that the same user owns all mobile devices [19], or

that performance alone is the main requirement [18]. The systems in [17][1] have briefly discussed data security issues and suggested to use encryption techniques or reputation system to protect against the malicious behaviour of collaborating nodes.

A preliminary RAW scheduler has been designed, keeping in mind the scenario requirements, resource limitations of mobile nodes and the ad-hoc environment of a MCC. Instead of using the computationally expensive encryption techniques, risk-aware scheduling has been designed for data security. A modular approach has been adapted to make the solution flexible, and components re-usable. Our proposed scheduler has five main components: (1) a DSM, to split the data according to its sensitivity levels and QoS requirements, and feed to corresponding WQs; (2) a WQM to manage WQ interactions and operations; (3) a TM to classify the collaborating nodes according to their trust levels; (4) an ACM to ensure that the collaborating nodes of a particular trust class can only access WQs of the corresponding sensitivity level; and (5) a CMM to coordinates the communication between the RAW scheduler and the community manager. It also provides an interface for the collaborating nodes and monitors long-term QoS changes in their WWAN channels.

The implementation and evaluation of the proposed MWQ scheduling algorithm have shown that it delivers better results for the average throughput and reduces total data transfer time by adapting to the changes in channel conditions. Three alternatives of the MWQ algorithm have been presented. The simulation as well as experimental methodology was used for the validation and comparison. A test-bed implementation was done to conduct the real world experiments. It has been found that, among the four alternatives the MWQ algorithm delivers the best results by adapting to changes in both long as well as short term channel conditions. These encouraging result have lead us to believe that by plugging a smart CAPAC and TM into the core RAW scheduler can generate even better results.

## 6.2 Suggestions for Future Research

The following presents four recommendations for the future research.

- **Energy Efficiency** Given that the battery power is a very expensive resource for mobile devices, a detailed study of energy efficiency gained by

using the proposed RAW scheduler is due for future work.

- **Access Control Studies** The results obtained from the MWQ evaluation are encouraging and suggest that better results can be produced after some further research. One possible future work could be the detailed design and implementation of an ACM that can define the smart access permissions for the MWQs.
- **Trust Management Studies** Further research is possible on the detailed design and evaluation of the TM. The evaluation of the proposed TM for the energy efficiency gain could be an interesting work for future.
- **Data Tagger Enhancements** Further fine grain enhancements can be done at the scheduler design level, specially in the data tagger module. It could be made more intelligent and energy efficient

In conclusion, the aim of this research is to achieve the best effort QoS in the MCCs during collaborative data transfer has been achieved by proposing the detailed design of the RAW scheduler, although a number of areas for the future work remain.

# Bibliography

- [1] K. H. Kim and K. G. Shin, “Prism: Improving the performance of inverse-multiplexed tcp in wireless networks,” *IEEE Transactions on Mobile Computing*, vol. 6, pp. 1297–1312, October 2007.
- [2] B. Solhaug and D. Elgesem, “Why trust is not proportional to risk,” *2nd International Conference on Availability, Reliability and Security ARES*, pp. 11–18, 2007.
- [3] S. Kozlov, P. van der Stok, and J. Lukkien, “Adaptive scheduling of mpeg video frames during real-time wireless video streaming,” in *Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, (Washington, DC, USA), pp. 460–462, IEEE, 2005.
- [4] S. Wang, C. Lin, and C. Huang, “The protocol developer manual for the nctuns 6.0 network simulator and emulator.” <http://csie.nqu.edu.tw/smallko/nctuns/DeveloperManual.pdf>, , visited = 15-01-2017.
- [5] S. Wang, C. Lin, and C. Huang, “Nctuns tool for evaluating the performances of real-life p2p applications,” in *Peer-to-Peer Networks and Internet Policies*, <http://nsl.cs.nctu.edu.tw/NSL/files/Nova2009.pdf>, visited = 15-01-2017: Nova Science Publishers, 2010.
- [6] “Gprs.” <http://www.gsmworld.com/technology/gprs.htm>, visited = 15-01-2017.
- [7] “The evolution of edge.” [http://www.ericsson.com/res/docs/whitepapers/evolution\\_to\\_edge.pdf](http://www.ericsson.com/res/docs/whitepapers/evolution_to_edge.pdf), visited = 15-01-2017.
- [8] “Ieee 802.11: Wireless lan medium access control (mac) and physical layer (phy) specifications,” tech. rep., <http://standards.ieee.org/getieee802/download/802.11-2007.pdf>, 2007.

- [9] “Bluetooth technology website.” <http://www.bluetooth.com/>, visited = 15-01-2017.
- [10] P. H. Fredette, “The past, present, and future of inverse multiplexing,” *IEEE Communications Magazine*, vol. 32, pp. 42–46, April 1994.
- [11] S. S. Channappayya, G. P. Abousleman, and L. J. Karam, “Coding of digital imagery for transmission over multiple noisy channels,” in *ICASSP ’01: Proceedings of the Acoustics, Speech, and Signal Processing, 2001. on IEEE International Conference*, (Washington, DC, USA), pp. 1729–1732, IEEE Computer Society, 2001.
- [12] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee, “Mar: a commuter router infrastructure for the mobile internet,” in *MobiSys ’04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, (New York, NY, USA), pp. 217–230, ACM, 2004.
- [13] R. Mahajan, J. Padhye, S. Agarwal, and B. Zill, “E PluriBus Unum: High Performance Connectivity On Buses,” tech. rep., Microsoft Research, 2008.
- [14] R. Chakravorty, S. Banerjee, and S. Ganguly, “MobiStream: Error-Resilient Video Streaming in Wireless WANs Using Virtual Channels,” in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–14, IEEE, April 2006.
- [15] Y. He and J. Brassil, “NATALIE: An Adaptive, Network-Aware Traffic Equalizer,” in *Proceedings of IEEE International Conference on Communications*, pp. 595–602, June 2007.
- [16] A. Qureshi and J. Guttag, “Horde: separating network striping policy from mechanism,” in *MobiSys ’05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, (New York, NY, USA), pp. 121–134, ACM, 2005.
- [17] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath, and C. A. Thekkath, “Combine: leveraging the power of wireless peers through collaborative downloading,” in *MobiSys ’07: Proceedings of the 5th international conference on Mobile systems, applications and services*, (New York, NY, USA), pp. 286–298, ACM, 2007.

- [18] P. Sharma, S. J. Lee, J. Brassil, and K. G. Shin, “Handheld routers: Intelligent bandwidth aggregation for mobile collaborative communities,” in *BROADNETS '04: Proceedings of the First International Conference on Broadband Networks*, (Washington, DC, USA), pp. 537–547, IEEE Computer Society, October 2004.
- [19] C. Carter and R. Kravets, “User devices cooperating to support resource aggregation,” *Mobile Computing Systems and Applications, IEEE Workshop on*, vol. 0, pp. 59–69, 2002.
- [20] H. Luo, R. Ramjee, P. Sinha, L. E. Li, and S. Lu, “Ucan: a unified cellular and ad-hoc network architecture,” in *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 353–367, ACM, 2003.
- [21] S. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, C. Chiou, and C. Lin, “The design and implementation of the {NCTUns} 1.0 network simulator,” *Computer Networks*, vol. 42, no. 2, pp. 175 – 197, 2003.
- [22] S. Saleem and N. Zhang, “A risk-aware workload scheduler to support secure and efficient collaborative data transfer in mobile communities.,” in *Proceedings of 9th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pp. 31–34, IEEE, 2012.
- [23] J. C. Funk, H. R. Duffin, L. Dai, and C. D. Knutson, “Inverse multiplexing in short-range multi-transport wireless communications,” in *Wireless Communications and Networking (WCNC)*, pp. 757–762, IEEE, 2003.
- [24] J. K. Petersen, *The telecommunications illustrated*. CRC Press, 2nd ed., 2002.
- [25] “Gsm.” <http://www.gsma.com/aboutus/gsm-technology/gsm>, visited = 15-01-2017.
- [26] L. B. Milstein, “Wideband code division multiple access,” *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 1344–1354, Aug 2000.
- [27] “Wimax ieee 802.16 technology tutorial.” <http://www.radio-electronics.com/info/wireless/wimax/wimax.php>, visited = 15-01-2017.

- [28] “Long term evolution (lte): A technical overview.” [http://www.3g4g.co.uk/Lte/LTE\\_WP\\_0706\\_Motorola.pdf](http://www.3g4g.co.uk/Lte/LTE_WP_0706_Motorola.pdf), published by Motorola Inc., visited = 15-01-2017.
- [29] A. C. Snoeren, “Adaptive inverse multiplexing for wide-area wireless networks,” in *GlobeCom’99*, IEEE, 1999.
- [30] K. Chebrolu and R. R. Rao, “Bandwidth aggregation for real-time applications in heterogeneous wireless networks,” *IEEE Transactions on Mobile Computing*, vol. 5, pp. 388–403, April 2006.
- [31] H. Y. Hsieh and R. Sivakumar, “ptcp: An end-to-end transport layer protocol for striped connections,” in *ICNP ’02: Proceedings of the 10th IEEE International Conference on Network Protocols*, (Washington, DC, USA), pp. 24–33, IEEE Computer Society, 2002.
- [32] “Information and communication technology usage in the 2010 pakistan floods,” tech. rep., <https://s3.amazonaws.com/nethope/ICTin2010PakistanFloods.pdf>, 2011.
- [33] F. Legendre, “30 years of ad hoc networking research: What about humanitarian and disaster relief solutions? what are we still missing?,” in *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*, ACWR ’11, (New York, NY, USA), pp. 217–217, ACM, 2011.
- [34] N. Pogkas, G. E. Karastergios, C. D. Antonopoulos, S. A. Koubias, and G. D. Papadopoulos, “Architecture design and implementation of an ad-hoc network for disaster relief operations,” *IEEE Trans. Industrial Informatics*, vol. 3, no. 1, pp. 63–72, 2007.
- [35] H. Adishesu, G. Parulkar, and G. Varghese, “A reliable and scalable striping protocol,” in *SIGCOMM ’96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 131–141, ACM, 1996.
- [36] Z. Zhou and M. Hamdi, “Randomized batch scheduling with minimum configurations for switches and routers,” in *2007 Workshop on High Performance Switching and Routing*, pp. 1–6, IEEE, May 2007.

- [37] G. Cheung, P. Sharma, and S.-J. Lee, “Smart media striping over multiple burst-loss channels,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, pp. 319–333, August 2007.
- [38] L. Magalhaes and R. Kravets, “Mmtp — multimedia multiplexing transport protocol,” in *SIGCOMM LA '01: Workshop on Data communication in Latin America and the Caribbean*, (New York, NY, USA), pp. 220–243, ACM, 2001.
- [39] R. K. Luiz Magalhaes, “End-to-end inverse multiplexing for mobile hosts,” *Journal of Brazilian Computer Science*, vol. 7, pp. 52–62, 2001.
- [40] S. Keshav, “A control-theoretic approach to flow control,” *SIGCOMM Comput. Commun. Rev.*, vol. 21, pp. 3–15, September 1991.
- [41] L. Magalhaes and R. Kravets, “Transport level mechanisms for bandwidth aggregation on mobile hosts,” in *ICNP '01: Proceedings of the Ninth International Conference on Network Protocols*, (Washington, DC, USA), pp. 165+, IEEE Computer Society, 2001.
- [42] “Coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s - part 2: Video iso/iec 11172-2,” tech. rep., [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=22411](http://www.iso.org/iso/catalogue_detail.htm?csnumber=22411), 1993.
- [43] J. Apostolopoulos, “Error-resilient video compression via multiple state streams,” *International Workshop on Very Low Bitrate Video Coding (VLBV)*, pp. 168–171, 1999.
- [44] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan, “System support for bandwidth management and content adaptation in internet applications,” in *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, (Berkeley, CA, USA), p. 15, USENIX Association, 2000.
- [45] “Iso guide: Risk management vocabulary.” [http://www.iso.org/iso/catalogue\\_detail.htm-?csnumber=44651](http://www.iso.org/iso/catalogue_detail.htm-?csnumber=44651), visited = 15-01-2017.
- [46] D. H. Mcknight and N. L. Chervany, “The meanings of trust,” tech. rep., <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.155.1213>, 1996.



- [47] Z. Yan, “Doctoral thesis: Trust management for mobile computing platforms,” tech. rep., Helsinki University of Technology, <http://lib.tkk.fi/Diss/2007/isbn9789512291205/>, 2007.
- [48] A. Jøsang and S. L. Presti, “Analysing the relationship between risk and trust,” in *Proceedings of the 2nd International Conference on Trust Management*, pp. 135–145, 2004.
- [49] R. V. Nehme, E. A. Rundensteiner, and E. Bertino, “Tagging stream data for rich real-time services,” In *Proceedings of the Very Large Data Bases (VLDB) Endowment*, vol. 2, pp. 73–84, Aug. 2009.
- [50] E. Choo, J. Lee, H. Lee, and G. Nam, “Srmt: A lightweight encryption scheme for secure real-time multimedia transmission,” In *Proceedings of International Conference on Multimedia and Ubiquitous Engineering (MUE)*, 2007.
- [51] X. Liu and A. M. Eskicioglu, “Selective encryption of multimedia content in distribution networks: challenges and new directions,” in *International Conference on Communication, Internet, and Information Technology*, pp. 527–533, 2003.
- [52] Y. Ren, A. Boukerche, and L. Mokdad, “Performance analysis of a selective encryption algorithm for wireless ad hoc networks,” in *Wireless Communications and Networking Conference (WCNC)*, pp. 1038–1043, IEEE, 2011.
- [53] A. Massoudi, F. Lefebvre, C. De Vleeschouwer, B. Macq, and J.-J. Quisquater, “Overview on selective encryption of image and video: Challenges and perspectives,” *EURASIP Journal on Information Security*, vol. 2008, pp. 5:1–5:18, Jan. 2008.
- [54] W. Wang, M. Hempel, D. Peng, H. Wang, H. Sharif, and H.-H. Chen, “On energy efficient encryption for video streaming in wireless sensor networks,” *IEEE Transactions on Multimedia*, vol. 12, pp. 417–426, Aug. 2010.
- [55] L. Eschenauer, V. D. Gligor, and J. Baras, “On trust establishment in mobile ad-hoc networks,” in *In Proceedings of the Security Protocols Workshop*, pp. 47–66, 2002.

- [56] H. Tran, M. Hitchens, V. Varadharajan, and P. Watters, "A trust based access control framework for p2p file-sharing systems," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS)*, (Washington, DC, USA), p. 302c, IEEE Computer Society, 2005.
- [57] F. Feng, C. Lin, D. Peng, and J. Li, "A trust and context based access control model for distributed systems," in *Proceedings of 10th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 629–634, 2008.
- [58] T. Beth, M. Borchering, and B. Klein, "Valuation of trust in open networks," in *Proceedings of Third European Symposium on Research in Computer Security (ESORICS)*, pp. 3–18, 1994.
- [59] L. Capra, "Engineering human trust in mobile system collaborations," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 6, pp. 107–116, 2004.
- [60] "Coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s - part 1: Systems." [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=19180](http://www.iso.org/iso/catalogue_detail.htm?csnumber=19180), visited = 15-01-2017.
- [61] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, pp. xviii–xxxiv, Feb 1992.
- [62] S. Boll, W. Klas, M. Menth, and C. Heinlein, "Mpeg-l/mrp: implementing adaptive streaming of mpeg videos for interactive internet applications," in *Proceedings of the ninth ACM international conference on Multimedia (MULTIMEDIA)*, (New York, NY, USA), pp. 626–627, ACM, 2001.
- [63] H. Zhang and S. Rangarajan, "Adaptive scheduling of streaming video over wireless networks," in *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 477–480, IEEE, June 2008.
- [64] E. Nordstrom and P. Gunningberg, "A testbed and methodology for experimental evaluation of wireless mobile ad hoc networks," in *Proceedings of 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pp. 100–109, IEEE, 2005.

- [65] J. Vitek and T. Kalibera, “Repeatability, reproducibility, and rigor in systems research,” in *Proceedings of the Ninth International Conference on Embedded Software*, EMSOFT ’11, (New York, NY, USA), pp. 33–38, ACM, 2011.
- [66] D. Maltz, J. Broch, and D. B. Johnson, “Experiences designing and building a multi-hop wireless ad hoc network testbed,” 1999. <https://www.cs.rice.edu/~dbj/pubs/CMU-CS-99-116.pdf>, visited = 15-01-2017.
- [67] E. Weingärtner, H. Vom Lehn, and K. Wehrle, “A performance comparison of recent network simulators,” in *Proceedings of International Conference on Communications*, ICC’09, (Piscataway, NJ, USA), pp. 1287–1291, IEEE Press, 2009.
- [68] L. F. Perrone, Y. Yuan, and D. M. Nicol, “Modeling and simulation best practices for wireless ad hoc networks,” in *Proceedings of the 35th Winter Simulation Conference (WSC)*, pp. 685–693, Winter Simulation Conference, 2003.
- [69] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. <https://bayanbox.ir/view/3697480237749666667/Introduction-to-Network-Simulator-NS2-2012.pdf>, , visited = 15-01-2017: Springer Publishing Company, 2nd ed., 2011.

# Appendix A

## WLANs Vs. WWANs: Throughput Comparison

### A.1 Typical Throughput Rate of WLAN Technologies

Technology	Data Rate
802.11a (Wi-Fi5)	54 Mbps
802.11b (Wi-Fi)	11 Mbps
802.11g	54 Mbps
bluetooth	1-3 Mbps
UWB	675 Mbps

Table A.1: Typical Throughput Rate of WLAN Technologies.

## A.2 Typical Throughput Rate of WWAN Technologies

Technology	Data Rate(Uplink)	Data Rate(Downlink)
GPRS	20-40 Kbps	60-80 Kbps
EDGE	59 Kbps	236 Kbps
EDGE-2	1.2 Mbps	474 Kbps
WCDMA	384 Kbps	384 Kbps
HSUPA	.73-11.5 Mbps	N/A
HSPA+	Up to 22 Mbps	Up to 42 Mbps
EVDO Rel. 0	.15 Mbps	2.45 Mbps
EVDO Rel. A	1.8 Mbps	3.1 Mbps
EVDO Rel. B	3.6 Mbps	9.6 Mbps
WiMAX (802.16e)	6.4 Mbps	9.5 Mbps
WiMAX II (802.16m)	Up to 130 Mbps	Up to 130 Mbps
LTE	Up to 86 Mbps	Up to 326 Mbps

Table A.2: Typical Throughput Rate of WWAN Technologies.

# Appendix B

## NCTUns Simulation Details

### B.1 NCTUns Network Scalability Problem

NCTUns is a very wise choice for our simulation scenario due to its support for multi-interface node and ability to run real time applications on any network node. The simulator has become a commercial product named Estinet since 2009. Due to this reason, fewer researchers are using it for the simulations recently. Hence, developer forums are not active. The developers manuals are outdated and incomplete. Furthermore, the simulator code available still has some bugs which might have been fixed in the recent releases but these are not available free of cost.

We did experience a major setback soon into our simulation, our network topology was not scaling according to our scenario requirements. The problem arises when we connect the sender with the receiver through multiple paths and try to send data on each path simultaneously. The simulator is supporting simultaneous transfer of data on two paths but when we try to use a third path, the data do not get transferred through it. It caused a hindrance in simulating more than two collaborators and several network scenarios. We spent few months trying to find and fix this bug. After lots of research and code debugging, we decided to contact Estinet (commercial version of NCTUns) support team to get the commercial product but during trial it became evident that problem lies with the software, and it does not support the functionality required by us. Later Estinet team confirmed through email that this scaling issue is a bug in their software and they will try to fix it in next release.

## B.2 Simulation Code

Given below are the files corresponding to the implementation of four scheduling algorithms. As NCTUns allows to run any application code on a network node, I run them on the sender node, one by one through GUI using commands in the following format. Here xxx is the abbreviation of algorithm's name.

```
xxx_sender /*path of config file*/xxx_config.txt
```

where xxx\_config.txt has a list of Local IPs and corresponding server IPs. At the end, it has size of data to be transferred. It is important to note that different algorithms have different configuration file formats for the data section depending upon the nature of scheduling algorithm.

### B.2.1 Round Robin Algorithm Code

```
***** rr_sender.c *****
*****
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <pthread.h>

#define PORT 3490 // the port client will be connecting to
#define MAXDATASIZE 200 // max number of bytes we can get at once
#define MAXCOLABORATORS 10
#define MAXLINELENGTH 256

char* remove_newline(char *s)
{
    int len = strlen(s);
```

```

        if (len > 0 && s[len-1] == '\n'){ // if there's a newline
            s[len-1] = '\0';           // truncate the string
        }
        return s;
    }

int main(int argc, char *argv[])
{
    /***/
    if(argc!=2){
printf("Usage: ./rr_sender Recvr_First_IP Recvr_Second_IP\n");
printf("sndr_First_IP sndr_Second_IP, data\n");
return 0;
    }
    int i = 0;
    int err, name_len=0;
    int linenum = 0;
    char line[MAXLINELENGTH];
    char *serverIPs[MAXCOLABORATORS] = {'\0'};
    char *localIPs[MAXCOLABORATORS] = {'\0'};
    long int data[MAXCOLABORATORS];
    int counter[MAXCOLABORATORS];
    int cur_colaborators=0;
    printf("File to open is %s\n", argv[1]);
    FILE *fin;
    if ((fin = fopen(argv[1], "r")) == NULL){
        fprintf(stderr, "Unable to open config file rr-config.txt!\n");
        exit(1);
    }
    /*** to read server IPs ***/
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        int testlen = 0;
        testlen = strlen(line);

```



```

        serverIPs[linenum] = malloc(testlen + 1);
        strcpy (serverIPs[linenum], remove_newline(line));
        linenum++;
    }

    /***** to read local IPs *****/
    linenum=0;
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        int testlen;
        testlen = strlen(line);
        localIPs[linenum] = malloc(testlen + 1);
        strcpy (localIPs[linenum], remove_newline(line));
        linenum++;
        cur_colaborators++;
    }

    /***** to get data size *****/
    linenum=0;
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        sscanf(line, "%ld", &data[linenum++]);
    }

    /*****/

    for(i=0;i<cur_colaborators;i++){
        // serverIPs[i]="127.0.0.1"; //Dummey
        //localIPs[i]="127.0.0.1"; //Dummey
        //data[0]=50000000; //Dummey
        counter[i]=0;
    }

    int sockfd[MAXCOLABORATORS], numbytes;
    char buf[MAXDATASIZE];
    int buf_len=0;

```

```

buf[0]='\0';
struct hostent *he[MAXCOLABORATORS];
struct sockaddr_in their_addr[MAXCOLABORATORS],
local_addr[MAXCOLABORATORS]; // connector's address information
/*if(argc!=6){
printf("Usage: ./rr_sender Recvr_First_IP Recvr_Second_IP
sndr_First_IP sndr_Second_IP, data\n");
return 0;
}*/
for(i=0;i<cur_colaborators; i++){
    printf("Server[%d]=%s\n", i, serverIPs[i]);
    printf("local[%d]=%s\n", i, localIPs[i]);
}
for(i=0;i<cur_colaborators; i++){
    if ((he[i]=gethostbyname(serverIPs[i])) == NULL) {
// get the host info
herror("gethostbyname");
exit(1);
    }

    if ((sockfd[i] = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket");
exit(1);
    }
    int yes =1;
    if (setsockopt(sockfd[i], SOL_SOCKET, SO_REUSEADDR,
&yes, sizeof(int)) == -1) {
perror("setsockopt");
exit(1);
    }

    local_addr[i].sin_family = AF_INET;

    local_addr[i].sin_addr.s_addr = inet_addr(localIPs[i]);

```

```

        if(bind(sockfd[i], (struct sockaddr *) &local_addr[i],
sizeof(local_addr[i]))==-1)
        {
perror("Client: bind");
exit(1);
        }

        their_addr[i].sin_family = AF_INET;    // host byte order
            their_addr[i].sin_port = htons(PORT);
// short, network byte order
        their_addr[i].sin_addr = *((struct in_addr *)he[i]->h_addr);
        memset(their_addr[i].sin_zero, '\0', sizeof their_addr[i].sin_zero);
        if (connect(sockfd[i], (struct sockaddr *)&their_addr[i],
sizeof their_addr[i]) == -1) {
perror("connect");
exit(1);
        }
    }

    sprintf(buf,"The Quick Brown Fox Jumps over the Lazy Dog. The Quick
Brown Fox Jumps over the Lazy Dog");
    int len=strlen(buf);
    int data_sent=0;

    while(data_sent < data[0]){
        for(i=0; i<cur_colaborators; i++){
if (send(sockfd[i], buf, strlen(buf), 0) == -1)
perror("Client: send");
        counter[i]+=len;
        data_sent+=len;
        if(data_sent>=data[0]) break;
        }
    }

    for(i=0; i<cur_colaborators; i++){
        printf("Client: Total Data Sent on path %d: %d\n", i, counter[i]);
        close(sockfd[i]);
    }

```

```

}
printf("\nClient: Total Data Sent: %d\n", data[0]);
    return 0;
}

```

```

***** rr_config.txt *****
*****
1.0.1.1
1.0.1.1
#
1.0.6.2
1.0.7.2
#
50000000
#

```

## B.2.2 Channel Pinned Algorithm Code

```

***** cp_sender.c *****
*****
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <pthread.h>

#define PORT 3490 // the port client will be connecting to
#define MAXDATASIZE 200 // max number of bytes we can get at once
#define MAXCOLABORATORS 10
#define MAXLINELENGTH 256

```

```

char* remove_newline(char *s)
{
    int len = strlen(s);
    if (len > 0 && s[len-1] == '\n'){ // if there's a newline
        s[len-1] = '\0';           // truncate the string
    }
    return s;
}

pthread_t tid[MAXCOLABORATORS];
struct arg {
    char *server_ip;
    char *local_ip;
    uint16_t port;
    long data_size;
};
typedef struct arg arg;
void* doSomething(void *);

int main(int argc, char *argv[])
{
    /*****/
    if(argc!=2){
printf("Usage: ./rr_sender Recvr_First_IP Recvr_Second_IP\n");
    sndr_First_IP sndr_Second_IP, data\n");
    return 0;
    }

    int i = 0;
    int err, linenum=0;
    char *serverIPs[MAXCOLABORATORS];
    char *localIPs[MAXCOLABORATORS];
    long data[MAXCOLABORATORS];
    char line[MAXLINELENGTH];

```

```

    int cur_colaborators=0;
    FILE *fin;
    if ((fin = fopen(argv[1], "r")) == NULL){
        fprintf(stderr, "Unable to open config file!\n");
        exit(1);
    }
    /***** to read server IPs *****/
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        int testlen = 0;
        testlen = strlen(line);
        serverIPs[linenum] = malloc(testlen + 1);
        strcpy (serverIPs[linenum], remove_newline(line));
        linenum++;
    }

    /***** to read local IPs *****/
    linenum=0;
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        int testlen;
        testlen = strlen(line);
        localIPs[linenum] = malloc(testlen + 1);
        strcpy (localIPs[linenum], remove_newline(line));
        linenum++;
        cur_colaborators++;
    }

    /***** to get data size *****/
    linenum=0;
    for(i=0; i<cur_colaborators; i++)
    data[i]=0;
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {

```

```

        if(line[0] == '#') break;
        sscanf(line, "%ld", &data[linenum++]);
    }

    /*****/

    for(i=0;i<cur_colaborators;i++){

        err = pthread_create(&(tid[i]), NULL, &doSomething,
        &(arg) {serverIPs[i], localIPs[i], 3490, data[i]});
        if (err != 0)
            printf("\ncan't create thread %d:[%s]", i, strerror(err));
    }
    for(i=0;i<cur_colaborators;i++){
        pthread_join(tid[i], NULL);
    }

    return 0;
}

void* doSomething(void *ptr)
{

    arg *x = ptr;
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    long counter=0;
    long queue_size;
    queue_size=x->data_size;
    struct hostent *he;
    struct sockaddr_in their_addr, local_addr;
    // connector's address information
    if ((he=gethostbyname(x->server_ip)) == NULL) {
        // get the host info
        perror("gethostbyname");
    }

```

```

        exit(1);
    }
    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
    int yes = 1;
        if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
&yes, sizeof(int)) == -1) {
            perror("setsockopt");
            exit(1);
        }

    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = inet_addr(x->local_ip);
    if(bind(sockfd, (struct sockaddr *) &local_addr,
sizeof(local_addr))== -1)
    {
        perror("Client: bind");
        exit(1);
    }

    their_addr.sin_family = AF_INET;    // host byte order
    their_addr.sin_port = htons(x->port);
    // short, network byte order
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(their_addr.sin_zero, '\0', sizeof their_addr.sin_zero);
    if (connect(sockfd, (struct sockaddr *)&their_addr,
sizeof their_addr) == -1) {
        perror("connect");
        exit(1);
    }
    int i;
    sprintf(buf, "The Quick Brown Fox Jumps over the Lazy Dog. The Quick
Brown Fox Jumps over the Lazy Dog");

```



```

int len=strlen(buf);
printf("This Data = %ld\n", queue_size);
while(counter < queue_size){

if (send(sockfd, buf, strlen(buf), 0) == -1)
perror("send");
counter+=len;
}
printf("Total Data Sent in this thread: %d\n", counter);

close(sockfd);

return NULL;
}

***** cp_config.txt *****
1.0.1.1
1.0.1.1
#
1.0.8.2
1.0.9.2
#
30000000
20000000
#
*****

```

### B.2.3 Single Work Queue Algorithm Code

```

***** swq_sender.c *****
*****
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

```

```

#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <pthread.h>

#define PORT 3490 // the port client will be connecting to
#define MAXDATASIZE 200 // max number of bytes we can get at once
#define MAXCOLABORATORS 10
#define MAXLINELENGTH 256

char* remove_newline(char *s)
{
    int len = strlen(s);
    if (len > 0 && s[len-1] == '\n'){ // if there's a newline
        s[len-1] = '\0';           // truncate the string
    }
    return s;
}

pthread_t tid[MAXCOLABORATORS];
long counter;
int line;
pthread_mutex_t lock;

struct arg {
    char *server_ip;
    char *local_ip;
    uint16_t port;
    long data;
};

typedef struct arg arg;
void* doSomething(void *);

```

```

int main(int argc, char *argv[])
{
/*****/
    if(argc!=2){
printf("Usage: ./rr_sender Recvr_First_IP Recvr_Second_IP
sndr_First_IP sndr_Second_IP, data\n");
return 0;
    }

    int i = 0;
    int err;
    line=0;

    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n mutex init failed\n");
        return 1;
    }
    char *serverIPs[MAXCOLABORATORS];
    char *localIPs[MAXCOLABORATORS];
    long data[MAXCOLABORATORS];
    char line[MAXLINELENGTH];
    int cur_colaborators=0;
    int linenum=0;
    FILE *fin;
    if ((fin = fopen(argv[1], "r")) == NULL){
        fprintf(stderr, "Unable to open config file!\n");
        exit(1);
    }
/***** to read server IPs *****/
while(fgets(line, MAXLINELENGTH, fin) != NULL)
{
    if(line[0] == '#') break;
    int testlen = 0;

```

```

        testlen = strlen(line);
        serverIPs[linenum] = malloc(testlen + 1);
        strcpy (serverIPs[linenum], remove_newline(line));
        linenum++;
    }

/***** to read local IPs *****/
linenum=0;
while(fgets(line, MAXLINELENGTH, fin) != NULL)
{
    if(line[0] == '#') break;
    int testlen;
    testlen = strlen(line);
    localIPs[linenum] = malloc(testlen + 1);
    strcpy (localIPs[linenum], remove_newline(line));
    linenum++;
    cur_colaborators++;
}

/***** to get data size *****/
linenum=0;
for(i=0; i<cur_colaborators; i++)
data[i]=0;
while(fgets(line, MAXLINELENGTH, fin) != NULL)
{
    if(line[0] == '#') break;
    sscanf(line, "%ld", &data[linenum++]);
}

/*****/
/* Dummy Entering of Colaborators and Sender/Receiver
Data until File program in incomplete */
for(i=0;i<cur_colaborators;i++){
    //serverIPs[i]="127.0.0.1"; //Dummy
    //localIPs[i]="127.0.0.1"; //Dummy
    //data[i]=50000000; //Dummy
    err = pthread_create(&(tid[i]), NULL, &doSomething,

```

```

&(arg) {serverIPs[i], localIPs[i], 3490, data[0]});
        if (err != 0)
            printf("\ncan't create thread %d:[%s]", i, strerror(err));
    }
    for(i=0;i<cur_colaborators;i++){
        pthread_join(tid[i], NULL);
    }
    /***** Till Here *****/
    pthread_mutex_destroy(&lock);
    /*****/
    return 0;
}

void* doSomething(void *ptr)
{

    arg *x = ptr;
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    int buf_len=0;
    int loc_counter=0;
    buf[0]='\0';
    struct hostent *he;
    struct sockaddr_in their_addr, local_addr;
    // connector's address information
    if ((he=gethostbyname(x->server_ip)) == NULL) {
        // get the host info
        perror("gethostbyname");
        exit(1);
    }
    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
    int yes =1;

```

```

        if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
&yes, sizeof(int)) == -1) {
            perror("setsockopt");
            exit(1);
        }
local_addr.sin_family = AF_INET;
local_addr.sin_addr.s_addr = inet_addr(x->local_ip);
if(bind(sockfd, (struct sockaddr *) &local_addr,
sizeof(local_addr))== -1)
{
    perror("Client: bind");
    exit(1);
}

their_addr.sin_family = AF_INET;
// host byte order
their_addr.sin_port = htons(x->port);
// short, network byte order
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(their_addr.sin_zero, '\0', sizeof their_addr.sin_zero);
if (connect(sockfd, (struct sockaddr *)&their_addr,
sizeof their_addr) == -1) {
    perror("connect");
    exit(1);
}

int i;
sprintf(buf, "The Quick Brown Fox Jumps over the Lazy Dog. The Quick
Brown Fox Jumps over the Lazy Dog");
int len=strlen(buf);

do{
    pthread_mutex_lock(&lock);
    if(counter > x->data) {pthread_mutex_unlock(&lock); break;}
    counter+=len;
    loc_counter+=len;

```

```

//x->data-=len;
pthread_mutex_unlock(&lock);
if (send(sockfd, buf, strlen(buf), 0) == -1)
perror("send");
}while(1);

printf("Total Data Sent in this thread: %d\n", loc_counter);
//printf("Total Data Sent by ALL threads: %d\n", counter);
close(sockfd);
return NULL;
}

```

```

***** swq_config.txt *****
*****
1.0.1.1
1.0.1.1
#
1.0.8.2
1.0.9.2
#
50000000
#

```

## B.2.4 Multi-level Work Queue Algorithm Code

```

***** mwq_sender.c *****
*****
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>

```

```

#include <netinet/in.h>
#include <sys/socket.h>
#include <pthread.h>
#include <time.h>

#define PORT 3490 // the port client will be connecting to
#define MAXDATASIZE 200 // max number of bytes we can get at once
#define MAXCOLABORATORS 10
#define MAXQUEUES 10
#define MAXLINELENGTH 256

pthread_t tid[MAXCOLABORATORS];

long queue_pos[MAXQUEUES];
long data[MAXQUEUES];
int cur_colaborators;
int total_queues;
pthread_mutex_t lock;

struct arg {
    char *server_ip;
    char *local_ip;
    uint16_t port;
};
typedef struct arg arg;
void* doSomething(void *);

char* remove_newline(char *s)
{
    int len = strlen(s);
    if (len > 0 && s[len-1] == '\n'){ // if there's a newline
        s[len-1] = '\0';           // truncate the string
    }
    return s;
}

```



```

int main(int argc, char *argv[])
{
/*****/
    if(argc!=2){
printf("Usage: ./rr_sender Recvr_First_IP Recvr_Second_IP
sndr_First_IP sndr_Second_IP, data\n");
return 0;
    }

    int i = 0;
    int err, linenum=0;
    char line[MAXLINELENGTH];
    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n mutex init failed\n");
        return 1;
    }

    char *serverIPs[MAXCOLABORATORS];
    char *localIPs[MAXCOLABORATORS];
    cur_colaborators=0;
    total_queues = 0;
    FILE *fin;
    if ((fin = fopen(argv[1], "r")) == NULL){
        fprintf(stderr, "Unable to open config file!\n");
        exit(1);
    }
/***** to read server IPs *****/
while(fgets(line, MAXLINELENGTH, fin) != NULL)
{
    if(line[0] == '#') break;
    int testlen = 0;
    testlen = strlen(line);

```

```

        serverIPs[linenum] = malloc(testlen + 1);
        strcpy (serverIPs[linenum], remove_newline(line));
        linenum++;
    }

    /***** to read local IPs *****/
    linenum=0;
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        int testlen;
        testlen = strlen(line);
        localIPs[linenum] = malloc(testlen + 1);
        strcpy (localIPs[linenum], remove_newline(line));
        linenum++;
        cur_colaborators++;
    }

    /***** to get data size *****/
    linenum=0;
    for(i=0; i<MAXQUEUES; i++)
    data[i]=0;
    while(fgets(line, MAXLINELENGTH, fin) != NULL)
    {
        if(line[0] == '#') break;
        sscanf(line, "%ld", &data[linenum++]);
        total_queues++;
    }

    /*****/

    /* Dummy Entering of Colaborators and Sender/Receiver
    Data until File program in incomplete */
    for(i=0;i<total_queues;i++)
    {
        queue_pos[i]=0;

```

```

        //data[i]=500000000;  //Dummey
    }

    for(i=0;i<cur_colaborators;i++){
        //serverIPs[i]="127.0.0.1"; //Dummey
        //localIPs[i]="127.0.0.1";  //Dummey

        err = pthread_create(&(tid[i]), NULL, &doSomething,
        &(arg) {serverIPs[i], localIPs[i], 3490});
        if (err != 0)
            printf("\ncan't create thread %d:[%s]", i, strerror(err));
    }
    for(i=0;i<cur_colaborators;i++){
        pthread_join(tid[i], NULL);
    }
    /***** Till Here *****/

    pthread_mutex_destroy(&lock);
    /*****/
    return 0;
}

void* doSomething(void *ptr)
{

    arg *x = ptr;
    int sockfd, numbytes;
    char buf[MAXDATASIZE];

    int counter=0;
    buf[0]='\0';
    struct hostent *he;
    struct sockaddr_in their_addr, local_addr;
    // connector's address information
    if ((he=gethostbyname(x->server_ip)) == NULL) {

```

```

// get the host info
herror("gethostbyname");
    exit(1);
}
if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
}
int yes =1;
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
&yes, sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }
local_addr.sin_family = AF_INET;
local_addr.sin_addr.s_addr = inet_addr(x->local_ip);
if(bind(sockfd, (struct sockaddr *) &local_addr,
sizeof(local_addr))== -1)
{
perror("Client: bind");
exit(1);
}

their_addr.sin_family = AF_INET;
// host byte order
their_addr.sin_port = htons(x->port);
// short, network byte order
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(their_addr.sin_zero, '\0', sizeof their_addr.sin_zero);
if (connect(sockfd, (struct sockaddr *)&their_addr,
sizeof their_addr) == -1) {
perror("connect");
exit(1);
}

```

```

sprintf(buf,"The Quick Brown Fox Jumps over the Lazy Dog. The Quick
Brown Fox Jumps over the Lazy Dog");
int len=strlen(buf);
int i;

for(i=0; i<total_queues;i++)
{

    do{
pthread_mutex_lock(&lock);
if(queue_pos[i] > data[i]) {pthread_mutex_unlock(&lock); break;}
counter+=len;
queue_pos[i]+=len;
pthread_mutex_unlock(&lock);
if (send(sockfd, buf, strlen(buf), 0) == -1)
perror("send");
    }while(1);

    printf("Total Data Sent of %i queue: %d\n", i, counter);
    counter=0;
}
close(sockfd);
return NULL;
}

```

```

***** mwq_config.txt *****
*****
1.0.1.1
1.0.1.1
#
1.0.8.2
1.0.9.2
#
30000000

```

20000000

#

## B.2.5 Receiver Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

#define MYPORT 3490 // the port users will be connecting to
#define BACKLOG 10  // how many pending connections queue will hold
#define MAXDATASIZE 200 // max number of bytes we can get at once

void sigchld_handler(int s)
{
    while(waitpid(-1, NULL, WNOHANG) > 0);
}

int main(void)
{
    int sockfd, new_fd; // listen on sock_fd, new connection on new_fd
    struct sockaddr_in my_addr; // my address information
    struct sockaddr_in their_addr; // connector's address information
    socklen_t sin_size;
    struct sigaction sa;
    char buf[MAXDATASIZE];
    int con_counter=0;
    int yes=1, numbytes;
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
```

```

        perror("socket");
        exit(1);
    }
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
&yes, sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }
    my_addr.sin_family = AF_INET;           // host byte order
    my_addr.sin_port = htons(MYPORT);
// short, network byte order
    my_addr.sin_addr.s_addr = INADDR_ANY;
// automatically fill with my IP
    memset(my_addr.sin_zero, '\0',
sizeof my_addr.sin_zero);
    if (bind(sockfd, (struct sockaddr *)&my_addr,
sizeof my_addr) == -1) {
        perror("bind");
        exit(1);
    }
    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }
    sa.sa_handler = sigchld_handler; // reap all dead processes
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART;
    if (sigaction(SIGCHLD, &sa, NULL) == -1) {
        perror("sigaction");
        exit(1);
    }
    while(1) { // main accept() loop
        sin_size = sizeof their_addr;
        if ((new_fd = accept(sockfd, (struct sockaddr *)
&their_addr, &sin_size)) == -1) {

```

```

        perror("accept");
        continue;
    }
    con_counter++;
    printf("server: got connection from %s, from port
%d\n", inet_ntoa(their_addr.sin_addr), ntohs(their_addr.sin_port));

    if (!fork()) { // this is the child process
        close(sockfd); // child doesn't need the listener

int i, datacounter;
datacounter = 0;
do{
    if ((numbytes=recv(new_fd, buf, MAXDATASIZE-1, 0)) == -1) {
        perror("Server: recv");
        exit(1);
    }
    datacounter+=numbytes;
    buf[numbytes] = '\0';
}while(numbytes!=0);
    close(new_fd);
    exit(0);
}
close(new_fd); // parent doesn't need this

}
return 0;
}

```



# Appendix C

## HTTP Downloader Implementation Details

### C.1 Bluetooth Pairing Process

The Bluetooth pairing process is typically triggered automatically the first time a device receives a connection request from a device with which it is not yet paired. In order that Bluetooth pairing may occur, a password has to be exchanged between the two devices. This password or "Passkey" as it is more correctly termed is a code shared by both Bluetooth devices. It is used to ensure that both users have agreed to pair with each other.

The process of Bluetooth pairing is summarised below:

1. Bluetooth device looks for other Bluetooth devices in range: To be found by other Bluetooth devices, the first device, Device 1 must be set to discoverable mode - this will allow other Bluetooth devices in the vicinity to detect its presence and attempt to establish a connection.
2. Two Bluetooth devices find each other: When the two devices: Device 1 and device 2 find each other it is possible to detect what they are. Normally the discoverable device will indicate what type of device it is - cellphone, headset, etc., along with its Bluetooth device name. The Bluetooth device name is the can be allocated by the user, or it will be the one allocated during manufacture.
3. Prompt for Passkey: Often the default passkey is set to "0000", but it is

advisable to use something else as hackers will assume most people will not change this.

However many more sophisticated devices - smartphones and computers - both users must agree on a code which must obviously be the same for both.

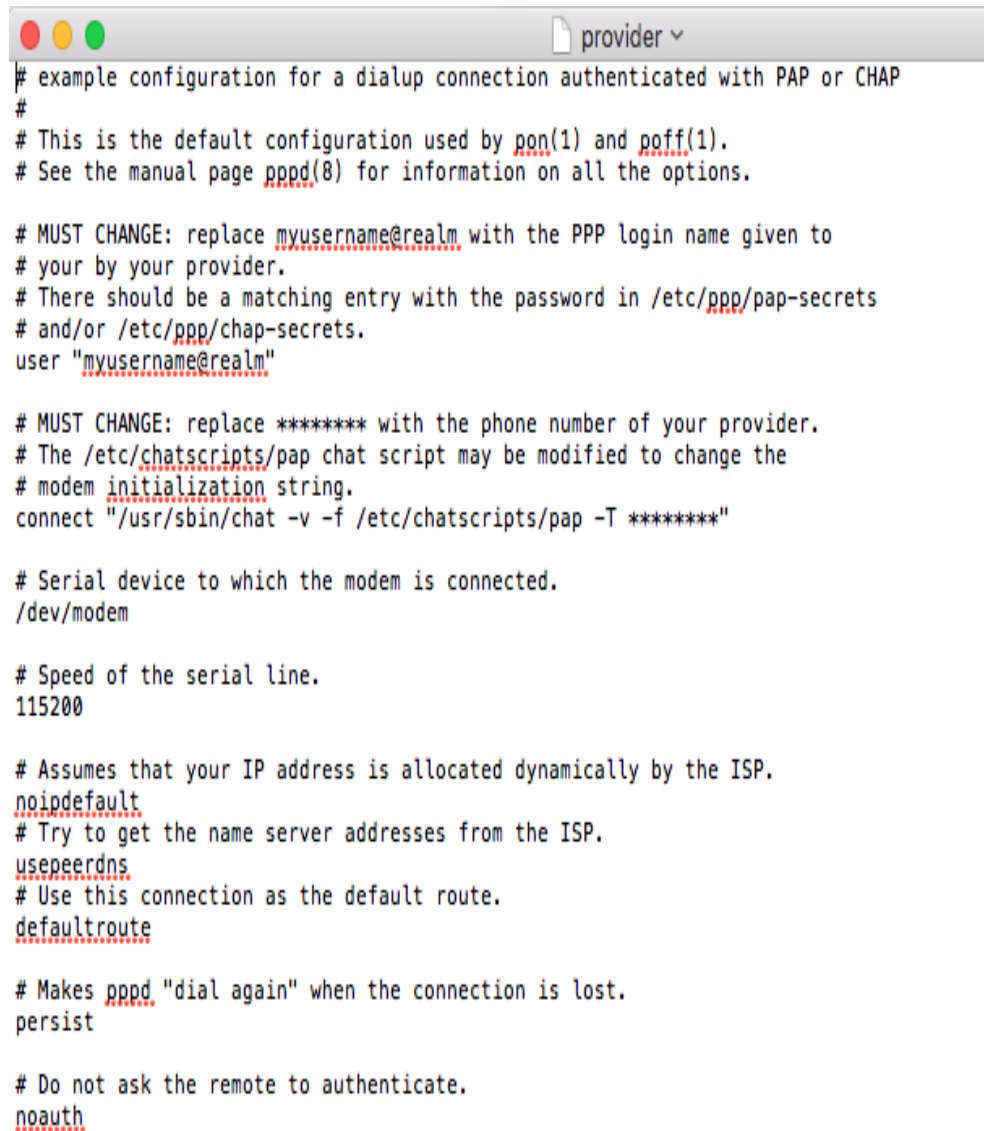
4. Device 1 sends passkey: The initiating device, Device 1 sends the passkey that has been entered to Device 2.
5. Device 2 sends passkey: The passkeys are compared and if they are both the same, a trusted pair is formed, Bluetooth pairing is established.
6. Communication is established: Once the Bluetooth pairing has occurred, data can be exchanged between the devices.

Once the Bluetooth pairing has been established it is remembered by the devices, which can then connect to each without user intervention. If necessary, the Bluetooth pairing relationship may be removed by the user at a later time if required.

## **C.2 Chat Script Example and Explanation**

Below is the details and explanation of configuration for a point to point connection authentication with PAP or CHAP. Each step is explained in comments.

### **C.2.1 Example Configuration for a PPP Connection Authentication**



```
# example configuration for a dialup connection authenticated with PAP or CHAP
#
# This is the default configuration used by pon(1) and poff(1).
# See the manual page pppd(8) for information on all the options.

# MUST CHANGE: replace myusername@realm with the PPP login name given to
# your by your provider.
# There should be a matching entry with the password in /etc/ppp/pap-secrets
# and/or /etc/ppp/chap-secrets.
user "myusername@realm"

# MUST CHANGE: replace ***** with the phone number of your provider.
# The /etc/chatscripts/pap chat script may be modified to change the
# modem initialization string.
connect "/usr/sbin/chat -v -f /etc/chatscripts/pap -T *****"

# Serial device to which the modem is connected.
/dev/modem

# Speed of the serial line.
115200

# Assumes that your IP address is allocated dynamically by the ISP.
noipdefault
# Try to get the name server addresses from the ISP.
usepeerdns
# Use this connection as the default route.
defaultroute

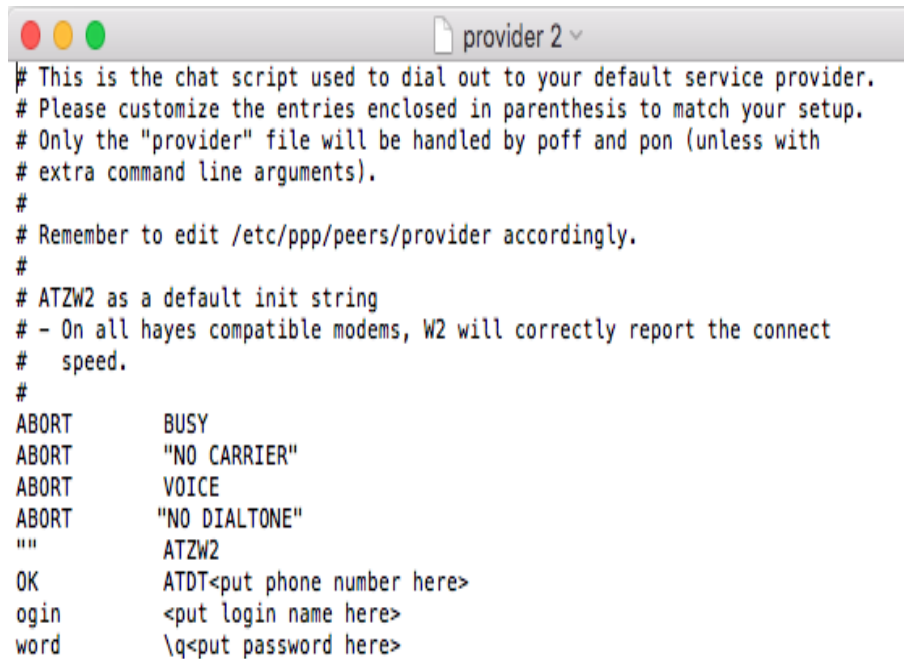
# Makes pppd "dial again" when the connection is lost.
persist

# Do not ask the remote to authenticate.
noauth
```

Figure C.1: Example Configuration for a PPP Connection Authenticated with PAP or CHAP

## C.2.2 Chat Script to Connect to Default Service Provider

Following is an example chat script to dial out to a default service provider. We followed same example to write our chat scripts for two mobile data service providers, Lyca and Virgin.

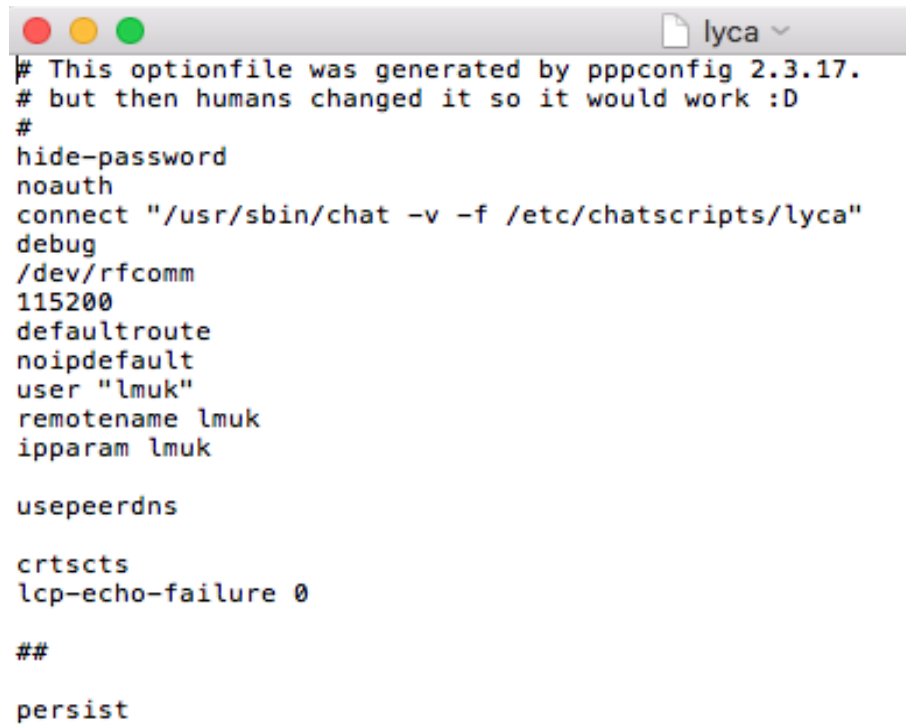


```
# This is the chat script used to dial out to your default service provider.
# Please customize the entries enclosed in parenthesis to match your setup.
# Only the "provider" file will be handled by poff and pon (unless with
# extra command line arguments).
#
# Remember to edit /etc/ppp/peers/provider accordingly.
#
# ATZW2 as a default init string
# - On all hayes compatible modems, W2 will correctly report the connect
#   speed.
#
ABORT      BUSY
ABORT      "NO CARRIER"
ABORT      VOICE
ABORT      "NO DIALTONE"
""         ATZW2
OK         ATDT<put phone number here>
ogin       <put login name here>
word       \q<put password here>
```

Figure C.2: Chat Script to Connect to Default Service Provider

### C.2.3 Configuration for a PPP Connection Authenticated with Lyca

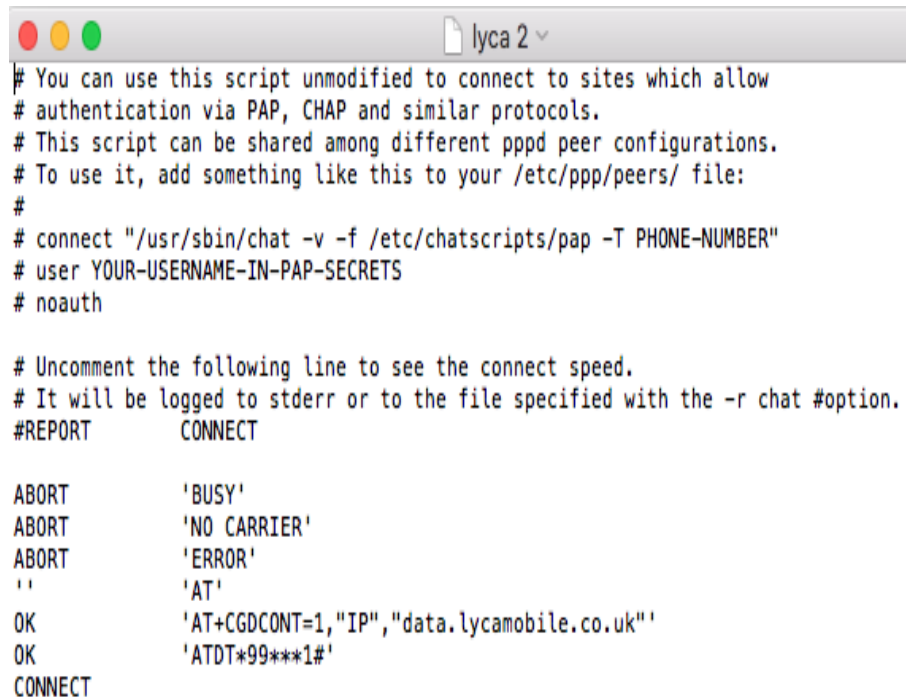
I have written connection authentication and chat scripts for Virgin and Lyca mobile service provider. Following are the scripts used for making connections with Lyca Mobile.



```
# This optionfile was generated by pppconfig 2.3.17.  
# but then humans changed it so it would work :D  
#  
hide-password  
noauth  
connect "/usr/sbin/chat -v -f /etc/chatscripts/lyca"  
debug  
/dev/rfcomm  
115200  
defaultroute  
noipdefault  
user "lmuk"  
remotename lmuk  
ipparam lmuk  
  
usepeerdns  
  
crtscts  
lcp-echo-failure 0  
  
##  
  
persist
```

Figure C.3: Configuration for a PPP Connection Authentication with Lyca

## C.2.4 Chat Script to Connect to Lyca



```
# You can use this script unmodified to connect to sites which allow
# authentication via PAP, CHAP and similar protocols.
# This script can be shared among different pppd peer configurations.
# To use it, add something like this to your /etc/ppp/peers/ file:
#
# connect "/usr/sbin/chat -v -f /etc/chatscripts/pap -T PHONE-NUMBER"
# user YOUR-USERNAME-IN-PAP-SECRETS
# noauth

# Uncomment the following line to see the connect speed.
# It will be logged to stderr or to the file specified with the -r chat #option.
#REPORT      CONNECT

ABORT        'BUSY'
ABORT        'NO CARRIER'
ABORT        'ERROR'
''           'AT'
OK           'AT+CGDCONT=1,"IP","data.lycamobile.co.uk"'
OK           'ATDT*99***1#'
CONNECT
```

Figure C.4: Chat Script to Connect to Lyca Mobile Service Provider

## C.2.5 Log File

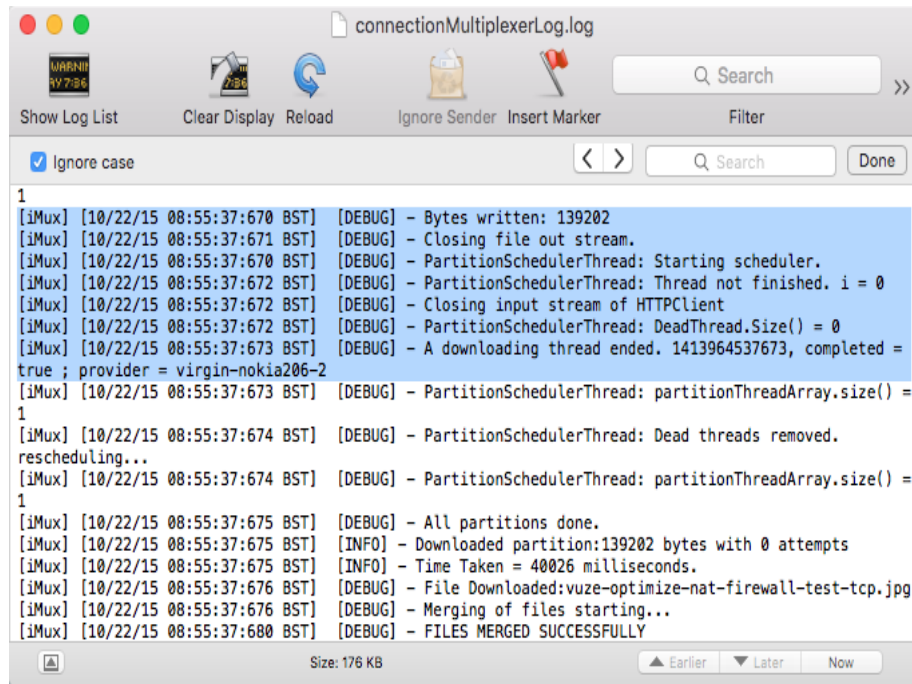


Figure C.5: Log File from a Test Run

## C.3 HTTP Downloader Code

### C.3.1 DownloadRequest.java

```
package com.imux.vo;
import java.util.ArrayList;
import java.util.List;

/**
 * Keeps the download request and its sub-requests i.e. partitions.
 * @author sadia
 *
 */
public class DownloadRequest {

    private String url;
    private int contentLength;
```

```

private String fileName;
private List<List<DownloadRequestPartition>> partitionLists;

public DownloadRequest(String argUrl, int argContentLength,
String argFileName){
this.url = new String(argUrl);
contentLength = argContentLength;
fileName = argFileName;
partitionLists = new ArrayList<List<DownloadRequestPartition>>();
}

public String getUrl() {
return url;
}

public void setUrl(String url) {
this.url = url;
}

public int getContentLength() {
return contentLength;
}

public void setContentLength(int contentLength) {
this.contentLength = contentLength;
}

public List<List<DownloadRequestPartition>> getParitionList() {
return partitionLists;
}

public void setParitionList(List<List<DownloadRequestPartition>>
paritionList) {
this.partitionLists = paritionList;
}

```



```

/**
 * Makes only work chunks into one queue as the passed number
 * of connection count.
 * @param argConnectionCount
 */
public void makePartitionsByConnectionCount(int argConnectionCount){
// We only make a single queue here and put all partitions in it.
List<DownloadRequestPartition> partitions = new ArrayList<
DownloadRequestPartition>();
this.partitionLists.add(partitions);
int partitionSize = this.contentLength/argConnectionCount;
int index = 0;
for( int i = 0 ; i < argConnectionCount ; i++){
DownloadRequestPartition partition = new DownloadRequestPartition();
partition.setStartingByte(index);
partition.setEndingByte(index + partitionSize);
partition.setPartitionFileName(fileName + i);
partition.setUrl(url);
partition.setPartitionCompleted(false);
partition.setPartitionDownloading(false);
partition.setPartitionStoped(true);
index = partitionSize + 1;
partitions.add(partition);
}
}

/**
 * Makes n number of partitions (work chunks) of passed partition size
 * to complete the request.
 * @param argPartitionSize
 */
public void makePartitionsBySize(int argPartitionSize){
int partitionSize = argPartitionSize;
int partitionCount = (int)(this.contentLength/argPartitionSize);

```

```

if(partitionCount != 0){
if( this.contentLength % partitionCount != 0)
partitionCount++;
}
else
partitionCount++;

// We only make a single queue here and put all partitions in it.
List<DownloadRequestPartition> partitions = new ArrayList<
DownloadRequestPartition>();
this.partitionLists.add(partitions);

int index = 0;
for( int i = 0 ; i < partitionCount ; i++){
DownloadRequestPartition partition = new DownloadRequestPartition();
partition.setStartingByte(index);
if(i == partitionCount - 1){
index = index + (contentLength % argPartitionSize) - 1;
partition.setEndingByte(index);
}
else{
index = index + partitionSize - 1;
partition.setEndingByte(index);
}
partition.setPartitionCompleted(false);
partition.setPartitionDownloading(false);
partition.setPartitionStoped(true);
partition.setPartitionFileName(fileName + i);
partition.setUrl(url);
index = index + 1;
partitions.add(partition);
}
}

/**

```

```

    * Makes n number of partitions (work chunks) of passed partition size
    * to complete the request.
    * @param argPartitionSize
    */
public void makePartitionsByPriority(int argPartitionSize, int
argPriorityQueueCount, int argPercentInQueue[]){
if(argPercentInQueue == null
|| argPriorityQueueCount != argPercentInQueue.length
|| sum(argPercentInQueue) != 100)
throw new RuntimeException("Number of queues should be same
as number of %ages and their sum must be 100.");
// Ok lets make the partitions first.
int partitionSize = argPartitionSize;
int partitionCount = (int)(this.contentLength/argPartitionSize);
if(partitionCount != 0){
if( this.contentLength % partitionCount != 0)
partitionCount++;
}
else
partitionCount++;

// We make multiple queues as required.
for( int i = 0 ; i < argPriorityQueueCount ; i++){
List<DownloadRequestPartition> partitions = new ArrayList<
DownloadRequestPartition>();
this.partitionLists.add(partitions);
}
// roughly calculate how many partitions each queue will get.
int partitionPerQueue[] = new int[argPriorityQueueCount];
for(int i = 0 ; i < argPriorityQueueCount ; i++){
// total is partitionCount
// each queue might be off by one partition but we want to round such
// that no queue at top has less data than percentage assigned to it.
partitionPerQueue[i] = (int)Math.ceil(partitionCount
* argPercentInQueue[i] / 100);

```

```

}
int index = 0;
int currentQueue = 0;
for( int i = 0 ; i < partitionCount ; i++){
// Create a partition.
DownloadRequestPartition partition = new DownloadRequestPartition();
partition.setStartingByte(index);
if(i == partitionCount - 1){
index = index + (contentLength % argPartitionSize) - 1;
partition.setEndingByte(index);
}
else{
index = index + partitionSize - 1;
partition.setEndingByte(index);
}
partition.setPartitionCompleted(false);
partition.setPartitionDownloading(false);
partition.setPartitionStoped(true);
partition.setPartitionFileName(fileName + i);
partition.setUrl(url);
index = index + 1;
// Put the partition in the correct queue. If current queue is full, move
// down to next queue.
if(partitionPerQueue[currentQueue] <= 0)
currentQueue++;
partitionLists.get(currentQueue).add(partition);
partitionPerQueue[currentQueue]--;

}
}

private int sum(int argValues[]){
int sum = 0;
for (int i : argValues) {
sum += i;

```

```

}
return sum;
}

public int getDownloadedByteCount(){
    int sum = 0;
    for (List<DownloadRequestPartition> list2 : partitionLists) {
        for (DownloadRequestPartition downloadRequestPartition : list2) {
            sum += downloadRequestPartition.getBytesDownloaded();
        }
    }
    return sum;
}

public DownloadRequestPartition getNextPartitionToDownload(Integer
argConPriority){
    DownloadRequestPartition request = null;
    List<DownloadRequestPartition> subQueue = null;
    // has this instance been initialised with priority queue
    // (i.e. more than one queue)
    if( partitionLists.size() > 1 && argConPriority != null){
        // this is priority scheduler.
        // Step 1, look for a partition in its level or the level below.
        int actualIndex = argConPriority < partitionLists.size() ?
argConPriority : partitionLists.size() - 1;
        for(int i = actualIndex ; i < partitionLists.size() ; i++){
            subQueue = partitionLists.get(i);
            for( int j = 0 ; j < subQueue.size() ; j++){
                DownloadRequestPartition temp = subQueue.get(j);
                if(!temp.isPartitionCompleted() && !temp.isPartitionDownloading()
&& temp.isPartitionStoped()){
                    request = temp;
                    break;
                }
            }
        }
    }
}

```

```

// Get the next available partition waiting to be downloaded.
if(request != null)
break;
}
if( request == null){
// a partition was not found in its level and below.
// Lets go towards the top.
for(int i = actualIndex - 1 ; i >= 0 ; i--){
subQueue = partitionLists.get(i);
for( int j = 0 ; j < subQueue.size() ; j++){
DownloadRequestPartition temp = subQueue.get(j);
if(!temp.isPartitionCompleted() && !temp.isPartitionDownloading()
&& temp.isPartitionStoped()){
request = temp;
break;
}
}
// get the next available partition waiting to be downloaded.
if(request != null)
break;
}
}

}else{
// there will always be only 1 queue for simple round robin request.
// But lets keep the code consistent.
for(int i = 0 ; i < partitionLists.size() ; i++){
subQueue = partitionLists.get(i);
for( int j = 0 ; j < subQueue.size() ; j++){
DownloadRequestPartition temp = subQueue.get(j);
if(!temp.isPartitionCompleted() && !temp.isPartitionDownloading()
&& temp.isPartitionStoped()){
request = temp;
break;
}
}
}
}

```

```

    }
    // get the next available partition waiting to be downloaded.
    if(request != null)
        break;
    }
    }
    return request;
}

public int getCompletedPartitionCount(){
    int count = 0;
    for (List<DownloadRequestPartition> list : partitionLists) {
        for (DownloadRequestPartition downloadRequestPartition : list) {
            if(downloadRequestPartition.isPartitionCompleted()
                && !downloadRequestPartition.isPartitionDownloading()
                && downloadRequestPartition.isPartitionStoped())
                count++;
        }
    }
    return count;
}

public boolean isMultiqueue(){
    return partitionLists.size() > 1;
}
}

```

### C.3.2 DownloadRequestPartition.java

```

/**
 *
 */
package com.imux.vo;

/**
 * @author sadia

```

```

*
*/
public class DownloadRequestPartition {

    private String url;
    private int startingByte;
    private int endingByte;
    private int bytesDownloaded;
    private String partitionFileName;
    private boolean partitionDownloading;
    private boolean partitionCompleted;
    private boolean partitionStoped;
    private int failureCount;

    public int getEndingByte() {
        return endingByte;
    }

    public void setEndingByte(int endingByte) {
        this.endingByte = endingByte;
    }

    public String getPartitionFileName() {
        return partitionFileName;
    }

    public void setPartitionFileName(String partitionFileName) {
        this.partitionFileName = partitionFileName;
    }

    public int getStartingByte() {
        return startingByte;
    }

    public void setStartingByte(int startingByte) {

```



```

this.startingByte = startingByte;
}

public boolean isPartitionCompleted() {
return partitionCompleted;
}

public void setPartitionCompleted(boolean partitionCompleted) {
this.partitionCompleted = partitionCompleted;
}

public String getUrl() {
return url;
}

public void setUrl(String url) {
this.url = url;
}

public boolean isPartitionDownloading() {
return partitionDownloading;
}

public void setPartitionDownloading(boolean partitionDownloading) {
this.partitionDownloading = partitionDownloading;
}

public boolean isPartitionStoped() {
return partitionStoped;
}

public void setPartitionStoped(boolean partitionStoped) {
this.partitionStoped = partitionStoped;
}

```

```

public int getBytesDownloaded() {
    return bytesDownloaded;
}

public void setBytesDownloaded(int bytesDownloaded) {
    this.bytesDownloaded = bytesDownloaded;
}

public int getFailureCount() {
    return failureCount;
}

public void setFailureCount(int failureCount) {
    this.failureCount = failureCount;
}
}

```

### C.3.3 HTTPRequestUtil.java

```

package com.imux.network;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.URL;

import org.apache.commons.httpclient.Header;
import org.apache.commons.httpclient.HostConfiguration;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpMethod;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.logging.Log;

```

```

import org.apache.commons.logging.LogFactory;

import com.imux.threads.SinglePartitionThread;
import com.imux.vo.DownloadRequestPartition;
import com.imux.utils.*;

public class HTTPRequestUtil {

    private Log log = LogFactory
        .getLog(HTTPRequestUtil.class);

    private int bytesDownloaded;

    public int getBytesDownloaded(){
        return bytesDownloaded;
    }

    public static int getContentLength(String argRequestURL)throws
        MalformedURLException, IOException{
        URL url = new URL(argRequestURL);
        return url.openConnection().getContentLength();
    }

    @SuppressWarnings("deprecation")
    public boolean downloadWithResume(DownloadRequestPartition
        argRequestPartition, SinglePartitionThread argPartialDownloadingThread)
    {
        argRequestPartition.setPartitionDownloading(true);
        argRequestPartition.setPartitionCompleted(false);
        argRequestPartition.setPartitionStoped(false);
        int argEndRange = argRequestPartition.getEndingByte();
        String argFragmentFileName = argRequestPartition.getPartitionFileName();
        int argStartRange = argRequestPartition.getStartingByte();

```

```

String argRequestURL = argRequestPartition.getUrl();
InetAddress argInterfaceInetAddress = argPartialDownloadingThread.
getInterfaceInetAddress();
log.debug("Using interface " + argInterfaceInetAddress.getHostAddress()
+ " downloading " + argStartRange + " - " + argEndRange + " of "
+ argRequestURL + " into file " + argFragmentFileName + " - Device name
- " + argPartialDownloadingThread.getChannel().getDeviceFriendlyName());

    long localFileSize = 0;
    long remoteFileSize = 0;
    HttpClient client = null;
    BufferedInputStream in = null;
    RandomAccessFile out = null;
    HttpMethod request = null;
    int bytesDownloadedThisTime = 0;
    try{
        File file = new File(ApplicationConstants.DOWNLOAD_FOLDER_PATH
, argFragmentFileName);
        if (file.exists()) {
            localFileSize = file.length();
            log.debug("File exists, size : " + localFileSize);
            if( localFileSize == argEndRange - argStartRange){
                log.info("Its already downloaded.");
                return true;
            }else if( localFileSize > argEndRange - argStartRange){
                log.debug("Should never happen. Something wrong! file size is
grater than chunk size! Not downloading.");
                return true;
            }
        }
        if( argPartialDownloadingThread.getChannel().getHttpClient() == null){
            client = new HttpClient();
            argPartialDownloadingThread.getChannel().setHttpClient(client);
            log.debug("Created new HTTPClient.");
        }else{

```

```

client = argPartialDownloadingThread.getChannel().getHttpClient();
log.debug("Reusing HttpClient");
}

client.setTimeout(ApplicationConstants.connectionTimeout);
client.setConnectionTimeout(ApplicationConstants.readTimeout);
request = new GetMethod(argRequestURL);

HostConfiguration configuration = new HostConfiguration();

configuration.setLocalAddress(argInterfaceInetAddress);

    if (localFileSize > 0) {
        /* Note: server must support partial content for resume
        * Now move the starting point of the reader so that you
        * read what you missed last time.
        */
        argStartRange += localFileSize;
        request.setRequestHeader("Range", "bytes=" + argStartRange
+ "-" + argEndRange);
        log.debug("New range = " + argStartRange + "-"
+ argEndRange);
        if (client.executeMethod(configuration, request) !=
HttpStatus.SC_PARTIAL_CONTENT)
            // the server does not support resume. So we will have
to download the complete file in one go.
            return false;
    } else {
        log.debug("File size was zero; range remains " + argStartRange
+ "-" + argEndRange);
        request.setRequestHeader("Range", "bytes=" + argStartRange
+ "-" + argEndRange);
        int status = client.executeMethod(configuration, request);
        if (status != HttpStatus.SC_OK && status !=
HttpStatus.SC_PARTIAL_CONTENT)

```

```

        // response not ok, server is either busy or having issues
        // we don't need to handle.
        return false;
    }

    Header contentLengthHeader = request.getResponseHeader("content-length");
    if (contentLengthHeader == null) {
        // We don't know what is the length of content, cannot start with that.
        return false;
    }
    remoteFileSize = Long.parseLong(contentLengthHeader.getValue());

    log.debug("Local file size is " + localFileSize + " bytes.");
    log.debug("Remote file size is " + remoteFileSize + " bytes.");
    log.debug("Bytes Remaining " + (remoteFileSize - localFileSize)
+ " bytes.");

    log.debug("The HTTPClient port being used : " +
client.getHostConfiguration().getPort());

    in = new BufferedInputStream(request.getResponseBodyAsStream());
    out = new RandomAccessFile(file, "rw");
    out.seek(localFileSize);

    int offset = 0;
    int len = ApplicationConstants.readBufferLength;
    /*
     *
     * Length is 4 K is the max we can receive in one go.
     * Webserver dictates e.g. in a .NET application running in
Windows 2002 server as of 01-August-2003 would be
     * dictated by the maxRequestLength setting in the
     * machine.config file or web.config file and is the total amount
     * of data that can be sent through HTTP Post to the server.
     * The default is 4MB (4096)...and is generally set low so that your

```

```

        * server will not be overwhelmed by possible DoS attacks.
        */

int bytes = 0;
long totalBytes = 0;
bytesDownloaded = argRequestPartition.getBytesDownloaded();
byte[] block = new byte[len];
while ( (bytes = in.read(block, offset, len)) > -1) {

    out.write(block, 0, bytes);
    totalBytes += bytes;
    bytesDownloaded += bytes;
    argRequestPartition.setBytesDownloaded(bytesDownloaded);
    bytesDownloadedThisTime += bytes;
}

boolean retVal = totalBytes == remoteFileSize;
log.debug("Bytes written: " + totalBytes);
argRequestPartition.setPartitionCompleted(retVal);
argRequestPartition.setPartitionStoped(!retVal);
argRequestPartition.setPartitionDownloading(false);
argPartialDownloadingThread.setDownloadThreadInterrupted(!retVal);
return retVal;

    } catch (Exception e) {
        argRequestPartition.setPartitionCompleted(false);
        argRequestPartition.setPartitionStoped(true);
        argRequestPartition.setPartitionDownloading(false);
        argPartialDownloadingThread.setDownloadThreadInterrupted(true);
        argPartialDownloadingThread.getChannel().setFaultCount(
            argPartialDownloadingThread.getChannel()
                .getFaultCount()+1); // for channel
        argRequestPartition.setFailureCount(argRequestPartition
            .getFailureCount()+1); // for thread
        e.printStackTrace();
    }
}

```

```

        return false;
    }catch(Error e){
        argRequestPartition.setPartitionCompleted(false);
        argRequestPartition.setPartitionStoped(true);
        argRequestPartition.setPartitionDownloading(false);
        argPartialDownloadingThread.setDownloadThreadInterrupted(true);
        argPartialDownloadingThread.getChannel().setFaultCount(
            argPartialDownloadingThread.getChannel()
            .getFaultCount()+1); // for channel
        argRequestPartition.setFailureCount(argRequestPartition
            .getFailureCount()+1);
        e.printStackTrace();
        return false;
    }finally {
        try {
            argPartialDownloadingThread.getChannel().setBytesDownloaded(
                argPartialDownloadingThread.getChannel().getBytesDownloaded()
                +bytesDownloadedThisTime);

            log.debug("Closing file out stream.");
            if (out != null ) out.close();
        }catch(IOException ioe) {
            log.debug("Closing file out stream FAILED.");
            ioe.printStackTrace();/* do nothing */
        }

        try {
            log.debug("Closing input stream of HTTPClient");
            if (in != null) in.close();
        }catch(IOException ioe) {
            log.debug("Clould not close input stream of HTTPClient");
            ioe.printStackTrace();/* do nothing */
        }

        // comment this for not closing the request connection
        ( which closes the HTTPClient channel )

```



```

//      try{
//      log.debug("Releasing request connection.");
//      if( request != null)
//      request.releaseConnection();
//      }catch(NullPointerException e){
//      log.error(e.getMessage());
//      e.printStackTrace();/* do nothing. */}
    }
}

public static boolean doesServerSupportPartialContent
(String argUrl) throws IOException{
    HttpClient client = null;
    HttpMethod request = null;

    client = new HttpClient();
    request = new GetMethod(argUrl);
    request.setRequestHeader("Range", "bytes=0-2");
    if (client.executeMethod(request) != HttpStatus.SC_PARTIAL_CONTENT)
        // the server does not support resume. So we will have to
download the complete file in one go... ( it doesnot support ranges )
        return false;
    else
        return true;
}
}

```

### C.3.4 PartitionScheduler.java

```

package com.imux.threads;

import java.util.ArrayList;
import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

```

```

import com.imux.bluetoothconnection.BlueChannel;
import com.imux.bluetoothconnection.DeviceList;
//import com.imux.gui.ConnectionMultiplexerMainGui.FileMergingThread;
import com.imux.vo.DownloadRequest;
import com.imux.vo.DownloadRequestPartition;

/**
 * This class handles thread scheduling and removes dead threads.
 * @author sadia
 *
 */
public class PartitionScheduler implements Runnable{

    private DownloadRequest request;
    private long timeTaken;

    private Log log = LogFactory
        .getLog(PartitionScheduler.class);

    //
    private List<SinglePartitionThread> singlePartitionThreadList;
    private DeviceList deviceList;
    private FileJoiningThread fileMergingThread;

    public PartitionScheduler(DownloadRequest argDownloadRequest,
        String argFileName, DeviceList argDeviceList,
        FileJoiningThread argFileMergingThread, List<SinglePartitionThread>
        argPartitionThreadArray){
        request = argDownloadRequest;
        timeTaken = 0;
        deviceList = argDeviceList;
        fileMergingThread = argFileMergingThread;
        singlePartitionThreadList = argPartitionThreadArray;
    }

```

```

public void run(){
    timeTaken = System.currentTimeMillis();

    while(true){
        log.debug("PartitionScheduler for thread: Starting scheduler.");
        // first remove dead threads
        List < SinglePartitionThread> deadThreads = new ArrayList<
        SinglePartitionThread>();
        for( int i = 0 ; i < singlePartitionThreadList.size() ; i++){
            SinglePartitionThread temp = singlePartitionThreadList.get(i);
            if( temp.isFinishedSuccessfully() == false ){
                log.debug("PartitionSchedulerThread:
                Thread not finished. i = " + i);
                if( temp.isDownloadThreadInterrupted() == true){
                    log.debug("PartitionSchedulerThread:
                    Thread intrupped too. i = " + i);
                    deadThreads.add(temp);
                }
            }else{
                log.debug("PartitionSchedulerThread: Thread Completed. i = " + i);
                deadThreads.add(temp);
            }
        }

        log.debug("PartitionSchedulerThread: DeadThread.Size() =
        " + deadThreads.size());
        log.debug("PartitionSchedulerThread: partitionThreadArray.size() =
        " + singlePartitionThreadList.size());
        for( int i = 0 ; i < deadThreads.size() ; i++)
            singlePartitionThreadList.remove(deadThreads.get(i));
        // dead threads removed.
        log.debug("PartitionSchedulerThread: Dead threads removed. rescheduling");
        log.debug("PartitionSchedulerThread: singlePartitionThreadList.size() =
        " + singlePartitionThreadList.size());
    }
}

```

```

// now reschedule the partitions
if(request.isMultiqueue())
multiQueueScheduler();
else
singleQueueScheduler();

if( request.getCompletedPartitionCount() == request.getPartitionList()
.size()){
log.debug("All partitions done.");
break;
}
else
try {
// Lets check progress in 5 seconds.
Thread.sleep(5000);
} catch (InterruptedException e) {
e.printStackTrace();
}
} // while(true) loop
timeTaken = System.currentTimeMillis() - timeTaken;
for( int i = 0 ; i < request.getPartitionList().size() ; i++){
List<List<DownloadRequestPartition>> queues = request.getPartitionList();
for (List<DownloadRequestPartition> list : queues)
for (DownloadRequestPartition drp : list)
log.info("Downloaded partition:" + drp.getBytesDownloaded()
+ " bytes with " + drp.getFailureCount() + " attempts");

}
log.info("Time Taken = " + timeTaken + " milliseconds.");
Thread fileMerger = new Thread(fileMergingThread);
fileMerger.start();
}

/**
 * Simply gets a partition thats not finished, finds a free interface

```

```

    * and assigns it to download this partition.
    */
private void singleQueueScheduler(){
for( int i = 0 ; i < request.getParitionList().size() ; i++){
DownloadRequestPartition temp = request.getParitionList().get(0).get(i);
if( temp.isPartitionDownloading())
continue; // this one is running, check the next one.
else if( temp.isPartitionCompleted())
continue;// completed partitions
else if(temp.isPartitionStoped()){
// reschedule it;
List<BlueChannel> channelList = deviceList.getDevices();
//log.debug("partitionThreadArraySize  = " + partitionThreadArray.size());
//for(int k = 0; k < listAllAddresses.size(); k++){
for(int k = 0; k < channelList.size(); k++){
boolean foundAddress = false;
// Check if an interface is free.
for(int l = 0; l < singlePartitionThreadList.size(); l++ ){
if((singlePartitionThreadList.get(l).getInterfaceInetAddress().equals
((channelList.get(k).getInetAddress())))){
foundAddress = true;
break;
}
}
}
if (!foundAddress){
log.debug("Found " + channelList.get(k).getInetAddress().
getHostAddress() + " free. Using it.");
SinglePartitionThread pdThread = new SinglePartitionThread(temp,
channelList.get(k));
Thread thread = new Thread(pdThread);
singlePartitionThreadList.add(pdThread);
thread.start();
log.debug("Thread started.");
break;
}
}
}

```

```

} // k
} // endif
} // partition loop
}

/**
 * Priority queue scheduler. Finds free connections, then finds a
 * partition corresponding to the quality and assigns it to
 * that link for downloading.
 */
private void multiQueueScheduler(){
    List<BlueChannel> channelList = deviceList.getDevices();
    for(int k = 0; k < channelList.size(); k++){
        boolean foundAddress = false;
        // Check if an interface is free.
        for(int l = 0; l < singlePartitionThreadList.size(); l++){
            if(singlePartitionThreadList.get(l).getInterfaceInetAddress() != null
            && (singlePartitionThreadList.get(l).
                getInterfaceInetAddress().equals((channelList.get(k)
                .getInetAddress())))){
                foundAddress = true;
                break;
            }
        } // l
        if (!foundAddress){
            // its better to just start a downloading thread here.
            log.debug("Found " + channelList.get(k).getInetAddress().getHostAddress()
            + " free. Using it.");
            DownloadRequestPartition temp = request.getNextPartitionToDownload(k);
            if( temp != null){
                SinglePartitionThread pdThread = new SinglePartitionThread(temp,
                channelList.get(k));
                Thread thread = new Thread(pdThread);
                singlePartitionThreadList.add(pdThread);
                thread.start();
            }
        }
    }
}

```

```

log.debug("Thread started.");
}else
log.debug("No more data to download for interface " + k + ".");
break;
}
} // k
}
}

```

### C.3.5 SinglePartitionThread.java

```

package com.imux.threads;

import java.net.InetAddress;
//import java.net.MalformedURLException;
//import java.net.URL;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.imux.bluetoothconnection.BlueChannel;
import com.imux.network.HTTPRequestUtil;
import com.imux.vo.DownloadRequestPartition;

public class SinglePartitionThread implements Runnable {

private Log log = LogFactory
.getLog(SinglePartitionThread.class);

private boolean finishedSuccessfully;
private InetAddress interfaceInetAddress;
private HTTPRequestUtil downloadUtil;
private boolean downloadThreadInterrupted;
private DownloadRequestPartition requestPartition;
private BlueChannel channel;

```

```

public SinglePartitionThread(DownloadRequestPartition
argDownloadRequestPartition, BlueChannel argBlueChannel){
requestPartition = argDownloadRequestPartition;
finishedSuccessfully = false;
channel = argBlueChannel;
interfaceInetAddress = argBlueChannel.getInetAddress();
}

public void run() {
log.debug("Downloading thread starting. " + System.
currentTimeMillis());
downloadUtil = new HTTPRequestUtil();
if( requestPartition != null){
requestPartition.setPartitionDownloading(true);
requestPartition.setPartitionCompleted(false);
requestPartition.setPartitionStoped(false);
}
finishedSuccessfully = downloadUtil.downloadWithResume(
requestPartition, this);
downloadThreadInterrupted = !finishedSuccessfully;

log.debug("A downloading thread ended. " + System.currentTimeMillis()
+ ", completed = " + finishedSuccessfully
+ " ; provider = " + this.channel.getDeviceFriendlyName());
}

public boolean isFinishedSuccessfully() {
return finishedSuccessfully;
}

public void setFinishedSuccessfully(boolean finishedSuccessfully) {
this.finishedSuccessfully = finishedSuccessfully;
}

```



```

public int getDownloadedBytes(){
return downloadUtil.getBytesDownloaded();
}

public InetAddress getInterfaceInetAddress() {
return interfaceInetAddress;
}

public boolean isDownloadThreadInterrupted(){
return downloadThreadInterrupted;
}

public void setDownloadThreadInterrupted(boolean
downloadThreadInterrupted) {
this.downloadThreadInterrupted = downloadThreadInterrupted;
}

public BlueChannel getChannel() {
return channel;
}

public void setChannel(BlueChannel channel) {
this.channel = channel;
}
}

```