Dual Degree Dissertation

GPRS Simulations using ns-Network Simulator

submitted in partial fulfillment of the
requirements for the degree of

Bachelor of Technolgy and

Master of Technology

under the Dual Degree Program in

Communications and Signal Processing

by

Ms. Richa Jain

Roll No. 96D07007

under the guidance of

Prof. Sridhar Iyer

Department of Electrical Engineering,
Indian Institute of Technology - Bombay,
Powai, Mumbai 400 076.

June, 2001.

Dissertation Approval Sheet

The Dual Degree Project report entitled " **Extensions for GPRS Simulator in** *ns* " submitted by **Ms. Richa Jain** (Roll No. 96D07007) may be accepted.

Chairman: ——————

External Examiner: ——————

Guide: ——————

Internal Examiner: ——————

Acknowledgement

I would like to thank my advisor, **Prof Sridhar Iyer**, for his continual encouragement, help and guidance during the course of this project.

I would also like to thank Mr Joerg Widmer, ICSI, Berkeley for his permission to incorporate the NOAH Agent; Mr Sandeep Kumar and Mr Kopparapu Suman for their patience and for their contributions to the code; Mr Srinath Perur and Dr Leena Chandran Wadia for their help and feedback.

I would also like the thank Prof Abhay Karandikar and members of the Infonet Lab.

Richa Jain

Abstract

Enabling wireless Internet access at data rates comparable to wired networks, is a growing concern in recent years. One attempt at this, is the General Packet Radio Service (GPRS), a packet based extension to GSM. The packet switched radio transmission enables efficient multiplexing of radio resources and data rates of up to 170 kbps can be achieved.

GPRS (General Packet Radio Service) Networks are currently being deployed in the market, and there is a need to study performance and network related issues. Simulations can provide a basis for evaluation and key decisions for deployment. In this project we undertake design and implementation of a GPRS simulator. We focus on the handling of the air interface and the Link Layer, Radio Link Layer and the Medium Access Layer for the Mobile Station - Base Station interaction. We use the *ns-Network Simulator* as a base for the implementation.

Using the simulator, we then study the effect of load conditions on the average packet delay and the number of users supported by a GPRS system vs a GSM system.

Contents

List of Figures

Abbreviations

ARQ	Automatic Repeat Request
BSC	Base Station Controller
BSS	Base Station System
BSSGP	BSS GPRS Protocol
BTS	Base Transceiver System
CCCH	Common Control CHannel
DLL	Data Link Layer
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GSN	GPRS Support Node
GTP	GPRS Tunneling Protocol
LL	Link Layer
MAC	Medium Access Control
MM_State	Memory Management State
MNRF	Mobile Not Reachable Flag
MS	Mobile Station
PACCH	Packet Associated Control CHannel
PAGCH	Packet Access Grant CHannel
PBCCH	Packet Broadcast CHannel
PCCCH	Packet Common Control CHannel
PDCH	Packet Data CHannel
PDTCH	Packet Data Traffic CHannel
PNCH	Packet Notification CHannel
PPCH	Packet Paging CHannel
PRACH	Packet Random Access CHannel
PDN	Packet Data Network
PDP	Packet Data Protocol (for eg IP or X.25)
RLC	Radio Control Layer
SGSN	Serving GPRS Support Node
SNDCP	SubNetwork Dependent Convergence Protocol
TDMA	Time Division Multiple Access
ns	Network Simulator

Chapter 1

Introduction

In view of the significant demand for wireless data services, providing faster, cheaper and more reliable access "on the move" is becoming an important concern. Currently, the most widely deployed mobile communication standard is the Global System for Mobile Communication (GSM). It is a circuit switched system and can provide data rates only up to 9.6 kbps. This severely limits the services and applications it can support.

An attempt at providing higher data rates is the General Packet Radio Service (GPRS). It is a packet-based enhancement to GSM and allows a maximum data rate of 170 kbps. The packet based approach is ideally suited for applications that generate bursty traffic for especially Internet applications. Efficient multiplexing of the radio resources enables it to support many more users. Also users are allowed to be 'connected' always and charging is based on to the volume of traffic. Thus it leads to lower call set-up times, and proves much cheaper.

1.1 Motivation

GPRS systems are in the process of being deployed world wide. Vendors and network operators are faced with key decisions regarding infra-structural setup, bandwidth allocation policies and QoS issues. The GPRS specifications allow for many such implementation choices by the manufacturer or network operator. A few of these are the optimal sharing of radio resources between GSM and GPRS; resource allocation policies to be used for single and multi-slot operation; parameter values for the resource release mechanism and so on. Choosing an appropriate mechanism or parameter value can greatly affect system performance and capacity.

Actual deployment and trials to find the optimal performance parameters for a GPRS system is quite infeasible.

In such a situation, a simulation environment is definitely a better option for preliminary testing of mechanisms and algorithms. It would not only provide much greater flexibility and range in studying the issues of interest, but also considerably narrow down the search for optimal parameters of the system. The simulations can then serve as a framework for a limited set of field trials.

1.2 Scope of the Project

In this project, we design and implement a simulator for GPRS. We use ns-Network Simulator, a widely used, public-domain simulator, as the base for our work.

We focus our attention on the main factors that give GPRS an edge over GSM - its packet-based approach and the multiplexing of radio resources. We simulate the network stack for the GPRS Mobile Station (MS) and Base Station (BS) and focus on the handling of radio resources.

In particular, modules for the Link Layer Control, the Radio Link Layer and the Medium Access Control between the Mobile Station and Base Station have been implemented in *ns*.

Our simulator supports GSM and GPRS mobiles in single-slot operation, two-way data transfer between the MS and BS (localized to single cell), amongst other features. Details of the simulator are described later in the thesis.

1.3 Organization of the Report

In the second chapter, we give an overview of GPRS - the network architecture, the protocol layers and the radio interface. The next chapter describes *ns-Network Simulator*, its features and internals. Modifications and additions to *ns* for GPRS simulations are described in chapter four. Chapter five gives the details of the MAC implementation done by the author. Simulations performed are described in chapter six. Though the scope of this thesis was limited to the MS-BS interaction, nevertheless we present a design for the implementation of the GPRS support network in chapter seven. Finally we give the conclusions and outlook in chapter eight. Psuedocode for the MAC implementation and Users' Manual can be found in the Appendix.

Chapter 2

An Overview of GPRS

Existing technologies like GSM are circuit switched - at the air interface, a complete traffic channel is allocated to a user for the entire duration of a call. It will remain idle, in case there is no data to be transmitted in certain intervals during the call. This limits both the data rates and the number of users that can be supported by circuit switched systems. Also, the connection setup can take up to several seconds, and data rates are restricted to 9.6 kbit/s. The search for a more efficient handling of the air-interface leads us to GPRS.

The General Packet Radio Service (GPRS), like the name suggests, is a mobile communication standard based on packet switched radio transmission. The main feature that gives it an edge over the existing circuit switched technologies like GSM is its handling of the radio resources. A traffic channel is alloted only when needed and is released immediately after the transmission of packets is over. GPRS also allows for a user to be alloted multiple channels, leading to higher data rates. Data rates up to 170 kbits/s can thus be achieved. GPRS call set up times are typically less than a second. Billing is based on the volume of traffic rather than connectivity and the users can thus be connected throughout.

We proceed to describe the details of the GPRS system in the following sections.

2.1 The GPRS Network Architecture

GPRS is built as an extension to GSM. It utilizes much of the GSM network infrastructure and shares the same radio resources. To enable direct routing of packets to external Packet Data Networks (PDN), two extra nodes have to be added to the GSM core network. These comprise the GPRS backbone network. The radio system comprises the Mobile Station (MS), which is the user terminal equipment, and the Base Station Subsystem (BSS) [1]. The overall GPRS network architecture is shown in Figure 2.1 [2].

The BSS consists of the Base Station Controller (BSC) and the Base Transceiver Station (BTS). The BSC supports all relevant GPRS protocols for communication over the air interface. It sets up, supervises and disconnects packet switched calls and handles cell change and channel assignment. The BTS is only a relay station (without any protocol functions) and performs modulation, demodulation and actual radio transmission and reception. The

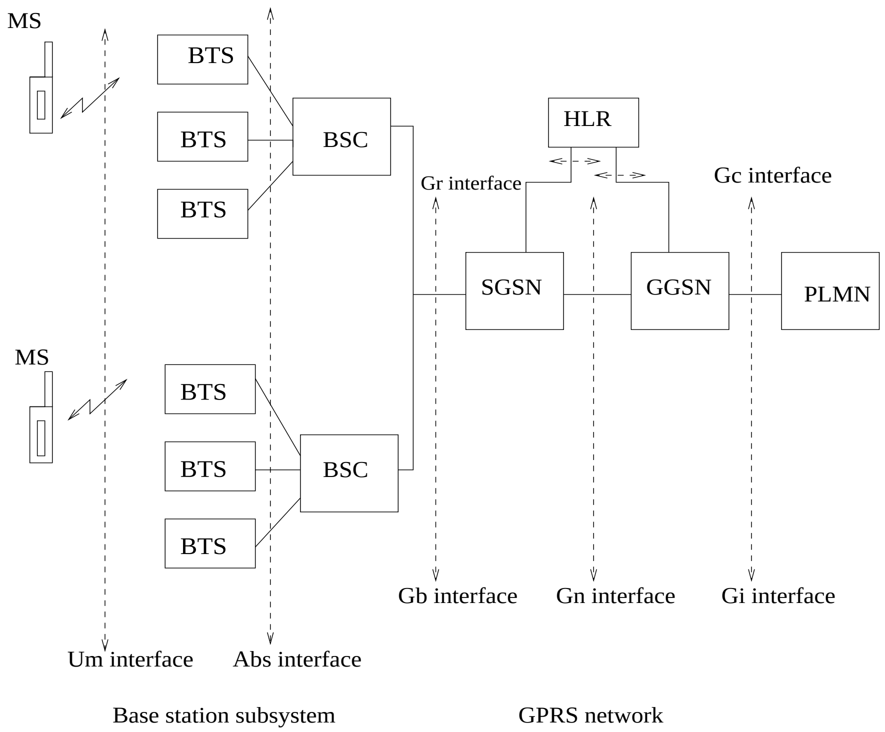

Figure 2.1: GPRS system architecture.

radio coverage area of a BTS is termed a cell.

The two logical nodes, the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN), serve as the GPRS core. The SSGN is responsible for the MS in its service area - data exchange to/from the MS, maintaining location information for the MS and detecting new GPRS MS in its service area. The GGSN provides inter networking with the external Packet Data Network (PDN) and is connected with the SGSNs via an IP-based GPRS backbone network. The PDN may be either IP based or X.25.

In order to interact with the PDN, MS have to be alloted PDN addresses (IP or X.25 address depending on the external PDN). This is done either from a pool of PDN addresses reserved for that GPRS network (ie available with the HLRs of the GPRS network) or dynamically from a pool of PDN addresses available at the SGSNs (similar to Care-of-Addressing in Mobile IP).

A Home Location Register (HLR) maintains all GPRS-user related data needed by the GGSN to perform routing and data transfer. Any node wanting to correspond with an MS first finds the MS' current location from the MS' HLR, and routes the packet to the appropriate SGSN via the GGSNs. The SGSN then looks up the Routing Area/ Cell from its Visitor Tables and routes the packet to the appropriate BSS which transmits it to the MS over the air interface. A simplified example of routing is shown in Figure 2.2 [3].

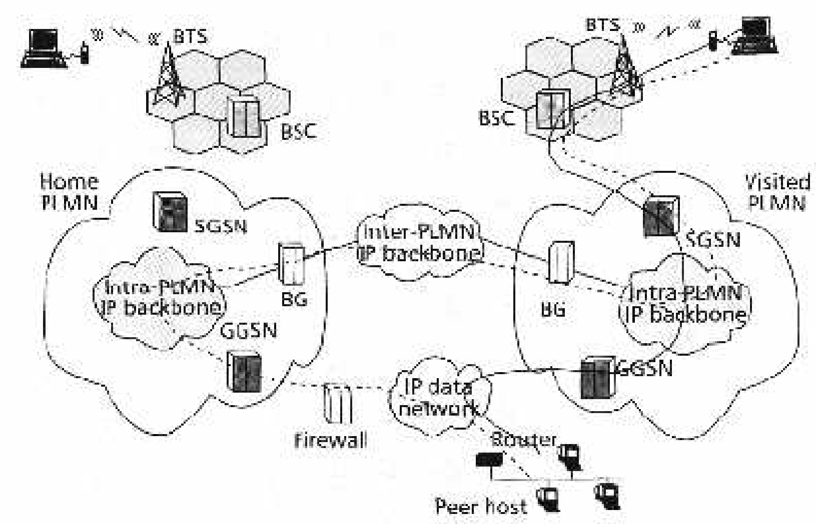

Figure 2.2: A simple routing example in GPRS.

2.2 Protocol Architecture

The network protocol stack for GPRS is shown in Figure 2.3 [4].

The GPRS Tunneling Protocol (GTP) encapsulates and tunnels user data and signalling within the GPRS backbone network. TCP carries GTP packet data units (PDUs) in the GPRS backbone network for protocols that require a reliable link (ex X.25) and UDP carries GTP PDUs for protocols that do not require a reliable link (ex IP). IP is used for routing within the GPRS backbone [5]. Ethernet, ISDN or ATM based protocols may be used below IP depending on the operator's network architecture.

Between the SGSN and MS, the Subnetwork Dependent Convergence Protocol (SNDCP) maps network level protocol characteristics onto the underlying LLC. It multiplexes network layer messages onto a single virtual logical connection and handles encryption, segmentation as well as data and header compression.

The BSS GPRS Protocol (BSSGP) conveys routing and QoS information between BSS and SGSN.

Radio communication between MS and the GPRS network is covered by physical and data link layer (DLL) functionalities. Between MS and BSS the DLL is split into the Logical Link Control (LLC) and the Radio Link Control/ Medium Access Control (RLC/MAC).

The LLC provides a logical link between the MS and SGSN over the *Um* and *Gb* interfaces [6]. It fragments the higher layer PDUs before sending them down to the RLC. Its functions comprise ciphering, flow control and sequence control. It can be used in acknowledged or unacknowledged mode.

The RLC/MAC handles the QoS control and BEC, along with segmentation and rearrangement of LLC PDUs [7]. The MAC operates between the MS and BTS and is derived from the slotted ALOHA system with selective, bit-map based ARQ. It handles access sig-

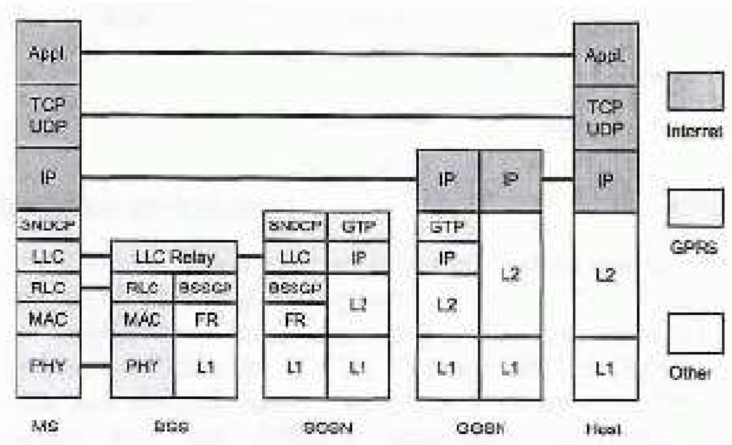

Figure 2.3: A simple routing example in GPRS.

nalling (request and grant), transmission of data blocks over the air interface and multiplexing data and signalling traffic onto the physical channel.

2.3 Air Interface

GPRS utilizes GSM radio-resources keeping the same hybrid TDMA/FDMA structure. Certain time slots on the TDMA are statically or dynamically reserved for GPRS. An MS can operate in single slot mode or in multi-slot mode (ie use multiple time slots of a TDMA frame).

A channel is taken to imply a particular TDMA slot on a certain frequency. One or more channels taken from the available pool of GSM channels may be dedicated to GPRS packet data traffic. Such channels are called Packet Data Channels (PDCH). These can be alloted to GPRS on a static or dynamic basis. One of these channels, called Packet Common Control Channel (PCCCH) is reserved for all the necessary control signalling, initiating packet transfer as well as user dedicated signals. The others are used only for data transfer. Note that the existence of PDCH does not imply the existence of a PCCCH. If no PCCCH is alloted in a cell, all GPRS MS will camp on the existing GSM CCCH.

The logical channels in GPRS are summarized in the Table 2.1. The PBCCH transmits system information to all GPRS MS in a cell. Of the common control channels, PRACH initiates packet transfer and response to paging messages; PPCH is used to page an MS prior to downlink packet transfer; PAGCH assigns resources prior to packet transfer; and the PNCH is for PTM multicast notification of resources assigned for the multicast. The packet transfer channels are the PDTCH (a MS may use more than one PDTCH in parallel for packet transfer). PACCH is for signalling information related to a particular MS for example acknowledgements, power control information and resource assignment and reassignment

Channel	Function	Group	Direction
PBCCH	Packet Broadcast Channel	Broadcast	Downlink
PRACH	Packet Random Access Channel	Control	Uplink
PPCH	Packet Paging Channel	Control	Downlink
PAGCH	Packet Access Grant Channel	Control	Downlink
PNCH	Packet Notification Channel	Control	Downlink
PDTCH	Packet Data Traffic Channel	Data Traffic	Uplink and Downlink
PACCH	Packet Associated Control Channel	Data Traffic	Uplink and Downlink

Table 2.1: GPRS Logical Channels

Scheme	Code rate	Payload	BCS	Pre-coded USF	Tail bits	Coded bits	Data rate (kbps)
CS-1	1/2	181	40	3	4	456	9.05
CS-2	2/3	268	16	6	4	588	13.4
CS-3	3/4	312	16	6	4	676	15.6
CS-4	1	428	16	12	0	456	21.4

Table 2.2: GPRS Coding Schemes

messages.

2.4 MultiFrame structure for Packet Data Channels

The Radio Block (RB) is the basic transmission unit of a PDCH. Radio blocks are transmitted in 4 time slots spanning 4 consecutive TDMA frames. A radio block is 456 bits long and this includes FEC and other header fields. The actual RB structure and the number of payload bits depends on the message type (data or control message) and the coding scheme used (GSM 05.03 specifies 4 different coding schemes as shown in the Table 2.2 [5]).

A PDCH is structured in multi frames each comprising 52 TDMA frames. Every 13th frame is left idle so that only 12 radio blocks fit in a multi frame. Thus the mean transmission time of a radio block is 20 ms.

More users can be supported than the number of available PDCHs since the PDCHs are alloted and released dynamically during idle periods. Also, each MS can transmit over multiple PDCH in parallel, thus allowing for higher data rates.

In this thesis, we limit ourselves to the MS-BS interface. We focus on on handling of the radio resources, and on the network stack, the LL, RLC and MAC operating between the MS and BS. These are simulated using *ns-Network Simulator*. We next go on to describe *ns* and its internals.

Chapter 3

About ns-Network Simulator

The ns-Network Simulator is a public-domain, event driven simulator [8]. *ns* was chosen as the base simulator for our work because of the range of features it provides, along with the fact that it has an open source code that can be modified and added to.

Ns supports the following features

- Elements for network topology - nodes and point to point links.
- Routing - unicast, multicast, hierarchical.
- Queueing schemes - drop-tail, RED, CBQ, FQ and SFQ.
- Transport protocols - TCP and UDP.
- Traffic generators - CBR, Exponential and Pareto.
- Applications - web cache.
- Basic mobility - ad hoc networks, mobile IP.

The simulator uses a split-language programming approach. OTcl is used for describing the simulation scenario, scheduling events and dynamic configuration of components during simulation. The actual core of the simulator, ie per-packet processing code, is written in C++. This approach allows for fast generation of large scenarios, but also adds complexity. To simply use the simulator, it is sufficient to know OTcl, but to add and extend the simulator, requires programming at both levels and debugging simultaneously in both languages.

3.1 Scenario Generation using *ns*

The generation of simulation scenarios in Tcl scripts *ns* is done as follows:

- Set the options for simulator configuration.

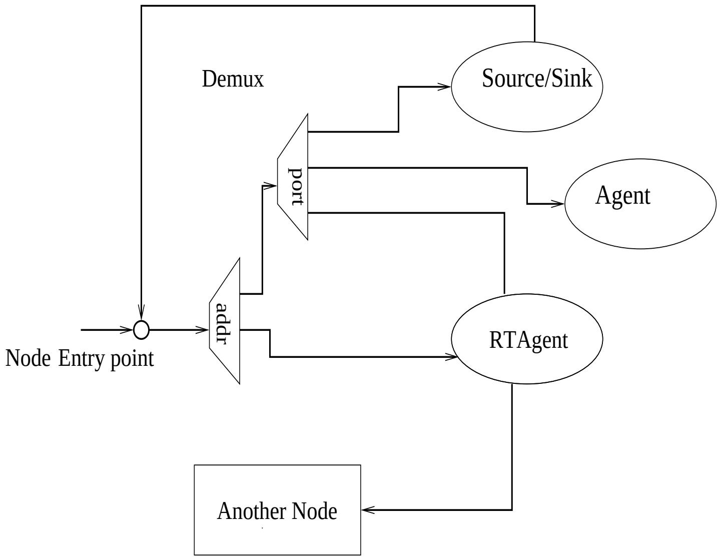

Figure 3.1: A fixed node in *ns*.

- Create a simulator instance.
- Set up the topology.
- Set up traffic.
- Schedule movement or other dynamic changes during the simulation.
- Tracing the events.

The topology consists of network nodes connected by links with certain bandwidth, delay and queueing strategy. The structure of a node is shown in Figure 3.1. To exchange packets amongst nodes, agents have to be attached to the nodes. Agents can be routing agents or sending/receiving agents for TCP or UDP packets.

A sample script is given in below:

```
# Using dynamic routing; using arrays to create nodes.

set ns [new Simulator]
$ns rtproto DV
set nf [open out.nam w]
$ns namtrace-all $nf

#this closes the trace file and starts nam
proc finish {} {
global ns nf
$ns flush-trace
```

```
close $nf
exec nam out.nam &
exit 0
}

#creating 7 nodes using an array.
for {set i 0} {$i<7} {incr i} {
  set n($i) [$ns node]
}

#linking each node to the next node with a 1Mbps link,
# 10ms delay, and Drop Tail at the queue to the link.
for {set i 0} {$i<7} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}

#starting CBR traffic, with packet size 500 bytes
# at intervals 0.005 s.
set cbr0 [new Agent/CBR]
$ns attach-agent $n(0) $cbr0
$cbr0 set packetSize- 500
$cbr0 set interval_ 0.005

#creating a sink
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

# now to connect the two agents!
$ns connect $cbr0 $null0

# now tell cbr0 when to start sending packets (and till when)
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up  $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
# note: write the times in order

#call the procedure "finish" to close the trace files.
$ns at 5.0 "finish"

#start the simulation!
```

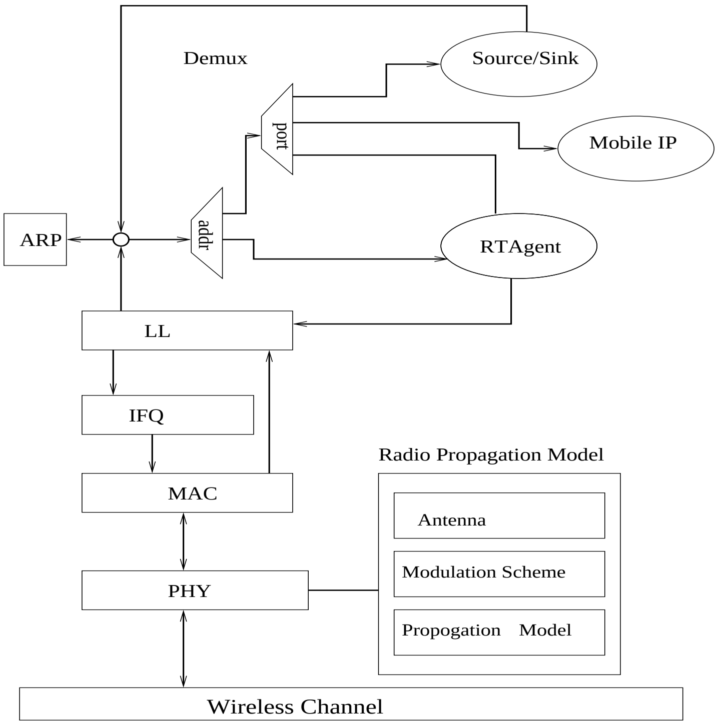

Figure 3.2: The network stack at an *ns* wireless node .

```
$ns run
```

3.2 *Ns* Internals - The Wireless Domain

In *ns* the main protocols, topology elements, per-packet processing actions etc are written as separate C++ files. These are then linked together in OTcl to build the simulator. For the purpose of this thesis, we concern ourselves only with the wireless domain in *ns*.

The wireless node is created differently from a fixed node. It has a layered structure ie a network stack as shown in figure 4.2 [9]. It has the mobility to move around in a given topology, and transmit onto and receive from a wireless channel. The mobility features include node movement (two dimensional), periodic position updates and maintaining topology boundary. These are implemented in C++ (*ns*/mobilenode.{cc,h}). Each of the layers is implemented individually in C++. The stack is linked together by OTcl in *ns*/tcl/lib/ns-mobilenode.tcl. By default, each node is configured as "ad-hoc" and can send or listen to any other mobile node within reach.

3.2.1 The Network Stack on an *ns* Wireless Nodes

The network stack is briefly described below

- **Link Layer(LL)** is responsible for simulating the data link layer protocols. It sets the Mac destination address in the Mac header of the packet. Currently, it simply passes packets down to and up from the Mac. It also has an ARP module connected to it to resolve IP address to hardware(Mac) addresses. (*ns*/ll.{cc,h}, *ns*/tcl/lib/ns-mobilenode.tcl)

- **Address Resolution Protocol (ARP)**. If the ARP has the Mac address for the destination, it writes it into the Mac header of the packet else it braodcasts an ARP Request and caches the packet temporarily. For each unknown destination, there is buffer space for a single packet only. If any other packets arrive for the same destination, the buffer is over written. (*ns*/arp.{cc,h}, *ns*/tcl/lib/ns-mobilenode.tcl)

- **Interface Queue (IFQ)**. This is implemented as a priority queue which gives priority to routing protocol packets and places them at the head of queue. Also supports a filter over all packets in the queue and can selectively remove packets for a certain destination. (*ns*/priqueue.{cc,h}, *ns*/tcl/lib/ns-mobilenode.tcl)

- **Medium Access Control (Mac) Layer**. Currently implemented in ns are IEEE 802.11 and 802.3, CSMA and multihop. (*ns*/mac*.{cc,h}, *ns*/tcl/lib/ns-mobilenode.tcl)

- **Network Interface (netif)** is used by the mobile node to access the channel. The shared wireless interface is implemented as Phy/WirelessPhy. This interface is subject to collisions and receives packets via the radio propagation model. The packet header is stamped with data related to the transmitting interface like the transmit power, wavelength etc. This information is used by the propagation model to determine if the packet has the power to be received. The model approximates the DSSS (Lucent WaveLan Direct-Sequence Spread Spectrum) radio interface. (*ns*/wirelessphy.{cc,h}, *ns*/tcl/lib/ns-mobilenode.tcl)

- **Radio Propagation Model**. It uses $1/r^2$ at near distances and an approximation to the two-ray-ground model ($1/r^4$) at far distances.

A sample OTcl script for wireless scenarios is given below:

```
# Define options

set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/802_11
set val(ifq)        Queue/DropTail/PriQueue
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
```

```
set val(x)              670   ;# X dimension of the topography
set val(y)              670   ;# Y dimension of the topography
set val(ifqlen)         50    ;# max packet in ifq
set val(seed)           0.0
set val(adhocRouting)   DSR
set val(nn)             3     ;# how many nodes are simulated
set val(cp)             "../scene/cbr-3-test"  ;#traffic patterns
set val(sc)             "../scene/scen-3-test" ;# movement patterns
set val(stop)           400.0                  ;# simulation time


# create simulator instance
set ns_ [new Simulator]

# setup topography object
set topo [new Topography]

# create trace object for ns
set tracefd [open ~/wireless1.tr w]
$ns_ trace-all $tracefd

# define topology
$topo load_flatgrid $val(x) $val(y)

# Create God
set god_ [create-god $val(nn)]

# define how node should be created

#global node setting
$ns_ node-config -adhocRouting $val(adhocRouting) \
                 -llType $val(ll) \
                  -macType $val(mac) \
                  -ifqType $val(ifq) \
                  -ifqLen $val(ifqlen) \
                  -antType $val(ant) \
                  -propType $val(prop) \
                  -phyType $val(netif) \
                  -channelType $val(chan) \
                  -topoInstance $topo  \
                  -agentTrace OFF \
```

```
                  -routerTrace OFF \
                  -macTrace ON

#  Create the specified number of nodes [$val(nn)] and
# "attach" them to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
set node_($i) [$ns_ node]
}

# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
     $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ run
```

3.2.2 Routing Agents

The mobile node is created according to the ad-hoc routing protocol to be used by it. It is configured to have the appropriate tables and data structures. *Ns* supports DSR, DSDV. AODV and TORA. Thus each node, by default, can communicate to all the other mobile nodes.

3.2.3 Wired-cum-wireless scenarios

If a mobile node desires to communicate to a fixed ie wired node, it can do so by attaching to a base-station. The base-station is implemented as a mobile node in *ns* but with wired routing switched on and motion disabled. This is done at the OTcl level in *ns*/tcl/lib/ns-bsnode.tcl. Hierarchical routing has to be used with the wired and wireless parts kept in different domains.

Note that the mobile nodes can still communicate amongst themselves, but would require to use a base station for interacting with a wired node. This works fine with Mac 802.11 but not for GPRS.

3.2.4 A Non-Adhoc Routing Agent : NOAH

For a GPRS scenario, we require that each MS talks only to its BS, and **not** to any other MS. This implies we cannot use the normal ad-hoc configuration for mobilenodes provided by *ns*. Instead, the Non-Adhoc routing agent *NOAH* built by Joerg Widmer [9] was used. This creates a simple mobile node (without the tables, signalling etc normally created along with dsr/dsdv nodes) which can communicate *only* to its BS. NOAH routes all packets to and from a mobilenode, through its BS. It inherently requires the use of *ns*' hierarchical routing. Thus a NOAH node can also be used in a wired-cum-wireless scenario. *ns*/noah/noah.{cc,h}

In the next chapter we describe the modifications and extensions we have made to *ns* to enable GPRS simulation.

Chapter 4

Extensions for GPRS simulation in ns

Recall the basic features of GPRS described earlier. Our focus is on the MS-BS interactions, specifically on the simulation of the Link Layer (LL), Radio Link Control (RLC) and Medium Access Control (MAC) layers. Features of the implemented simulator are described below.

4.1 Scope of our simulator

We allow for operation within a single cell. The Base Station (BS) may be interfaced with wired nodes (using hierarchical routing) to simulate the GPRS scenario. Multiple cells are not supported since the implementation of Home Location Register (HLR), Visitor Location Register (VLR), hand-offs etc was beyond the scope of this thesis.

Traffic can be set-up in either direction from MS to BS or vice-versa. The number of frequencies available in a cell can be varied by the user through the OTcl script. The maximum number of mobile nodes permissible in the cell can also be set through the users' script. Mobile nodes can be configured as GSM MS or GPRS MS. The difference here is that GPRS MS release slots when there is no active packet transfer, while GSM MS retain their slots till the call ends. Each MS can transmit/receive only on one frequency at a time. The BS however, can listen to/transmit on many frequencies simultaneously.

In order to provide these, the LL, RLC [1] and MAC layers have been implemented with the following features

- LL: Fragmentation and reassembly of higher layer PDUs; a stop-and-wait retransmit mechanism.
- RLC: Fragmentation and reassembly of LL PDUs; a selective retransmit mechanism.
- MAC: Separate uplink and downlink frequencies; TDMA frames with 8 time slots on each; slot allocation by request; symmetrically allocation of uplink and downlink channels; slot release.

[1] The code for the LL and RLC was contributed by Mr Sandeep Kumar and Mr Kopparapu Suman as part of their Mini-Project.

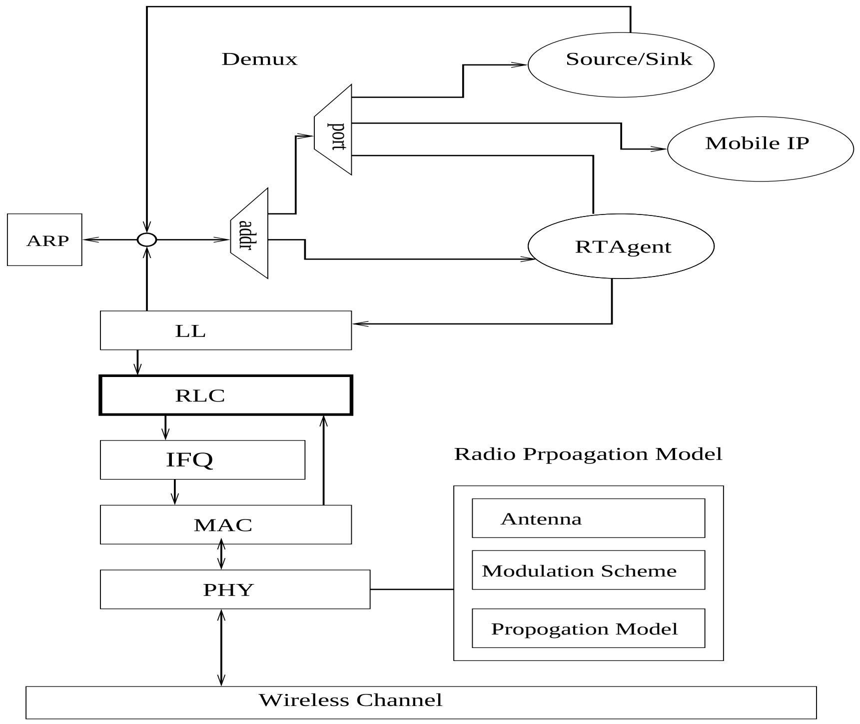

Figure 4.1: The modified network stack for a wireless node in GPRS.

These are further described in the following sections.

4.2 Features added to ns

4.2.1 Changes in the node structure

We introduce an RLC into the default *ns* mobile node structure as shown in the Figure 4.1. This involved changing the nodal structure in *ns*/tcl/lib/ns-mobilenode.tcl and other changes in *ns*/tcl/lib/ns-lib.tcl and *ns*/tcl/lib/ns-default.tcl.

4.2.2 Link Layer

The default LL provided by *ns* has been modified to include fragmentation and acknowledged mode as options.

The LL can be used in fragmented or unfragmented mode. This can be configured from the OTcl script. In fragmented mode, packets received from the sending Agent are segmented before sending down to the RLC and are reassembled at the other end, before being passed up to the receiving Agent. The fragment size can be set by the user. It is currently set at a default of 1500 bytes in accordance with the maximum LL PDU size specified in GPRS.

The LL can be used in acked or unacked mode. A stop-and-wait retransmit mechanism is implemented. The LL header has been modified to accommodate these changes. *ns*/ll.{cc,h}

4.2.3 Radio Link Control

Here again, the main features included are fragmentation and assembly, along with RLC retransmissions. The mechanism is a simplified form of Selective Retransmits. Again, fragmentation-reassembly and acknowledgements may be configured ON or OFF. The RLC fragment size can be set. A new header, the RLC header is introduced. *ns*/rlc.{cc,h}

In case an RLC fragment is dropped by the MAC, in acknowledged mode, a duplicate acknowledgment for the last correctly received RLC fragment is sent back to the sender, which then retransmits the expected RLC fragment. In unacknowledged mode, if an RLC fragment (of an LL PDU) is missing, the RLC does not pass *any* of the fragments to the LL. The LL in this case, would re-send the LL PDU (if in acknowledged mode) or will let the higher layers (ie TCP) handle it.

As mentioned earlier, the actual size of an RLC PDU's payload depends on the coding scheme used. We take the average GPRS RLC PDU size to be 200 bytes. In GPRS these 200 bytes are transmitted over four slots in consecutive TDMA frames, amounting to 50 bytes per slot. To model this in our simulator, we configure our RLC PDUs to be of 50 bytes and thus transmit 50 bytes in each TDMA slot.

4.2.4 Medium Access Control

The MAC is similar to a reservation based slotted Aloha. One RLC fragment (50 bytes) is transmitted per slot. Slot 0 on both the uplink and downlink is reserved for signalling and broadcasts. The user can set apart a number of slots for GPRS traffic. The rest are left for GSM mobiles.

We also have a random error model which allows the BS to randomly drop packets or mark them as erroneous. Details of the MAC implementation are described in the next chapter. *ns*/mac-gprs.{cc,h}

Chapter 5

MAC Implementation Details

In this chapter we give a description of how we have implemented the MAC layer. We describe the basic channel, the TDMA slot structure, modeling of packet transmission and reception, along with the slot handling and call handling mechanism used. We also describe how we have dealt with exceptions like collisions and errors on the channel.

The relevant code for our work can be found in *ns*/mac-gprs.{cc,h} A psuedocode for the implementation is given as Appendix 2.

5.1 Channels

The physical air interface provided by *ns* is used. The number of frequency channels to be created for the Uplink and Downlink can be set by the user through *max_num_freq* in the OTcl script. We differentiate between the uplink and downlink channels by creating separate timers to clock the TDMA on each. A hard-coded skew of 3 time-slots is maintained between the uplink and downlink TDMA frames. On each frequency (uplink as well as downlink), slot 0 is reserved for signalling and broadcasts. The user can decide the number of slots to reserve for GPRS traffic on each frequency through *gprs_slots_per_frame* in the OTcl script. The remaining slots are left for GSM mobiles. The frequency channel a packet is to be transmitted on is stamped onto a new field called *chan_* in the common header of the packet.

5.2 TDMA Slot Structure

Every TDMA frame has 8 slots (defined as SLOTS_PER_FRAME). Slot duration is set as 577 microseconds. We transmit 50 bytes ie one (simulated) RLC PDU, in each slot. This models the transmission of one GPRS Radio Block (RB) (of size 200 bytes) over four slots in consecutive frames. For packets smaller than the size of an RLC PDU (ex *rlc_acks* or *ll_acks*), the packet transmission time is taken as $packet\_size/transmission\_rate$.

We use *Up_Slot_Gprs_Timer* and *Down_Slot_Gprs_Timer* to signal the start and end of each slot on the uplink and downlink respectively, and the processing is handled by the *upslotHandler()* and *downslotHandler()* respectively.

The *upslotHandler()* checks whether any MS has a packet to transmit in the current upslot. If it does, the packet is passed onto *tx_onto_PHY()* that starts the 'transmission' of the packet. Similarly the *downslotHandler()* checks at the BS.

5.2.1 Timing Advance

The upslot/downslot TDMA frames at the MS and the BS should be synchronized. But the finite propagation delay between the MS and BS causes a mismatch. Therefore, the clock at the MS would have to be advanced by the finite propagation delay time, in order to maintain synchronism. Implementing this would have introduced undue complexity in our simulator. Instead, we work a way around this problem by setting the propagation delay to zero (in *ns*/wirelessphy.cc).

5.3 Packet Transmission and Reception

We model packet transmission over the air-interface by a timer that keeps track of how long the radio transmission should take and signals when the transmission is over. The packet transmit timer (*TxPktGprsTimer*) is started at the beginning of the appropriate slot by *tx_onto_PHY()*. On expiry of the transmit timer, the *sendHandler()* is called, which frees the packet, switches off the radio, and unlocks the IFQ.

If a MS or the BS senses a packet destined for it at the air-interface, it calls *rx_from_PHY()*. This starts a receive timer (*RxPktGprsTimer*) to model the actual radio reception and also checks for collisions. On expiry of the receive timer, the *recvHandler()* is called, which checks whether the packet is in error or can be received, and sends it on to be processed by *fwd_DATA_to_LL()* and forwarded to the RLC.

Since we take propagation delay to be zero, the transmit and receive timers effectively start (and end) together.

5.4 Call Set-up and Handling

The following messages are used for call set-up and handling.

5.4.1 Resource Request

The first packet received by the MAC of an MS (from the IFQ) triggers a *resource_request* message to the BS. We buffer the packet and the lock the IFQ to prevent it from sending down further packets. A wait_timer is started to keep track of the time an MS waits for a *resource_reply*. The *resource_request* message is transmitted on slot 0 on uplink frequency 0.

5.4.2 Resource Reply

The BS allots a channel (slot-frequency) to an MS either on the receipt of a *resource_request* from the MS or on the receipt of a packet from its own IFQ for the MS (slot allocation is described in the next section). Information about the slot-frequency channel is written onto the MAC header of the *resource_reply* packet and it is transmitted on slot 0 of downlink frequency 0, to the appropriate MS.

On receipt of a *resource_reply*, an MS checks the slot-frequency channel alloted to it and stores it in *tx_chan[]* and *rx_chan[]* for future reference. It then kills the wait timer; schedules the old buffered packet to be transmited on the appropriate slot-frequency and unblocks the IFQ. Other waiting packets can now be passed down from the IFQ to the MAC and transmitted.

In case the BS has a second *resource_reply* message to be transmitted in slot 0 of the upcoming frame (this situation can arise when the BS receives a *resource_request* from an MS and a packet destined to some hitherto unattached MS from its IFQ in the same TDMA frame), the second reply is stored in a temporary buffer and transmitted in the next TDMA frame with slot 0 free.

5.4.3 Resource Release

Since Internet traffic is mostly bursty, we have implemented a slot release mechanism for GPRS MS. If the IFQ of the MS is empty and no packet is transmitted or received for four TDMA frames while the MS is holding a channel, we initiate a *resource_release*. The MS clears its *tx_chan[]* and *rx_chan[]* entries and sends a *resource_release*. On receipt of a *resource_release*, the BS purges its *vlr_.up_table* and *vlr_.down_table* entries. Only an MS can initiate a *resource_release*.

If the MS later wants to restart transmission, we send another *resource_request*. This request is treated on par with any other fresh requests by the BS. In the case of traffic from the BS to MS, the BS will allot fresh resources to the MS and send a *resource_reply* informing the MS.

5.5 Slot Handling

5.5.1 Allocation

At the BS we maintain a table recording which Upslot/Downslot has been alloted to which MS (in *vlr_*, specifically *vlr_.upslot[][]* and *vlr_downslot[][]*). When the BS receives a *resource_request* from an MS or a packet from its own IFQ (to be sent to an MS), it allocates the *first free slot* available to that MS. This is done in *slot_allot()*. If the MS is a GPRS MS, a slot is allocated from the pool of GPRS slots, else from the pool of GSM slots. By default, an MS is GPRS and four slots on each frequency are reserved for GPRS (this leaves only three slots on each frequency for GSM). Allocation is symmetric on the uplink and the downlink frequencies. Only single slot operation is supported ie each MS can be alloted only one slot.

5.5.2 Release

Slot release is possible only with GPRS mobile nodes. We maintain a release timer (*SlotReleaseTimer*) at the MS. On receiving a packet (in either direction) the MS checks the IFQ. If the IFQ is empty, the *SlotReleaseTimer* is started. It is reset if the MS gets another packet (in either direction) within four TDMA frames. Otherwise, on expiry of the timer, the *releaseHandler()* is called.

Currently, the value of four TDMA frames for the release timer is taken based on simulations performed. It was found optimal in preventing spurious time-outs and avoiding waiting too long. However, the optimal value may differ according to traffic generation pattern and needs to be further explored.

5.6 Dealing with Collisions

Since this is a reservation based slotted Aloha system, the only place collisions can occur is during contention on the Packet Random Access CHannel (ie slot 0 on uplink frequency 0) when more than one MS send a *resource_request* at the same time. We use *chan0_0* to maintain the state information about the PRACH. In case of collision, the colliding *resource_request* packets are dropped by *rx_from_phy()* at the BS. The MS wait one TDMA frame for a reply. Since no reply is received, the MS set the *Backoff Timer* for a random interval (generated using *Random::integer(k)*). On expiry this calls the *backoffHandler()*, which then schedules another *resource_request*.

5.7 Error model

The error model provided by *ns* can introduce errors into packets created by Agents like TCP or UDP. It cannot introduce errors at the lower layers. In order to test our acknowledgement mechanism, we introduce an error model that produces random errors in (simulated) RLC PDUs ie for slot level transmissions.

This error model (in *bs_recv()*) marks a randomly chosen RLC fragment (either going up or down) as erroneous. This causes the packet to be dropped at its destination. At each drop, we use *Random::integer(error_rate_)* to decide the next drop.

The user may choose to include this error model through *rlc_error_* in the OTcl script. The *error_rate_* can also be set by the user from the OTcl script.

5.8 Handling ARPs

At the start of a simulation, the first packet to be received at the Mac of an MS is an ARP request. Though this is a broadcast message, it is not transmitted directly, but resources are requested and the ARP request is sent out only on the alloted slot-frequency channel. This is to prevent other MS from receiving an ARP request from an MS. The BS however, goes ahead and broadcasts the ARP request on the broadcast channel ie slot 0 on downlink frequency 0.

Also, in *ns*, a node sends an ARP request each time a packet is received at the LL, even though an ARP request has already been sent and it is waiting for a reply. To prevent our MAC from transmitting such duplicate ARP requests and wasting precious radio-resources, we simply drop duplicate ARPs in *ms_recv()*

5.9 MAC Code Summary

The elements of the code - the storage structures, the methods and the timers used - are as follows

5.9.1 Storage Structures

In this section, we summarize the important storage structures used.

- At the MS

`Packet *pktTx[i]`: pointer to the packet to be sent in Upslot i
`Packet *pktRx[i]`: pointer to the packet received in Downslot i
`int tx_chan[i]`: which channel should the MS transmit on in Upslot i
`int rx_chan[i]`: which channel should the MS listen to in Downslot i

Note: `tx_chan[0]` = 0 for Random Access
`rx_chan[0]`=0 for Broadcasts and other messages from BS
This structure also allows for multi-slot allocation to be included later.

- At the BS

`Packet *txQ[i][j]`: pointer to the packet to be sent in Downslot j, frequency i
`Packet *rxQ[i][j]`: pointer to the packet received in Upslot j, frequency i
`int vlr_.hier_addr_[i]`: hierarchical address of MS with MAC index i
`int vlr_.up_table[i][j]`: MAC index of MS alloted Upslot j on frequency i
`int vlr_.down_table[i][j]`: MAC index of MS alloted Downslot j on frequency i

Note: These structures have to be unique for each BS. Since *ns* does not currently have a separate BS node at the C++ level (*ns* configures a node as BS only at the OTcl level), these structures had to be made static. This limits our simulator to supporting just one BS and consequently, just one cell.

5.9.2 Methods

The methods created for the class Mac/GPRS are

- `recv(Packet *p, Handler *h)` : the entry point for the MAC
- `ms_recv(Packet *p, Handler *h)` : packet processing at the MS
- `bs_recv(Packet *p, Handler *h)` : processing a packet at the BS
- `slot_allot(int ms_, int &freq, int &slot)` : allot a slot to ms_, return values through &freq, &slot
- `send_res_reply (int dst, int freq, int slot)` : creating a resource_reply packet, to be sent on slot/freq
- `send_res_request ()` : creating a resource_reply packet
- `send_let_go()` : creating a resource_release packet
- `downslotHandler(Event *e)` : actions at the end of every down slot
- `upslotHandler(Event *e)` : actions at the end of every up slot
- `sendHandler(Event *e)` : actions after packet has been transmitted
- `recvHandler(Event *e)` : actions after packet has been completely received.
- `releaseHandler(void)` : on the expiry of the slot release timer
- `backoffHandler(void)` : on the expiry of the back off timer
- `waitHandler(void)` : on the expiry of the wait timer
- `radioSwitch(int i)` : switching the radio on/off to conserve energy.
- `rx_from_phy(Packet* p)` : remove MAC headers, check for collisions
- `rx_from_ll(Packet* p)` : add MAC header
- `fwd_DATA_to_LL(Packet *p)` : handle data according to type and pass to upper layer.
- `tx_onto_PHY(Packet *p )` : transmit onto air interface

5.9.3 Timers

The following is a list of timers used by the system

- Up-Slot Timer : clocks slots on the Uplink frequencies.
- Down-Slot Timer : clocks slots on the Downlink frequencies.
- Packet Transmit Timer : times the transmission of a packet.
- Packet Receive Timer : keeps track while a packet is being received.
- Wait Timer : the amount of time to wait for a *resource_reply*.
- Backoff Timer : to back off for a random interval before retransmitting a *resource_request*.

The psuedocode for the implementation is listed in Appendix II. We describe the simulations performed using our extensions for GPRS in *ns* in the next chapter.

Chapter 6

Testing and Experimentation

We tested the features of the simulator through various simulations and performed experiments to study the system delay, the effect of ARQ mechanisms below TCP and a capacity analysis of GPRS vs GSM systems. These were run on a machine with 750 MHz, Pentium III processor. A summary of these tests and experiments is given below.

6.1 The Average Packet Delay Experiment

6.1.1 Studying the Average Packet Delay

Objective: In this experiment, we study the effect of traffic rate generation on the average packet delay. This is indicative of the load that can be supported by the system. We calculate the average packet delay for the TCP packets sent from all the MS to the BS.

Scenario: Five GPRS MS were used, with exponential traffic from each MS to the BS. The mean packet burst duration and the mean idle burst duration were set at 500ms.

TCP packet size was set at 1500 bytes. The LL and RLC were used in a fragmented, unacknowledged mode. The LL PDU size was set to 1520 bytes while the RLC PDU size was kept at 50 bytes (as explained in the previous chapter). The OTcl script used for this experiment is given in Appendix 1.

The data generation rate was varied from 2 kbps to 40 kbps. Simulations could be run for a maximum duration of 80s due to memory constraints.

Observations: Figure 6.1 shows how the average packet delay varies with time for different data generation rates. In Figure 6.2, we show the variation of the delay with date rate, at different instants of time. The following observations can be made from the graphs:

- The average packet delay for data rates up to 10 kbps shows little variation with time and has a steady state value around 0.15 s Figure 6.1 (i).
- For data rates greater than 10 kbps, the delay increases suddenly after about 5 s, Figure 6.1 (ii).

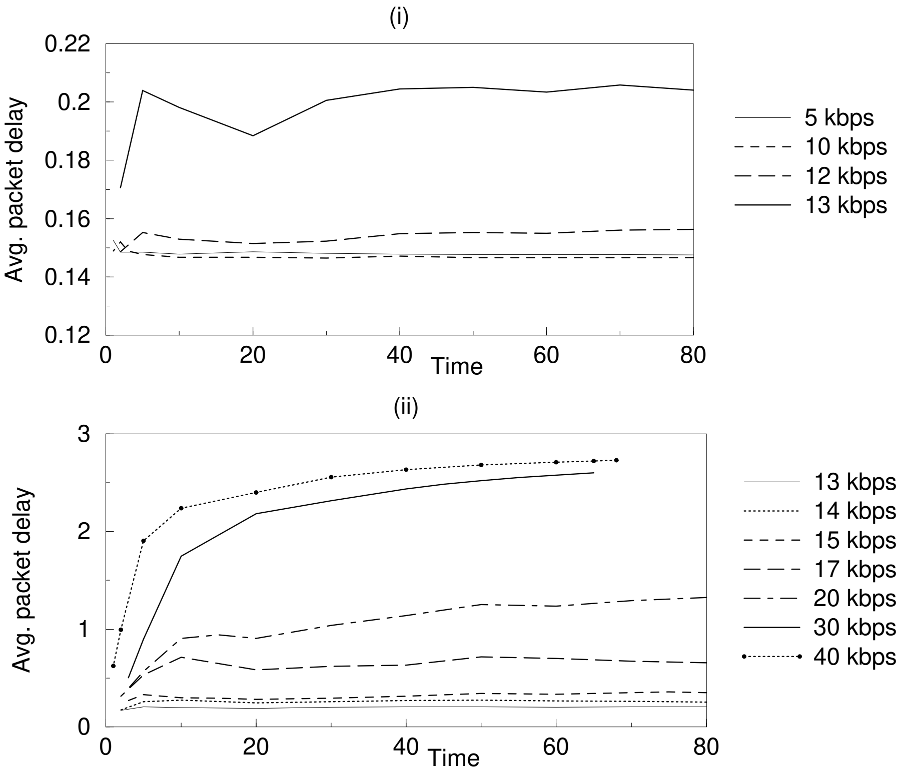

Figure 6.1: Average packet delay for various data generation rates

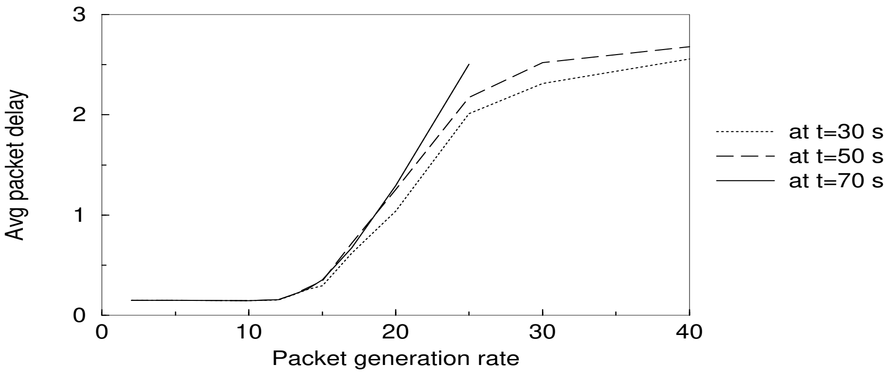

Figure 6.2: Average packet delay vs data generation rates

- The delay seems to approach a steady value of 2.6 s for data rates of 30 kbps and higher.
- There is a sharp increase in the average packet delay for data rates between 20 kbps to 18 kbps, Figure 6.2.

Inferences: We infer that traffic data rates up to 12 kbps may be supported by the system while keeping the average packet delay within a reasonable bound. This may be because at lower rates, the MAC handles packets as they come, and the delay is just the mean transmission time required for all the RLC PDUs of a TCP packet. However at larger data rates, the MAC, being TDMA, cannot handle the influx of RLC PDUs and a backlog occurs. The RLC PDUs get queued up at the MAC leading to large delays.

6.1.2 Resource Release Mechanism

As a subset of the simulations performed for this experiment, we experimented with the *SlotReleaseTimer* . This is described below.

Objective: To find an optimal value for the *SlotReleaseTimer*.

Scenario: A constant data rate of 10 kbps was used and the the value of the *SlotReleaseTimer* varied in the C++ code.

Observations:

- For each node, at the end of the bursty period after all the packets buffered at the IFQ have been transmitted, the *SlotReleaseTimer* is observed to start.
- Setting the timer value at say 9 or 15 TDMA slots did not make any difference, since the *resource_release* message can be transmitted only after every 8 slots. Hence, we tried further experiments with the timer value as multiples of a TDMA frame.
- For timer values of 2 and 3 TDMA frames, many spurious timeouts were observed. Often, a *resource_request* message was seen just after a slot release, with the BS having to allot resources to the MS in order to send back the last acknowledgment packet.
- Timer value of 5 TDMA frames and greater, resulted in the MS holding on to resources very long into their idle periods.
- Optimal operation was observed with a timer value of 4 TDMA frames.

Inferences: A value of 4 TDMA frames was found to be optimal for data rate generation of 10 kbps. This timer values has been used for the rest of our simulations. We note that, for higher data rates, the optimal release time should be lower.

6.1.3 Validation of Fragmentation and Reassembly

The above mentioned simulations use the LL and RLC in fragmented mode.

Objective: To test the fragmentation and reassembly mechanisms implemented.

Scenario: As a deviation from the above experiments, various TCP packet sizes were tried out. Also, behavior under erroneous conditions was tested using the *rlc_error* model.

Observations: The following were observed from the MAC traces and the verbose output for the LL, RLC and MAC.

- An appropriate number of fragments were created according to the size of the TCP packet and delivered in order.
- In case of receiving out of sequence RLC PDUs, the PDUs were buffered until the LL fragment was completed, and only then passed on to the LL. In LL acked mode, the LL triggered a retransmit for a missing LL PDU. In LL unacked mode, TCP triggers a retransmit.

Inferences: We conclude that the fragmentation and reassembly mechanisms work correctly.

6.2 Interaction of the LL/RLC and TCP ARQs

Objective: To study the interaction of the ARQ mechanisms at the LL and RLC with TCP's ARQ.

Scenario: A single MS was used, with exponential traffic from the MS to the BS. The mean packet burst duration and the mean idle burst duration were set at 500ms, with a data rate of 12 kbps. TCP packet size was set at 1500 bytes. The LL and RLC were used in a fragmented mode. The LL PDU size was set to 1520 bytes while the RLC PDU size was kept at 50 bytes. Simulations were run for all the combinations of LL and RLC in Acked/Unacked mode. *rlc_error* model was used to introduce slot level errors, with *error_rate* set to 10000. The simulation was run for 80 s.

Observations:

- LL unacked, RLC unacked: Two RLC PDUs were dropped. Consequently 62 RLC PDUs (31 for each TCP retransmit invoked) were observed.
- LL acked, RLC unacked: Two RLC PDUs, different from the ones in the previous case, were dropped. No TCP retransmits occurred.
- LL unacked, RLC acked: Six RLC PDUs were dropped.
- LL acked, RLC acked: Six RLC PDUs were dropped, different from the ones in the previous case. However, appropriate RLC retransmissions were observed. No TCP re-transmissions occurred.

Inferences: Since different RLC PDUs were dropped in each case, we cannot do a comparison or study the effect of the lower ARQs on TCP. However, this experiment does validate our RLC and LL acknowledged modes.

6.3 GPRS vs GSM capacity analysis

Objective: To find the capacity of a GPRS system in terms of the number of users supported vs that of a GSM system for similar load conditions.

Scenario: For the GSM simulations, *gprs_slots_per_frame* was set to zero so that all slots were reserved for GSM. The number of GSM MS was varied. Exponential traffic flow was simulated from each MS to the BS. The mean packet burst duration and the mean idle burst duration were set at 500ms for a data rate of 12 kbps. Two frequency channels were used.

For GPRS simulations, *gprs_slots_per_frame* was set to seven so that all slots were reserved for GPRS. The number of GPRS MS was varied. Each GPRS MS generated traffic similar to that for GSM.

The simulations were run for 50 s.

Observations:

- For GSM MS a slot once allotted, was held till the end of the simulation.
- Again, for GSM, a number of slots equal to the number of GSM MS were always reserved, even through idle bursts.
- For GPRS MS, slot allocation was dynamic. Slots were released during idle bursts and reallotted to other MS.
- For the case with 6 GPRS MS, each with exponential traffic, not more than 3 were seen to be transmitting at any given time. Also the *vlr_* tables showed not more than 3 slots in use at any time.

- In a variation using just one frequency channel with *gprs_slots_per_frame* set to 4, GPRS was observed to support up to 6 MS.

Simulations could not be carried out for more than 6 MS due to memory constraints.

Inferences: Since simulations for more than 6 MS could not be carried out, we cannot draw any certain inferences about the capacity supported by GSM/GPRS systems. However, the observations are in accordance with expected results. GSM can support only as many users as the number of slots available (on all frequencies), since each user books a slot for the entire duration of a call. This is confirmed by our observations for the GSM simulations. GPRS on the other hand, has the capacity to support many more users since a slot is released during idle bursts and can be given to another MS. Observation (v) shows that a GPRS system can support more users than the number of slots available to it.

Thus we have implemented and validated a simulator for the MS-BS interaction in GPRS. Though the scope of our project was limited to this, nevertheless, we propose a design for the implementation of the other features of the GPRS support network in the next chapter.

Chapter 7

Design for GPRS Support Network

In this chapter, we propose a design for the other network elements of GPRS not implemented in our simulator.

Recall the GPRS network architecture described in earlier chapters. For this design, we assume each GPRS network has only one GGSN, that is connected to the external PDNs. Also, we maintain Only one HLR for each GPRS network, and is assume that it resides at the GGSN.

7.1 Addressing

For a wired-cum-wireless network interaction, *ns* specifies the use of hierarchical addressing (only 3 tier addressing is currently allowed). We stick to this basic structure and specify the following format:

x.0.0 : address for 'the' GGSN of GPRS network 'x'
x.y.0 : address of the y^{th} SGSN of GPRS network 'x'
x.y.z : address of the z^{th} BS under SGSN 'y' of GPRS network 'x'
x.0.a : address of MS 'a' with home network 'x'
0.0.b or 0.b.c : address of nodes within the PDN

We need to specify an address format in order to distinguish various kinds of nodes by their address (and for ease of implementation).

7.2 A model for the GPRS nodes

In this section, we describe the functionalities to be implemented at each GPRS node. For each node, we describe the actions and signalling to be done on the occurrence of specific events. When a node receives/sends a packet, the relevant headers (GTP, BSSGP, SNDCP) are added and encapsulation/decapsulation done accordingly. The added headers contain a field MESG_TYPE that indicates the kind of information carried in the packet, along with other parameters needed for specific message types. Information like the source, destination, packet sender and next hop can be obtained from the default *ns* packet header.

Functionalities already implemented at the MS and BS are not repeated here. We mention only the extra functions required to complete the GPRS architecture.

7.2.1 The Mobile Station

Functionalities:

Receive multicast messages.
Receive paging messages, and respond.
Receive and send data packets.
Send registration requests (and resource request) and process reply.

Additional data needed at MS

```
MM_State            //IDLE, STANDBY, or READY
Dynamic_Address_allowed      // Flag.
QoS_profile_required
QoS_profile_negotiated
nsaddr_t current_BS
boolean DATA_TO_SEND
```

Events

1.Packet received : Note that the MS can receive packets (directly) only from the BS [2].

From	BSS
Message type	PAGE_FOR_REG^{R}

Action	`if ( MM_State == STANDBY) MM_State=READY;`
	`if ( MM_State == READY)`
	`send mesg=RESOURCE_REQ to BS`

[2] MESS_TYPE^{S,R}(XXX,YYY) implies that this message can be received only in states STANDBY or READY, and the parameters XXX, YYY are required to process the message.

From Message type	BSS BEACON^{S,R} (sending_BS)

Action	`note BS address of the 2 beacons`
	`stronger than min_rx_power;`
	`if (current_BS==NULL)`
	`      send mesg=REG_REQUEST to strongest BS;`
	`else if ((current_BS_beacon ≤ min_rx_power) &`
	`   other stronger beacon heard ) {`
	`      initiate hand-over`
	`      ie send mesg=REG_REQUEST to stronger BS. }`

Message Type	MULTICAST
Action	`if (MM_State==IDLE) call GPRS_attach();` `accept packet; process.`
Message Type	DATA^{S,R}(sending_BS)

Action	`if (current_BS==NULL)`
	`      send mesg=REG_REQUEST) to sending_BS ;`
	`if ( MM_State==STANDBY) MM_State = READY;`
	`accept data; process.`

2. Data packet generated at MS. Only in READY or STANDBY states.

To	PDN
Message Type	PDN_DATA

Action	`if (MM_State==STANDBY) MM_State=READY ;`
	`if (MM_State==READY) {`
	`      if (current_BS==NULL)`
	`         listen for beacons and register;`
	`      if ( channel==NULL)`
	`         send mesg=RESOURCE_REQ to current_BS ;`
	`      send mesg=PDN_DATA (and data) to current_BS ; }`

To	MS
Message Type	MS_DATA
Action	`same as for PDN_DATA`

3. READY time out.

Description	Can occur only in READY state.
Action	`set MM_State=STANDBY;` `send mesg=READY_TIME_OUT to current_BS.`

4. GPRS_attach().

Description	. Done at set_up or when called by user, to establish an MM context at the MS and SGSN with MM_State initially set to IDLE.
Action	`send mesg=GPRS_ATTACH to current_BS → SGSN.` `SGSN then sets MM_State to IDLE for the MS.`

5.GPRS_detach().

Description	`Called when MNRF_timer expires or the user wants to force the mobile into IDLE state.`

Action	`send mesg=GPRS_DETACH to current_BS → SGSN.`
	`// SGSN then sets MM_State to IDLE for the MS`
	`// and the Route_area and cell_identity fields to NULL.`

Note: Periodic Routing_Area updates, multiple PDP contexts, security features, dynamic address allocation have not been looked into.

7.2.2 The Base Station Subsystem

Functionalities:

Contention resolution amongst the MS sending resource requests.
Authentication of MS resource request from HLR.
Resource allocation.
Call Set up and maintenance.
- Send multicasts and data packets to MS
- Page the MS when it receives new packets to send to MS.
- Periodic beacons.

Inform SGSN (and HLR) of location updates.
Perform hand-overs.
Disconnect calls.

Additional data at the BS

```
BS visitor information:
  MS_id
  PPF        // Flag indicating whether or not paging can be initiated
```

Events

1.Packet received : The BS communicates directly only with the MS and SGSN. Let BS address be x.y.z.

From	MS
Message type	REG_REQUEST (current_BS, QoS_requested)

Action	if (current_BS==NULL) {
	add MS_ to BS_visitor_table;
	send mesg=NEW_MS(cell_id_, routing_area) to SGSN→HLR;}
	else { // Hand-over!
	send mesg=VERIFY(MS_, QoS_requested) to SGSN → HLR;
	wait for VERIFY_REPLY;
	if (VERIFY_REPLY==OK) {
	add MS_ to BS_visitor_table;
	if (y≠ SGSN of current_BS) {
	send mesg=NEW_MS(cell_id_) to SGSN y →HLR;
	SGSN y also informs old SGSN.}
	else // SGSN unchanged
	send mesg=CELL_UPDATE(cell_id,routing_area) to SGSN;
	if (mesg==UPDATED)
	send mesg=REG_REPLY(OK) to MS;
	}
	else // MS not authorized
	send mesg=REG_REPLY(NOK) to MS.

Message Type	LEAVING (new_SGSN)
Action	remove MS_ from BS_visitor_table; free the slot alloted to it.
Message Type	PDN_DATA
Action	forward to SGSN → GGSN/s → PDN. // can use *ns* internal routing.
Message Type	MS_DATA
Action	send mesg=WHERE_IS(MS_) to SGSN → HLR.

From	SGSN
Message type	DATA

Action	if (MS does not have a channel) // find from BS_visitor_table ;
	if (PPF)
	send mesg=PAGE_FOR_REG to MS;
	else mesg=DATA on channel_ to MS.

Message Type	MS_LOCATION
Action	send mesg=MS_DATA to relevant SGSN (SGSN_addr) via the GPRS network.

7.2.3 The Serving GPRS Support Node

Functionalities

To route packets from BS to the GPRS network and vice-versa
Mobility management :
  Maintain location information about the MS; its Routing Area and Cell.
  Send updates to HLR
Verify MS subscription information from HLR

Data Maintained at SGSN

List of BS it is connected to;

The following information about each MS attached (in READY state or STANDBY)to it. Note that this list is not exhaustive.

```
MS_id
MM_State             // Mobility Management State :IDLE/STANDBY/READY
Routing_area         // Current routing_area of MS
Cell_identity        // Current cell of MS
cell_identity_age    // Time since last LLC PDU was received
READY_timer
MNRF_timer
delete_MS_timer
                     // from MS at the Serving GSN
new_SGSN_address     // Address of the new SGSN where buffered and
                     // not sent PDUs should be forwarded.
MNRF                 // Mobile Not Reachable Flag
PPF                  // Whether Paging can be initiated

QoS_ssubscribed
QoS_requested
QoS_negotiated
radio-priority
SND                  // GTP seq_no of next downlink PDU for the
SNU                  // GTP seq_no of next uplink PDU for the MS
```

Events

1.Packet received. SGSN can receive packets only from the local GGSN, HLR and its BS.

From	BSS
Message Type	NEW_MS (MS_id_, cell_id, routing_area_, PPF_)

Action	`add MS_id_ to SGSN_visitor_list;`
	`in SGSN_visitor_table`
	   `set cell_identity= cell_id_ ;`
	   `set routing_area=routing_area_ ;`
	`set MNRF=0;`
	`set PPF= PPF_;`
	`send mesg=SGSN_UPDATE(MS_id_) to HLR;`

Message Type	CELL_UPDATE(MS_id_, new_cell_id_)
Action	`in SGSN_visitor_table set cell_identity= cell_id_ ;`
Message Type	VERIFY(MS_id_, QoS_requested)
Action	`send mesg=VERIFY(MS_id_, QoS_requested) to HLR;`
Message Type	WHERE_IS(MS_id_)
Action	`send mesg=WHERE_IS(MS_id_) to HLR;`
Message Type	PDN_DATA
Action	`use` *ns* `routing to route it to SGSN/GGSN` → `PDN`
Message Type	MS_DATA
Action	`send to relevant SGSN (via GGSNs if needed) ;`
From	Other SGSN in same GPRS network // if allowed
Message Type	DATA
Action	`pass on - ie use` *ns* `routing`
Message Type	MS_AT_NEW_SGSN(MS_id_, new_SGSN_)
Action	`set new_SGSN_addr= new_SGSN_ in SGSN_visitor_table;` `start delete_MS timer ;`
From	GGSN
Message Type	MS_DATA
Action	`look up MS from SGSN_visitor_table;` `send to relevant BS/cell ;`
From	HLR
Message Type	VERIFY_REPLY
Action	`send to BS that sent VERIFY query`
Message Type	MS_LOCATION
Action	`send to BS that sent WHERE_IS query`

2.Time-outs:

MNRF time-out	`change MS' MNRF to 1`
READY time-out	`change MS' MM_State to STANDBY`
Delete_MS time-out	`// MS has shifted to new SGSN` `remove MS from SGSN_visitor_list`

7.2.4 The Gateway GPRS Support Node

Functionalities:

Route packets between the PDNs and GPRS Mobile Stations. May involve routing amongst different GPRS subnetworks.

Data Maintained at GGSN

List of PDNs it is connected to.

List of other GGSNs it is connected to.

List of SGSNs it is connected to.

Note: This information is required for routing and can be obtained from the routing tables maintained by *ns*

Events

1.PACKET RECEIVED. GGSN can receive packets from external PDNs, other GPRS subnetworks, the local and external HLRs, and local SGSNs.

From	external PDNs
Message Type	NULL // since external packets do not have message field
Action	`//packet would be data meant for a mobile station` `send mesg=WHERE_IS to HLR of MS_`
From	other GGSNs
Message Type	PDN_DATA
Action	`//data meant for a PDN the GGSN is connected to` `forward to PDN`
Message Type	MS_DATA(current_SGSN)
Action	`//data meant for an MS visiting the local GPRS subnetwork` `send mesg=MS_DATA to SGSN_`
From	SGSNs
Message Type	PDN_DATA
Action	`//data meant for a PDN the GGSN is connected to` `forward to PDN`
Message Type	MS_DATA(current_SGSN)

Action	`//data meant for an MS visiting another SGSN`
	`send mesg=MS_DATA to appropriate SGSN,`
	`via another GGSN if needed`

From	HLRs
Message Type	MS_LOCATION
Action	`send to GGSN/SGSN that sent the WHERE_IS query`
Message Type	VERIFY_REPLY
Action	`send to GGSN/SGSN that sent the VERIFY query`

7.2.5 The Home Location Register

Functionalities

Maintain subscription information about the MS.
Keep track of MS roaming ie the Serving GSN, and accessibility.

Data Maintained at HLR

HLR maintains a table with the following fields:

MS_id
SGNS_addr
MNRF
QoS_subscribed

Events

1.QUERY RECEIVED. The HLR can receive queries directly only from GGSNs or SGSNs belonging to the local GPRS network. Other nodes will have to route queries through these.

From	`Local GGNS or SGNSs`
Message Type	`VERIFY(MS_id,QoS_requested)`

Action	`if MS_ is in HLR table {`
	`if (QoS_requested < QoS_subscribed){`
	`send mesg=VERIFY_REPLY(OK) } }`
	`else send mesg=VERIFY_REPLY(NOK)`

From	`Local GGNS or SGNSs`
Message Type	WHERE_IS(MS_)

Action	`if MS_ is in HLR table {`
	`send mesg=MS_LOCATION(SGSN_addr for MS_)`
	`else send mesg=MS_LOCATION( wrong_hlr)`

From	SGSNs
Message Type	SGSN_UPDATE(MS_,SGSN)
Action	`look up MS_ in HLR table` `change SGSN_addr to SGSN_`

7.3 Incorporating GPRS nodes into *ns*

Each GPRS node may be implemented as an *Agent* in *ns*. These are the end-points, where network level packets are created and received. A 'GPRS Agent' attached to an *ns-node* will make it function as a GPRS node. The GPRS nodes add a header corresponding to GTP, SNDCP or BSSGP whichever is relevant, on outgoing packets. Each GPRS *Agent*'s *recv()* module should be configured to read and decapsulate the appropriate GPRS headers. The headers added have the following basic structure:

- A Message_Type field
- Address of the sending node.
- Any other parameters required for specific message types that cannot be accessed from the default *ns packet header*.

The nodal agents would have to be implemented in C++.
At the OTcl level, the GPRS module would take care of the following:

- Initializing the data structures at each node.
- Creating default linkages from the GGSN to SGSNs in the GPRS subnetwork, and from each SGSN to the BS under it.
- Filling the HLR at set-up time.
- Enabling user to call GPRS_attach() and GPRS_detach().
- Enabling user to change MM_State.

This sums up our design for the GPRS Support Network. We conclude this thesis with a summary of our work in the next chapter.

Chapter 8

Concluding Remarks

In this project, we have designed and implemented a simulator for GPRS. Our focus was on the interaction between a Mobile Station and the Base Station and the multiplexing of radio resources in GPRS.

For this, the Link Layer, Radio Link Control[3]. and the Medium Access Control protocols for GPRS were implemented in *ns-Network Simulator*.

The LL and RLC handle fragmentation and reassembly of PDUs, along with retransmissions if used in acknowledged mode.

Our MAC models separate uplink and downlink frequencies. It supports a 8 slot TDMA frame on each frequency. We have modeled packet transmission and reception using timers. The appropriate messaging for call set up and handling is supported. Single slot operation is supported with symmetric uplink and downlink allocation. The first free slot is alloted to a requesting MS. A slot release mechanism is supported for GPRS MS. Possible collisions on the Random Access grant channel are appropriately taken care of. We have also incorporated an error mechanism for introducing slot level errors.

We performed experiments to study the average packet delay of the system; the interaction of the ARQ mechanisms at the LL and RLC with TCP's ARQ and the capacity analysis for GPRS vs GSM.

We found that that traffic data rates up to 12 kbps may be supported by the system while keeping the average packet delay within a reasonable bound. We also found that a GPRS system can support more mobiles than the number of TDMA slots reserved for it. The performance of various other features of the simulator was also validated.

This implementation of MAC, RLC and LL is being submitted to the *ns* distribution at ISI. This work may be further extended to incorporate other features of the GPRS network architecture as described in chapter seven.

[3]The code for the LL and RLC was contributed by Mr Sandeep Kumar and Mr Kopparapu Suman

Appendix 1: Users' Manual

1.1 Configuring the Nodes

Configuration is similar to any wired-cum-wireless scenario and hierarchical addressing. The following changes apply:

- Using a Non-Adhoc Agent [9].

  `set opt(adhoc) NOAH`

- Setting the MAC

  `set opt(mac) Mac/Gprs`

- Setting the RLC

  `set opt(rlc) RLC`

- While configuring the nodes, the following line has to be included in the normal list

  `$ns_ node-config -rlcType $opt(rlc)`

Note: The BS should be configured first, ie before the other wireless nodes so that it has a hierarchical address of the form 0.0.0 or 1.0.0 .

1.2 Parameters for the LL

The following parameters can be used to configure our LL

```
LL set acked_ 1;
LL set llfraged_ 1;
LL set llfragsz_ 500;
LL set llverbose_ 0;
```

`acked_` set to 1 implies LL is to be run in ACKed mode, 0 implies non-ACK.
`llfraged_` set to 1 indicates LL is to fragment packets before sending down to RLC; 0 implies no fragmentation.
`llfragsz_` indicates the size of the LL fragment.
`llverbose_` set to 1 implies that the LL will give a verbose output indicating each action performed.

1.3 Parameters for the RLC

The following parameters can be used to configure other RLC

```
RLC set acked_ 1;
RLC set rlcfraged_ 1;
RLC set rlcfragsz_ 50;
RLC set rlcverbose_ 0;
```

`acked_` set to 1 implies RLC is to be run in ACKed mode, 0 implies non-ACK.
`rlcfraged_` set to 1 indicates RLC is to fragment packets before sending down to the MAC; 0 implies no fragmentation.
`rlcfragsz_` indicates the size of the RLC fragment.
`rlcverbose_` set to 1 implies that the RLC will give a verbose output indicating each action performed.

1.4 Parameters for the MAC

The following parameters are used to configure the MAC for GPRS

```
Mac/Gprs set gprs_ 1
Mac/Gprs set max_num_ms_ 64
Mac/Gprs set max_num_freq_ 2
Mac/Gprs set slot_packet_len_ 50
Mac/Gprs set gprs_slots_per_frame_ 4
Mac/Gprs set rlc_error_ 1
Mac/Gprs set error_rate_ 1000
Mac/Gprs set verbose_ 0
```

`gprs_` set to 1 indicates that the node is a GPRS mobile, while 0 implies a GSM mobile.
`max_num_ms_` is the maximum number of MS the BS can handle.
`max_num_freq_` is the number of frequencies available in the cell.
`slot_packet_len_` is the maximum packet size accepted by the MAC.
`gprs_slots_per_frame_` is the number of slots available to GPRS mobiles. Note that slot 0 is always reserved for Broadcast/Random Access/ Signalling.
`rlc_error_` indicates whether we want to introduce errors in RLC PDUs
`error_rate_` sets the maximum value of the random variable used to generate the errors.
`verbose_` set to 1 will make the MAC give a verbose output, printing information about each action performed.

1.5 Setting up Traffic

Traffic for GPRS systems is usually bursty. In *ns* Exponential or Pareto distributions may be used as Applications over TCP/UDP Agents. This can be set up in the following way.

```
set tcp($j) [new Agent/TCP]
$ns_ attach-agent $node_($j) $tcp($j)
set sink($j) [new Agent/TCPSink]
$ns_ attach-agent $BS(0) $sink($j)
$ns_ connect $tcp($j) $sink($j)


set exp($j) [new Application/Traffic/Exponential]
$exp($j) set burst_time_ 500ms
$exp($j) set idle_time_ 100ms
$exp($j) set rate_ 10k


$exp($j) attach-agent $s($j)
$ns_ at $opt(start) "$exp($j) start"
```

1.6 A script for GPRS simulation

We give a sample script for using the GPRS simulations features we have provided.

```
# Define options

set opt(chan)     Channel/WirelessChannel     ;# channel type
set opt(prop)     Propagation/TwoRayGround    ;# radio-propagation model
set opt(netif)    Phy/WirelessPhy             ;# network interface type
set opt(mac)      Mac/Gprs;# MAC type
set opt(ifq)      Queue/DropTail/PriQueue     ;# interface queue type
set opt(ll)       LL                          ;# Link layer type
set opt(rlc)      RLC
set opt(ant)      Antenna/OmniAntenna         ;# antenna model
set opt(ifqlen)   5000                        ;# max packet in ifq
set opt(adhoc)    NOAH                        ;# routing protocol
set opt(x)        70                          ;# x coordinate of topology
set opt(y)        70                          ;# y coordinate of topology
set opt(seed)     0.0                         ;# seed for random num gen.
set opt(tr)       "/tmp/richa/sim1.tr"
set opt(start)    0.0
set opt(stop)     80                          ;# time to stop simulation
```

```
set num_bs_nodes 1
set opt(nn)      5                          ;# number of mobilenodes
set opt(rate)    15k

Mac/Gprs set gprs_slots_per_frame_  7
Mac/Gprs set slot_packet_len_      53
Mac/Gprs set max_num_ms_           15
Mac/Gprs set max_num_freq_          2
Mac/Gprs set gprs_                  1
Mac/Gprs set rlc_error_             0
Mac/Gprs set error_rate_         1000
Mac/Gprs set verbose_               0

LL set acked_        0
LL set llfraged_     1
LL set llfragsz_  1520
LL set llverbose_    0

RLC set acked_       0
RLC set rlcfraged_   1
RLC set rlcfragsz_  50
RLC set rlcverbose_  0

#remove unnecessary packet headers, else each pkt takes 2kb!
remove-packet-header LDP MPLS Snoop
remove-packet-header Ping TFRC TFRC_ACK
remove-packet-header Diffusion RAP IMEP
remove-packet-header AODV SR TORA IPinIP
remove-packet-header MIP HttpInval
remove-packet-header MFTP SRMEXT SRM aSRM
remove-packet-header mcastCtrl CtrMcast IVS
remove-packet-header Resv UMP Flags


# create simulator instance
set ns_   [new Simulator]

# set up for hierarchical routing
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 1           ;# number of domains
lappend cluster_num 1                ;# number of clusters in each domain
```

```
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 6               ;# number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel  ;# of each domain

set tracefd  [open $opt(tr) w]
$ns_ trace-all $tracefd

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)

# create God
create-god $opt(nn)

set chan1 [new $opt(chan)]
# configure for base-station node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -rlcType $opt(rlc) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif)\
                 -topoInstance $topo \
                 -wiredRouting ON \
                 -agentTrace ON \
                 -routerTrace OFF  \
                 -macTrace OFF  \
                 -movementTrace OFF \
                 -channel $chan1

#create base-station node
set temp {1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 }
# hier address to be used for wireless domain

set BS(0) [$ns_ node [lindex $temp 0]]
$BS(0) random-motion 0  ;# disable random motion
```

```
#provide some co-ord (fixed) to base station node
$BS(0) set X_ 1.0
$BS(0) set Y_ 2.0
$BS(0) set Z_ 0.0

# configure for mobilenodes
$ns_ node-config -wiredRouting OFF

# create mobilenodes in the same domain as BS(0)
for {set j 0} {$j < $opt(nn)} {incr j} {
    set node_($j) [ $ns_ node [lindex $temp [expr $j+1]] ]
    $node_($j) base-station [AddrParams addr2id [$BS(0) node-addr]]
}

for {set j 0} {$j < $opt(nn)} {incr j} {

set s($j) [new Agent/TCP]
$ns_ attach-agent $node_($j) $s($j)
$s($j) set packetSize_ 1500

set null($j) [new Agent/TCPSink]
$ns_ attach-agent $BS(0) $null($j)
$null($j) set packetSize_ 30
$ns_ connect $s($j) $null($j)

set exp($j) [new Application/Traffic/Exponential]
$exp($j) set burst_time_ 500ms
$exp($j) set idle_time_ 500ms
$exp($j) set rate_ $opt(rate)
$exp($j) attach-agent $s($j)

$ns_ at $opt(start) "$exp($j) start"
}

# Tell all nodes when the simulation ends
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0 "$BS(0) reset";
```

```
$ns_ at $opt(stop).0002 "puts \" \" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}

#puts "Starting Simulation..."
$ns_ run

--------------------
```

Appendix 2 : Psuedocode for Packet Processing

The psuedocode for packet processing at the MAC is described below.

2.1 Packet at MS MAC

Event: Packet received at an MS

Method: ms_recv()

Comment: It can be a packet from the IFQ to be sent to the BS
or a packet from the Air Interface to be sent up the stack.
The latter case occurs only at the start of a down-slot.

Action:

```
Check direction in packet header
If packet direction == UP
    Check mac_source and mac_destination
    If mac_source == BS and mac_destination == (self or broadcast)
        Stop the Release timer if it is On
        Restart Release Timer if the IFQ is empty.
        Store packet in pktRx_[downslot]. (see Note)
        Switch the radio On.
        Pass pktRx_[downslot] to rx_from_phy().
    Else ignore packet.

Else if packet direction == DOWN
    If packet is a broadcast
        Stamp it to be transmitted on upslot0, frequency0.
    Else
        check tx_chan[] for slot/frequency to transmit packet.
        If no slot-frequency found
            Store packet in p_temp
            Call send_resource_request().
```

Else

If Release Timer is ON, stop it.
Stamp frequency channel onto packet header
Store packet in pktTx[up_slot alloted]. (see Note)
Pass pktTx[up_slot alloted] to rx_from_ll()

Note : In the case where the MS receives two packets on concurrent slots, the packet received later will over-write the previous packet, at the instance at the end of the old slot and start of the new one. At this instance, the `pktRxTimer` is not yet done with the old packet and still requires the pointer. It triggers an error. To avoid this, we use an array (could also use a linked list), that stores pointers to packets received in each slot. The array entry is cleared by the `recvHandler` after the packet reception is over.

A similar situation holds for the `pktTx[]`. We use an array to avoid over-writing and thus loosing packets to be transmitted in adjacent slots.

Note that this structure would also suffice for multi-slot operation.

2.2 Packet received at an BS

Method: bs_recv()

Comment: It can be a packet from the IFQ to be sent to an MS
or a packet from an MS at the Air Interface.
The latter case occurs only at the start of an up-slot.

Action: Check direction in packet header
If packet direction == UP
Check mac_destination
If mac_destination == (self or Broadcast)
Extract the frequency channel the packet was sent on from its header
. Store packet in rxQ[freq][upslot]. (see Note)
Switch the radio On.
Pass rxQ_[freq][upslot] to rx_from_phy().
Else ignore packet.

Else if packet direction == DOWN
If packet is a broadcast
Stamp it to be transmitted in downslot0, frequency0.
Else
Look up slot reserved for this destination from the vlr_downtable[][]
If no slot reserved for destination
Mark packet as waiting.

Call slot_allot()
Call send_resource_reply()
Else
If *resource_reply* not yet received by MS
Mark packet as waiting.
Stamp frequency channel onto packet header
Store packet in txQ[freq][downslot]. (see Note)
Pass txQ[freq][downslot] to rx_from_ll()

Note:
The situation here is similar to that at the MS. There is also the added complexity that the BS may have to transmit/receive packets meant for/from different MS at a time; and that it can transmit/receive on many frequencies at a time. We sort and store all packets to be transmitted, and those received, in a two-dimensional array, in order to keep track of all the packets. Again, the entries are cleared after the packets have been fully processed.

2.3 Accepting a packet from the IFQ

Method: `rx_from_ll ()`

Comment: This is where the MAC header is added and its values set

Action: Add MAC header subtype
Configure the other MAC header fields
Set the wait flag to zero, unless the packet has already been marked waiting

2.4 End of a Down Slot

Method: downslotHandler(Event *e)

Comment: It is the BS' turn to transmit. See if it has any packet scheduled to be transmitted in this slot. All frequencies needed to be checked.

Action: Restart downslot timer to clock the next slot.
Start the Upslot Timer if this downslot 3 in the first TDMA frame.
Compare node_address with the BS_address
If node is a BS
Check `txQ[freq][downslot]` for all frequencies
If `txQ[freq][downslot]` !=NULL (see Note)
Check if the packet has been marked as waiting

If not, then pass packet to tx_onto_PHY()
Else do nothing
Increment downslot counter. Take care of wrap around.

Note: `txQ[freq_][downslot_]` stores a pointer to the packet to be transmitted by the BS in downslot in `downslot_` on frequency `freq_`.

2.5 End of a Up Slot

Method: upslotHandler(Event *e)

Comment: An MS can transmit a packet if it has one scheduled for transmission.

Action: Restart upslot timer to clock the next slot
Compare node_address with the BS_address
If node is an MS
  Check if it has a packet to transmit in `pktTx[up_slot_]`
  if `pktTx[up_slot_]` is not NULL
    Start slot release timer, if IFQ is empty
    Switch the radio ON
    Pass the packet in `pktTx[up_slot_]` to `tx_onto_PHY ()`
    Free `pktTx[up_slot_]`
Else do nothing
Increment upslot counter. Take care of wrap around.

2.6 Packet to be transmitted onto the Air-Interface

Method: `tx_onto_PHY ()`

Comment: This is where the packet is actually passed down to the physical layer.

Action: Find packet transmission time
Assert that it is non-negative but less than `slot_time_`
Check if this is the PRACH and this is an MS
If yes, check for collision on `chan0_0`
If collision
  Change tt chan0_0 to COLL
  Increment `coll_count`
  Mark error in packet header
Check the MAC header subtype

If it is a *resource_request*

Schedule the `Wait timer` for 1 TDMA frame

Start the `pktTxTimer`

Pass the packet to `downtarget_`

2.7 Packet to be received from the Air-Interface

Method: `rx_from_PHY ()`

Comment: This is where a packet is received from the Air-Interface by the receiver.

Action: Check if this is the PRACH and this is the BS

If yes, check for collision on `chan0_0`

If collision

Drop the colliding packets

Decrement the `coll_count`

Change the tt chan0_0 to IDLE when all colliding packets have been dropped

Else

Drop packet header if marked in error

Find packet reception interval

Assert that it is non-negative but less than `slot_time_`

Start the `pktRxTimer`

2.8 Packet send completed

Method: `sendHandler ()`

Comment: Simply winds up the transmission

Action: Free the packet

Switch the radio OFF

Unlock the IFQ if the `wait timer` is not busy

2.9 Packet receive completed

Method: `sendHandler ()`

Comment: Wind up reception and pass the packet onto higher layers

Action: Switch the radio OFF
Double check whether I should receive the packet
If not, drop the packet
Pass the packet to `fwd_DATA_to_LL()`

2.10 A packet has to be passed to the RLC

Method: `fwd_DATA_to_LL()`

Comment: The packet can be either a MAC control packet or data packet
It has to be handled accordingly.

Action: Check MAC type
If MAC type == DATA
    Remove MAC header
    Increment hop count in the packet's common header
    Pass it on to the `uptarget`, the RLC in this case
If MAC type == Control, check for subtype
If MAC sub type == *res_request*
    Allot a slot to the source MS, through `slot_allot()`
    Schedule a *res_reply* to be sent
If MAC sub type == *res_reply*
    Stop the `wait timer`
    Obtain the slot and frequency reserved for me from the *res_reply* header
    Set `tx_chan[slot]=rx_chan[slot] = frequency`
    Unmark the waiting packet at the BS
    Move the packet buffered at the MS to `pktTx[slot]`
    Pass the `pktTx[slot]` to `rx_from_ll()`
    Unlock the IFQ
If MAC sub type == *tx_end*
    Clear the corresponding MS' entry from the `vlr_.down_table` and `vlr_.up_table`

2.10 Resources have to be alloted to an MS

Method: `slot_allot (int ms_, int &freq, int &slot)`

Comment: A slot has to be alloted to the mobile from the pool of GPRS
and GSM slots depending on which type the mobile is.

Action: Check if this is a GPRS mobile
If it is a GPRS MS

Find the first available GPRS slot in the `vlr_.up_table`
Refuse connection if no GPRS slot- frequency available
Else if it is a GSM MS
Find the first available GSM slot in the `vlr_.up_table`
Refuse connection if no GSM slot- frequency available
Enter the MS' MAC `index_` into the `vlr_.up_table` and `vlr_.down_table`
Return the value of the frequency and slot to the caller

This completes the description of the psuedocode for packet processing at the MAC.

References

[1] C Bettstetter, H J Vogel and Jorg Eberspacher "GSM Phase 2+ General Pakcet Radio Service GPRS: Architecture,Protocols, and Air Interface",*IEEE Communications Surveys*, 1999.

[2] J Cai, D J Goodman, "General Packet Radio Service in GSM", *IEEE Communications Magazine*, Oct. 1997, pp. 122-31.

[3] G Brasche, B Walke, "Concepts, Services and Protocols for the New GSM Phase 2+ General Packet Radio Service", *IEEE Communications Magazine*, Nov. 1997, pp. 92-104

[4] M Meyer, "TCP Performance over GPRS", *IEEE Wireless Comm. and Networking Conference"*,New Orleans, LA, Sept 1999.

[5] GSM 03.60: "Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Service Description; Stage 2".

[6] GSM 04.64: "Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Logical Link Control (LLC)".

[7] GSM 04.60: "Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Mobile Station (MS) - Base Station System (BSS) interface; Radio Link Control/Medium Access Control (RLC/MAC) protocol".

[8] *ns* Manual,
http://www.isi.edu/nsnam/ns/doc/index.html

[9] Joerg Widmer,
http://www.icsi.berkeley.edu/ widmer/mnav/ns-extension/