SMSList

Sandeep Gupta
M.Tech-WCC, 2nd sem
Indian Institute of Information Technology, Allahabad

sandeepgupta@iiita.ac.in

May, 2003

Abstract

The following report is based on project work done as a part of Mobile Intelligent Network course. Here I introduce the notion of SMS Lists, a concept similar to e-mail lists. I have successfully implemented the same in C using MySQL database and an open source WAP and SMS gateway, maintained by the Kannel Group.

1 Introduction

The concept of SMSList is similar to e-mail list. Here, a mobile user can create an SMSList and invite others to be a part of it. The invited parties can either accept or reject the invitation. Once, a list is created and others have joined the list, one can start sending SMS on the list. The SMS sent to the list would be delivered to all the members of the list (with default list and individual settings).

2 Kannel and SMS

SMS, short messaging service, is a way to send short messages, logos, ring tones from one GSM phone to another. SMS services are content services initiated by SMS message to certain phone number, which then answers with requested content. eg. 8888 service provided by Indiatimes.

Whenever such SMS is received by the SMSC, it forwards it to the content provider. This is done through various protocols like SMPP, CIMD etc. This is where Kannel gateway sits and provides an abstraction to the actual application. The content provider can write applications without worrying about different protocols being used by different SMSCs.

Kannel gateway invokes different services according to the keyword in an SMS. The keyword is the first word in the SMS. It can also send and receive messages via HTTP.

Figure 1[1] shows the logical position of SMS gateway between a phone and a content server.

More details about Kannel Gateway can be found out at www.kannel.org

[1]Source: Kannel Documentation

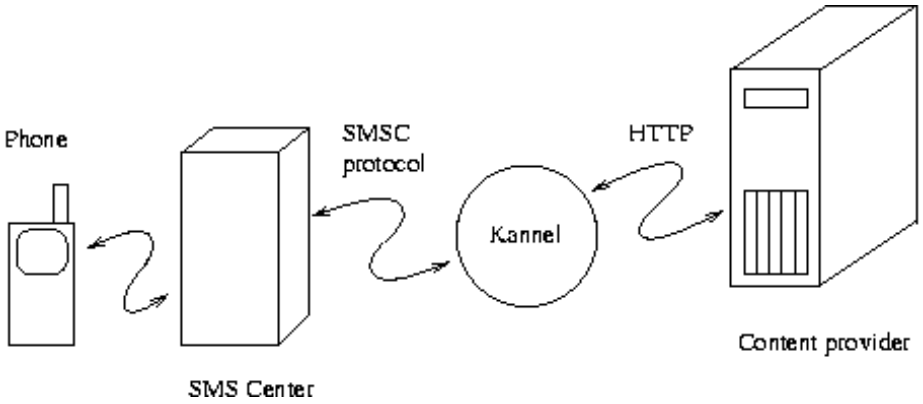

Figure 1: Logical position of SMS gateway

3 SMSList Reference Model

The following figure shows the reference model of SMSList.

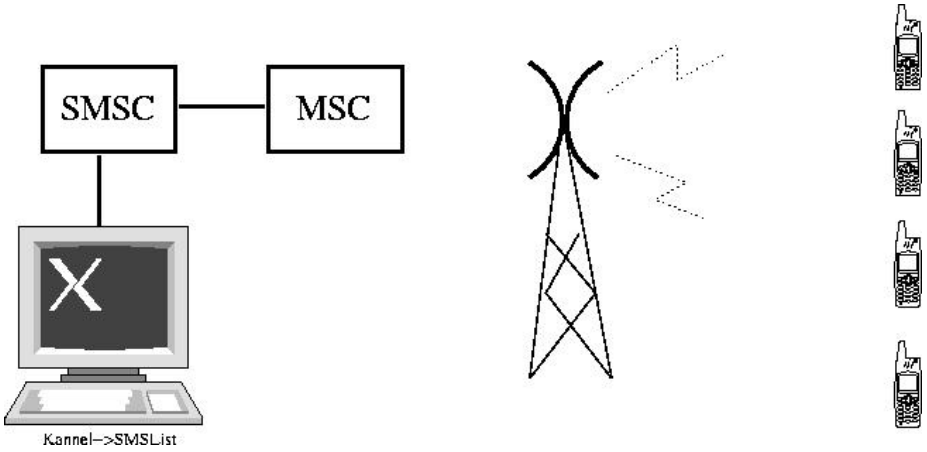

Figure 2: Reference model of SMSList

Mobile phones send messages to BS. BS passes these SMS to MSC which further sends it to SMS Center. SMSC is a store and forward platform. It checks the intended recepient of the list. If the recepient is available, it forwards the SMS, otherwise it stores it and tries to forward it later.

Now when the SMS gets delivered to kannel gateway (through HTTP), it checks for the keyword 'List'. If the keyword is present in the SMS, the message is passed to the SMSList application. The application processes it and sends back appropriate reply/replies.

4 Database

The MySQL database consists of the following tables:

```
mysql> show tables;
+-------------------+
| Tables_in_SMSLIST |
+-------------------+
| ack_pending       |
| list              |
| member            |
| sms_sent          |
| user              |
+-------------------+
5 rows in set (0.00 sec)
```

4.1 Table *ack_pending*

This table contains records of those phone nos. to which invitation has been sent to join a list and an 'accept' has not been received. Enteries in this table are flushed out after 24 hours.

```
mysql> desc ack_pending;
+------------+----------+------+-----+---------------------+-------+
| Field      | Type     | Null | Key | Default             | Extra |
+------------+----------+------+-----+---------------------+-------+
| ph_no      | char(15) |      | PRI |                     |       |
| for_list   | char(32) |      | PRI |                     |       |
| time_stamp | datetime |      | PRI | 0000-00-00 00:00:00 |       |
+------------+----------+------+-----+---------------------+-------+
3 rows in set (0.00 sec)
```

4.2 Table *list*

Table *list* consists the settings pertaining to a list like owner, whether the list is private or public and if the list is a broadcast-only medium.

```
mysql> desc list;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| name     | char(32)      |      | PRI |         |       |
| owner_no | char(15)      |      | PRI |         |       |
| is_pvt   | enum('Y','N') | YES  |     | Y       |       |
| is_bcast | enum('Y','N') | YES  |     | N       |       |
+----------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

4.3 Table *member*

This contains settings of members of a list. Owner of the list can decide whether a member can send messages on the list. Broadcast-only lists do not allow members except the owner to send messages on the list. Similarly, a member can decide whether to receive messages from a particular list.

```
mysql> desc member;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| ph_no    | char(15)      |      | PRI |         |       |
| list     | char(32)      |      | PRI |         |       |
| can_send | enum('Y','N') | YES  |     | Y       |       |
| recv     | enum('Y','N') | YES  |     | Y       |       |
+----------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

4.4 Table *sms_sent*

All the SMS that are sent through SMSList get stored here.

```
mysql> desc sms_sent;
+-----------+-----------+------+-----+---------+----------------+
| Field     | Type      | Null | Key | Default | Extra          |
+-----------+-----------+------+-----+---------+----------------+
| id        | int(11)   |      | PRI | NULL    | auto_increment |
| from_no   | char(15)  |      |     |         |                |
| for_list  | char(32)  | YES  |     | NULL    |                |
| time_sent | datetime  | YES  |     | NULL    |                |
| sms       | char(255) | YES  |     | NULL    |                |
+-----------+-----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

4.5 Table *user*

Table *user* contains records of all the users who are authorized to use SMSList service. This table is assumed to be populated before users can actually access this service.

```
mysql> desc user;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| ph_no | char(15) |      | PRI |         |       |
| name  | char(32) | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

5 Message Flows

5.1 Creating an 'SMSList'

Figure 3 shows the message flow of SMS when a user wishes to create an SMSList.

1. User creates an SMS of the format 'List *listname* create' where *listname* is the name of the list and sends it to BS.

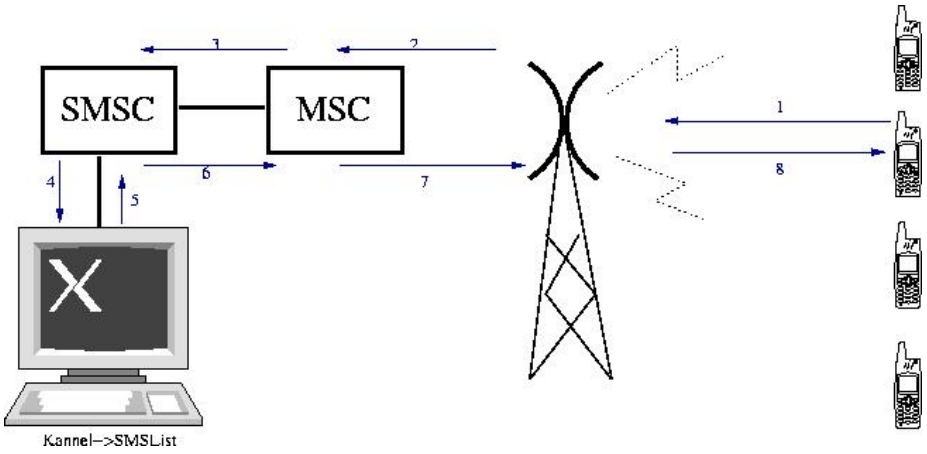

Figure 3: Message Flow : Creating an 'SMSList'

2. BS forwards the SMS to MSC.

3. MSC notices the SMS and sends it to SMS Center.

4. SMSC sees the number which the SMS is addressed to and decides to forward the data to the content provider.

5. Kannel receives the SMS, checks the keyword. It finds the keyword 'List', so it invokes the SMSList application. SMSList parses the SMS. The request is for creating a new 'SMS List'. It checks its database to see if the user is a registered member of the service and whether there is any list created with same name. If the conditions are satisfied, a list is created with *listname* with owner phone no. as the one from which the SMS was received. An SMS stating whether list creation was successful or not is created and passed to SMSC.

6. SMSC forwards message to MSC.

7. MSC sends message to BS.

8. BS finally transmits the SMS to the initiator of the request.

5.2 Inviting others to join the list

Once the list is created, next logical step is to invite other mobile phone users to join the list. The owner of the list can invite members. The SMS should be of this format: 'List *listname* invite *phno1 phno2...*'

1. The SMS follows the usual BS → MSC → SMSC → SMSList path.

2. SMSList parses the phone nos. from the SMS. After performing various checks, invitations are sent to other phone nos.

3. - 5. Others receive an invitation to join the list.

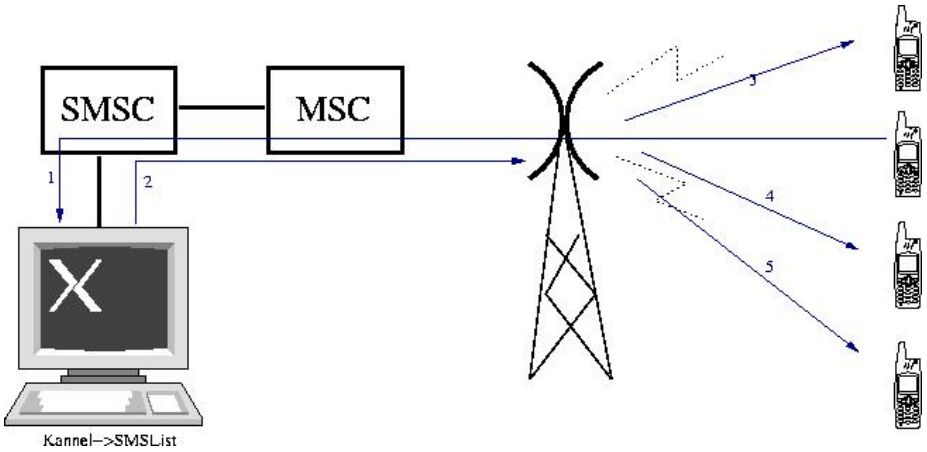

Figure 4: Message Flow : Inviting others to join the list

5.3 Accepting an invitation

The invited phone user can accept the invitation by sending an SMS of the format: 'List *listname* accept'. To reject the invitation, the phone user can simply ignore the message. Figure 5 shows the message flow

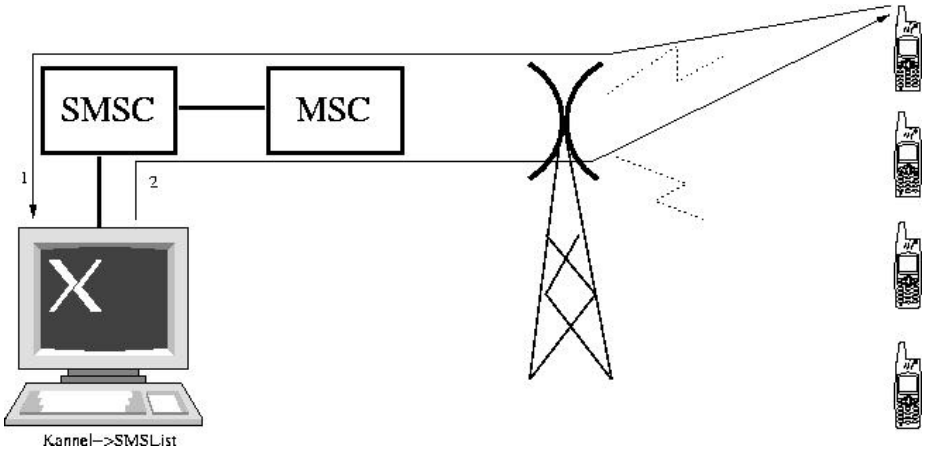

Figure 5: Message Flow : Accepting an invitation

1. Invitee sends an accept message.
2. After performing necessary checks, SMSList adds the phone no. as a member of the list. Newly inducted member can now send messages on the list.

5.4 Sending SMS on the list

Once a phone no. is a member of a list, it can send and receive messages on the list (provided it has the necessary permissions). Format for this SMS is : 'List

listname msg *message*'.

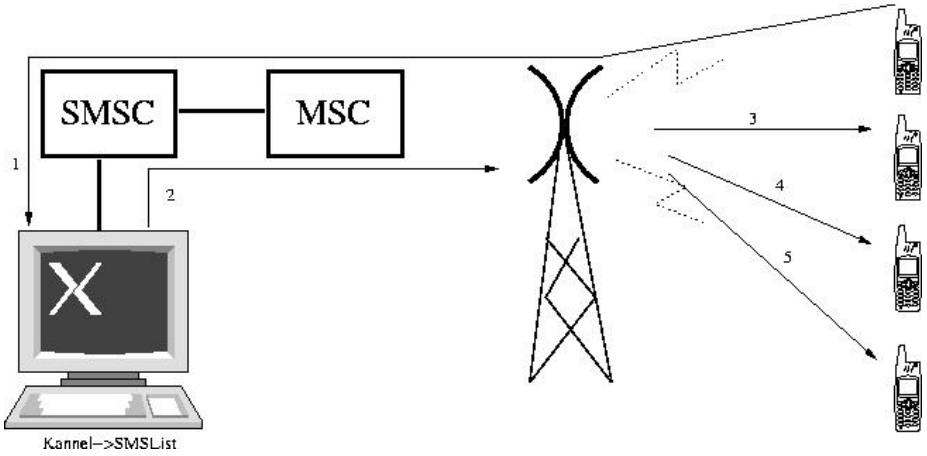

Figure 6: Message Flow : Sending SMS on the list

1. List member sends an SMS addressed to the list.
2. SMSList extracts the member phone nos. of that list and creates Group SMS to be sent to the members.
3. - 5. The members receive the SMS.

6 Simulation Setup

For simulations, a phone simulator program was written that could receive and send simple text SMS. Also, a 'fakesmsc' program, that comes with Kannel gateway distribution, was modified so that it could work with phone simulator program. The architecture that simulations follow is shown in figure 7

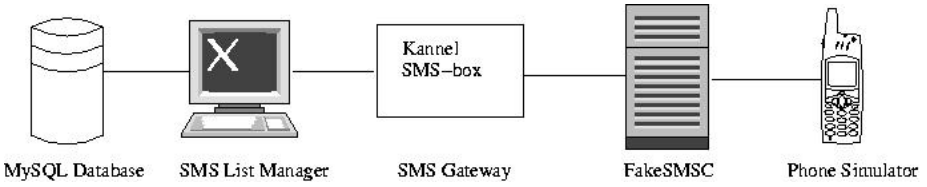

Figure 7: Simulation Architecture

The phone simulator can send SMS directly to Fake SMSC. The fakesmsc program sends this SMS to Kannel SMS-box which invokes SMS List Manger program. This list manger is responsible for processing these SMS using MySQL database.

Reverse procedure is followed for sending SMS from List Manager to phone simulator.

Since, Fake SMSC does not implement storing feature of SMSC, all the phone simulators should be connected to Fake SMSC to receive SMS.

7 Future Work and Conclusion

SMS Lists creates a new paradigm of services that can be offered through SMS. A plethora of services can be built over this concept. Current implementation only demonstrates simple list creation and sending messages on list. Other services can be polling options, group event reminders, location based alerts for groupmembers, etc., to name a few.

As far as business model is concerned, the GSM company providing this service can offer discounts for using SMS Lists, like paying for only 4 SMS if a message was sent to a list having 10 members.

SMSList can also find a practical use in large corporate offices and educational institutes. Event/lecture reminders would be the main use of SMSList in this scenario.

8 Acknowledgment

Thanks to Prof. Manish Singh, under whose guidance this project was successfully carried out.

References

1. Wireless Intelligent Networking by Christensen, Florack & Duncan
2. The Kannel Documentation (available at www.kannel.org)
3. MySQL & mSQL by Randy Jay Yarger, George Reese & Tim King

A Acronyms

- BS: Base Station
- GSM: Global System for Mobile communication
- HTTP: Hyper Text Transfer Protocol
- MSC: Mobile Switching Center
- SMS: Short Messaging Service (Also, used for referring the short message)
- SMSC: SMS Center

B Source Code

B.1 smslistmgr.c

```
#include <stdio.h>
#include <sys/time.h>
#include <mysql.h>

int opr (const char *, const char *, const char *, char *);
int sendsms (const char *, const char *);
void hex_conv (char *);
main (int argc, char *argv[])
{
  char lname[32], ph_no[15], *op, *sms, *errmsg;
  int temp;
  op = (char *) malloc (sizeof (char) * 15);
  sms = (char *) malloc (sizeof (char) * 255);


  if (argc < 4)
    {
      printf
("ERROR. send \"List <listname> <op>\". op can be msg, create, invite, accept.\n");
      return 1;
    }
  //for(temp=1;temp<argc;temp++)
  //      hex_conv(argv[temp]);
  strcpy (ph_no, argv[1]);
  strcpy (lname, argv[2]);
  strcpy (sms, argv[3]);
  temp = strcspn (sms, "+");
  strncpy (op, sms, temp + 1);
  strcpy (sms, sms + temp + 1);
  *(op + temp) = '\0';
  hex_conv (ph_no);
  opr (op, lname, ph_no, sms);
}


int
opr (const char *op, const char *lname, const char *ph_no, char *sms)
{
  MYSQL *connection, mysql;
  MYSQL_RES *result;
  MYSQL_ROW row;
  int state, i, j, k;
  char query[255], uname[32], phnos[100][20], nos[100][20], *tim_sent,
    message[1000], tempphno[20];

  mysql_init (&mysql);
  connection =
    mysql_real_connect (&mysql, "localhost", "root", "root123", "SMSLIST", 0,
0, 0);
  if (connection == NULL)
```

```
    {
      perror (mysql_error (&mysql));
      return 1;
    }
  //aging out ack
  sprintf (query,
   "delete from ack_pending where time_stamp < date_sub(now(), interval 1 day)");
  state = mysql_query (connection, query);
  if (state != 0)
    {
      perror (mysql_error (connection));
      return 1;
    }
  if (!strcasecmp (op, "msg")) //broadcast sms
    {
      //getting user information
      sprintf (query, "select ph_no, name from user where ph_no = \"%s\"",
       ph_no);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("You are not registered member of this service.\n");
  return 1;
}
      while ((row = mysql_fetch_row (result)) != NULL)
{
  strcpy (uname, row[1]);
}
      mysql_free_result (result);

      //getting list information
      sprintf (query,
       "select name, owner_no, is_bcast from list where name = \"%s\"",
       lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("List \'%s\' doesnot exist.\n", lname);
  return 1;
}
```

```
      row = mysql_fetch_row (result);
      if (!strcasecmp (row[2], "Y") && strcasecmp (row[1], ph_no))
{
  printf ("You are not authorised to send sms on \'%s\' list.\n",
  lname);
}
      mysql_free_result (result);

      //getting membership information
      sprintf (query,
       "select list, can_send from member where ph_no = \"%s\" and list = \"%s\"",
       ph_no, lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("You are not a part of \'%s\' smslist.\n", lname);
  return 1;
}
      row = mysql_fetch_row (result);
      if (!strcasecmp (row[1], "N"))
{
  printf
    ("You don\'t have permissions to send sms on \'%s\' smslist.\n",
     lname);
  return 1;
}
      mysql_free_result (result);

      //all's well send sms to the list
      sprintf (query,
       "select ph_no from member where list = \"%s\" and recv=\'Y\' and ph_no != \"%s\"",
       lname, ph_no);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      i = 0;
      while ((row = mysql_fetch_row (result)) != NULL)
{
  strcpy (phnos[i], row[0]);
  if (phnos[i][0] == '+')
    {
      strcpy (tempphno, "%2B");
```

```
      strcat (tempphno, phnos[i] + 1);
      strcpy (phnos[i], tempphno);
    }
  i++;
}
      strcpy (tempphno, ph_no);
      if (ph_no[0] == '+')
{
  strcpy (tempphno, "%2B");
  strcat (tempphno, ph_no + 1);
}
      sprintf (message, "Group=%s%%0ASender=%s%%0A%%0A%s", lname, tempphno,
       sms);
      sendgrpsms (phnos, i, message);
      mysql_free_result (result);
      //hex_conv (sms);
      sprintf (query,
       "insert into sms_sent(from_no,for_list,time_sent,sms) values (\"%s\",\"%s\",now(),\"%s\")",
       ph_no, lname, sms);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}
      printf ("Your message has been sent to members of \'%s\' list.\n",
      lname);
    }
  else if (!strcasecmp (op, "create")) //creating new list
    {
      //getting user information
      sprintf (query, "select ph_no, name from user where ph_no = \"%s\"",
       ph_no);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("You are not registered member of this service.\n");
  return 1;
}
      //checking if list already exists
      sprintf (query, "select * from list where name = \"%s\"", lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}
```

```
      result = mysql_store_result (connection);
      if (mysql_num_rows (result))
{
  printf
    ("List with name \'%s\' already exists. Please choose another name.",
     lname);
  return 1;
}

      //creating list
      sprintf (query,
       "insert into list (name, owner_no) values (\"%s\", \"%s\")",
       lname, ph_no);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}
      sprintf (query,
       "insert into member values (\"%s\", \"%s\",\'Y\',\'Y\')",
       ph_no, lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}
      printf
("List \'%s\' successfully created. Invite members by sending
      sms of following format: List %s invite <ph_no1> <ph_no2>
      ... ph_no should be in international format.eg. +919415262000\n",
 lname, lname);
    }
  else if (!strcasecmp (op, "invite")) //inviting members
    {
      //getting user information
      sprintf (query,
       "select * from list where owner_no = \"%s\" and name = \"%s\"",
       ph_no, lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("Error processing invite request. Invite not send.\n");
  return 1;
}
      //sending invites
```

```
      j = parse_phno (phnos, sms);
      for (i = 0, k = 0; i < j; i++)
{
  strcpy (tempphno, phnos[i]);
  hex_conv (tempphno);
  sprintf (query, "select * from user where ph_no = \"%s\"",
   tempphno);
  state = mysql_query (connection, query);
  if (state != 0)
    {
      perror (mysql_error (connection));
      return 1;
    }
  result = mysql_store_result (connection);
  if (!mysql_num_rows (result)) //invitee not a registerd user
    continue;
  sprintf (query,
   "select * from member where ph_no=\"%s\" and list=\"%s\"",
   tempphno, lname);
  state = mysql_query (connection, query);
  if (state != 0)
    {
      perror (mysql_error (connection));
      return 1;
    }
  result = mysql_store_result (connection);
  if (mysql_num_rows (result)) //already a member
    continue;
  strcpy (nos[k++], phnos[i]);
  sprintf (query,
   "insert into ack_pending(ph_no,for_list,time_stamp) values (\"%s\",\"%s\",now())",
   tempphno, lname);
  state = mysql_query (connection, query);
  if (state != 0)
    {
      perror (mysql_error (connection));
      return 1;
    }
}
      if (ph_no[0] == '+')
sprintf (message,
 "You+have+been+invited+by+%%22%2B%s%%22+to+join+%%22%s%%22+smslist.
+Reply+%%22List+%s+accept%%22+to+join+the+list.+Ignore+this+message+if+u+dont+want+to+join.",
 ph_no + 1, lname, lname);
      else
sprintf (message,
 "You+have+been+invited+by+%%22%s%%22+to+join+%%22%s%%22+smslist.
+Reply+%%22List+%s+accept%%22+to+join+the+list.+Ignore+this+message+if+u+dont+want+to+join.",
 ph_no, lname, lname);
      sendgrpsms (nos, k, message);
      printf ("Invite send.\n");
    }
  else if (!strcasecmp (op, "accept")) //accepting invitation
    {
```

```
      //getting user information
      char can_send = 'Y';
      sprintf (query,
       "select * from ack_pending where ph_no = \"%s\" and for_list = \"%s\"",
       ph_no, lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("ERROR. No acknowledgements pending.\n");
  return 1;
}
      sprintf (query, "select is_bcast from list where name = \"%s\"", lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}

      result = mysql_store_result (connection);
      if (!mysql_num_rows (result))
{
  printf ("ERROR. No such list.\n");
  return 1;
}
      row = mysql_fetch_row (result);
      can_send = strcasecmp (row[0], "Y") ? 'Y' : 'N';
      sprintf (query,
       "select * from member where ph_no = \"%s\" and list = \"%s\"",
       ph_no, lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}
      result = mysql_store_result (connection);
      if (mysql_num_rows (result))
return 1; //already a member
      sprintf (query,
       "insert into member(ph_no,list,can_send) values (\"%s\", \"%s\", '%c')",
       ph_no, lname, can_send);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
```

```
}

      sprintf (query,
       "delete from ack_pending where ph_no = \"%s\" and for_list = \"%s\"",
       ph_no, lname);
      state = mysql_query (connection, query);
      if (state != 0)
{
  perror (mysql_error (connection));
  return 1;
}
      printf ("You are now a member of \'%s\' smslist.\n", lname);

    }
  else //default
    {
      printf
("ERROR. send \"List <listname> <op>\". op can be msg, create, invite, accept.\n");
    }
  mysql_close (connection);
}

sendgrpsms (const char phnos[][20], int i, const char *message)
{
  if (i <= 0)
    return 0;
  char url[3000];

  sprintf (url,
   "lynx -dump \"http://localhost:13013/cgi-bin/sendsms\?username=a&password=b&to=");
  while (i > 0)
    {
      strcat (url, phnos[--i]);
      strcat (url, "%20");
    }
  strcat (url, "&text=\"");
  strcat (url, message);
  strcat (url, "\" \">/dev/null");
  return system (url);

}

void
hex_conv (char *arg)
{
  int i = 0, j = 0, num;
  char newarg[1000], hex[3];
  hex[2] = '\0';
  while (arg[i] != '\0')
    {
      if (arg[i] == '%')
{
  hex[0] = arg[++i];
  hex[1] = arg[++i];
```

```
  sscanf (hex, "%x", &num);
  newarg[j++] = num;
}
      else
newarg[j++] = arg[i];
      i++;
    }
  newarg[j] = '\0';
  strcpy (arg, newarg);
}

int
parse_phno (char phno[][20], char *sms)
{
  int i, j, k = 0;

  //hex_conv(sms);
  for (i = 0; sms[k] != '\0'; i++)
    for (j = 0;; j++)
      {
if (sms[k] != '+' && sms[k] != '\0')
  phno[i][j] = sms[k++];
else
  {
    phno[i][j] = '\0';
    if (sms[k] != '\0')
      k++;
    break;
  }
      }
  return i;
}
```

B.2 myfakesmsc.c - modified from fakesmsc.c

```
//myfakesmsc.c - modified by Sandeep Gupta

/*
 * fakesmsc.c - simulate an SMS center, using a trivial protocol
 *
 * The protocol:
 *
 * Client sends each message on its own line (terminated with \r\n or \n).
 * The line begins with 3 space-separated fields:
 * sender's phone number, receiver's phone number,
 * type of message. Type of message can be one of "text", "data", or
 * "udh". If type == "text", the rest of the line is taken as the message.
 * If type == "data", the next field is taken to be the text of the
 * message in urlcoded form. Space is coded as '+'. If type == "udh",
 * the following 2 fields are taken to be the UDH and normal portions
 * in urlcoded form. Space is again coded as '+'.
 * The server sends replies back in the same format.
 *
 * Lars Wirzenius, later edition by Kalle Marjola
```

```
 * Largely rewritten by Uoti Urpala
 */

static char usage[] = "\n\
Usage: fakesmsc [-H host] [-p port] [-i interval] [-m max] <msg> ... \n\
\n\
* 'host' and 'port' define bearerbox connection (default localhost:10000),\n\
* 'interval' is time in seconds (floats allowed) between generated messages,\n\
* 'max' is the total number sent (-1, default, means unlimited),\n\
* <msg> is message to send, if several are given, they are sent randomly.\n\
\n\
msg format: \"sender receiver type(text/data/udh/route) [udhdata|route] msgdata\"\n\
\n\
Type \"text\" means plaintext msgdata, \"data\" urlcoded, \"udh\" urlcoded udh+msg\n\
and \"route\" means smsbox-id routed plaintext msgdata\n\
Examples: \n\
\n\
fakesmsc -m 1 \"123 345 udh %04udh%3f message+data+here\"\n\
fakesmsc -m 1 \"123 345 route smsbox1 message+data+here\"\n\
fakesmsc -i 0.01 -m 1000 \"123 345 text nop\" \"1 2 text another message here\"\n\
\n\
Server replies are shown in the same message format.\n";

#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include <limits.h>
#include <signal.h>
#include <fcntl.h>//sg

#include <sys/param.h>

#include "gwlib/gwlib.h"

static int port = 10000;
static int mport = 10001; //sg
static Octstr *host;
static long max_send = LONG_MAX;
static double interval = 1.0;
static int sigint_received;

static struct cache
{ //sg
  char phno[20];
  int sockfd;
  int valid;
}
cachetable[10];
```

```
void getphno (char *, char *); //sg

static void
signal_handler (int signum)
{
  if (signum == SIGINT)
    sigint_received = 1;
  else
    panic (0, "Caught signal with no handler?!");
}


static void
setup_signal_handlers (void)
{
  struct sigaction act;

  act.sa_handler = signal_handler;
  sigemptyset (&act.sa_mask);
  act.sa_flags = 0;
  sigaction (SIGINT, &act, NULL);
}


/* Choose a random message from a table of messages. */
static Octstr *
choose_message (Octstr ** msgs, int num_msgs)
{
  /* the following doesn't give an even distribution, but who cares */
  return msgs[gw_rand () % num_msgs];
}


/* Get current time, as double. */
static double
get_current_time (void)
{
  struct timezone tz;
  struct timeval now;

  gettimeofday (&now, &tz);
  return (double) now.tv_sec + now.tv_usec / 1e6;
}

/* our arguments */
static int
check_args (int i, int argc, char **argv)
{
  if (strcmp (argv[i], "-p") == 0 || strcmp (argv[i], "--port") == 0)
    port = atoi (argv[i + 1]);
  else if (!strcmp (argv[i], "-H") || !strcmp (argv[i], "--host"))
    host = octstr_create (argv[i + 1]);
  else if (strcmp (argv[i], "-m") == 0 || strcmp (argv[i], "--messages") == 0)
    {
```

```
      max_send = atoi (argv[i + 1]);
      if (max_send < 0)
max_send = LONG_MAX;
    }
  else if (strcmp (argv[i], "-i") == 0 || strcmp (argv[i], "--interval") == 0)
    interval = atof (argv[i + 1]);
  else
    {
      panic (0, "%s", usage);
      return 0;
    }

  return 1;
}


/* The main program. */
int
main (int argc, char **argv)
{
  Connection *server;
  Octstr *line;
  Octstr **msgs;
  int i;
  int mptr, num_msgs;
  long num_received, num_sent;
  double first_received_at, last_received_at;
  double first_sent_at, last_sent_at;
  double start_time, end_time;
  double delta;

  int ssockfd, connfd, val, j, clilen; //sg
  struct sockaddr_in cliaddr; //sg
  char msg[255], phno[20]; //sg
  Octstr *msgoctstr; //sg


  gwlib_init ();
  setup_signal_handlers ();
  host = octstr_create ("localhost");
  start_time = get_current_time ();

  mptr = get_and_set_debugs (argc, argv, check_args);
  num_msgs = argc - mptr;
  if (num_msgs <= 0)
    panic (0, "%s", usage);
  msgs = gw_malloc (sizeof (Octstr *) * num_msgs);
  for (i = 0; i < num_msgs; i++)
    {
      msgs[i] = octstr_create (argv[mptr + i]);
      octstr_append_char (msgs[i], 10); /* End of line */
    }
  info (0, "Host %s Port %d interval %.3f max-messages %ld",
octstr_get_cstr (host), port, interval, max_send);
```

```
  srand ((unsigned int) time (NULL));

  info (0, "fakesmsc starting");
  server = conn_open_tcp (host, port, NULL);
  if (server == NULL)
    panic (0, "Failed to open connection");

  //sg
  if ((ssockfd = make_server_socket (mport, "*")) == -1)
    panic (0, "Failed to make server for phones");
  val = fcntl (ssockfd, F_GETFL, 0);
  fcntl (ssockfd, F_SETFL, val | O_NONBLOCK);
  for (j = 0; j < 10; j++)
    cachetable[j].valid = 0;

  num_sent = 0;
  num_received = 0;

  first_received_at = 0;
  first_sent_at = 0;
  last_received_at = 0;
  last_sent_at = 0;

  num_sent = 0;
  while (1)
    {
      if (num_sent < max_send)
{
  if (conn_write (server, choose_message (msgs, num_msgs)) == -1)
    panic (0, "write failed");
  ++num_sent;
  if (num_sent == max_send)
    info (0, "fakesmsc: sent message %ld", num_sent);
  else
    debug ("send", 0, "fakesmsc: sent message %ld", num_sent);
  last_sent_at = get_current_time ();
  if (first_sent_at == 0)
    first_sent_at = last_sent_at;
}

      //sg
      clilen = sizeof (cliaddr);
      if ((connfd =
   accept (ssockfd, (struct sockaddr *) &cliaddr, &clilen)) < 0)
{
  if (errno != EWOULDBLOCK)
    printf ("accept error");
}
      else
{
  for (j = 0; j < 10; j++)
    if (cachetable[j].valid == 0)
      break;
```

```
  if (j == 10)
    {
      write (connfd, "server busy\0", 12);
      close (connfd);
    }
  else
    {
      read (connfd, cachetable[j].phno, 20);
      cachetable[j].sockfd = connfd;
      cachetable[j].valid = 1;
      val = fcntl (connfd, F_GETFL, 0);
      fcntl (connfd, F_SETFL, val | O_NONBLOCK);
    }
}
      for (j = 0; j < 10; j++)
{
  if (cachetable[j].valid == 1)
    if ((val = read (cachetable[j].sockfd, msg, 255)) < 0)
      {
if (errno != EWOULDBLOCK)
  printf ("read error");
      }
    else
      {
if (val == 0) //connection closed
  cachetable[j].valid = 0;
else
  {
    msgoctstr = octstr_create (msg);
    octstr_append_char (msgoctstr, 10);
    if (conn_write (server, msgoctstr) == -1)
      panic (0, "write failed");
    debug ("send", 0,
   "fakesmsc phonesim: sent message recd from %s",
    cachetable[j].phno);
    octstr_destroy (msgoctstr);
  }
      }
}
      //sg

      //do {
      delta = interval * num_sent - (get_current_time () - first_sent_at);
      if (delta < 0)
delta = 0;
      /*if (num_sent >= max_send)
         delta = -1; */
      conn_wait (server, delta);
      if (conn_read_error (server) || conn_eof (server) || sigint_received)
goto over;
      while ((line = conn_read_line (server)))
{

  //sg
```

```
  strcpy (msg, octstr_get_cstr (line));
  getphno (msg, phno);
  for (j = 0; j < 10; j++)
    {
      if (cachetable[j].valid == 1)
if (!strcasecmp (phno, cachetable[j].phno))
  {
    write (cachetable[j].sockfd, msg, strlen (msg) + 1);
    break;
  }
    }
  //sg

  last_received_at = get_current_time ();
  if (first_received_at == 0)
    first_received_at = last_received_at;
  ++num_received;
  if (num_received == max_send)
    {
      info (0, "Got message %ld: <%s>", num_received,
    octstr_get_cstr (line));
    }
  else
    {
      debug ("receive", 0, "Got message %ld: <%s>", num_received,
     octstr_get_cstr (line));
    }
  octstr_destroy (line);
}
      //} while (delta > 0 || num_sent >= max_send);
    }

over:
  conn_destroy (server);

  for (i = 0; i < num_msgs; i++)
    octstr_destroy (msgs[i]);
  gw_free (msgs);

  end_time = get_current_time ();

  info (0, "fakesmsc: %ld messages sent and %ld received",
num_sent, num_received);
  info (0, "fakesmsc: total running time %.1f seconds",
end_time - start_time);
  delta = last_sent_at - first_sent_at;
  if (delta == 0)
    delta = .01;
  if (num_sent > 1)
    info (0, "fakesmsc: from first to last sent message %.1f s, "
  "%.1f msgs/s", delta, (num_sent - 1) / delta);
  delta = last_received_at - first_received_at;
  if (delta == 0)
    delta = .01;
```

```
  if (num_received > 1)
    info (0, "fakesmsc: from first to last received message %.1f s, "
  "%.1f msgs/s", delta, (num_received - 1) / delta);
  info (0, "fakesmsc: terminating");
  return 0;
}

//sg
void
getphno (char *msg, char *phno)
{
  char *i = index (msg, ' ');
  char *j = index (i + 1, ' ');
  strncpy (phno, i + 1, j - i - 1);
  phno[j-i-1]='\0';
}
```

B.3 phonesim.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define SIZE 255

int mport = 10001;
char smscno[] = "13013";
void formatsms (char *);
void makesms (const char *, char *);
void chkplus (char *);

main (int argc, char *argv[])
{
  int sockfd, cpid;
  struct sockaddr_in serv_addr;
  char send[SIZE], recv[SIZE], host[20], phno[30];
  if (argc < 2)
    {
      printf
("usage: %s phone_no [host_ip_add]\neg. %s +919415222222 172.16.1.100\n",
 argv[0], argv[0]);
      exit (1);
    }
  strcpy (phno, argv[1]);
  strcat (phno, "\0");
  if (argc == 2)
    strcpy (host, "127.0.0.1");
  else
    strcpy (host, argv[2]);

  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
    {
      printf ("client: cannot open socket stream\n");
```

```
      exit (1);
    }

  bzero ((char *) &serv_addr, sizeof (serv_addr));
  serv_addr.sin_family = AF_INET;
  inet_pton (AF_INET, host, &serv_addr.sin_addr);
  serv_addr.sin_port = htons (mport);

  if (connect (sockfd, (struct sockaddr *) &serv_addr, sizeof (serv_addr)) <
      0)
    {
      printf ("client: can't connect to server\n");
      exit (2);
    }
  write (sockfd, phno, strlen (phno));
  if ((cpid = fork ()) == 0)
    {
      while (read (sockfd, recv, SIZE) > 0) /*wait for response */
{
  printf ("\n\aSMS Received...\nOpening...\n\n");
  formatsms (recv);
  printf ("%s\n\n", recv);
}

      exit (0);
    }
  while (fgets (send, SIZE, stdin) != NULL)
    {
      if (strcmp (send, ""))
{
  chkplus (send);
  makesms (phno, send);
  write (sockfd, send, strlen (send));
}
    }
}

void
formatsms (char *arg)
{
  int i = 0, j = 0, num;
  char newarg[1000], hex[3], *p;
  hex[2] = '\0';
  while (arg[i] != '\0')
    {
      if (arg[i] == '%')
{
  hex[0] = arg[++i];
  hex[1] = arg[++i];
  sscanf (hex, "%x", &num);
  newarg[j++] = num;
}
      else if (arg[i] == '+')
newarg[j++] = ' ';
```

```
      else
newarg[j++] = arg[i];
      i++;
    }
  newarg[j] = '\0';
  p = strstr (newarg, "data");
  if (p == NULL)
    p = strstr (newarg, "text");
  strcpy (arg, p + 5);
}

void
makesms (const char *phno, char *arg)
{
  char newarg[1000];
  strcpy (newarg, phno);
  strcat (newarg, " ");
  strcat (newarg, smscno);
  strcat (newarg, " data ");
  strcat (newarg, arg);
  strcpy (arg, newarg);
}

void
chkplus (char *arg)
{
  char newarg[1000];
  int i = 0, j = 0;
  while (arg[i] != '\0')
    {
      if (arg[i] == '+')
{
  newarg[j++] = '%';
  newarg[j++] = '2';
  newarg[j] = 'B';
}
      else
newarg[j] = arg[i];
      i++;
      j++;
    }
  strcpy (arg, newarg);
}
```