A LEO Satellite Email Gateway

Fabian Schulte
Grupo Bioingeniería y Telemedicina
ETSI Telecomunicación
Universidad Politécnica de Madrid

December 11, 2004

Contents

1 Introduction **3**
- 1.1 The Project EHAS . . . 3
- 1.2 Low Earth Orbit Satellites . . . 4
  - 1.2.1 Digital satellites . . . 4
  - 1.2.2 Analog satellites . . . 4
- 1.3 A LEO gateway . . . 5
  - 1.3.1 International gateway . . . 5
  - 1.3.2 Isolated gateway . . . 5

2 Foundations **6**
- 2.1 Email . . . 6
  - 2.1.1 Envelopes and messages . . . 6
  - 2.1.2 RFC-822-MIME Format . . . 6
  - 2.1.3 Mail addresses . . . 7
  - 2.1.4 Mail system components . . . 7
- 2.2 Amateur radio protocols . . . 8
  - 2.2.1 AX.25 . . . 8
- 2.3 Pacsat file protocols . . . 8
  - 2.3.1 FTL0 . . . 8
  - 2.3.2 Pacsat file header . . . 9
  - 2.3.3 Pacsat broadcast protocol . . . 10
  - 2.3.4 Earth mail gateways . . . 11

3 Tools **12**
- 3.1 sendmail . . . 12
  - 3.1.1 Basics . . . 12
  - 3.1.2 Smarthosts . . . 12
  - 3.1.3 Routing with mailer tables . . . 12
  - 3.1.4 Interfacing with a mailer . . . 13
- 3.2 PFH . . . 13
- 3.3 Satellite message transfer . . . 14
  - 3.3.1 Satellite tracking . . . 14
  - 3.3.2 Uploading/downloading tools . . . 14
  - 3.3.3 Tools used in this project . . . 14
- 3.4 Linux and debian . . . 14

4 Implementation **15**
4.1 The Pacsat mailer 15
4.1.1 Interface 15
4.1.2 Implementation 15
4.1.3 Interfacing with sendmail 16
4.2 The fromPacsat program 18
4.2.1 Interface 18
4.2.2 Implementation 19
4.3 The structure of the gateway 20
4.4 Installation 20
4.4.1 PB/PG 20
4.4.2 Pacsat gateway 21
4.5 Future work 22

5 Programs **23**
5.1 toPacsat 23
5.2 fromPacsat 30
5.3 The test scripts 35

Chapter 1

Introduction

1.1 The Project EHAS

The project "Enlace Hispanoamericano de Salud" (EHAS) is carried out by the Group of Bioengineering and Telemedicine of the Technical University of Madrid and the organization Engineers Without Frontiers. Its goal is to improve the medical care in isolated rural areas in South America by installing a communication-network.

There are two kind of medical centers. In the Centros de Salud, which are situated in towns, work the doctors. They have medical equipment and usually access to a telephone line. The smaller Puestos de Salud are dependent on each Centro de Salud because they have an inferior infrastructure. The personal here often need to communicate with its Centro de Salud in order to do a diagnosis. Because of the bad access situation and the absence of a telephone line, it is necessary to find another possibility of communication.

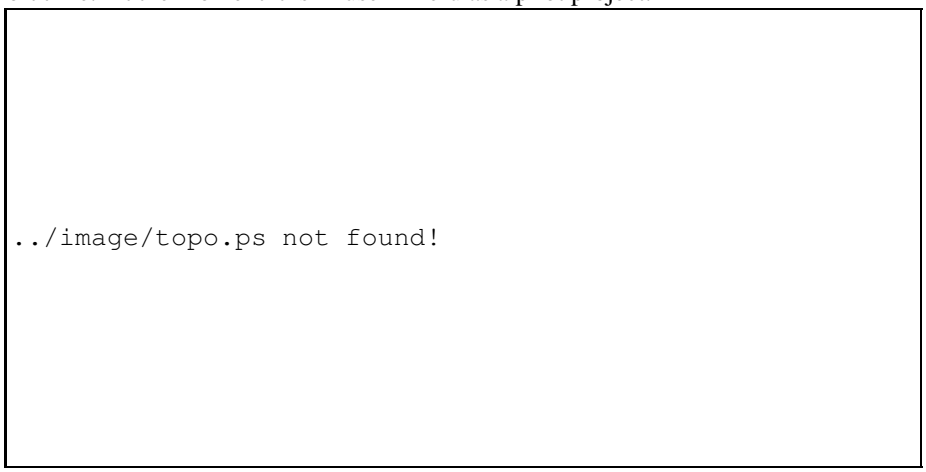
The EHAS network connects all the Centros de Salud and the Puestos de Salud with the Internet and makes communication possible via email. In order to achieve this, the Puestos de Salud are connected with their Centro de Salud by VHF radio. The Centros de Salud usually have telephone access with which a connection to the Internet and the national coordination center is possible. They hold mailboxes for their Puestos de Salud. In special cases, the Puestos de Salud are connected directly to this national center by using HF radio.

Unfortunately, there are some Centros de Salud which do not have a telephone line and where a connection by VHF radio is not possible. They are connected by using a low earth orbit satellite (LEO). When this satellite is in view of a Centro de Salud it receives the sent messages. As soon as possible it sends them to the NodoInternacional in Madrid. Here the messages are downloaded and sent as emails on the Internet. In the other direction emails with a destination of one of these Centros de Salud are first sent to the NodoInternacional where they are forwarded to the satellite. At this point they can be received from the satellite by the Centros de Salud and sent to the Puestos de Salud.

Throughout this process the satellite HealthSat II is used. The connection via satellite should only be used in special cases because of the high cost and because of the high complexity of satellite communication.

With this network it is possible for the Puestos de Salud to have access to medical documents and databases, to consult a doctor or to exchange important informations in

a short time. At the moment it is in use in Peru as a pilot project.



1.2 Low Earth Orbit Satellites

There are two kinds of Low Earth Orbit Satellites (LEO), analog ones and the newer digital ones.

1.2.1 Digital satellites

Low Earth Orbit Satellites circle round the earth at a distance between 500 and 900 km on a polar orbit. Because of this small distance they need to be very fast and therefore are only visible 4 times a day for nearly 15 minutes. However you need only a power of 50W to contact them and to transmit information. They are typical cubics with a length of about 23 cm and a weight of about 9kg. They consist of a receptor at usually about 145MHz, a transmitter at about 435MHz, a power supply and a computer with a RAM large enough to store the information. The batteries can be recharged by a solar panel. The protocol which is used for this connection is the AX.25, which is an amateur radio version of X.25 at a speed of 9600bps.

The digital Pacsat satellites (a project that grew up from AMSAT, the Radio Amateur Satellite Corporation) can store all the information (e.g. files) they receive until it is downloaded by another ground station. Pacsat is a term to describe a digital store and forward satellite.

For the transmission of files you need, besides a computer, hardware equipment, which consists of a TNC, a transceiver, the antenna and a controlling unit. The TNC (Terminal Node Controller) is connected to a serial port of the computer and to the transceiver. It has a modem inside. The modem is necessary to transform the digital information of the computer into analog signals, which the transceiver can send. This transceiver receives the signals from the TNC and the controlling information for the frequency shifting. Then it sends the signals to the antenna. The controlling unit of the antenna is connected directly to the computer. It tracks the antenna.

1.2.2 Analog satellites

Besides the digital Pacsat satellites there are also analog satellites. Even though they were the first LEO satellites in the sky, they are still used. Within their range they work

as a transponder to repeat voice signals immediately on another frequency without storing them.

1.3 A LEO gateway

Communication via LEO satellites, as described in section 1.1, needs an appropriate gateway. There are two kinds of gateways, one in the NodoInternacional in Spain and the gateways in the Centros de Salud.

1.3.1 International gateway

This gateway should receive messages with destination of the Puestos de Salud as emails from the Internet. Then it has to perform several steps to prepare the message for uploading. These are transformation of the message into a format which the satellite understand, compressing and encapsulation of the message and finally queueing it to the uploading program.

To do the processing of the opposite direction, the gateway must look up in the directory which contains the messages downloaded. Then it has to carry out the same steps like above in turned order and to make sure that the email is sent correctly into the Internet.

1.3.2 Isolated gateway

The Internet isolated gateway in the Centros de Salud is similar. It should take the emails received by the VHF radio connection from the Puestos de Salud and transform them in the same way like above. Also the transformation steps for the opposite direction are similar. At the end the gateway is responsible for the correct delivery to the accounts of the corresponding Puestos de Salud, from where it can be transmitted via the VHF radio connection.

Chapter 2

Foundations

2.1 Email

2.1.1 Envelopes and messages

An email consists of an envelope and the encapsulated message. The envelope is created by the mailer which is responsible for delivery. Because there are different possibilities to route an email there are also different envelopes corresponding to the needs of the receiving mail transport agent. The envelope usually contains the recipient's address, the priority, information about security and perhaps the sender's address. With this information the mail transport agent can route the message. The following message itself consists of a header and the body. The header holds the information about the message, the body the real information for the recipient.

2.1.2 RFC-822-MIME Format

The old RFC-822 standard defines several header fields for an email[11]. The most important for our purposes are:

- *To*: the addresses of the most important recipients
- *Cc*: addresses of other recipients; there is no difference in delivery to the *To*: address
- *Bcc*: the same like *Cc*:, but these addresses will be deleted from the header before delivery
- *From*: author of the mail
- *Sender*: sender of the mail, need not to be same as the *From*:
- *Received*: every MTA that forwards this mail inserts this line with information about itself
- *Return-Path*: how to reply to the sender, usually only the sender's address

Besides these, there are many other possible fields like Date:, Reply-To:, Message-Id:, In-Reply-To:, References:, Keywords: or Subject:. Moreover new header fields may be introduced with the string "X-". After a blank line the body of the message follows.

RFC-822 only defines a mail format with a ASCII body. MIME (Multipurpose Internet Mail Extensions) is an extension to RFC-822 to make it possible to send emails with a body written in an non-English alphabet or with a body which contains other data than text. The new header fields are:

- MIME-Version: to indicate that it is a MIME-mail
- Content-Description: description of the mail content
- Content-Id: clear identifier of the mail
- Content-Transfer-Encoding: the way the body is encoded: ASCII with 7 bits (the old way) or with 8 bits, base64 encoding for binary files, quoted-printable encoding for ASCII-binary mixed files
- Content-Type: the type of the body: text, image, audio, video, application, message, multipart; for each type exist several subtypes

2.1.3 Mail addresses

Most mail addresses consist of a user name, a @ symbol and a domain name. The user name is the name of the account the user has on the server and to whose mailbox the email will be delivered. The domain name is based on the Domain Name System (DNS) which is defined in RFC 1034/1035[11]. The whole Internet is divided into several domains on the highest level. Each of these top-level domains itself is divided in many domains on the second level and so on. A hierarchy tree is built. The single domains are separated by points. DNS-Servers change such a DNS-address into a IP-address, which is necessary for connecting to the other host[1].

It is not necessary that a server with the specified domain name really exists. In this case MX (mail exchange) records change the domain name into the real name of the corresponding mail server. There may be several mail servers with different priorities for one domain name.

2.1.4 Mail system components

An email system consists of at least three parts, the Mail User Agent, the Mail Transfer Agent and the Mail Delivery Agents.

The Mail User Agent (MUA) can compose, edit, view and reply to messages. It looks in the user's mailbox for new mail, shows and archives it. It provides an editor for writing mails, adds the mail header and forwards it to the Mail Transfer Agent.

The Mail Transfer Agent (MTA) takes the mail and is responsible for its correct delivery. It has to prepare it, add an envelope and choose a Mail Delivery Agent for routing it to its destination.

There are several Mail Delivery Agents (MDA) for the different kinds of mail and the different destinations. They are responsible for the correct routing of the mail to its destination. For example there is a MDA which delivers the local mail to the user's mailbox and another MDA for routing the mail to another host.

Moreover, there may be other mail system components like e.g. mailbox servers (POP-IMAP), which are not used in this project.

2.2 Amateur radio protocols

The most important and most used amateur radio protocol is AX.25.

2.2.1 AX.25

AX.25 is the amateur radio version of X.25. It provides link-layer communication between stations. These may be two individual stations or an individual one and a multi-port controller. Further on it permits multiple link-layer connections at the same time. It covers the layers physical and data link of the ISO/OSI reference model. Each layer can be divided in sublayers which consist of finite state machines. Any upper-layer protocol (e.g. FTL0 or TCP/IP) may be specified independently. On the link layer the data is transmitted in frames. There are three base types:

- Information Frame (I or UI), transmits the data
- Supervisory Frame (S), responsible for link control like acknowledging and so on
- Unnumbered Frame (U), provides establishing and terminating link connections

Each frame itself consists of several fields, which are:

- flag, 8 bit 01111110, indicating the beginning of a frame
- address, 112/224 bit, the source and the destination of the frame
- control, 8/16 bit, the type of the frame
- PID (Protocol Identifier field, only I frame), 8 bit, indicates which layer 3 protocol is used
- info, 256 bit, the user data
- FCD (Frame Check Sequence field), 16 bit, a checksum for the frame
- flag, the same as above, indicating the end of the frame

AX.25 can work in a connection-oriented mode (with I frames) or in a connection-less mode (with UI frames). It represents an error-free channel[3]. Further on each AX.25 port on a machine must be assigned a AX.25 callsign (e.g. EB4GLO).

2.3 Pacsat file protocols

All here presented Pacsat protocols require an AX.25 connection.

2.3.1 FTL0

The File Transfer Level 0[4] protocol is used for uploading and downloading files to and from a Pacsat. It communicates with a single AX.25 connection and provides bidirectional, full-duplex message transfer. It assumes that the Pacsat File Header for the files to be transmitted is used. The server (satellite) is able to store the uploaded files together with a directory entry and can forward them if a request is transmitted.

FTL0 sends the information in packets. Each AX.25 I frame can contain one or several FTL0 packets. A packet consists of a header and a body. The header consists of 2 bytes and supplies the size and the type of the packet. The body may consist of between 0 and 2047 bytes. A packet has the the following form:

`<length_lsb><h1>[<info>]`

`<length_lsb>` is an 8 bit unsigned integer providing the 8 least significant bits of the length of `<info>`. The bits 7-5 of `<h1>` form the 3 most significant bits of the length, the bits 4-0 encode the packet type (e.g. UPLOAD_CMD, SELECT_CMD, DATA). No checksumming is done because it is assumed that the AX.25 link is error-free.

FTL0 uses two state machines. One is responsible for the uploading, the other one for downloading, selecting and directory administration. Because of this uploading and downloading can be done simultaneously. After a LOGIN_RESP packet and the following response LOGIN_RESP of the server the client (ground station) sends a SELECT_CMD with a description of the desired files (e.g. all files with its callsign as destination and not larger than 500 kB). The server responds with a SELECT_RESP directory list. Following this, the client may send a DL_CMD and the server starts transmission with DATA packets. After successful reception the client sends a DL_ACK_CMD and the server responds with a final handshake DL_COMPLETED_RESP. The uploading procedure of the other state machine is similar. The client sends an UPLOAD_CMD and the server responds with a UL_GO_RESP if the upload is possible. If allowed the client sends the whole file and at the end the server answers with an UL_ACK_RESP. FTL0 can continue interrupted transfer which is necessary since it is possible that a satellite leaves the visible range of a ground station during transmission. The link may be closed by either the server or the client.

2.3.2 Pacsat file header

The PFH[5] is a standardized header for all files to be transmitted from or to Pacsats. It provides full separation of the message header and the message body.

The header starts with the two constant bytes `0xaa` and `0x55` to indicate a PFH follows. Then the mandatory header follows, possibly the extended header and parts of the optional header. After the special end item the body follows. This means the header looks like the following:

`<0xaa> <0x55> <MandatoryH> [<ExtendedH> [<OptionalItems>]] <0x00> <0x00> <0x00>`

Every header item consists of a 2-byte integer Id specifying the item, the 8-bit unsigned length of the data and the data of the item itself. It looks like the following:

`<id><length><data>`

The mandatory header holds

- the `file_number` (4 byte), the `file_name` (8 byte) and the `file_extension` (3 byte), all set by the Pacsat.
- the complete `file_size` (4 byte) indicating the size in byte, set by the uploading ground station.
- the `create_time` and `last_modified_time` (4 byte), like all times in seconds since Jan 1, 1970, set by the uploading ground station.
- a `single_event_upset` flag (1 byte), set by the Pacsat.

- the file_type (1 byte), e.g. ASCII...., set by the uploading ground station.
- the body_checksum and the header_checksum and the body_offset in byte from the beginning, set by the uploading ground station, a 16 bit checksum formed by adding all bytes of the header/body (ignoring overflow).

The extended header is not necessary, but recommended. If used all parts must be present. It consists of

- the callsign of the source of the message (variable length) and the callsign of the ax25_uploader of the message (6 byte), which may be different.
- the upload_time (4 byte), set by the Pacsat.
- the download_count (1 byte) which counts how many times a message have been downloaded, set by the Pacsat.
- the callsign of the destination (variable length) and the callsign of the ax25_downloader (6 byte), which also may be different.
- the download_time (4 byte), set by the Pacsat.
- the expiry_time (4 byte), after which the Pacsat may remove the message if free space is needed, may be set by the uploading ground station.
- the priority of the message (1 byte), higher numbers indicate higher priority, set by the uploading ground station.

Then optional items may follow.

- compression_type (1 byte) and possibly a compression_description (variable length) if a non-standard compression is used.
- the bbs_message_type (1 byte) and the bulletin id number (variable length), set by the uploading ground station.
- title and describing keywords of the message and a file description (all variable length), if a non-standard file is used, set by the uploading ground station.
- a user file_name of the message (variable length), which is used to transfer named files, set by the uploading ground station.

For a list of the Pacsat file types and the compression types see [5].

2.3.3 Pacsat broadcast protocol

A satellite is naturally a broadcast medium. Everyone in range of the Pacsat can hear everything. If there is a message of general interest there may be many people who would like to download it. Using the FTL0 protocol everyone himself has to establish a connection to the satellite which takes a considerable amount of time. With the Pacsat Broadcast Protocol it is possible to send messages from a Pacsat to several users at the same time.

Files, which should be broadcasted, are assigned a 32 bit unique file ID. The protocol uses the unconnected <UI> frame mode of AX.25 by dividing the file in frames.

Each frame has sufficient information so that it can be put in the correct place and in the correct file by the receiving ground station. Files are put in the round-robin broadcast by the Pacsat if they are uploaded with a broadcast rotation priority. Files with a higher priority will be sent more often. For more information about the protocol see[6].

2.3.4 Earth mail gateways

Earth mail gateways route mail over satellites. In a sytem of several earth mail gateways at different places, it is important that only one gateway downloads the message to forward it into the other earth-based network. To achieve this it is possible to lock a download. A gateway should check first the destination field if it is able to deliver the message. If `download_time` is zero it should download the message. If a non-zero `download_time` and a blank `ax25_downloader` occurs, then this indicates that another gateway has started a locked downloading but not yet finished. If also `ax25_downloader` is filled out, then the download has already been carried out successfully. To start a locked downloading the gateway has to set the `lock_destination` flag in the DOWNLOAD_CMD packet.

Chapter 3

Tools

3.1 sendmail

The gateway uses sendmail version 8.9.3.

3.1.1 Basics

`sendmail` is a complex and a configurable Mail Transfer Agent to receive and route emails. It usually runs as a daemon waiting for incoming mail. Once it has received a mail, it uses several Mail Delivery Agents (mailer) to deliver it to the correct address. For example, mail with a local recipient address is processed by the local mailer, which puts the mail in the corresponding mailbox. Mail, which should be sent to another host, can be delivered for example by the SMTP (simple mail transport protocol) mailer.

`sendmail` is configured by the `sendmail.cf` file, which is read every time that sendmail is started. Usually `sendmail.cf` is created by a Makefile out of the `sendmail.mc` file and other configuration files (e.g. the MDA definitions). All these files are written in M4, the GNU macro processor.

3.1.2 Smarthosts

If your host can deliver local mail but cannot look up other hosts on the Internet with DNS, it must first send all non-local email to another host, which performs the delivery. This host is called *smart host* and defined in the sendmail configuration file with:

define('SMART_HOST', *agent*:*host*)

Then all non-local mail will be sent to *host* using the mailer *agent*.[2]

3.1.3 Routing with mailer tables

If you wish to treat certain domains or subdomains specially, you may build a mailertable. An entry in a mailertable consists of a key/value pair. The key is the domain of the address or the subdomain with a leading point. The value is the mailer, a colon and the host, to which it should be sent. For example this mailertable will send all mail with destination addresses of the subdomain *foo.com* to the host *yyy* using the mailer *xxx*.

.foo.com xxx:yyy

3.1.4 Interfacing with a mailer

`sendmail` uses other programs called mailers (MDA) to deliver the mail. A new mailer is defined in the `sendmail` configuration file with MAILER(*mailer*). The definition itself is written in a separate file. The most important definitions are:

- the symbolic name of the mailer (M=)
- the path to the mailer program (P=)
- the arguments (A=) with which it is called, sendmail provides several macros for this, for example the sender's address, the recipients' addresses or the host to which to be sent
- several flags (F=), describing the mailer and its needs
- rewriting rules for the recipients' addresses (R=) and for the sender's address (S=)

Rewriting rules change the addresses following defined rules in a form so that the correct mailer can be chosen and so that the mailer understands them. Every rule consists of a left-hand side value and a right-hand side value. If the address matches the condition of the LHS value it will be replaced by the form given by the RHS value. For building the rules sendmail provides a meta syntax and meta symbols. All addresses run through a stream of several rulesets, which can be extended by defining own rules with the R/S= option[2].

Once sendmail has decided which mailer to use, the set of rewriting rules is performed and the mailer is executed with the given arguments. The email itself is written to the standard input of the mailer program. Then the mailer can read the email and deliver it.

3.2 PFH

`PFH` is a C program written by Rob Janssen[9]. It can add a Pacsat File Header (PFH) to a message for uploading and can interpret a downloaded message and its PFH header.

There are five modes, which are set by command-line options:

- extract the PFH header and view the message on the display or write it to a file.
- write the PFH items of all messages in this directory in a file to get an overview of all downloaded messages.
- the same like above, but only one message is viewed.
- sort and clean up the directory.
- create a PFH and add it to the message; prompts for header values.

3.3 Satellite message transfer

3.3.1 Satellite tracking

As mentioned LEO satellites are only visible for nearly 15 minutes. Because of this the track of the satellite has to be calculated to know when it is visible. There is a mathematical model for doing this, which needs the correct time, the exact position of the ground station and the so called Keplerian elements. Named after Johann Kepler, these are the seven numbers which define an ellipse. For the satellite orbit additional numbers are introduced to compensate several influences. They can be found at www.amsat.org or received by broadcast via satellite and should be updated every week. With these inputs it is possible to calculate the track of the satellite. Further on during the time when the satellite is visible and a transmission is carried out, the antennas have to be controlled in order to provide at every time an optimal position towards the satellite.

3.3.2 Uploading/downloading tools

For the message upload and download tools are necessary. I studied two tools, `WiSP` for Windows95[7] and pb/pg for Linux[8].

`WiSP` is a program package which provides uploading/downloading, file administration, file creation and viewing and satellite tracking. You can write messages and `WiSP` adds the PFH header and puts them in a queue for uploading. Further on `WiSP` calculates the track of several satellites. If a satellite is visible, `WiSP` does the satellite tracking and sends the message to the satellite. At the same time it downloads new messages. Moreover it controls the upload and download frequency since they have to be adjusted because of the Doppler effect. Then `WiSP` extracts the PFH header and saves the message in a directory and views it.

`pb`/`pg` is a program package only for uploading and downloading. `pg` looks in its working directory if there are new files to upload (files with extension `.out`) or files which are only uploaded partially (`.pul`). It then tries to open a connection to the satellite which is specified in the `pg.conf` configuration file. If successful it uploads all files and exits. If the connection fails it tries again for a configurable time. `pb` is the downloading part. It also waits until a connection to the satellite is successful. Then it sends a directory request which is set in the `autoload.dat` file. Here you can specify which files you want, e.g. your destination, a special title or keyword or a maximum file size. These files are then downloaded and saved in the working directory with the extension `.dl`. `pb/pg` does not provide neither file creation and viewing nor satellite tracking nor frequency adjustment.

3.3.3 Tools used in this project

In the EHAS project both `WiSP` and `pb/pg` are used at this time. `pb/pg` is running on a Linux machine, configured with an AX.25 port. It is used for uploading and downloading of the messages. At the same time `WiSP`, running on another machine, does the satellite tracking and the frequency shifting.

3.4 Linux and debian

The whole gateway is developed and tested on debian Linux, kernel version 2.2.14. The installation and configuration of the AX.25 protocol is required.

Chapter 4

Implementation

4.1 The Pacsat mailer

The Mail Transport Agent (mailer) `toPacsat` can be used both in the NodoInternacional and in the Centros de Salud to prepare mail messages for uploading to a Pacsat satellite.

The mailer program itself `toPacsat` is based on the program `PFH` by Rob Janssen [9] and adapted for the purposes of the EHAS project.

4.1.1 Interface

The mailer `pacsat` uses the program `toPacsat` to do the processing. `toPacsat` is called with the callsign of the destination, the senders's address, all the recipients' addresses as arguments and optionally, a priority for the message. Its usage is:

toPacsat [-p *priority*] *destination sender recipient...*

`toPacsat` reads the message to be delivered from its standard input.
It has a header file called `pacsat.h`. This file looks like the following:

```
#define TMPFILE "/var/tmp/XXXXXX"
#define SPOOLDIR "/var/spool/pg"
#define ZIPCALL "/usr/bin/zip -q"
#define TITLE "EHAS message"
#define PBDIR "/var/spool/pb"
#define SENDMAILCALL "/usr/sbin/sendmail"
#define UNZIPCALL "/usr/bin/unzip -p"
```

TMPFILE defines the directory for all necessary temporary files, SPOOLDIR and PBDIR define the directories of the uploader and the downloader. TITLE is a constant which is written in the Pacsat File Header. ZIPCALL, UNZIPCALL, and SENDMAILCALL define the needed system calls with appropriate arguments.

4.1.2 Implementation

First of all, `toPacsat` creates a unique filename for the output file to avoid mutual exclusion problems. Then, it writes the PFH header as described in section 2.3.2 into this file. This includes the mandatory header, the extended header and some parts of the optional header. The message is read from its standard input and stored in a temporary file. The envelope, which will be used by the downloading site of the gateway, is

created and the message is encapsulated in this envelope. The whole temporary file is compressed using the program `zip`. This compressed file is appended to the output file. The missing PFH header, which could not be calculated before (e.g. the checksum of the compressed body), are written. Now the output file is put into the spool directory of the uploading program `pg`. At the end, all temporary files are removed.

`toPacsat` contains the following procedures. The `create_unique_file` function builds a file name by using a timestamp with seconds and microseconds and additionally the process number of this `toPacsat`. This should always guarantee a unique filename, even if there are several invocations of `toPacsat`. It gives the extension `.tmp` to avoid `pg` from uploading the no complete file.

The function `write_pfh` writes all header items to the output file. The header begins with the two characteristic constants. For each item the `id`, the `length` of the value and the value itself are written. The value may be a specific number of zeros or spaces for later initialization of this field or data like the actual time, the file type, the compression type, the priority, the source and the destination callsign or the title of the message. After writing all items, the header is terminated with constants. The source callsign is taken from the `pg.conf` configuration file.

`write_message_to_tmp` reads the email from the `stdin`. A temporary file is created using `mkstemp`. The envelope is then written to this temporary file. It starts with *From* and the sender's address. The second line consists of *To* and all recipients' addresses. Now the complete email is copied to this file.

The `compress_message` function calls the compression program zip in order to get a compressed file with the extension `.zip`. Here I used the option -q to put zip in the quiet mode to produce no output. The message should be compressed to achieve a faster transmission to and from the satellite.

`append_message` only copies the compressed file to the output file. A checksum is calculated by adding all written characters.

The function `write_rest` fills the header with the remaining items, which are now known like the file size, the body offset or the body checksum or now can be calculated like the header checksum, which must be done at the very end. It is calculated in the same way as the body checksum.

At the end, the `clean` function removes the temporary file, the zip file and changes the extension of the output file to `.out`, so that pg detects it as a file to be uploaded.

`toPacsat` includes the `pacsat.h` file. It reports all errors and warnings to the mail facility of the facility `syslog`. The `clean` function is defined as a handler for the signals SIGTERM, SIGINT and SIGHUP. In the case of one of these signals it is called to make sure that all temporary files are deleted.

4.1.3 Interfacing with sendmail

The mailer definition of `pacsat` contains the path of the mailer `toPacsat`, its arguments, the maximum message size, the mailer flags and the rewriting rules.

`The` arguments are the callsign of the destination ($h macro), the sender's address ($g macro), the recipients' addresses ($u macro), which are needed to create an envelope and optionally the priority. I used the $g flag instead of $f because it provides the full address including the local host name.

The mailer flags are set with the option *F=*

- *D* the mailer needs a Date: line (this and the following two to maintain all information which might be necessary later)

- *F* the mailer needs a From: line
- *M* the mailer needs a Message-Id: line
- *h* upper case should be preserved in host name (in order to maintain the correct callsign)
- *m* mailer can deliver several messages to the same host on one transaction. With this flag set, it is possible to send several emails with destination of one Centro de Salud in one satellite message. This strongly improves the time needed to transmit all emails. In this case the $u macro will be expanded to a list of recipients' addresses.

The rewriting rule 55 for the sender's address is set. Below the rewriting rule 55 is defined[2]. RR 55 adds the full host name to the address in the From: line in order to prevent that only a local user name is written. The left hand side (LHS) of the first line looks to see if there is exactly one token, for example user@ would be two tokens and not rewritten. The right hand side (RHS) adds the @ and the host name given by the macro $j. For example *user* is rewritten to *user@mailserver.com*. The LHS of the second line checks for any token with an following @ and the local host name (macro $w). The RHS does the same rewriting as above. *user@local* will be rewritten to *user@mailserver.com*. For the recipients' addresses, no rewriting rule is necessary.

Further on the default maximum size of a message is set to 100,000.

The path of the mailer contains the full path of the binary file.

```
PUSHDIVERT(-1)

ifdef(‘PACSAT_MAILER_ARGS’„
        ‘define(‘PACSAT_MAILER_ARGS’, toPacsat $h $g $u)’)
ifdef(‘PACSAT_MAILER_PATH’„
        ‘define(‘PACSAT_MAILER_PATH’, /usr/local/sbin/toPacsat)’)
ifdef(‘PACSAT_MAILER_MAX’„
        ‘define(‘PACSAT_MAILER_MAX’, 100000)’)
POPDIVERT
####################################
### PACSAT Mailer specification  ###
####################################

VERSIONID(‘@(#)pacsat.m4        1.1 (FS) 05/16/2000’)

Mpacsat,                P=PACSAT_MAILER_PATH, F=DFMhm, S=55, R=0,
M=PACSAT_MAILER_MAX, A=PACSAT_MAILER_ARGS

# Rewriting Rule 55
# adds full host name to local address
S55
R$-     $@$1@$j         user -> user@hub
R$-@$w  $@$1@$j         user@local -> user@hub


LOCAL_CONFIG
```

Moreover, the mailer `pacsat` has to be added to the sendmail configuration file `sendmail.mc`.

The international gateway in the NodoInternacional has to route all emails with destinations of one of the different Centros de Salud to the mailer `pacsat`. For this, a mailertable is required. In this mailertable every possible domain of the Centros de Salud is written together with the mailer `pacsat` and the corresponding host, which is in this case the callsign of the Centro de Salud.

cs1.es	pacsat:EB5GLO
cs2.es	pacsat:EB6GLO
.	.
.	.

In the isolated gateways in the Centros de Salud every email has to be processed by the mailer `pacsat`. This is achieved by setting the smarthost to the callsign of the NodoInternacional and the corresponding mailer to `pacsat` with :

define('SMART_HOST', 'pacsat:EB4GLO')dnl

4.2 The fromPacsat program

The `fromPacsat` program does the opposite direction of the transformation than `toPacsat`. `fromPacsat` is also based on `PFH[9]` and adapted to the EHAS project. It is used both in the NodoInternacional and in the Centros de Salud identically.

4.2.1 Interface

`fromPacsat` takes no arguments.

It is called periodically to check to see if new messages have arrived. Because of this, the only necessary configuration is to add `fromPacsat` to the crontable. `Cron` is a program which executes other programs in given intervals or at a fixed time every day, every week, etc. This information for `Cron` is saved in the crontable `/etc/crontab`. The new crontable looks like the following:

```
   # /etc/crontab:  system-wide crontab
   # Unlike any other crontab you don't have to run the 'crontab'
   # command to install the new version when you edit this file.
   # This file also has a username field, that none of the other crontabs
do.

   SHELL=/bin/sh
   PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
   MAILTO=""

   # m h dom mon dow user  command
   25 6    * * *   root    test -e /usr/sbin/anacron || run-parts
--report /etc/cron.daily
   47 6    * * 7   root    test -e /usr/sbin/anacron || run-parts
--report /etc/cron.weekly
   52 6    1 * *   root    test -e /usr/sbin/anacron || run-parts
--report /etc/cron.monthly
```

```
*/15 *   * * *   mail    /usr/local/sbin/fromPacsat
#
```

Now the program /usr/local/sbin/fromPacsat is called as the user mail every 15 minutes, every hour, every day. I decided to use Cron instead of a forever loop inside the program because this guarantees to be more robust. In the case of a problem during a run, later execution is not affected. Further on, the user mail has to be added to the list of trusted-users in order to be able to use the -f flag of sendmail, which is described later. This is done by adding the user mail to the file /etc/mail/trusted-users.

4.2.2 Implementation

fromPacsat searches the working directory of pb for new messages with the extension .dl, which indicates a downloaded file. This is done in a while loop until all messages are processed. First of all, the PFH header is extracted from every message and the checksums, given by the PFH, are saved. Following this, the remaining body, which is the compressed message, is copied to a temporary file. The program unzip decompresses the file and writes the result in another temporary file. From this file the envelope is extracted and interpreted. After this, sendmail is called with addresses of the envelope and the decompressed message. Now sendmail decides how to deliver the email. In most cases it will use the mailer local to pass it to one of the local accounts, but it is also possible to use another mailer for further delivery. At the end, all temporary files are removed.

read_pfh first checks if a Pacsat File Header follows by reading the first characteristic constants. After this, for each item the id, length and the value are read. All items except the header checksum and the body checksum are rejected because they are no longer needed. At the end, a header checksum check is carried out.

The function copy_message_to_tmpfile creates a temporary file. Then, the rest of the input file, which is the compressed email, is read and copied to this temporary file. At the same time the body checksum is calculated in the same way like in toPacsat. Again, at the end, the current checksum is compared with read one.

The decompress function creates a second temporary file. Unzip decompresses the first temporary file and writes the output to stdout, which is redirected into the second temporary file. This detour is necessary because the original file name, to which unzip usually decompresses, is not known in fromPacsat.

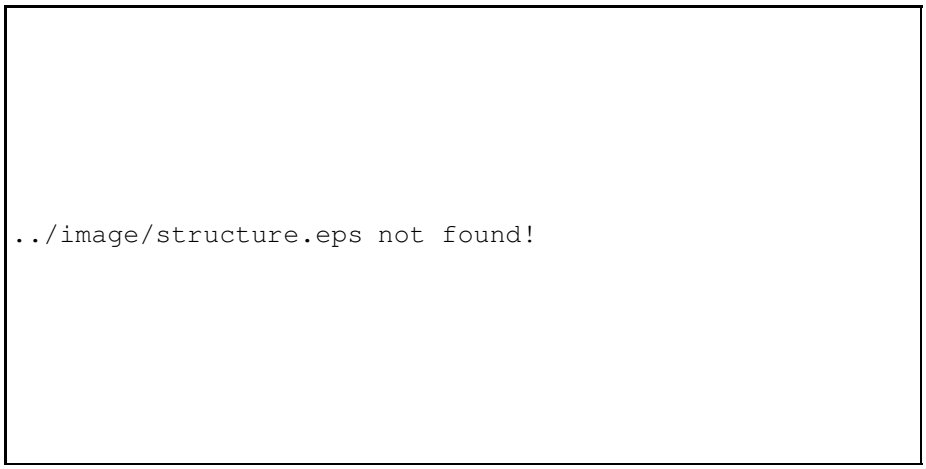
extract_envelope reads the first two lines, which are always the envelope, of this second temporary file. The sender's and the recipients' addresses are extracted.

In the function call_sendmail a pipe to the system call, which executes sendmail with the from and to addresses of the envelope, is opened for writing ('w'). Then, the complete email is read from the second temporary file and copied to this pipe. The from address in the call to sendmail is set by the flag -f, which only trusted-users are allowed to use.

The last function clean removes the two temporary files and the input file.

As in toPacsat first the same header file is included. fromPacsat runs as a daemon. This detaches the program from the terminal control and puts itself as a daemon into the background. The same signal handlers as in toPacsat are installed. All possible errors are reported to syslog, too.

4.3 The structure of the gateway



4.4 Installation

In choosing the right directories for all files I followed in all parts of the gateway the Filesystem Hierarchy Standard (FHS)[10]. In this standard, the locations for directories and files of all possible types are specified. These include all temporary files to `/var/tmp`, all configuration files to `/etc`, all files for further processing to `/var/spool/...` and the binaries to `/usr/local/sbin`.

4.4.1 PB/PG

As mentioned `pb/pg` only works on the current directory. This means that the `pg.conf` file has to be copied to the spool directory set in the `pacsat.h` file and also `that pg` has to be executed here. I used the configuration file as follows:

```
SATELLITE       UOSAT5   #the callsign of the satellite
SATNAME         UO-22 # the name of the satellite
MYCALL          EB4GLO # the callsign of the ground station
MAXIDLE         900 # pg waits 900s without hearing anything
MAXWAIT         0 # tries to establish the connection immediately
```

`pg` stops after MAXIDLE seconds if it does not hear anything of the satellite. Because of this it has to be started periodically. This is achieved by adding an entry in the crontab. It can look like:

```
*/15 * * * * root /var/spool/pg/pg
```

`pb` also has to be executed in its spool directory, where the `pb.conf` file has to be placed. It looks like:

```
SATELLITE       HL02     # the same as above
SATNAME         KO-25
MYCALL          EB4GLO
MAXDAYS         5 # Files older than 5 days are ignored by pb
UDPPORT         5100 # port for controlling commands
KISSLOG         yes # write a KISSLOG file
```

In contrast to pg it never stops automatically. Because of this it is sufficient to make an entry in /etc/init.d, which starts pb every time the system is booted (see section 4.4.2 The Makefile).

4.4.2 Pacsat gateway

The hierarchy

The main directory of the pacsat gateway has five subdirectories.

- src : the two sources toPacsat and fromPacsat, the pacsat.h file and the pg.conf and pb.conf files
- doc : a documentation file describing the use, the installation and the test
- examples: two sendmail.mc example files for use in the NodoInternacional and in the Centros de Salud, an example mailertable and example emails
- mailer : the pacsat.m4 mailer definition
- test : scripts for the test

The Makefile

The main targets of the Makefile are all, install and distrib.

All compiles the sources toPacsat and fromPacsat. Install installs them in the /usr/local/sbin directory. Further on, it copies the pacsat.m4 mailer definition to /usr/share/sendmail/sendmail.cf/mailer and the pacsat.conf file to /etc. The spool directories for pg and pb are created under /var/spool and the owner is set to mail, so that the mailer is able to access them. The spool directory of pg must have write permissions for all users because sendmail takes the sender's name as user for the toPacsat mailer. Only if the sender is not a local user, daemon will be used. After that, the configuration files for pb/pg are copied to these directories. The init.d script pb is copied to /etc/init.d and activated with update-rc.d pb defaults. This will call pb every time the system is started. The path and the call to pb can be found in the script pb. At the end, a symbolic link with origin in the configuration file of pg to the /etc directory is created in order to make the gateway able to read this file. Distrib compresses the whole directory structure to a tar distribution file.

The test

There are two different tests, one for use with satellite (gw_test_sat) and one for use without satellite (gw_test). These tests configure the system both as NodoInternacional and as Centro de Salud and send emails to each other. With them, it is possible to check if the programs and the mailer are installed correctly and if they work. The test with satellite implies an explicit message transfer by the satellite. The test without satellite can be carried out in a shorter time by simulation of the satellite. Both take as arguments :

- the from email address (sender)
- a first email address somewhere on the Internet

- a second email address somewhere on the Internet
- a first local address in the Centro de Salud
- a local address in the NodoInternacional
- a second local address in the Centro de Salud

To simplify the test there are the two scripts `test1` and `test2`, corresponding to `gw_test` and `gw_test_sat`. They already include the arguments.

The tests store the original sendmail configuration (`sendmail.mc, mailertable, local-host-names`) as a backup by calling the script `storeOrig`. After this, the system is configured as a Centro de Salud with the script `confCS`. This includes the smart host is set to the `pacsat` mailer with the host argument EB4GLO, which is the callsign of the NodoInternacional. At this point, all mail will be delivered by the `pacsat` mailer. The domain cs1.es, the domain of the Centro de Salud1, is added to the list of `local-host-names`. With this, all mail to cs1.es will be delivered locally. A mailertable is not necessary since all mail except the local one has to be processed by the smart host. The script `mail_to_IN` then sends different test emails to the NodoInternacional. After sleeping 30s for delivery, the system will be configured like a NodoInternacional with `confIN`. The smart host is now the usual email server. A mailertable is created to route the emails for the Centros de Salud to the mailer `pacsat`. Now the two tests differ. The test with satellite waits until the programs pg and pb have been successfully executed, in other words until the messages have been uploaded and re-downloaded. The test without the satellite does the same as what the satellite would do. It moves all files from the uploader directory to the downloader directory and changes the extension from `.out` to `.dl`. Then both tests send several test emails with `send_to_CS` to the Centro de Salud. After sleeping for 15m 30s, to make sure that cron has called `fromPacsat`, the system is configured again as a Centro de Salud and the upload/download or the file moving is carried out. Again, it waits 15m 30s and the original configuration is restored by `restoreOrig`.

At this point, the addresses in the Internet should have received four emails, the local Centro de Salud addresses four emails and the local NodoInternacional address one email.

The tests have to be executed as root since they change the sendmail configuration.

4.5 Future work

In the future the two programs `pb` and `pg` should also be adapted to the purposes of the EHAS project. This includes the following points

- they should run as daemons in the background without the need to restart them
- they should not produce any output to the screen
- they should report to the syslog
- they should obey the FHS standard

Further on, it is desirable to do the satellite tracking also under Linux. For these two problems I contacted the author of `pb/pg` and he told me that he is already working on these two. So if a new version of `pg/pb` is available, it should be included to this gateway.

Chapter 5

Programs

5.1 toPacsat

```
/* adds a PFH header and compresses the message */
   /* copyright 1991 by PE1CHL */
   /* adapted to the EHAS project by FS 2000 <fabianschulte@web.de>
*/

   #include <stdio.h>
   #include <time.h>
   #include <stdlib.h>
   #include <string.h>
   #include <malloc.h>
   #include <sys/time.h>
   #include <unistd.h>
   #include <sys/stat.h>
   #include <syslog.h>
   #include <stdarg.h>
   #include <signal.h>

   #include "pacsat.h"

   /* define exit error codes */
   int dir_err=1;
   int file_err=2;
   int zip_err=3;
   int ren_err=4;

   int errno,i;
   unsigned int csum;                  /* checksum */
   unsigned long fsizepos, bcsumpos, hcsumpos, boffpos, bodypos;
   unsigned char ftype=0, ctype=0;
   char tmp_file[256]=TMPFILE, file_without_ext[256],zip_file[256],output_file[256];


   /* create unique output file name using timestamp and process id
*/
   int create_unique_file()
```

```
{
  struct timeval tv;
  struct timezone tz;
  gettimeofday(&tv,&tz);
  sprintf(file_without_ext,"%s%s%d-%d-%d",SPOOLDIR,"/",
          tv.tv_sec,
          tv.tv_usec,
          getpid());
  return 0;
}

/* write item with id, length and value */
void my_fputc(int id, int len, char* data, FILE *stream)
{
  fputc((unsigned char)id,stream);   /* write the item */
  fputc((unsigned char)(id>> 8),stream);
  fputc((unsigned char)len,stream);
  fwrite(data,1,len,stream);
}

/* write the Pacsat File Header */
void write_pfh(FILE *out,char *destination, char* priority)
{
  char source[16];
  char s1[256],s2[156];
  const char conf_file[16]="/etc/pg.conf";
  FILE* conf;
  char data[16];
  time_t now;
  int len;
  unsigned long val;

  /* get source callsign of pg.conf file */
  if ((conf=fopen(conf_file,"r"))==NULL)
    {
      syslog(LOG_ERR,"%s %s%s","unable to open pg.conf file:",conf_file,"\n");
      fclose(conf);
      exit(2);
    }
  while (!feof(conf))
    {
      fscanf(conf,"%100s %100s\n",s1,s2);
      if (!strcmp(s1,"MYCALL"))
        strcpy(source,s2);
    }
  fclose(conf);

  time(&now);                          /* get default time */
  fputc(0xaa,out);                     /* PFH magic number */
  fputc(0x55,out);

  /* file number */
  memset(data,'\0',4);
  my_fputc(0x01,4,data,out);
```

```
    /* file name */
    memset(data,' ',8);
    my_fputc(0x02,8,data,out);

    /* file extension */
    memset(data,' ',3);
    my_fputc(0x03,3,data,out);

    /* file size */
    fsizepos = ftell(out) + 3;
    memset(data,'\0',4);
    my_fputc(0x04,4,data,out);

    /* file create time */
    data[0] = (unsigned char)now; /* convert time to little-endian
*/
    data[1] = (unsigned char)(now >> 8);
    data[2] = (unsigned char)(now >> 16);
    data[3] = (unsigned char)(now >> 24);
    my_fputc(0x05,4,data,out);

    /* last modified time */
    memset(data,'\0',4);
    my_fputc(0x06,4,data,out);

    /* file type */
    memset(data,'\0',1);
    my_fputc(0x07,1,data,out);

    /* SEU flag *7
    ftype=1;
    val = atol((char *)"10");
    data[0] = (unsigned char)val;
    data[1] = (unsigned char)(val >> 8);
    data[2] = (unsigned char)(val >> 16);
    data[3] = (unsigned char)(val >> 24);
    my_fputc(0x08,1,data,out);

    /* body checksum */
    bcsumpos = ftell(out) + 3;
    memset(data,'\0',2);
    my_fputc(0x09,2,data,out);

    /* header checksum */
    hcsumpos = ftell(out) + 3;
    memset(data,'\0',2);
    my_fputc(0x0a,2,data,out);

    /* body offset */
    boffpos = ftell(out) + 3;
    memset(data,'\0',2);
    my_fputc(0x0b,2,data,out);
```

```
/* source callsign */
len = (int)strlen((char *)source);
my_fputc(0x10,len,source,out);

/* ax25 uploader */
memset(data,' ',6);
my_fputc(0x11,6,data,out);

/* upload time */
memset(data,'\0',4);
my_fputc(0x12,4,data,out);

/* download count */
memset(data,'\0',1);
my_fputc(0x13,1,data,out);

/* destination callsign */
len = (int)strlen((char *)destination);
my_fputc(0x14,len,destination,out);

/* ax25 downloader */
memset(data,' ',6);
my_fputc(0x15,6,data,out);

/* download time */
memset(data,'\0',4);
my_fputc(0x16,4,data,out);

/* expire time */
memset(data,'\0',4);
my_fputc(0x17,4,data,out);

/* priority */
val = atol((char *)priority); /* get value of decimal input */
data[0] = (unsigned char)val; /* convert to little-endian */
data[1] = (unsigned char)(val >> 8);
data[2] = (unsigned char)(val >> 16);
data[3] = (unsigned char)(val >> 24);
my_fputc(0x18,1,data,out);

/* compression type */
ctype=2;
data[0] = (unsigned char)atoi((char *)"2");
my_fputc(0x19,1,data,out);

/* title */
my_fputc(0x22,strlen(TITLE),TITLE,out);

fputc(0,out);                 /* terminate PFH */
fputc(0,out);
fputc(0,out);

/* get body offset */
bodypos = ftell(out);
```

```
}

/* read message and write it to a temporary file */
int write_message_to_tmp(int argc, char **argv,int offset)
{
  int j,len;
  char data[1024];
  FILE *tmp_out;

  /* make and open temp file */
  mkstemp(tmp_file);
  tmp_out=fopen(tmp_file,"w+b");
  if (tmp_out == NULL)
    {
      syslog(LOG_ERR,"%s %s %d%s","unable to open temporary file:",tmp_file,errno,"\n
      fclose(tmp_out);
      return file_err;
    }
  /* write envelope */
  fwrite("From ",1,5,tmp_out);
  fwrite(argv[offset+1],1,(int)strlen(argv[offset+1]),tmp_out);
  fwrite("\nTo",1,3,tmp_out);
  for (j=offset+2;j<argc;j++)
    {
      fwrite(" ",1,1,tmp_out);
      fwrite(argv[j],1,(int)strlen(argv[j]),tmp_out);
    }
  fwrite("\n",1,1,tmp_out);
  /* write message */
  do
    {
      len=(int)fread(data,1,sizeof(data),stdin);
      fwrite(data,1,len,tmp_out);
    }
  while (feof(stdin)==0);
  fclose(tmp_out);
  return(0);
}

/* compress message using zip */
int compress_message()
{
  int zip_value;
  char complete_call[256];

  strcpy(zip_file,tmp_file);
  strcat(zip_file,".zip");
  sprintf(complete_call,"%s %s %s",ZIPCALL,zip_file,tmp_file);
  zip_value=system(complete_call);
  if (zip_value!=0 && zip_value!=2)
    {
      syslog(LOG_ERR,"%s %d%s","error while compressing using zip:",zip_value,"\n");
      return zip_err;
    }
```

```
}

/* write compressed message into output file */
void append_message(FILE *out)
{
  FILE *zip_in;
  int len;
  unsigned char data[2048];

  zip_in=fopen(zip_file,"r+b");
  csum=0;
  do
    {
      len=(int)fread(data,1,sizeof(data),zip_in);
      fwrite(data,1,len,out);
      while (len>0) csum+=data[--len];
    }
  while(feof(zip_in)==0);
  fclose(zip_in);
}

/* write the last items */
int write_rest(FILE *out)
{
 /* store total file size */
  unsigned long val;

  val = ftell(out);
  fseek(out,fsizepos,0);
  fputc((unsigned char)val,out);
  fputc((unsigned char)(val >> 8),out);
  fputc((unsigned char)(val >> 16),out);
  fputc((unsigned char)(val >> 24),out);

  /* store calculated body checksum */

  fseek(out,bcsumpos,0);
  fputc((unsigned char)csum,out);
  fputc((unsigned char)(csum >> 8),out);

  /* set correct body offset */

  fseek(out,boffpos,0);
  fputc((unsigned char)bodypos,out);
  fputc((unsigned char)(bodypos >> 8),out);

  /* calculate header checksum */

  csum = 0;
  fseek(out,0L,0);

  while (bodypos-- > 0)
    csum += fgetc(out);
```

```
    /* store calculated header checksum */

    fseek(out,hcsumpos,0);
    fputc((unsigned char)csum,out);
    if (fputc((unsigned char)(csum >> 8),out)==EOF)
      {
        syslog(LOG_ERR,"%s\n","error while writing file");
        return file_err;
      }
    fclose(out);
    return 0;
  }

  /* remove temp files and rename output file */
  void clean()
  {
    if (remove(tmp_file)!=0)
      syslog(LOG_WARNING,"%s %s %d%s","error while removing",tmp_file,errno,"\n");
    if (remove(zip_file)!=0)
      syslog(LOG_WARNING,"%s %s %d%s","error while removing",zip_file,errno,"\n");
    if (rename(output_file,strcat(file_without_ext,".out"))!=0)
      {
        syslog(LOG_ERR,"%s %s %d%s","unable to rename file",file_without_ext,errno,"\n'
        exit(ren_err);
      }
  }

  /* main takes the destination callsign, the from and all to addresses
as arguments and optional the priority with -p xxx */

  int main(int argc, char **argv)
  {
    int error;
    char priority[1];
    FILE *out;

    /* open log connection */
    openlog(argv[0],LOG_PID,LOG_MAIL);
    /* get priority */
    i=1;
    strcpy(priority,"0");   /* default */
    while (argv[i][0]=='-')
      {
        if (argv[i][1]=='p')
          strcpy(priority,argv[++i]);
        i++;
      }
    if ((error=create_unique_file())!=0)
      {
        clean(0);
        exit(error);
      }
    /* open file */
    sprintf(output_file,"%s%s",file_without_ext,".tmp");
```

```
     if ((out = fopen(output_file,"w+b")) == NULL)
       {
         syslog(LOG_ERR,"%s %s %d%s","could not open outputfile:",output_file,errno,"\n"
         fclose(out);
         clean(0);
         exit(file_err);
       }
     write_pfh(out,argv[i],priority);
     if ((error=write_message_to_tmp(argc,argv,i))!=0)
       {
         clean(0);
         exit(error);
       }
     if ((error=compress_message())!=0)
       {
         clean(0);
         exit(error);
       }
     append_message(out);
     if ((error=write_rest(out))!=0)
       {
         clean(0);
         exit(error);
       }
     clean(0);
     syslog(LOG_INFO,"%s %s %s%s","file",file_without_ext,"created
correctly","\n");
     closelog();
     return(error);
   }
```

5.2 fromPacsat

```
/* extract email from PFH file and delivers it */
   /* copyright 1991 by PE1CHL */

   /* adapted to the EHAS Project 2000 by FS <fabianschulte@web.de
*/


   #include <stdio.h>
   #include <time.h>
   #include <syslog.h>
   #include <sys/types.h>
   #include <dirent.h>
   #include <unistd.h>
   #include <stdlib.h>
   #include <string.h>

   #include "pacsat.h"
```

```
   /* define exit error codes */
   int file_err=1;
   int nopfh_err=2;
   int dir_err=3;
   int env_err=4;
   int unzip_err=5;
   int sm_err=6;
   int errno;

   int bsum;         /* body checksum */

   char tmp_file1[256]=TMPFILE;
   char tmp_file2[256]=TMPFILE;
   char complete_file[256];                    /* input file */

   /* read the pfh and do checksum calc */
   int read_pfh(FILE *inp)
   {
     unsigned char data[2048];                    /* buffer */
     unsigned long val;
     unsigned int hsum,csum;  /* header checksum and counter for checksum
*/
     int id, id2, len, i;

     /* check magic number AA55 */
     if (fgetc(inp) != 0xaa || fgetc(inp) != 0x55)
       {
         syslog(LOG_ERR,"%s\n","no Pacsat header file");
         fclose(inp);
         return nopfh_err;
       }
     csum = 0xaa + 0x55;

     /* read header items */
     while ((id = fgetc(inp)) != EOF &&
            (id2 = fgetc(inp)) != EOF &&
            (len = fgetc(inp)) != EOF)
       {
         csum += id + id2 + len;                  /* update header
checksum */
         id |= id2 << 8;
         if (id == 0 && len == 0)          /* end of header?  */
           break;
         if (fread(data,1,len,inp) != len) /* read data */
           break;
         data[len] = '\0';
         if (len <= 4)
           {
             val = 0;
             for (i = 0; i < len; i++)
               val |= (unsigned long) data[i] << (8 * i);
           }
         /* special treatment for some header items */
         switch (id)
```

```
          {
          case 0x0a:                              /* header checksum
*/
            hsum = (unsigned)val;         /* get it's value */
            break;                                /* and don't update
csum */
          case 0x09:                              /* body checksum
*/
            bsum = (unsigned)val;         /* get value & update csum
*/
          default:
            for (i = 0; i < len; i++)
              csum += data[i];            /* update header checksum
*/
            break;
          }
      }
    /* checksum error ?  */
    if (hsum != csum)
      syslog(LOG_WARNING,"%s %d\n","header checksum error:",csum);
    return 0;
  }

  /* read the message and copy it to tmp outputfile */
  int copy_message_to_tmpfile(FILE* inp)
  {
    unsigned char data[2048];                 /* buffer */
    int len, i;
    unsigned int csum=0;
    FILE *out;                                /* file stream
for tmp file */

    /* create temporary outputfile */
    mkstemp(tmp_file1);
    if ((out = fopen(tmp_file1,"wb")) == NULL)
      {
        syslog(LOG_ERR,"%s %s %d%s","could not open output file",tmp_file1,errno,"\n");
        fclose(inp);
        return file_err;
      }
    /* copy rest of inputfile to outputfile */
    while ((len = (int)fread(data,1,sizeof(data),inp)) != 0) /* read
data */
      {
        for (i = 0; i < len; i++)
          {
            csum += data[i];              /* calc checksum */
          }
        if (fwrite(data,1,len,out) != len)        /* write to output
*/
          {
            syslog(LOG_ERR,"%s %s %d%s","error while writing to outputfile",tmp_file1,e
            return file_err;
          }
```

```
      }
    /*checksum error ?*/
    if (bsum != csum)
      syslog(LOG_WARNING,"%s %d%s","body checksum error:",csum,"\n");
    fclose(out);
  return 0;
  }

  /* decompress message */
  int decompress()
  {
    int unzip_value;
    char complete_call[256];

    /* create temp outputfile2 */
    mkstemp(tmp_file2);
    sprintf(complete_call,"%s %s %s%s",UNZIPCALL,tmp_file1,">",tmp_file2);
    /* call unzip */
    unzip_value=system(complete_call);
    if (unzip_value!=0 && unzip_value!=1)
      {
        syslog(LOG_ERR,"%s %d%s","error while decompressing using
unzip:",unzip_value,"\n");
        return unzip_err;
      }
    return 0;
  }

  /* extract envelope of message */
  void extract_envelope(FILE *inp, char *sender, char *recipients)
  {
    char line1[256], line2[4096];
    /* extract sender's and recipients' addresses */
    fgets(line1,256,inp);
    strcpy(sender,line1+5);
    sender[strlen(sender)-1]='\0';
    fgets(line2,4096,inp);
    strcpy(recipients,line2+3);
    recipients[strlen(recipients)-1]='\0';
  }

  /* call sendmail with message */
  int call_sendmail(char *sender, char *recipients, FILE *inp)
  {
    FILE *sm_inp;                        /* descriptor for pipe */
    int i;
    char complete_call[256];

    /* call sendmail */
    sprintf(complete_call,"%s %s%s %s",SENDMAILCALL,"-f",sender,recipients);
    /* open pipe */
    sm_inp=popen(complete_call,"w");
    if (sm_inp==NULL)
      {
```

```
        syslog(LOG_ERR,"%s","error while calling sendmail\n");
        return sm_err;
      }
    /* copy message to pipe of sendmail */
    while((i=fgetc(inp))!=EOF)
      fprintf(sm_inp,"%c",i);
    pclose(sm_inp);
    return 0;
  }

  /* remove all files */
  void clean(int i)
  {
    /* remove temporary files */
    if (remove(tmp_file1)!=0)
      syslog(LOG_WARNING,"%s %s %d%s","error while removing",tmp_file1,errno,"\n");
    if (remove(tmp_file2)!=0)
      syslog(LOG_WARNING,"%s %s %d%s","error while removing",tmp_file2,errno,"\n");
    remove(complete_file);
  }

  int main (int argc, char **argv)
  {
    struct dirent *dir_entry;
    FILE *inp, *inp2;
    DIR *dir_stream;                              /* directory descriptor
*/
    char sender[256],recipients[4096];
    int error=0;

    /* set program in daemon mode */
    daemon(1,1);
    /* install signal handler */
    /*signal(SIGTERM,clean(int));
    signal(SIGINT,clean(int));
    signal(SIGHUP,clean(int));*/
    /* open log connection */
    openlog(argv[0],LOG_PID,LOG_MAIL);
    /* open spool directory and look for downloaded files */
    dir_stream=opendir(PBDIR);
    if (dir_stream==NULL)
      {
        syslog(LOG_ERR,"%s %s %s%s","could not open pb-directory",PBDIR,errno,"\n");
        clean(0);
        exit(dir_err);
      }
    dir_entry=readdir(dir_stream);
    while (dir_entry!=NULL)
      {
        /* search completely downloaded files .dl */
        if (strstr(dir_entry->d_name,".dl")!=NULL)
          {
            bsum=0;
            sprintf(complete_file,"%s%s%s",PBDIR,"/",dir_entry->d_name);
```

```
            /* open inputfile */
            if ((inp = fopen(complete_file,"rb")) == NULL)
              {
                syslog(LOG_ERR,"%s %s %d%s","could not open input
file", complete_file,errno,"\n");
                clean(0);
                exit(file_err);
              }
            if ((error=read_pfh(inp))!=0)
               {
                 clean(0);
                 exit(error);
               }
            if ((error=copy_message_to_tmpfile(inp))!=0)
              {
                clean(0);
                exit(error);
              }
            fclose(inp);
            if ((error=decompress())!=0)
              {
                clean(0);
                exit(error);
              }
            if ((inp2 = fopen(tmp_file2,"r")) == NULL)
              {
                syslog(LOG_ERR,"%s %s %d%s","could not open input
file",tmp_file2,errno,"\n");
                clean(0);
                exit(file_err);
              }
            extract_envelope(inp2, sender, recipients);
            if ((error=call_sendmail(sender, recipients, inp2))!=0)
              {
                clean(0);
                exit(error);
              }
            fclose(inp2);
            clean(0);
          }
        dir_entry=readdir(dir_stream);
      }
    closedir(dir_stream);
    closelog();
    return error;
  }
```

5.3 The test scripts

The test1 script:

```
#!/bin/sh
```

```
   ./gw_test you@your.domain a@internet.domain b@internet.domain c@CS1
d@IN e@cs1;
```

The test2 script:

```
   #!/bin/sh
   ./gw_test_sat you@your.domain a@internet.domain b@internet.domain
c@CS1 d@IN e@cs1;
```

The gw_test script:

```
   #!/bin/sh
   echo Usage:  gw_test myaddress Internetaddress1 Internetaddress2
CS1address1 INaddress CS1address2;
   echo PacsatMailGateway test without uploading/downloading;
   echo These test will take about 3 minutes, please wait.....;
   ./store_orig;
   ./confCS;
   ./mail_to_IN $1 $2 $3 $4;
   sleep 30s;
   ./confIN;
   for a in ‘ls /var/spool/pg‘;
   do
      if [ $a != “pg.conf” ]
      then
         b=‘echo $a | sed ’s/.out/.dl/g’‘;
         ‘mv /var/spool/pg/$a /var/spool/pb/$b‘;
      fi
   done
   ./mail_to_CS $1 $2 $4 $6 $5;
   sleep 1m 30s;
   ./confCS;
   for a in ‘ls /var/spool/pg‘;
   do
      if [ $a != “pg.conf” ]
      then
         b=‘echo $a | sed ’s/.out/.dl/g’‘;
         ‘mv /var/spool/pg/$a /var/spool/pb/$b‘;
      fi
   done
   sleep 1m 30s;
   ./restore_orig
```

The gw_test_sat script:

```
   #!/bin/sh
   echo Usage:  gw_test_sat myaddress Internetaddress1 Internetaddress2
CS1address1 INaddress CS1address2;
   echo PacsatMailGateway test with uploading/downloading;
   ./store_orig;
   ./confCS;
   ./mail_to_IN $1 $2 $3 $4;
   sleep 30s;
   ./confIN;
   echo Now start uploading with pg and downloading with pb and after
press Return;
```

```
   read;
   ./mail_to_CS $1 $2 $4 $6 $5;
   sleep 15m 30s;
   ./confCS;
   echo Now start uploading with pg and downloading with pb and after
press Return;
   read;
   sleep 15m 30s;
   ./restore_orig;
```

The confCS script:

```
   #!/bin/sh
   cp ../examples/sendmailCS.mc /etc/mail
   mv /etc/mail/sendmailCS.mc /etc/mail/sendmail.mc
   rm /etc/mail/mailertable
   touch /etc/mail/mailertable
   cp ../examples/local-host-names /etc/mail
   sendmailconfig
```

The confIN script:

```
   #!/bin/sh
   cp ../examples/sendmailIN.mc /etc/mail
   mv /etc/mail/sendmailIN.mc /etc/mail/sendmail.mc
   cp ../examples/mailertable /etc/mail/mailertable
   rm /etc/mail/local-host-names
   touch /etc/mail/local-host-names
   sendmailconfig
```

The mail_to_IN script:

```
   #!/bin/sh
   sendmail -f$1 $2 $3 <../examples/mail1
   sendmail -f$1 $2 $4 <../examples/mail2
```

The mail_to_CS script:

```
   #!/bin/sh
   sendmail -f$1 $2 <../examples/mail3
   sendmail -f$1 $3 $4 <../examples/mail4
   sendmail -f$1 $3 $5 <../examples/mail4
```

The storeOrig script:

```
   #!/bin/sh
   mv /etc/mail/sendmail.mc /etc/mail/sendmail.mc.orig
   mv /etc/mail/mailertable /etc/mail/mailertable.orig
   mv /etc/mail/local-host-names /etc/local-host-names.orig
```

The restoreOrig script:

```
   #!/bin/sh
   mv /etc/mail/sendmail.mc.orig /etc/mail/sendmail.mc
   mv /etc/mail/mailertable.orig /etc/mail/mailertable
   mv /etc/mail/local-host-names.orig /etc/local-host-names
```

Bibliography

[1] Computer Networks, 3rd edition, Andrew S. Tanenbaum, Prentice Hall, 1996

[2] sendmail by Bryan Costales O'Reilly & Associates, Inc. 1993

[3] AX.25 Link Access Protocol for Amateur Packet Radio V2.2, 11/11/1997 by Tucson amateur Radio Corporation

[4] Pacsat protocol: File Transfer Level 0 by Jeff Ward, Harold E. Price, available at http://www.amsat.org/amsat/sats/nk6k/msatpro.html

[5] Pacsat File Header definition by Jeff Ward, Harold E. Price, available at http://www.amsat.org/amsat/sats/nk6k/msatpro.html

[6] Pacsat Broadcast Protocol by Harold E. Price and Jeff Ward, available at http://www.amsat.org/amsat/sats/nk6k/msatpro.html

[7] WiSP for Windows95/NT, available at http://www.amsat.org

[8] pb/pg 1.4. by Bent Bagger, available at http://www.amsat.org

[9] PFH by Rob Janssen 1991

[10] Filesystem Hierarchy Standard 2.1 edited by Daniel Quinlan, available at http://www.pathname.com/fhs

[11] RFC (Request for comment) standard archive: e.g. http://www.faqs.org/rfcs