Implementation of The Pacsat Protocol Suite on SUNSAT

J. Boot and S. Mostert
Electronic Systems Laboratory
Department of Electronic Engineering
University of Stellenbosch

Abstract *- The Pacsat protocol was devised to effectively communicate with low earth orbiting satellites carrying a store and forward facility. This article gives a very brief overview of the protocol and discusses the implementation of a server based on this protocol on SUNSAT, the micro satellite being developed at the University of Stellenbosch.*

1. Introduction

Low earth orbiting satellites enable us to communicate with people all over the globe. The satellites can receive and store messages while over one part of the earth and these messages can be downloaded later when the satellites are over different parts of the earth.

Low earth orbiting satellites have restricted availability of resources to groundstations. Because of a LEO satellite's orbit, it is only visible to a certain groundstation for limited times, typically ten to twenty minutes at a time. Furthermore a groundstation seldom has sole access to that satellite during these times - the satellite has to communicate with all the groundstations in its footprint demanding its attention. In addition, the communication channel bandwidth between the satellite and the groundstations is restricted.

Effective use of a satellite channel requires an effective protocol. The first access protocol used was the command line interpreter style of the bulletin board system (BBS). This protocol proved to be too ineffective, as the link became very congested. The Pacsat protocol suite was then developed to better utilise the satellite bandwidth. To do further research into more optimal protocols, we will start by implementing the Pacsat protocol suite on SUNSAT.

SUNSAT, the micro satellite being developed at the University of Stellenbosch, is an excellent vehicle for developing communication protocols. It will offer an opportunity to experiment with multiple protocols, such as a store and forward facility as well as a bulletin board system.

This paper gives a background on Pacsat and an overview of its implementation on SUNSAT.

2. Background

Pacsat is a generic term in the radio amateur service for a low earth orbiting satellite that carries a large on-board memory for the purpose of data storage and retrieval by groundstations[1]. A low earth orbit satellite can be described as a multi-channel, full duplex device, with short, periodic access times dictated by orbital mechanics. If it also has a large storage capacity dedicated to data storage and retrieval by groundstations, it can be called a PACSAT. Examples of these are UOSAT5, KITSAT-A and KITSAT-B. SUNSAT, the micro satellite being developed at the University of Stellenbosch, will also be a PACSAT.

2.1 The Pacsat protocol suite

The attributes of a PACSAT, mentioned above, mandate a different access approach than the standard

command-line interpreter style of a bulletin board system. This led to the development of the Pacsat Protocol Suite at the University of Surrey. According to the protocol, the satellite acts solely as a file server - it does not interpret files and plays no part in the forwarding of mail. The groundstations act as clients and can request a directory and files from the server. The suite consists of two protocols, namely the File Transfer Level 0 (FTL0) protocol[2] and the Pacsat Broadcast Protocol (PBP)[3].

2.2 The files on a PACSAT

Files that are stored on a PACSAT server, consists of a header and body. The header of every file is the so called Pacsat File Header (PFH) and contains information concerning the file, for example the length of the file, the date of creation, the destination of the file, the source of the file, some keywords, the file compression technique used, and more[4]. The body of the PACSAT-file is the file itself. It can be in ascii or binary format and may be compressed. Files found on the server can be files uploaded by clients or files generated by the spacecraft itself. Typical files that are found on PACSATs are personal messages, small programs, images, digitized voice, satellite log files and BBS mail files. Theoretically anything that can be stored digitally can be placed on a PACSAT. The PFH enables a groundstation to identify a file and its contents when it did not request the file itself, but captured it when it was broadcast on another groundstations request.

2.3 The communication method

The Pacsat protocol suite runs on the AX.25 packet-radio link-layer protocol. Two modes of communication are used. The first is point to point communication (Figure 1a), which is used solely when a groundstation uploads a file to the server. The second is point to multipoint communication (Figure 1b), which is used by the server to transmit directories, files and other information.

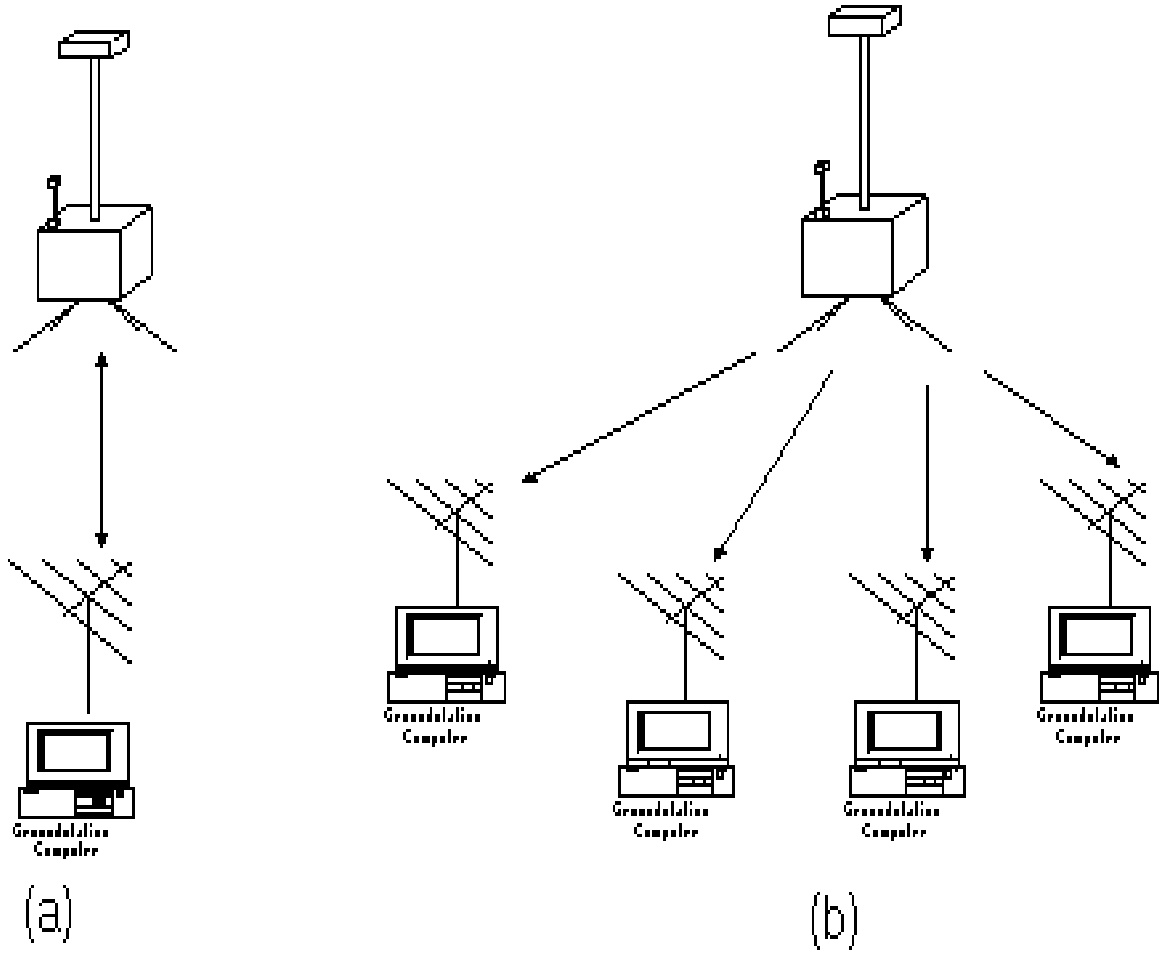

Figure 1

The point to point communication occurs using AX.25 connected mode information frames and the FTL0 protocol. It is solely used for the uploading of files to the server. The FTL0 protocol provides that a file that is only partly uploaded in a session can be completed later by sending only the missing data. It is not necessary

to resend correctly received data.

In a typical session a groundstation will connect to the satellite and the satellite will acknowledge the connection. The groundstation will then request to upload its file and will provide the size of the file. If there is space on the storage facility, the server will allocate a file number to the file and pass it on to the client. The client will now continue to upload the file. When the upload is completed, the server will validate the file and add it to the directory. If some error has occurred, the file will be discarded and the client will be informed accordingly.

The point to multipoint communication occurs using AX.25 unnumbered information frames and the PBP protocol. It is used by the client (or groundstation) for transmitting directory and file requests, and by the server (or satellite) for broadcasting all data packets. Packets transmitted this way can be received and stored by any client (groundstation). The protocol guarantees the following :

- any frame received correctly can be stored at the correct position of the file to which it belongs,
- when all frames are received, the client will know that the file is complete.

A typical session will start with a client transmitting a request. The server will decode the request, acknowledge it and add the client to the broadcast rotation list. A client can only have one request in the rotation list at a time. Broadcast time is allocated to each rotation list entry. An entry is deleted from the list when

- it has completed
- after 10 minutes
- another request is received from the same client
- a requested file can not be opened.

3. The implementation on SUNSAT

As a service to the amateur radio society, it was decided to provide a store and forward facility on SUNSAT. Very good groundstation software, implementing the Pacsat Protocol Suite, already exists and is widely used, therefore it was decided to implement a PACSAT server on SUNSAT. SUNSAT has two on board computers and a 64 megabyte storage facility, that can be used for data storage. This makes SUNSAT an excellent vehicle for a PACSAT server.

3.1 Schedulability

SUNSAT uses an earliest-deadline-first scheduling kernel on which its software runs[5]. Since the PACSAT server will only be a part of a large collection of software tasks that has to run on the satellite (Figure 2) the server-engine had to be disassembled into small schedulable processes. The protocol had to be studied intently to find the best way to do the disassembly. If the processes become too slow then the schedulability of the system will be severely handicapped. Similar to existing PACSATs, the FTL0 and PBP engines were separated and implemented individually. Each of these were broken down further, making the system scheduling-friendly.

3.2 Memory usage

A second consideration in the implementation of the protocol suite was memory usage. Although there are still only a few groundstations in South Africa communicating with satellites, there are many more in Europe and the Americas. The server must be able to handle at least twenty requests simultaneously, as well as servicing up to four clients uploading data. To keep track of all this communication takes up a fair amount of memory. Also, the large collection of software dictates that each system has to use as little memory as possible. For example, the PBP-engine allocates 128 bytes of memory for a directory request, and 96 bytes for a file request. The worst case scenario would be to handle 20 directory requests, which would consume 2560

bytes of memory. (Because of the broadcast mode of the protocol, this should never happen, since most groundstations can get the directory by just listening to some other groundstations' requests). A more typical scenario for a loaded system would be say 4 (20%) directory requests and the rest file requests, consuming a total of 1536 bytes of memory.

To keep track of the PACSAT files, the server keeps a directory list. This list uses 32 bytes for every entry, therefore every 32 files will consume 1 kilobyte of memory. Typically there can be around 400 files on the server, in total around four and a half megabyte in size. The list that the server will keep of its files will take 400*32 = 12.5 kilobytes of memory. Obviously, memory has to be allocated dynamically as the need for it arises, and must be freed when not needed, so other processes can use it.

3.3 Processes and the kernel

Different processes performing different tasks has to compete for the resources of the satellites computer system. The kernel takes care of all this. Processes get queued, each with a certain deadline. The process with the earliest deadline will be scheduled first. To keep the kernel schedulable, long tasks has to be broken down into smaller, simpler pieces. Information has to be passed between the different pieces. This limits the amount to which one can sensibly break down the tasks.

The Pacsat server software is broken down in 2 major parts: one implementing the FTL0 protocol and the other the PBP protocol (Figure 3). The further breakdown of the PBP engine is shown in Figure 4. One process handles the broadcast requests received by clients and the acknowledging thereof. Another is responsible for the transmission (broadcasting) of the requested data. A third process handles files generated on board the satellite and incorporates them into the server. A fourth process may be used to mix other information such as telemetry data, status reports etc. into the broadcast traffic. A similar breakdown is made for the FTL0 engine.

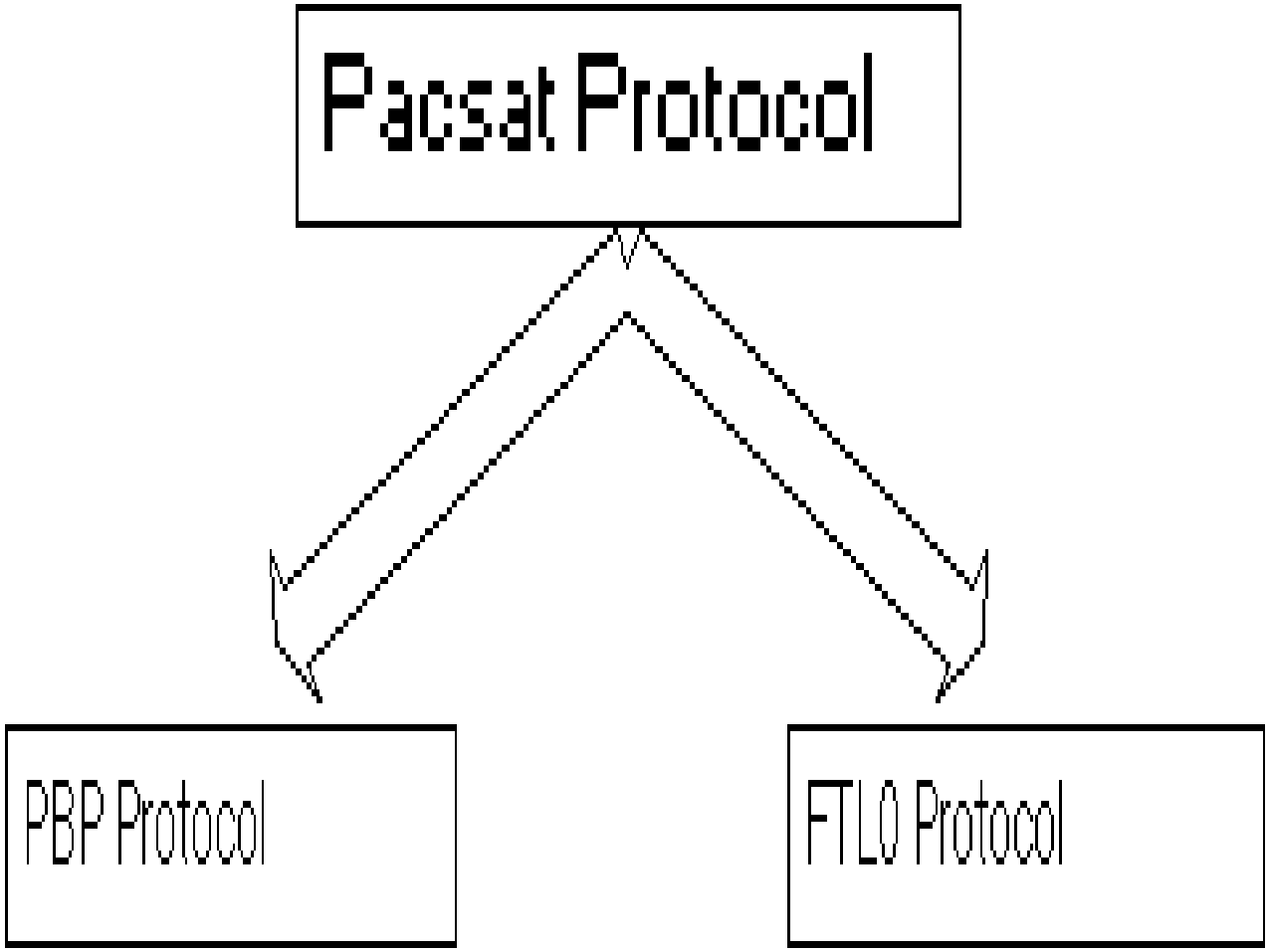

Figure 3.

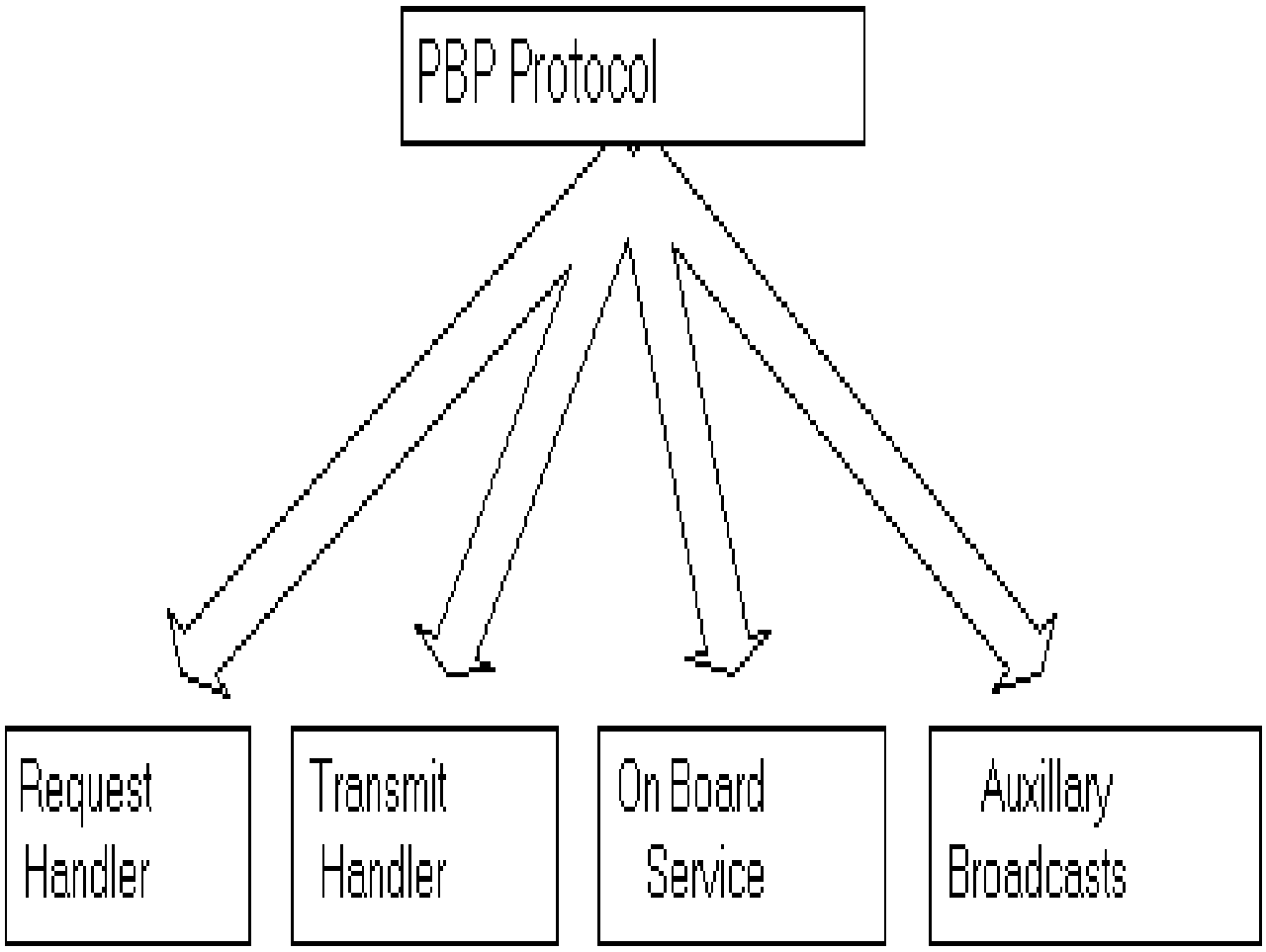

Figure 4.

4. Conclusion

To optimally utilise a low earth orbiting satellite's data storage and retrieval facility one can use the Pacsat Protocol suite, because of the following features: the suite tries to minimise communication by ensuring that data requested by one groundstation is usable to all groundstations in the footprint[3], it provides that files can be uploaded to the satellite in more than one session without resending correctly received data[1]. SUNSAT will be an excellent vehicle for carrying PACSAT. In implementing the protocol suite on SUNSAT, the two factors that dictate the software structure are the scheduling kernel[5] that the software has to run on and the dynamic allocation of memory. Tasks have to be broken down into short pieces to keep the system schedulable.

5. References

[1] HE Price, J Ward, "Pacsat Protocol Suite - An overview", ARRL 9th Computer Networking Conference, pp. 203-206, August 1990.

[2] J Ward, HE Price, "Pacsat Protocol: File Transfer Level 0", ARRL 9th Computer Networking Conference, pp. 209-231, August 1990.

[3] HE Price, J Ward, "Pacsat Broadcast Protocol", ARRL 9th Computer Networking Conference, pp. 232-238, August 1990.

[4] J Ward, HE Price, "Pacsat File Header Definition", ARRL 9th Computer Networking Conference, pp.

245-252, August 1990.

[5] M Ackerman, "The RTX programmer's reference manual", Ver 1.1, Nov 1991.