PROTOCOLS FOR STORE-AND-FORWARD MESSAGE SWITCHING VIA MICROSATELLITES

J. W. Ward
Surrey Satellite Technology Ltd.
Centre for Satellite Engineering Research
University of Surrey, Guildford, Surrey GU2 5XH, UK

H. E. Price
Quadron Service Corporation
133 E. De La Guerra #10
Santa Barbara, CA 93101

Abstract

The authors have developed a suite of protocols specifically optimized for use on store-and-forward microsatellite communications missions. The protocols support networks in which user terminals directly access the store-and-forward satellite to transfer electronic mail. The authors' PACSAT Protocol Suite includes a message format standard, a virtual-circuit directory and file transfer service, and a datagram based point-to-multipoint "broadcast" protocol.

An implementation of this suite has been operating on the UoSAT-3 PACSAT Communications Experiment (PCE) for twelve months and on the AMSAT Microsats for seven months. During this period, hundreds of small and medium-sized ground terminals in all parts of the world have accessed the satellites.

On-board software monitors network activity closely, and usage statistics for UoSAT-3 are gathered regularly at the UoSAT Command Station. Based on this data and the reported experiences of regular system users, we compare the effectiveness of the various protocols. From these comparisons we make some recommendations which are generally applicable to store-and-forward microsatellite missions.

This paper describes the design of the PACSAT Protocol Suite and the UoSAT-3 and Microsat implementations. It summarizes the in-orbit performance results, and concludes with recommendations for future store-and-forward microsatellite missions.

Introduction

The authors have been working on the problem of store and forward communication via small low-earth orbiting satellites within a world-wide heterogeneous user base since December of 1984. Since that time, experience has been gained on the following spacecraft:

UoSAT-2	(UO-11)	Launched March 1984 by Delta 174.
UoSAT-3	(UO-14)	Launched January 1990 by Ariane V35.
PACSAT	(AO-16)	Launched January 1990 by Ariane V35.
LUSAT	(LO-19)	Launched January 1990 by Ariane V35.
UoSAT-5	(UO-22)	Launched July 1991 by Ariane V44.

The UoSAT spacecraft were designed by the University of Surrey, UoSAT-3 and UoSAT-5 were funded through Surrey Satellite Technology Ltd. PACSAT and LUSAT are Microsat spacecraft designed by the Radio Amateur Satellite Corporation (AMSAT-NA), PACSAT was funded by AMSAT-NA, LUSAT was funded by AMSAT-Argentina.

All of these spacecraft are currently on orbit and active. UO-11 contains a rudimentary protocol system. The particular system described in this paper has accumulated 26 orbit-months of use on UoSAT-3, PACSAT, and LUSAT. The most recent user, UoSAT-5, launched just a few days ago, was broadcasting files using the protocols discussed here within 48 hours of launch. Two additional satellites with confirmed launch slots will employ these protocols, and several more are in the planning stages.

In their message passing role, these satellites are generically referred to as PACSATs, for Packet Radio Satellite, and the protocols are called "The PACSAT Protocol Suite".

The target user community is made up of numerous portable and fixed ground terminals without centralized control. These stations make up ever shifting ad hoc groups of similar interest. All stations use NBFM voice-grade RF gear with an Amateur Radio Service standard RF modem called a Terminal Node Controller (TNC). The largest portion the ground terminals are based on IBM PC-class computers running DOS, some use Apple Macintosh or Amiga, a few are UNIX based. There are currently 366 active amateur radio stations that access the satellites directly, and an unknown number of stations that use the facilities of the satellites via gateways and message relays, or that only receive the broadcast data.

Two non-amateur user groups will begin accessing the UoSAT-3 and UoSAT-5 spacecraft (using frequencies allocated in a service other than the Amateur Radio Service) later this year.

As will be discussed below, the unit of information transfer is a "file", wrapped in a standard "envelope". Files contain messages, telemetry, digitized voice, digitized images (uploaded from the ground, or generated with an on-board CCD camera), programs, or anything else commonly found on terrestrial computer networks.

The average file size is less than 9k, although some messages are more than 300k. In many cases, messages are combined and compressed, usually using the popular ZIP program, before being sent to the spacecraft. An early draft of this paper, along with Lotus 1-2-3 spreadsheet data files, was sent from the UK to California with a delay of less than 12 hours, and downloaded in less than one 14 minute pass; the compressed file size was 105,000 bytes, an uncompressed size of 249,000 bytes.

The main body of this paper will discuss the various design goals and constraints which guided the design of the protocols. We will also discuss the implementation methods, the hardware used, and provide some on-orbit results.

Design Drivers

We wanted to provide a space-based store and forward messaging system that fit in well with the existing terrestrial Amateur Radio Service packet networks. The problem was that the ground network is constantly changing, it is an ad hoc collection of OSI Stack and TCP/IP family protocols. It includes large multi-user systems and portable lap-tops. It includes 56kb UHF links and intercontinental 300 baud HF links. Routing protocols and conventions change frequently, and there is a high turn over rate in network routing nodes, hosts, users, and applications. It is, in other words, a standard general purpose wide area net, and not a specialized application.

The type and size of messages sent by the users changes as ground equipment prices drop and sophistication rises. For example, one user community has recently taken to digitizing photos of themselves and placing them on the spacecraft, the average size of these files is 80k.

It was clear that if we were to avoid constant upgrading of the satellite software, and if we wanted to avoid complex schemes for the updating of routing tables for a network node that was moving at 17,000 mph, we would have to do two things:

1) Design the spacecraft software so that there was one generic data entity that it knew how to deal with and deal with well, and

2) Leave all routing decisions to the ground segment.

To these ends, we developed an encapsulation specification. Anything relayed via the satellite would be placed in a file. The file would have sufficient information added to the front to describe the file contents such that consenting ground stations could determine how to deal with the file, where it came from, and where it was going to. The satellite is a mere carrier of the data, and plays no active role in determining where the data is to be sent. It does not handle files differently based on their contents.

The encapsulation is implemented by using a standardized header which is placed at the front of all data sent to the satellite. The only function the satellite must perform is receiving files, allowing files to be downloaded on request, and allowing ground stations to search the headers to determine which, if any, files it wants to download. Files are given unique identifiers by the satellite, which are used in all transactions. Any other identifier of interest to the ground stations is placed in the header. The header specification is discussed in detail later in this paper.

We would also have to make a concession to the major difference between a LEO satellite and a terrestrial network resource. A LEO satellite is accessible for a short period of time several times a day. The system must allow for a file upload or download to be terminated at any point in the process, and the process must be able to be continued from that point without requiring retransmission of data heard once before. While protocols such as TCP/IP would allow a session to remain open until the next time the satellite was in view, keeping session state for a large number of users for an indeterminate length of time was thought to be impractical. The only state we elected to maintain is file state, that is, is the file complete, and if not, what was the last byte received. Ground stations preserve similar state information for files being downloaded.

We also chose to take advantage of another attribute of LEO spacecraft -- they are inherently a broadcast medium. There are many potential users of the spacecraft in view at any one time, and these users may be interested in accessing the same files. A point to multipoint facility is then in order.

Finally, we designed with the assumption that the minimum ground terminal would have at least an IBM-PC class computer providing intelligence. In our network, the ground stations "poll" the satellite. The same hardware and software support systems could also support another model, where the satellite polls ground stations, such as small weather stations, data gathering buoys, etc., but we chose to interface with the existing network as described.

Our store-and-forward message passing suite of protocols contains three major elements:

1) The encapsulation protocol (PACSAT File Header)

2) The point to multipoint protocol for downloading (Broadcast Protocol)

3) The point to point protocol used for uploading, downloading, and for getting lists of files. (FTL0 protocol)

Protocols

The protocols are summarized in this section. The protocols are completely described in [1]. In the description of these protocols, "server" refers to the process in the satellite that implements the store-and-forward facility, "client" refers to the ground station process that uses this facility. Groundstation software provides the human interface to the network. Ground terminals communicate with the satellite using computer-to-computer protocols.

The Amateur Radio Service standard link layer protocol AX.25 [2] is used as the low level data transmission standard. The protocols described here function above the AX.25 protocol. The AX.25 sliding-window ARQ "virtual circuit" mode is used to provide error-free, correctly-sequenced data streams. The AX.25 UI frame is used as a standard datagram format. Other protocols providing virtual circuits and datagrams could also be used below the PACSAT Protocol Suite, but the existing user base uses AX.25 as the lowest common denominator.

Above AX.25 the Suite diverges into two distinct protocols: the PACSAT Broadcast Protocol (PBP) employs datagrams to transfer messages from the satellite to the groundstations, correcting errors by selective-repeat ARQ. The File Transfer Level 0 (FTL0) Protocol uses virtual circuits to provide a message directory search service, message uploading and message downloading. Both of these protocols are implemented using a client-server model; the satellite computer executes a server process, which acts upon requests made by the client process running on the groundstation computer.

Above these transfer protocols, a common message format -- a protocol for standard message encapsulation -- unites PBP and FTL0. The encapsulation protocol is called the PACSAT File Header (PFH). The PFH is the protocol header for the entire satellite-based store-and-forward network; it is added when the originator places data into the network and removed before the data is finally delivered to the destination.

The following figures illustrate the PACSAT Protocol Suite hierarchy, which is clarified in the following sections.

Point to Point

Higher layer ▲

PFH Encapsulation Protocol

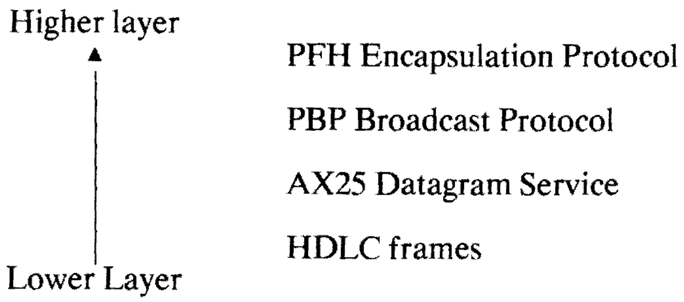
FTL0 File Transfer Protocol

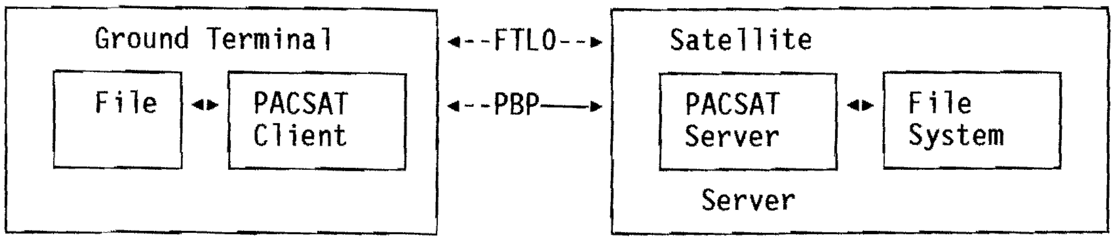
AX25 Virtual Circuit Service

HDLC frames

Lower Layer

Point to Multi-Point



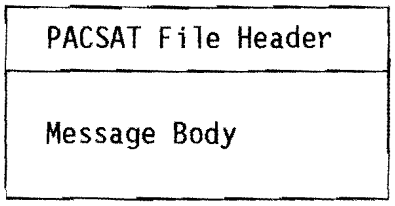


PACSAT File Header Specification.

The store-and-forward satellite exists to transfer messages from one ground terminal to one or more other terminals. End-to-end message transfer is implemented by a number of software processes in ground terminals and on the satellite. In order for these processes to exchange messages, there must be a definition of what constitutes a valid message.

The PACSAT File header provides a standard transfer unit for all information passed though the satellite. Except for realtime engineering telemetry, all data generated on-board by these satellites is also stored using this specification; this includes stored telemetry, data generated by on-board experiments such as charged particle detectors and dosimeters, and images generated by the UoSAT-5 CCD camera.

Each file sent to the satellite has a prepended PFH. As sent, stored, and later downloaded, a file looks like this:



The message *body* is what one user wishes to transfer to other users. The PACSAT Protocol Suite will transparently relay any binary message body.

The header contains fields which define the contents of the file, and provide information on addressing and routing. The syntax unambiguously separates the PFH from the message body and permits significant future expansion of the number of header items -- both on a system-wide basis and by any user group.

The PFH offers the following features:

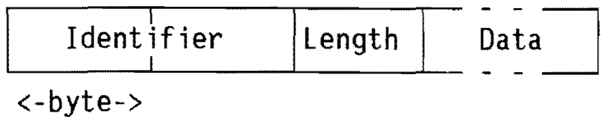
- Places no restrictions on the content or format of the message body.
- Provides for a large number header fields.
- Provides a structure that can be easily searched by server protocols to select files of interest based on the contents of the header.

The PFH syntax is generic, and equally suitable for satellite and terrestrial applications; the currently defined header items clearly reflect the satellite-based nature of the network.

PFH Item Syntax

PACSAT File Header Items employ the full range of 8-bit binary characters. The ground-station software converts between a human-interpretable representation of the header, and the more efficient PFH syntax.

The PFH Item syntax is illustrated below



PFH Item Syntax

Identifier The Identifier is a 16 bit integer, providing a range of 65536 possible different PFH Items.

Length. Length is an 8-bit integer, which tells how many data bytes there are in the Item. Items longer than 256 bytes can be spread over a sequence of Identifiers.

Data. The Data field is the information content of the Item.

The PACSAT Header Specification strives to separate the mechanics of storing the file on the server from the actual form of the data on the client. For example, the server is constrained to use the file naming conventions of the underlying operating system. To avoid conflicts between a client system's file name conventions and the server's conventions, and also to avoid duplication of names among clients and among other on-board users of the satellite file system, the satellite file server assigns a unique file number to each file. The actual name of the file as view by the client is kept in a different Item.

PACSAT File Header Item Definitions

Identifier	Interpretation	Required Field
1	file serial number assigned by satellite	required
2	file name in server file system	required
3	file name extension in server file system	required
4	file size, including PFH and body	required
11	length of PFH, in bytes	required
8	message body type	required
7	single-event upset severity flag	required
9	additive checksum of body bytes	required
10	additive checksum of header bytes	required
5	time of file creation	required
6	time of last file modification	required
18	time at which upload was completed	required
22	time at which message was last downloaded	
23	time at which message becomes outdated	
17	AX.25 link address of message uploader	
21	AX.25 link address of message downloader	
19	number of message downloads	
16	network address of message source	
20	network address of message destination	
24	priority of the message	
33	message title	
34	key words describing the message	
37	File name from message originator's file system	
25	compression applied to message body	
36	ASCII description of non-standard compression technique applied to body	
32	message ``type'' for PBBS gateway messages	
35	ASCII description of non-standard body	
0	last item in PFH	required

Other features of the header fields:

- File Tracing -- timestamps track a file's movement through the system.
- Message Addressing -- the PFH allows user addresses to be anything up to 256 bytes long, with no constraint on the address interpretation or presentation. The FTL0 Protocol's Directory Search capability can be used by groundstation clients to search for messages regardless of the address format chosen.
- Message Identification -- files can carry a title, key words, and other descriptors defining the content and format of a file.

The PFH provides a well-defined method for joining an arbitrary binary message body to standard envelope and header items. The standard syntax provides both uniformity and flexibility, and the system-wide item definitions are specifically tailored to the store-and-forward satellite environment.

The FTL0 Protocol

"File Transfer Level 0", or "FTL0" is the name of the PACSAT protocol which uses the AX.25 virtual circuit mode.[1] Clients use FTL0 to upload and download messages, and to obtain directories of messages on the server. These activities embody store-and-forward communications. FTL0 plays the central role of data transfer in the PACSAT Protocol Suite.

There are many existing protocols for computer file transfer. Some of these (e.g. Kermit, FTP) are suitable for connecting heterogeneous networks of microcomputers, and it might have been possible to adopt one of these protocols for the satellite store-and-forward network. Proponents of this approach argued that it would save the effort otherwise spent on protocol specification and groundstation software implementation.

The characteristics of the LEO satellite communications link and the particular set of services required from the store-and-forward server made existing protocols unsuitable. The server should provide simultaneous file uploading and downloading to take advantage of limited pass time, and also allow a flexible directory search facility. The end of a satellite pass, though predictable, can interrupt communications during any phase of a client/server transaction. Interrupted transactions must be continued and not simply abandoned. Furthermore, these features should not consume unreasonable amounts of memory or computing time in the server or client computers. No existing protocol was seen to possess all of these traits.

FTL0 is the result of a custom protocol design. The services provided are those specifically required by a satellite store and forward mission with the design goals as previously discussed. The client/server transactions can be interrupted at any phase. Interrupted transactions are continued during later satellite passes, and consume no memory while suspended.

FTL0 uses the AX.25 connected-mode virtual circuit to provide an error-free communications channel between the ground terminal client and the satellite server. Adhering to the layered protocol model, the FTL0 processes view the channel as an ordered byte stream, with no visible link-layer frame boundaries. The downlink stream is logically separate from the uplink stream, and the two can operate simultaneously.

FTL0 offers the following features:

- Simultaneous bi-directional transfer over a single virtual circuit.
- Recovers from Loss of Signal at any point in a transfer with no data loss. The server is responsible for upload continue offsets, the client is responsible for download continue offsets.
- Directory Search service. Allows for arbitrary select criteria, search for files with headers the meet the criteria, and download of selected headers or files.

The FTL0 protocol contains very little overhead. A client or server using 256-byte FTL0 information packets experiences 0.8% overhead on a file transfer. FTL0 relies on the

[1] The somewhat confusing "Level 0" portion of this name was added when several levels of service were under consideration. It does not refer to any OSI level or layer.

AX.25 link layer protocol to provide protection from data corruption and notification of any link failures.

Using FTL0

To initiate communications using FTL0, a link-layer virtual circuit is established between the client and the server. The client then initiates any uploads, downloads, directory searches or message directory transfers which it requires. The client or the server may terminate the virtual circuit upon encountering a protocol error or completing the desired activities.

Directory Search Service

The FTL0 directory search service identifies a subset of the messages on the server which are of interest to a particular client. A network devoted wholly to person-to-person messages would require only a vestigial directory search service to scan the destination Items in the PFH for the current client's address. Clients in a more open network -- e.g. a bulletin board or conferencing network -- require more powerful directory searches. Users may wish to identify messages about a certain subject, messages uploaded after a particular time, etc. Since the PACSAT Protocols serve the needs of many types of users, they include a directory search facility which permits the client to select messages using search equations involving the contents of any set of PFH Items.

Following the design goal that the satellite not "know" anything about the meaning of the contents of the header, the directory search specifies a *selection equation* which specifies PHF Item numbers, constants, and comparison operators. For example, a user may want "all messages stored since 0930 on the 15th of July which have the word "measles" in their Title or Keyword fields, but not messages over 200,000 bytes long". This translates to a symbolic Boolean equation such as:

(UPLOAD_TIME > 15/7/91 09:30:00) AND ((KEYWORD="*measles*") OR (TITLE="*measles*")) AND (FILE_LENGTH < 200000)

Each *term* of this equation contains four elements: a PFH Item identifier, a relational operator, a comparison type (numeric or string), and a constant value. The flexibility of the PFH Item syntax described above now becomes apparent, files can be placed in the satellite file system, and without any explicit knowledge of field definitions, the satellite software can select files for download based on a arbitrarily complex selection criteria. The knowledge of the field types and formats, complex parsers and human interfaces, and even different National Language conversions are kept on the ground.

The ground client software translates a selection request to postfix notation, the above equation is presented to the satellite as:

```
numeric >
UPLOAD_TIME 15/7/91 09:30:00
string =
KEYWORD "*measles*"
string =
TITLE "*measles*"
OR
AND
numeric <
FILE_LENGTH 200000
AND
```

The server processes the equation and determines if the message being evaluated meets the client's selection criteria. By evaluating the selection equation for every message in its file system, the server builds a list of *selected messages* for which the equation is true. The client can then request the satellite to send a directory listing of the selected messages or to begin downloading them.

PACSAT Broadcast Protocol

The downlink of a store-and-forward satellite is a broadcast channel, in the sense that all ground terminals in the footprint can receive all transmitted frames (aside from local reception errors). This characteristic can be exploited when a message on the satellite is to be delivered to more than one ground terminal in the same footprint. To make the most efficient use of the downlink bandwidth, a message should be transmitted only once for all ground terminals which need that message, not once for each terminal. This is particularly useful in electronic conferencing and news dissemination, where most messages will be of interest to more than one client. The PACSAT Broadcast Protocol (PBP) was designed to provide this service.

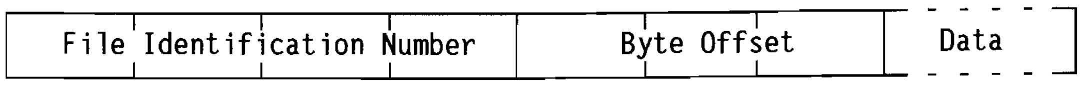
PBP is a selective repeat, automatic repeat request (SR-ARQ) protocol which uses AX.25 datagrams instead of virtual circuits. The protocol consists of a packet format and a number of procedures. The packet format allows each information packet transmitted by the satellite to be used by all clients in the footprint. The PBP procedures allow stations to initiate broadcasts and send selective repeat requests. PBP does not rely on the lower layer protocol to provide data integrity, PBP provides its own SR-ARQ facility for replacing lost datagrams.

PBP is a synthesis of customization and backward compatibility. By using the AX.25 datagram protocol PBP remains compatible with the existing Amateur Radio Service installed base of ground terminal AX.25 TNCs. By moving the ARQ process into a higher layer protocol, the ARQ can be modified to suit the LEO satellite downlink. The protocol fully exploits the broadcast nature of the downlink without requiring additional equipment in the ground terminals.

The PBP Datagram

Generally, messages (files) will not fit into a single datagram, but will have to be broken into several. If each message fragment transmitted by the PBP server is to be useful in isolation, each must indicate what message it is part of, and where in the message the data should be placed. The main elements of the PBP datagram format are illustrated below:

Simplified PBP Server Datagram Format



The File Identification Number is the same number used by the PFH and FTL0 protocols. Byte Offset indicates where in the file the Data belongs.

Clients transmit their requests to the PBP server using another datagram format. This datagram contains the File Identification Number, a block size telling the server how many data bytes to put in each datagram, and one of the following commands:

Start broadcast transmission
End broadcast transmission
Selective repeat request
Start permanent broadcast

Retransmission Requests

Selective repeat ARQ, in which the transmitting station retransmits only those data frames actually missed by the receiving station, is the most efficient form of pure ARQ. Unlike sliding window schemes, no correctly-received frames are wasted. Unlike stop-and-wait ARQ, the transmitting station can send data at full link speed. Experience with SR-ARQ on UoSAT-2 showed that it was very efficient for file transfer to and from LEO satellites, where a combination of short error bursts and long fades plagued other schemes.

Our earliest implementation of a satellite store and forward protocol, the UoSAT-2 MSG2 protocol [3], used fixed length datagrams and bitmaps for retransmission requests. As the management of bitmaps is cumbersome with variable size PBP datagrams, we now use a hole list concept. A hole is defined by a starting byte number and a length describing a gap in the received file. A hole list can be viewed as a table of hole locations and sizes.

PBP Procedures

PBP procedures are less strictly defined than those for a go-back-n or stop-and-wait ARQ protocol. The server responds only to requests from the client; there is no stream of acknowledgments from client to server or timeout timer after which the server retransmits outstanding packets. Every datagram transmitted by the server has been requested by one client or another, so little downlink bandwidth is wasted.

Once a client has identified a message which it requires (using the FTL0 directory service), it sends a broadcast request to the server. This request includes a suggested number of data bytes for each datagram, which allows the client to tailor the broadcast to prevailing link quality.

The PBP client will receive some or all of the datagrams from its request. Upon receiving these (or other) datagrams, the client places them at the indicated byte offset in the appropriate file. The most efficient client software will capture all datagrams on the downlink, even if they are not for a file specifically requested. The stored datagrams can then be used to build files which the user may desire later. Thus, the client gradually builds messages -- some complete, and some with missing datagrams.

When a client wishes to have the server retransmit missing datagrams, the client composes and transmits a repeat request datagram. If some of the desired datagrams are not received, then the client sends the request again. The client repeats this process until the message is complete.

Receive-only ground terminals can also gather messages from the satellite downlink using PBP. Such simple clients cannot request repeats or specific messages, but they will receive complete messages as a consequence of other clients' activity. The server can also transmit important news bulletins repeatedly to increase the chances of reception by receive-only stations.

PBP brings the network highly efficient ARQ downlinking. It eliminates the waste inherent in other ARQ schemes, and makes each transmitted frame potentially useful for all clients. It is also takes up little memory in the server for procedures or data structures. The level-two protocol is virtually null, and the PBP itself requires only data structures to store current hole lists. In contrast, an FTL0 transaction requires a complete AX.25 level-two implementation, plus data structures to accumulate FTL0 packets which may be spread over several AX.25 frames.

Implementation Details

The PACSAT Protocols require both server (satellite) software and client (ground) software. One of us (Ward) has implemented the server on UoSAT-3 and UoSAT-5 satellites, and the other (Price) has ported this software to the AO-16 and LO-19 satellites. Ward has also written client software for the IBM-PC family of computers. Other implementations of the client software have been written by independent third parties, these run on Apple Macintosh and UNIX systems, and well as IBM PC class systems.

There are many design choices in the server that are not unduly constrained by the protocols themselves. The various limitations imposed by the target hardware (the UoSAT OBC or the Microsat CPU) greatly affect the implementation of the server software. These include the total number of directory entries available in the satellite file system, the number of simultaneous connections permitted, the number of simultaneous broadcast files, the depth of the stack for FTL0 Selection Equations, the processing power available for directory searches, checksumming, Single Event Upset corrections, etc.

Security

The basic FTL0 protocol includes no explicit data security mechanisms. There is no user authentication system and no message encryption. In a fast changing network using non-secure radio uplinks and a broadcast downlink, implementation of complete security measures would have significantly increased the complexity the protocol specification, server and client implementation, and network administration.

Ultimately, data security is the responsibility of the user, just as message routing and data compression is. Since the PACSAT server is a transparent data channel, end users are free to devise end-to-end encryption schemes which suit their needs.[2] Only the PFH must be left unencrypted. Thus, the PACSAT Suite does not force a particular level of security on the user. Since many user groups are not communicating sensitive information, and others might require very tight security, this solution seemed best for a general-purpose network.[3]

A secure handshake is supplied for the spacecraft command and control functions and for some server functions (file delete, for example), but this is not a function of the file transfer features of the spacecraft, and is not "file security".

The secure handshake is based on a method described by Newland [4] for verifying user identity in a "party-line" or broadcast environment. The server supplies a password which must be properly encrypted and returned with the next command, authorized clients will have a password and an encryption algorithm not available to other clients. This protects against direct access, and also against attack by retransmission of a previous command, which is no longer valid because the string to be encrypted and returned changes with each transaction.

The lack of a general user authentication service has an adverse effect on the management of file space, although the resulting code simplification more than makes up for this. Because it implements no user authentication, the FTL0 server cannot provide conclusive confirmation that a message has reached its intended destination. The client to client notification task is appropriately and easily handled above FTL0. When a message reaches its destination, the destination station composes a very short return message (perhaps just PFH with no body), which verifies delivery. Of course, to be certain, the ground stations must have their own authentication scheme. Without the satellite participating in this scheme however, messages can never be safely deleted based on delivery. The servers always delete files when file space is needed, oldest file first. Clients can never request file deletion.

File security, User Authentication, and similar topics have been given a great deal of thought in study projects for commercial use of the PACSAT Protocols, but these are beyond the scope of this paper.

Satellite CPU Hardware.

The current server implementation runs on two different hardware platforms (by virtue of a common operating system). The basics of each system are:

Name	CPU Type	I/O ports	Program Space	File Space
UoSAT-3	8 MHz 80c186	2 @ 9600bps	256k	4MB
UoSAT-5	8 MHz 80c186	2 @ 9600bps	512k	13MB
Microsat	4 MHz NEC V40	6 @ 1200pbs	256k	8MB

[2] Amateur Radio Service users are, in fact, constrained not to encrypt their messages!
[3] For many purposes, shareware programs such as PKZIP.EXE can provide both encryption and data compression at no additional "expense" to the user. A data file ready for transmission would be passed through the encryption utility, have a PFH added and then be uploaded. When received by the destination, the PFH would be stripped off and the file decrypted. These operations can be automated using the PC's batch processing commands.

Full descriptions of the UoSAT CPU hardware can be found in [5], Microsat in [6].

Satellite Operating System.

Concurrent with the development of more sophisticated data handling protocols than had previously flown on amateur spacecraft came a requirement for a more sophisticated operating system and associated ground simulator environment. Previous digital missions had relied on 1802 and Z-80 microprocessors with small program memories. All software was coded in assembler. Ground support was minimal, with generally unsophisticated development environments. Better ground equipment may have existed, but was in general beyond the budget of the low cost missions.

The latest satellites used microprocessors that were instruction compatible with the Intel 80xx family used in IBM PCs. One of us (Price) is a principal at Quadron Service Corporation where he and others develop a real time operating system for this class of device. This operating system, called qCF, and the associated development tools allow programs to be written using Microsoft C; a well known language and compiler. In addition, the qCF system allows for several separate tasks to run in a multi-tasking environment, which lends itself well to the independent development of different aspects of the total spacecraft software load. Through a memorandum of understanding between Surrey Satellite Technology and Quadron, the qCF operating system was tailored to the UoSAT-3 PCE (as well as to the AMSAT-NA V-40 onboard computer).

The UoSAT-5 spacecraft carries the following major software programs, each running as an separate task (though several exchange data using qCF message passing facilities):

FTL0 -- file transfer protocol server.

HIT -- Telemetry task, reads data from the telemetry system, stores for retrieval via PBP. For historical reasons, this module also contains the PBP server.

TDE -- reads data from the total radiation dose experiment

SSTE -- reads data from the advanced solar cell technology experiment

ADCS -- Attitude control system, reads data from navigation magnetometers and controls magnetorquing. Uses the standard Microsoft C floating point library.

CCD -- communicates with the transputers in the CCD camera module over a 9600 bps serial bus. Gathers processed image data from this module and stores it for retrieval via the PBP.

MFILE -- uses 13MB mass memory to emulate a RAM disk, supplies file system services to all other tasks.

qAX25 -- provide link layer services for all other tasks.

Using a commercially available 80186 co-processor adapter card in the PC bus, the spacecraft CPU can be simulated with reasonably high fidelity. The several different student and staff researchers that developed the UoSAT-5 software could debug using off-the-shelve software and hardware. Including the cost of an IBM PC clone, the adapter card, and all required software, an entire UoSAT CPU simulator/development work station can be purchased for less than $7000, matching the overall budget of a lightsat project. Programmers skilled in the C language and IBM DOS are also more readily available than those skilled in 1802 assembler.

In Orbit Experience

Any operational public access bulletin board system will generate copious amounts of usage statistics. We have just begun to sift through the first years worth of data. Here are some numbers which serve to illustrate the utility and level of use these systems are attaining. Most of the figures are based on the UoSAT-3 spacecraft. As this satellite is controlled from a university environment, more man power is available (in theory) to at least acquire and catalog the data.

Factoids

UoSAT-3 statistics for the most recent 100 days (1/3 of the operational time) show:

> 177 individual stations active at least once (calls heard)
>
> 119 stations regularly active (calls heard in the past 2 weeks)
>
> 23 countries are active, with specific concentrations in Europe, Japan, USA, Australia and New Zealand.
>
> 9.7 Mbytes were uploaded.
>
> 55 Mbytes were downloaded with FTL0.
>
> 32 Mbytes of directories were downloaded.
>
> 3.3 Mbytes of data were generated by the Charged Particle Experiment and downloaded using the PBP. [7][8]
>
> 4 Mbytes of data were generated by the server activity log and downloaded using PBP.

Unfortunately, data on broadcast requests is not currently logged.

Differences between UoSAT and Microsat Usage.

The Microsats (AO-16 and LO-19) and UoSAT-3 offer identical user interfaces and facilities, the only difference is in the baud rate and modulation format. Microsats use PSK at 1200 bps, UoSAT-3 uses FSK at 9600 bps. It is somewhat easier to interface the Microsat modem with a radio, as it connects via the standard microphone and speaker connections. The UoSAT modem, which requires slightly more bandwidth and a flatter response curve than is provided by the audio ports, requires opening the radio and tapping into the modulator and demodulator stages. Otherwise, exactly the same radios, antennas, computers, and software can be used on both types of spacecraft. All three spacecraft are in the same orbit, and have similar transmitter power levels. Both types of ground station modems are available for about the same price ($120), in kit form, or assembled and tested from commercial sources.

The perceived difference in difficultly of interfacing to the radio by the user community is shown by the difference in the number of users, 177 for UoSAT and 290 for Microsat, 101 users are active on both.

UoSAT users receive a better level of service, if service is measured by total bytes received, or by the length of time it takes to receive a file of a given length. This is intuitive, based on the 8:1 difference in data rates. Both types of spacecraft offer very low bit error rates over an entire pass to even minimally equipped ground stations.

One difference in usage patterns caused by the baud rate differential can be seen by comparing the average file size uploaded to each type of spacecraft.

Satellite	Average Upload File Size	Average Uploads per day
UoSAT-3	7500	26
Microsat	2800	19

The user communities adjust to the level of service offered by the satellite and expand to fill the available resources, as is usual in the computer industry. As stated earlier, the UoSAT-3 users are currently uploading digitized images of themselves, an activity that takes a few minutes at 9600 baud, but requires more than one pass under normal loads for Microsat.

Effect of Features on the User Community.

Graph 1 shows the number of upload, download, and directory select transactions per day over the same time period of time as the above data. The number of broadcast download requests are not logged. Various server features have been implemented in a phased manner, the hole fill feature of the server was not implemented until mid-May. As can be seen from Graph 1, a smaller number of files were downloaded per day starting in May, even though the number of uploads and directory searches increased after that point.

We infer from this that users perceive a benefit to using PBP over FTL0 to download files. We have also noticed peer pressure being applied as some users view FTL0 downloads as "wasted" time, since the data can only be used by the FTL0 downloader. The activity log files are visible to any user, and the identity of stations using the FTL0 download command is revealed. FTL0 download users have been receiving messages from other users pointing out the total network efficiencies to be gained from preferential use of the PBP for most downloading.

The PBP download protocol has been shown to be more efficient than FTL0 downloading in two respects. First, with the low bit error rate afforded by the three to five watt transmitters on the spacecraft, an entire pass can be copied with almost no data dropouts. Since the AX.25 sliding-window ARQ protocol uplinks acknowledgment frames (at least one per seven downlink frames), there is always some activity on the uplink, with the possibility that the satellite will miss an acknowledgment and be forced to stop and wait, and then retransmit some data which had already been properly received. In a perfect PBP downlink, only one frame is uplinked -- the original begin request.

Second, multiple ground stations can be served by the multipoint aspect of the broadcast protocol. In practice, the ground station at the University of Surrey almost never needs to transmit a "begin broadcast" request. It is able to eavesdrop on broadcast requests from other stations in the UK and Europe, and receives most of the files that way. This also shows the efficacy of receive-only ground stations.

Uptime

While the developers view the satellite systems as research tools and test beds, the users view them as public utilities expected to be present 100% of the time. Over the last year, UoSAT-3 has been available 93% of the time, over the last six months the figure was 97%.

FINAL CONCLUSIONS

The protocols are working, and we have a network of meaningful size communicating meaningful amounts of data over inexpensive satellites, using inexpensive groundstation equipment.

Some user terminals are using omnidirectional antennas with near horizon to horizon coverage. The most important limiting factor in access is not the link margins, but uplink interference. This can be from stations in the network or from stations which just happen to be on the frequency. This much is clear: while it is possible to build a satellite sensitive enough to hear a few hundred milliwatts EIRP, if the network isn't on a clear channel it can't function efficiently. Differing frequency allocations and levels of adherence to International Law throughout the world make this a difficult problem. For example, the UoSAT-3 spacecraft can be placed in a RX to TX audio transfer mode, and it becomes an effective repeater of Spanish Taxi Cab dispatchers.

The protocols have proven themselves in several ways:

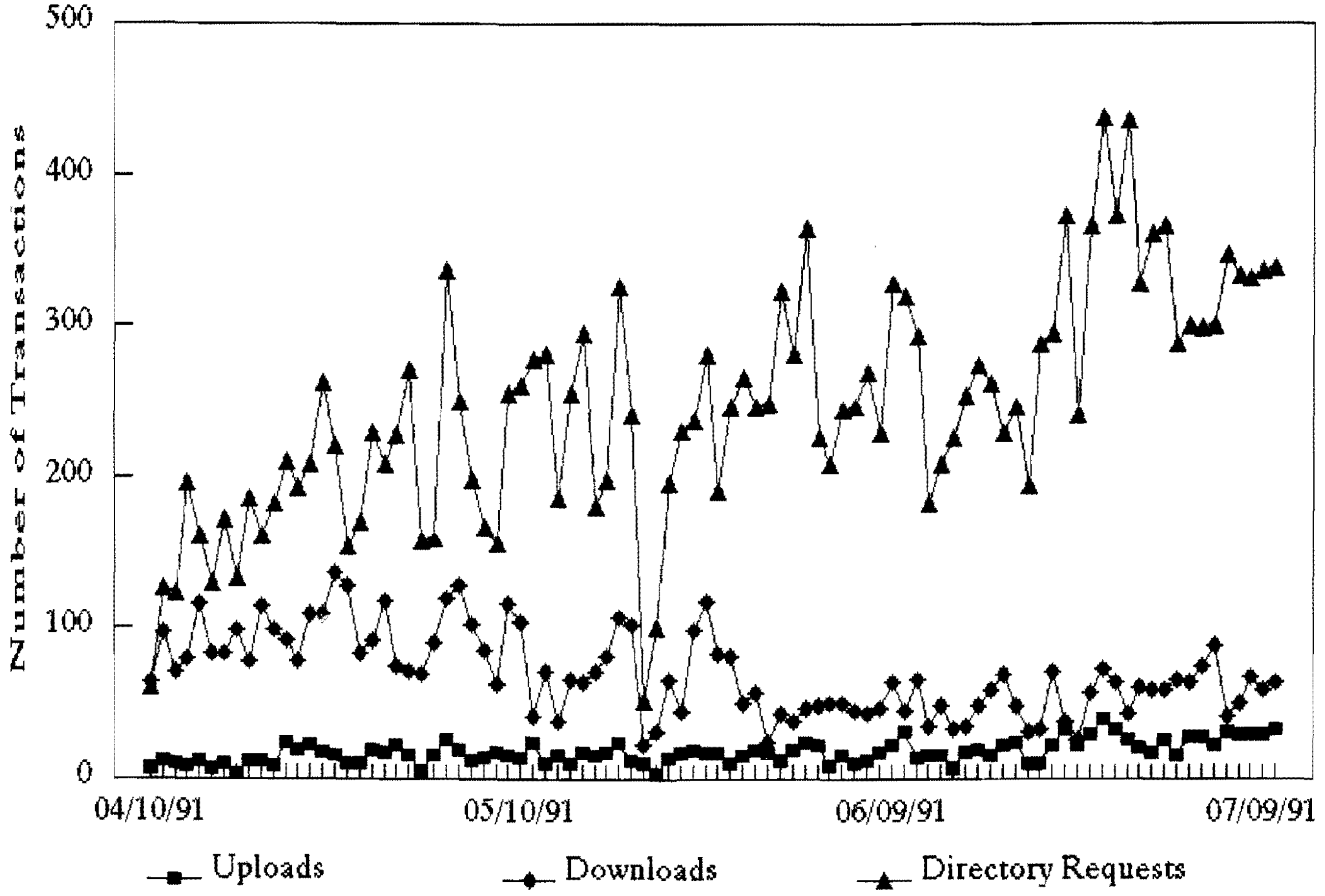
- They are relatively easy to implement (3 full implementations of FTL0 now available).
- They are flexible: the network has been used for image transfer, database transfers, simple text messaging, sending computer programs, relaying terrestrial BBS mail, all with one set of protocols.
- The broadcast protocol has been shown to be effective in allowing multipoint delivery of data, as well as being an efficient transport mechanism for a single station. It is also simpler to implement than the FTL0 protocol.

In the near future, we will perform more monitoring to further characterize the network.

The upload, download, and directory statistics have shown what we feel is a disproportionate amount of directory download activity. A large part of the current user community desires to perform eye-ball semantic analysis on the titles of files to determine what they want to download, resulting in "show me all" selection equations. To address this need, we will enhance the broadcast protocol to include the broadcast of file headers, thus removing the need for most stations to request individual directory downloads, and freeing up more downlink time.

Activities Per Day

UoSAT-3



In the coming year, two non-amateur networks will be commissioned. VITA and SATELLIFE will be bringing field stations on line, mostly in Africa. VITA will share UoSAT-3 with the amateurs on a roughly equal footing by demand-based satellite transmitter frequency switching. SatelLife, who funded 60% of the UoSAT-5 mission, will get 60% of orbit availability on a stricter rotation, the transmitter frequency will be switched by a clock. These two non-amateur networks will probably point up the need for some modifications to groundstation software, and refinement of groundstation software will become the primary evolutionary path for the network.

UoSAT-5's use of the broadcast protocol for transmission of large (350k) image files has already pointed out modifications necessary to the broadcast server, primarily in how it manages hole fill requests and schedules multiple transmission requests on the downlink.

UoSAT-5's being brought on-line in less than one week after launch was largely due to the existing base of software and the flexibility of the development tools and communications protocols. The commissioning process involved collecting initial data from several on-board experiments, as well as allowing autonomous on-board processes to stabilize the spacecraft, establish a constant spin rate, and attain and maintain a gravity gradient earth-pointing attitude.

We feel a good base has been laid for continued study into the practical applications of small low earth orbit store and forward satellites.

References

[1] Harold E. Price and Jeff Ward, **PACSAT Protocol Suite, PACSAT Data Specification Standards, PACSAT Protocol: File Transfer Level 0, PACSAT Broadcast Protocol, PACSAT File Header Definition**, Proc. 9th ARRL computer Networking Conference, London, Ontario Canada, 1990

[2] T. Fox, **AX.25 Amateur Packet-Radio Link Layer Protocol**, American Radio Relay League, Newington CT, USA, 1984.

[3] J. W. Ward and H. E. Price, **UoSAT-2 Digital Communications Experiment**, Journal of the Institution of Electronic and Radio Engineers, Vol 57, No 5 (Supplement), pp. S163-S173, September/October 1987.

[4] P. Newland, **A Few Thoughts on User Verification Within a Party-Line Network**, Proc 4th ARRL Computer Networking Conference, San Francisco, March 1985.

[5] J. W. Ward, **Store-and-Forward Message Relay Using Microsatellites: The UoSAT-3 PACSAT Communications Payload**, Proc. 4th AIAA/USU Conference on Small Satellites Vol I, Logan, Utah, August 1990.

[6] J. A. King, R. McGwier, H. Price, and F. White, **The In-Orbit Performance of Four Microsat Spacecraft**, Proc. 4th AIAA/USU Conference on Small Satellites Vol I, Logan, Utah, August 1990.

[7] Craig I. Underwood, **In-Orbit Radiation Effects Monitoring on the UoSAT Satellites**, Proc. 4th AIAA/USU Conference on Small Satellites Vol II, Logan, Utah, August 1990.

[8] Craig I. Underwood, et. al., **Space Science and Micro Satellites -- A Case Study: Observation of the Near Earth Radiation Environment using the Cosmic Ray Effects and Dosimeter Payload (CREDO) on UoSAT-3**, submitted for publication in the 5th AIAA/USU Conference on Small Satellites, Logan, Utah, August 1991.