A TERMINAL CONCENTRATOR BASED ON THE SM4-20 COMPUTER.

Walter Kunft
Yuri Plotnikov
Peter Pronay

February 1983
WP-83-44

CONTENTS

A TERMINAL CONCENTRATOR BASED ON THE SM4-20 COMPUTER.

Walter Kunft, Yuri Plotnikov and Peter Pronay

1. The place of the SM 4-20 in the IIASA node.

The SM4-20, a minicomputer made in Czechoslovakia, has been installed in IIASA to serve as a communication machine. The aim was to enhance the capabilities of the IIASA node, which is still based on TPA-70, a Hungarian minicomputer, by upgrading the number of input ports of the node and by sharing part of its communication load. At present, TPA is well equipped with fast DMA synchronous bit-stuffing adaptors making it well suitable for handling more packet-switching lines, but it isn't so well furnished with asynchronous interfaces. In contrary to that, SM4-20 has enough asynchronous ports (4 single line interfaces and 2 x 8 channels on multiplexors). Thus, it is logical to attach the asynchronous connections to SM4 and to relieve in this way more capacity of the TPA for external packet- and circuit- (DATEX-300) switching lines. This goal can be achieved by equipping the SM4 at IIASA with communication software capable of handling

- the asynchronous terminal lines (1st step)
- the asynchronous host lines (2nd step).

The first step has already been made during the last year, when a new software package has been developed and installed on SM 4. It allows the SM 4 to serve as a network terminal concentrator, concentrating asynchronous terminals to an X.25 DTE-DCE interface. The procedures performed at the interface between SM-4 and the supported terminals conform to the CCITT recommendation X.28 . The parameters characterizing the behaviour of the concentrator towards the terminals have been designed with respect to the recommendation X.3 . So, via a fast packet switched line more (8-16) local and remote terminals, including for example those, which are currently hooked directly to TPA-70 can access the network, represented in IIASA case by the TPA gateway system (fig. 1). Among these can be the terminal lines coming from the VAX and the PDP, to enable IIASA users to call external facilities. At the present time there is one such line established from each of the IIASA hosts. This number of lines could easily become not satisfactory as the demand for external computation and communication would grow. In the presently finished implementation the SM-4 based concentrator is not capable of handling incoming calls. That is why only terminal lines can be supported. However,implementation of Incoming Call packets' support is envisaged (2nd step) to be finished soon, together with the necessary routing modules. Simultaneously, support of so called "Host-PAD" functions will

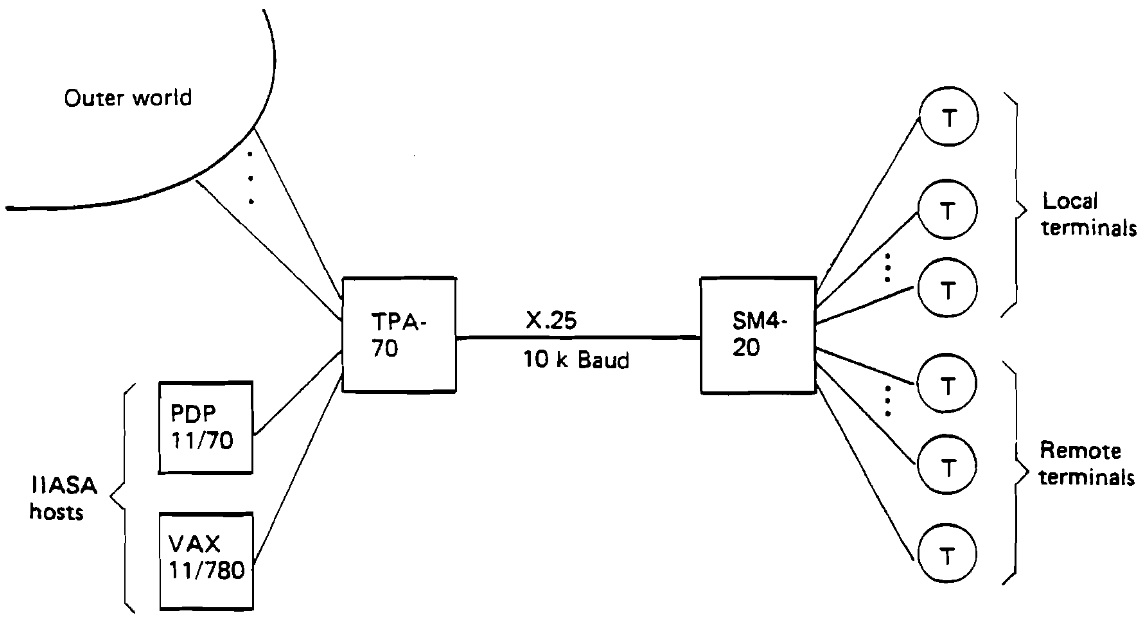

Fig. 1

be incorporated. The latter will enable to connect asynchronous lines from hosts to some of the concentrator ports. In this way are VAX, PDP and the Moscow NORD presently connected to the TPA-70. Summarizing, we can expect fig. 2 to reflect the capabilities of the SM-4 after finishing the present development.

The above mentioned software permits a few alternatives on the top of it. One is the linking of TPA-70 and SM-4 in symmetrical tandem arrangement in which each machine would act as a "hot backup" for the another one. It would, however, require that SM-4 be equipped with functionally equivalent software to that of the TPA-70. In addition some (probably HW+SW) mechanism would have to monitor the state of both machines to assure rerouting of the respective part of the traffic in case

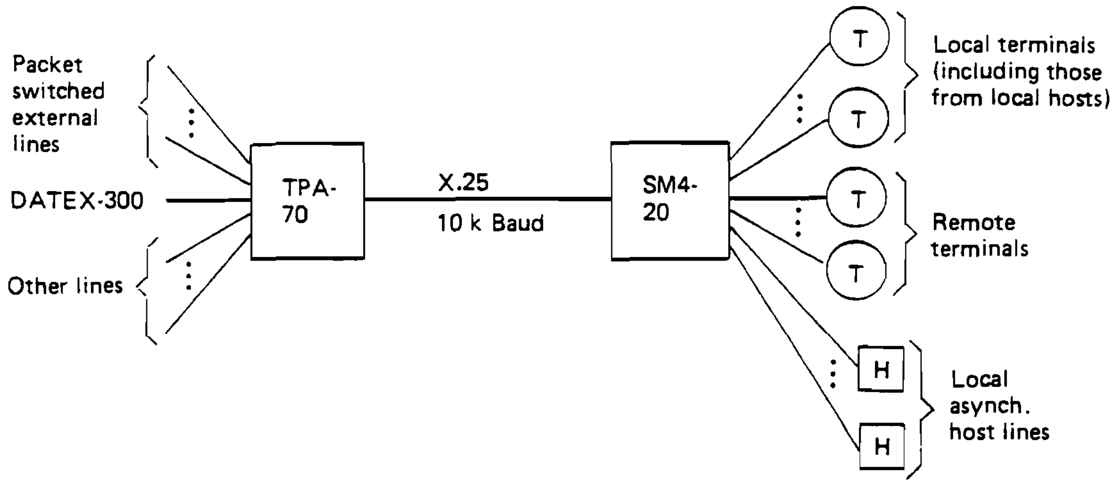

Fig. 2

of malfunction of one of the computers.

The other alternative may appear if the evolution of the IIASA computing facilities will lead towards development of a local area computer network. In this case one of the machines would have to serve as a local-to-external network gateway. The availability of an interface to the local transport medium could become the determining factor.

Both of the mentioned alternatives can be achieved by building on top of the presently installed software.

2. The packet switching software package for the SM4-20.

2.1. Implementation remarks (*).

The package described here consists of several tasks and can run on SM or PDP-11 series computers under DOSRV or RSX-11M operating systems (OS in further text). The particular tasks communicate together, using the standard means of intertask communication of OS. The tasks are in hierarchical interrelationship, corresponding to the hierarchy of performed functions of layered protocol levels. Events generated or detected by a lower level task are reported - if necessary-to the higher level task by the means of a software interrupt called an AST (Asynchronous System Trap). This includes also passing data to the upper task. The flow of control information as well as passing data in a downward direction is accomplished via sending so called orders, containing a description of action to be taken and possibly of the passed data to the lower level task. Here is also a standard OS facility used - the SEND DATA directive. These mechanisms use FIFO queues for passed objects, ensuring sequential data flow. Hardware interrupts are handled in a mixed manner. The interrupts coming from the interface(s) driving the packet line(s) are handled by self written interrupt service routines included in the FRAME task (see later),while driving the terminal lines is done using standard I/O macros of the OS terminal driver. These macros interact with the driver using FIFO queues, too. This approach was forced by the lack of standard

(*)
The text of this WP, starting from here assumes some knowledge of involved X. series of CCITT recommendations and of the hosting operating system.

drivers for communication interfaces, which could fulfill the requirements of the line protocol.

2.2. Running environment.

The described package can be used on computers with or without a memory management unit. It supports different combinations of communication interfaces. Any terminal device supported by the OS terminal driver can be attached to as a terminal. One to sixteen packet-switched lines can be supported, using either of DL-11 / ASAD single-line asynchronous interface, or DUP-11 / SAD synchronous interface. The latter can be operated with bit-stuffing, thus achieving the requirements of the X.25, or with BSC framing with DLE-s stuffed for data transparency. In the former case, CRC is generated and checked by hardware, in the latter this is done by software. This variety of possibilities on the packet-switched interface side is provided in order to be as flexible as possible and was initiated by changing possibilities of the TPA-70. Regarding terminal-driving: OS supports two kinds of terminal drivers. This package can run both using the 'half-duplex' and the 'full-duplex' driver. However, usage of the latter is strongly encouraged, as it allows smoother operation and will only be supported in further development of the package.

3. Package structure and functions.

3.1. Basic tasks of the package.

The following are condensed functions of the five main tasks which build up the kernel of the package. Somewhere, we also point to their place in the OSI reference model [1].

TASK *FRAME:*

Referring to ISO's seven layer Open System Interconnection (OSI) reference model, the FRAME task with the interface used houses the functions of the two lowest layers. It controls the functioning of the communication interface(-s) driving the packet switched line(-s). This makes up the lowest- physical layer of OSI. Besides that, the major part of FRAME performs the functions of the line access procedure LAPB according to the X.25 recommendation. If some situation in the future would require usage of the older LAP procedure, this could easily be implemented.

TASK *PACKET:*

Stays for the 3rd (network) layer of OSI. It performs the functions of the packet level of X.25. In the presently implemented form of the package there is no support for incoming calls, because normally the user of a terminal served by the concentrator initiates the establishment of virtual calls, rather than the remote DTE. This will change soon with the implementation of the necessary routing task and addition of some code in PACKET itself.

TASK *PADTSK:*

PADTSK is in fact an implementation of X.3 ,X.28 and X.29 recommendations of CCITT, but can hardly found its place in the ISO reference architecture. It contains the functions of a PAD (Packet Assembler/Disassembler). In the close future the same task will become capable of handling host- asynchronous ports besides the terminal ones.

TASK *ACCESS:*

Checks the access rights of the users approaching the PAD and maintains accounting files (one for each user).

TASK *OPER:*

Is the command interpreter for operator intervention. It also is the only means for attracting the attention of the whole package at the very beginning of a terminal session.

3.2. The FRAME task.

FRAME performs all the procedures and operations necessary to exchange data according to LAPB of CCITT X.25 standard. The task can be used to drive up to 16 communication lines. The task accepts orders from a higher layer using the interprocess-communication mechanism of the OS. Responses are sent to the higher layer by means of Asynchronous System Traps. Interrupt service routines are included to handle communication via synchronous (DUP-11 or SAD) or asynchronous (DL-11 or ASAD) interfaces. In the latter case a BSC-framing protocol compatible with the one used by the IIASA Gateway has been implemented with character stuffing/destuffing mechanisms and the format:

*DLE,SOH,FRAME- HEADER,INFORMATION(*if any*),DLE,ETB,CRC1,CRC2,PAD*

If a synchronous interface is used, the standard LAPB procedures can be performed, or it can be operated with DLE - stuffing as in the asynchronous case. The task is structured into the following modules

a) The main module, which performs the whole initialization, contains the interrupt service routines and decodes orders received from PACKET (higher layer). Further a fork process is contained in the main module which initiates the state modules using different entry-points associated with special groups of occuring events.

b) All possible events which may occur during link setup, transmission and reception of I-frames and disconnection of the link according to the LAPB are divided into several event classes (as 'frame received','frame transmitted',etc.). New orders from the higher layer task (PACKET) are handled as special events by the state modules of the FRAME task. The state modules have different entry-points, each of them associated with a separate event-class.

They perform the following steps:

- event recognition
- necessary operations according to LAPB :
  - start,reset or restart of a timer
  - generation and transmission of protocol data units, i.e. I-,S- or U-frames
- queue management operations
- queueing of Asynchronous System Traps to the higher layer task PACKET if there is a situation to be reported to the higher layer
- performing state transitions to change the current state of the link and to make a different state module responsible for the event processing.

GENERATION OF FRAME TASK.

To make the task FRAME fit the current requirements, to choose several facilities and include the desired options a prefix file called RSXMC.MAC must be set up. This file contains the information of the file [200,200]RSXMC.MAC or [200,200]DOSRV.MAC or [200,200]DOSRV3.MAC , which respectively is a product of operating system generation plus symbols added to specify conditions for assembly of FRAME. By defining different prefix files various versions of FRAME can be easily generated.

The generation of the FRAME task must commence with preparation of the RSXMC.MAC prefix file (its editing) to tailor it to the actual environment. To do this a file named RSXGEN.MAC is provided which contains all symbols which are necessary to define the features of FRAME. Then, the generation command file FRAMEGEN.CMD can be used. Before starting this command file observe the following:

1) Generate the file RSXMC.MAC in the target directory (i.e. directory which should contain the resulting task-image).

2) Include the task-builder command file FRAMETKB.CMD in the target directory. It can be edited eventually to ex/include the online debugger.

3) To generate FRAME for an RSX-11S system, include the system image and the symbol-table file of RSX-11S in the target directory.

According to the questions asked by the command file, the following can be specified:

1) Input-UIC (Directory containing the source files of FRAME)
2) Output-UIC (Target directory)
3) Listfile-UIC (Directory to contain the listing files)
4) Asynchronous interface used or not
5) System equipped with memory management or not

3.3. The PACKET task.

PACKET performs the functions of the packet level of CCITT X.25. That means in this package the establishment, operation and release of virtual circuits between DTEs. The new mechanisms specified in the 1980 release of the X.25 are not built in yet. The D-bit mechanism allowing end-to-end acknowledgement is not supported, as it decreases the performance of the network. The datagram service mechanisms are not provided, either.

The present implementation of the PACKET task assumes only one task - PADTSK - representing the transport layer, i.e. using the network layer services. In the future enhancements, the principle of single task interfacing PACKET will probably be maintained, just PADTSK will be replaced by a 'routing' task. Similarly to FRAME and PADTSK, PACKET has been programmed as a finite state machine, consisting of a main and several state modules. The main module does the necessary initializations, etc. In the following, the main functions of particular state modules are spelled.

- RST - is responsible for handling RESTARTs. If the FRAME task is up and the data link is operational, a RESTART is performed by RST - this happens after starting the PACKET task anew. Further actions - e.g. call establishment - are abandoned until any of sent RESTART REQUESTs is confirmed. A RESTART is also performed by PACKET/RST if the task should be terminated. RST handles RESTARTs initiated by network and solves RESTART collisions.

- XP1 - is the module corresponding to the *Ready state* of the logical channel. The most important events handled by this module are orders from PADTSK to set up a new virtual circuit (VC) and receipt of INCOMING CALLs. In the case of a VC-setup order, XP1 generates a CALL REQUEST packet, sends it using the services of FRAME and does a state transition to XP2. If an INCOMING CALL is received, XP1 indicates it to the network layer user in the transport layer. The normal case would be that the task representing the transport layer would accept the call and would order the PACKET task to complete the call setup. This order causes XP1 to send a CALL ACCEPTED packet to the network node (- TPA). Additionally XP1 does a state transition to the *Flow control ready state* (XD1). But as mentioned above PADTSK does not support incoming calls yet and therefore these are not accepted by the higher layer task in this version of the package.

- XP2 - materializes the *DTE Waiting state* of the logical channel. The expected event (we omit abnormal cases is this short description) is reception of a CALL CONNECTED packet from the network node. In this case XP2 sets the logical channel to state P5 (*Call collision* - module XP5).

- XP5 - solves events occuring in the *Call collision state*. The DTE's CALL REQUEST packet will be processed by the network and the INCOMING CALL will be cancelled. Therefore XP5 expect a CALL CONNECTED packet to come. In that case the requested VC is established and XP5 transits to XD1.

- XD1 - stays for the *Flow control ready state*. It transfers DATA and INTERRUPT packets via the logical channel bearing an established VC. Network Service Data Units (NSDUs) to be transmitted through the VC are segmented by XD1 and packed into DATA packets. The DATA packets are queued in a transmit queue dedicated to the VC. A straight flow control mechanism has been imposed on the interface between PACKET and PADTSK. The higher layer task (PADTSK in our case) is not allowed to request transmission of another NSDU until the transmission of the previous NSDU has been completed. Therefore, the transmit queue of a particular logical channel mentioned above, contains only DATA packets of the same NSDU. XD1 sends one packet after the other from the transmit queues of each VC, depending on the state of the windows opened by the network node for each VC, according to the flow control rules of the X.25. The boundaries of a NSDU during its transport through the network are indicated by using the M-bit in the DATA packet headers. The FRAME task needs a receive buffer for each VC. This must be specified by the PADTSK in a Receive order. Received DATA packets are queued-up in the receive queue of the concerned VC. The window opened for the network node on a particular logical channel is rotated only after one DATA packet has been transferred from the receive queue into the receive buffer of the PADTSK. Therefore the number of DATA packets in the receive queue of a VC cannot exceed the window size W agreed with the network administration (Albert Labadi). If no receive orders come from the PADTSK, up to W DATA packets can be received but the window will not be rotated by PACKET and the network node will stop transmission of DATA packets. XD1 sends updated lower window edge values as soon as possible - not to slow down the data flow from the node. That is why it uses RECEIVE READY packets to carry the acknowledgements if there are no DATA packets to be sent. XD1 also sends INTERRUPT packets on VC-user's request (order). A new INTERRUPT packet can only be sent if the previous one has been acknowledged by the network node (that means that the peer network entity has confirmed the interrupt). INTERRUPT packets are sent and received independently from the flow control applying to DATA packets. They have their own flow control as mentioned above.

If a RESET INDICATION is received, XD1 confirms it and carries out a reset operation of the particular logical channel according to X.25. Additionally it indicates this condition to the PADTSK to enable it to start error recovery procedures in that layer. If any protocol errors are detected, or if requested by the PADTSK , XD1 initiates a Reset operation on the associated logical channel by sending a RESET REQUEST packet to the network node. A state transition occurs then to state D2 - module XD2.

- XD2 - this module handles all events which can occur in the *DTE Reset request state.* A RESET CONFIRMATION packet is expected to be received from the network node. If this happens, the reset is completed by XD2 and the condition is reported to the higher layer by appropriate AST. A state transition occurs back to the D1 state and the normal data exchange continues.

If there comes to a serious protocol violation in the P1,P5,D1 or

D2 states or if the higher layer tasks request so, the VC is cleared by PACKET. The actual state module sends a CLEAR REQUEST packet and transits the logical channel state to P6 to wait for a CLEAR CONFIRMATION packet.

- XP6 - corresponds to the *DTE Clear request state* (P6). If the expected CLEAR CONFIRMATION or a CLEAR INDICATION is received, the associated virtual call is cleared, the data structures allocated for it are returned to the task's dynamic pool and the higher layer is informed that the VC has been cleared. All the state modules mentioned above except XP1 must be able to process the event 'CLEAR INDICATION packet received'. This event leads to the termination of given VC by the current state module in the same manner as XP6 does it.

- XR1 - finally this module handles all the events which can occur in the *DTE Restart request state* (R1) of X.25 . If a Restart is performed (see RST module), all active logical channels are set to the R1 state. This is a frozen state--it means that further events are ignored, until the Restart operation has been completed and all VCs have been terminated (=cleared).

GENERATION OF TASK PACKET.

Any parameter settings must be done in code. Newly assembled module should replace the old ones in the PACKET.OLB object library. The main module is kept separately. There is a command file PACKETBLD.CMD for linking the task PACKET.

3.4. The PADTSK task.

The PADTSK is a realization of a Packet Assembly/Disassembly facility (PAD). It is a converter between packet streams on different logical channels of the packet interface(s) and their respective character streams through the terminal ports. It allows the terminal user to

- select a network address to establish a connection to the addressed DTE;
- send & receive data over the established path;
- issue PAD commands to modify the behaviour of it or to enquire about it's parameters.

The full description of the PAD (which is supposed to be a facility of a public data network) is in X.3 recommendation of CCITT. The relevant formats and procedures are described in X.28 . In the SM-4 package only an essential subset of all the combinations of PAD parameters is supported.

PADTSK uses the same intertask communication means of the OS as do the other tasks in the package. It is also implemented in a form of a finite state machine and has the usual structure of one main and more (=3) status modules.

a) The ***main module*** receives and analyses orders from the OPER and ACCESS tasks. The orders can require the following actions:

Ordered action	*Sending task*
PAD initialization	ACCESS
Connection of a terminal to the PAD	ACCESS
Kill PADTSK	OPER
List connected terminals	OPER
Disconnect defined terminals	OPER
Mount DATEX 300 ports	OPER

Connection of a teminal to the PAD consists mainly of:

- allocation and initialization of a data structure describing the terminal status;
- attaching the PADTSK to the terminal - so that all terminal input is passed to it. Here, a feature of the 'full duplex' terminal driver called the 'typeahead buffer' is used with advantage. This is one of the main reasons for preferring RSX-11M v.3.2 or DOSRV3 as the host OS, as only these support the 'full duplex' TT driver.
- putting the PAD into the ***PAD Waiting state*** in which is is ready to accept commands from the terminal (ST1 is made the current module).

b) The state modules ST1 - 3 handle events connected with terminal input/ output and with the traffic through the interface PADTSK/PACKET. Separate entry points in the state modules are used for these two groups of events (ASTs).

- ST1 corresponds to the *PAD Waiting state*, when any terminal input is interpreted as a PAD command. So it is actually the command interpreter of the PAD. If the user entered a valid selection command, the necessary data from the command is passed to the PACKET task to be used for building the CALL REQUEST packet. A state transition to ST2 is made.
- ST2 remains the current state module while the *Connection in Progress* is the current state of the PAD. At this time, any further input from the terminal is ignored, just the bell rings on the terminal to indicate this. Receipt of a CALL CONNECTED packet by PACKET is reported to PADTSK and it transits to ST3. If the call was unsuccesful, a transition back to ST1 takes place.
- ST3 handles the data transfer phase. It is left at clearing of the call by either of the parties (user entered the CLR command or PACKET indicated the receipt of CLEAR INDICATION packet from the network).

The functioning of PADTSK depends very much on the momentary setting of the PAD parameters. The actual values of these are kept in each terminal's data structure. The main module contains also tables called profiles. These are sets of predefined values of PAD parameters. The user can change the parameters individually using the SET command of the PAD, or he can issue the PROF command to change them all at once. The incompleteness of the implementation of X.3, X.28 consists of the fact, that merely those values of PAD parameters contained in the predefined profiles are supported. Enhancement of this support will depend on operational experience and requirements imposed in routine operation.

The value of some parameters is of major significance for the PAD behaviour and some of them are checked even before the data is passed to state modules - in the AST service routines: *e.g.* the user can recall the attention of the PAD's command interpreter being in the data transfer phase entering a special character (CTRL^P) - if the PAD recall parameter (par.no.1) is set. The PAD echoes terminal input only if the par.2 is set, etc.

The PADTSK task supports another recommendation of CCITT - the X.29. This allows the far end DTE (can be e.g. a host) to communicate with the PAD in SM-4 in order to set or read the PAD parameters,etc. One typical usage of this could be to abandon echoing when the host password is being entered. DATA packets carrying this 'remote DTE-PAD' conversation are distinguished from those carrying user data by the Q-bit set in the general format identifier field of the DATA packet header.

A few words about the interface between PADTSK and PACKET tasks. The following are the valid event codes carried to PADTSK in the AST parameter block:

connected	a call has been established on PAD request
reset request sent	see 3.3
send complete	xmission of another NSDU has been finished, PACKET can accept another NSDU

data packet (Q=0) recvd.	it's reception by PADTSK will be confirmed to PACKET by an order, packet is typed on the terminal
interrupt packet recvd.	ignored by PADTSK (the interrpt. mechanism of PACKET isn't used by PADTSK)
clear confirmation recvd.	virtual call has been cleared, this is reported to the terminal
clear indication recvd.	call has been cleared by the other party, clearing cause is typed on the term.
reset confirmation recvd.	this is indicated to the term. as reset by PAD unless requested by user
reset indication recvd.	indicated to the user together with the initiator of the reset
incoming call recvd.	not supported yet - ignored
interrupt conf.recvd.	ignored
data packet (Q=1) recvd.	processed according to X.29

And here we just summarize the order codes sent to PACKET:

- initialization order
- set up X.25 virtual call

- clear logical channel
- reset logical channel
- receive DATA packet
- perform exit
- send DATA packet with Q=1
- set up X.29 virtual call

To facilitate necessary communication with ACCESS there are appropriate codes defined at that interface, too. The communication there mainly insures proper updating of accounting files for each user.

The specifics of **PADTSK generation** concern mainly proper choice of profiles for different terminal ports, or perhaps defining some new, more suitable ones to fit some particular needs. This must be done in the source code of the main module.

3.5. The ACCESS task.

The ACCESS task has two main functions:

-to check access rights of the users;

-to record accounting information during each session.

The access control is done on a local account number and associated password basis. It has nothing to do with access to particular DTEs. Once the user passes through this checkpoint, ACCESS instructs PADTSK to take further care of the user's terminal. Two files exist for each account on the accounting device :

- The first file contains identifying information about the account number holder - like name,password,address,etc.
- The second file is the accounting file written in fixed record length format. It contains information about date&time of each VC's and session's establishment, about the number of xmitted and received data segments,etc. When a new session is begun or a new VC is established, a new record is appended to the accounting file. The segment counters and date+time items are periodically updated based on information carried to ACCESS in ASTs initiated by PADTSK.

3.6. The OPER task.

Allows operator intervention. At the same time OPER passes the initial user request for connection to the ACCESS task. It has a defined set of commands for each controlled task. Based on the entered command OPER builds an order and sends it to the addressed task. OPER is installed as a utility (like a command in UNIX terminology). A command line for

OPER has the following syntax:

>opr taskname/function[:param1][:param2]...	;where taskname is one of PADTSK,PACKET,FRAME,ACCESS or others (see later); function is one of a particular set of functions defined for the given task.

We do not list here the total overview of OPER commands, instead, we show some frequently used ones:

>opr access/connect:1234567	;is the initial user command to establish a session via the concentrator. 1234567 is a user account number (7 digits). Pseudoname IIASA can be used instead of ACCESS in this command.
>opr padtsk/list	;lists all terminals connected (i.e. having a session) to the concentrator.
>opr padtsk/kill	;to shut down PADTSK if things go wrong or if it should not be used anymore.

OPER plays a key role in usage of different debugging tools in the package.

4. Development support tools.

There are two classes of development support tools in the package:

- tasks simulating particular layers;
- the tracing facility.

The principle of the former is based on the layered structure of the package. The point is to enable insulation of a particular layer (represented by a task) from the real running environment and to place it instead between two simulated layers - represented by these debugging tasks. In such a test environment the designer can "manually" trigger the appearance of various events and control the flow of data units (frames, packets, NSDUs) to and from the observed layer. In this manner he can force the system into non-common states and observe its behaviour in them. The package contains the following simulation tasks:

- TFRAME - simulates the interrupt service routines of the FRAME task and so allows to run it without real line. Driven by orders from OPER, TFRAME displays "outgoing" frames and injects "incoming" ones. It indicates on the terminal different occuring events and according to them triggers the state modules of Level2. (They are identical with FRAME).
- TPACK - simulates the PACKET task. Simultaneous use of TFRAME,TPACK and OPER represents a complete testbed for the 2nd layer task.
- L4SIM - is a simple simulation task capable to receive, transmit and display data segments to be carried in 3rd level packets. Orders from OPER can instruct L4SIM to require from PACKET all the

functions provided by the network layer. Thus, proper functioning of PACKET can be tested in off-line (with TFRAME) or on-line (with FRAME) regime.

The second main development support tool is the tracing facility. For debugging of dynamic, especially time-critical, sections of programs it is sometimes invaluable. If desired and defined so at generation time (by editing RSXMC.MAC), FRAME can collect tracing information (level 2 and 3 headers) on all transmitted and received frames/packets. The tracing data are periodically passed to the WTRACE task (if installed), which in turn writes them onto two disk files - one for transmission, one for reception. Additionally, relative time in 20ms ticks accompanies the stored header data. After stopping the system, RTRACE task can be run to convert the collected binary data into human-readable format. One can print-out the tracing protocol using the PIP utility of the OS. Appendix shows a fragment of a trace.

5. Operation.

In this chapter we describe actions performed by the SM-4 operator wishing to start, stop or control the running package. We do not speak here about using different development support tools, just about normal operation.

The following is the sequence of commands to be entered by the operator to start the system (suppose OS is running and a disk volume containing the package is mounted and assigned to sy0:; each command is terminated by pressing RETURN).

```
>@[1,64]ins        ;initiates a command (script) file, which installs the tasks of the package and makes the necessary assignments,
>run access        ;starts ACCESS, which initiates a chain reaction of starting subsequently PADTSK, PACKET and FRAME.
```

After this, FRAME automatically begins attempts to establish connection on its level, sending SABM frames or answering with UA the incoming SABMs. If a given number of attempts, separated by timeout fail, or if it even recognizes any abnormal state of the physical line, it prints a message on the console terminal. Here "LINE ii NOT READY" means that the modem did not set the CLEAR TO SEND bit, or there is no CARRIER. "SERIOUS ERROR 4 ON LINE ii" (e.g.) means that FRAME couldn't establish connection, though the line doesn't show any apparent malfunction.

The PACKET task, after being started immediately sends a RESTART REQUEST packet. It cannot be triggered to any further state until the restart procedure is completed. Then, the system is ready to accept users. If any malfunction occurs, and it is not clear from the printed diagnostic messages, the extensive comments in the source programs come to their value. If it becomes necessary to stop the blundering system it is good to know, which tasks are still active. The MCR (RSX shell)

```
>act /all          ;command can serve well here.
```

Then

```
>opr padtsk/kill            ;should normally stop PADTSK and
                            ACCESS;
>run x25kil                 ;should stop PACKET;
>opr frame/exit             ;should stop FRAME (and WTRACE-if
                            used).
```

Sometimes, the MCR

```
>abo taskname               ;command can be also used for
                            aborting tasks.
```

Each task's exit is reported on the console terminal.

6. User commands.

A user, logged into the SM-4 OS can establish a session with the concentrator package entering

```
>tic iiasa/connect:iiiiii          ;where iiiiii is his network access
                                    account number.
```

He is than asked to

```
ENTER CONNECTION PASSWORD:
```

His reply must be the password recorded in his user data file (see first file mentioned in 3.5). If it doesn't, or if there is no such file at all, his access is barred.

After passing the access check, PADTSK attaches to his terminal: i.e. the session has begun and all the user input is directed to PADTSK. The terminal will remain in 'comand state' until the user instructs the PADTSK to build up a virtual call. In the command state every line of input is treated as a PAD command. (For extensive description of PAD commands see the X.28 recommendation of CCITT).

Entering a 'selection command' tells the PAD to establish a virtual call to the addressed network entity (remote DTE) and sets the terminal into 'connection in progress state'. The format of selection command should be (according to X.28) :

```
[facility field]-address field<P>|<D>[call user data]
```

where fields in [] are optional and <>|<> are alternatives. The present version of PADTSK does not support user entered facility field, so that the actual format is

-address field<P>|<D>[call user data]

The address field may contain max. 15 digits. The data from this command are used further by PACKET to build up a CALL REQUEST packet. After entering the connection in progress state, the PAD rejects any further user input by ringing a bell instad of echoing it. This applies until

- either the VC is established and a COM indicates this on the terminal screen,
- or the VC couldn't be built up and an appropriate PAD service signal informs the user about this and about the reason (see X.28).

Being in the data transfer state, the user can (if the PAD parameter No.1 is set) escape from it by entering CTRL^P . This recalls the attention of PAD's command interpreter for the subsequently entered single command line.

An established VC can be cleared entering

CTRL^P CLR

or by instructing the remote DTE to clear the call. The user should then wait for indication of successful completion of the clearing (CLR CONF or CLR DTE). Then he can either enter another selection command or finish the session by issuing the EXIT command.

The PADTSK supports user input editing (character delete by DEL in command state and by CTRL^H=backspace in data xfer state, line delete by CTRL^U, line display by CTRL^R). The default profile is a UNIX oriented one, which provides PAD echo. The data forwarding characters are

RETURN, LINE FEED, CTRL^C, CTRL^S, CTRL^Q, CTRL^U, CTRL^R and DEL. In this profile, however, data xfer state input editing is disabled by default (can be enabled using the SET: command of PAD). This, and the choice of data forwarding characters allows the user to use UNIX terminal driver editing possibilities (after setting the editing characters to fit to the data forwarding ones by the STTY command). In this way a straightforward way of input editing when working with UNIX via concentrator is achieved. The output from UNIX, stopped previously via CTRL^S can, however, be unblocked only by CTRL^Q.

REFERENCES

[1] ISO DIS 7498, *Open Systems Interconnection - Basic Reference Model.*

[2] CCITT X.3, *Packet assembly/disassembly facility (PAD) in a public data network, 1980.*

[3] CCITT X.28, *DTE / DCE interface for a start- stop mode DTE accessing the PAD in a public data network situated in the same country, 1980.*

[4] CCITT X.29, *Procedures for exchange of control information and user data between a PAD and a packet- mode DTE or another PAD, 1980.*

[5] CCITT X.25, *Interface between DTE and DCE for terminals operating in packet mode on public data networks, 1980.*