TYMNET*—A terminal oriented communication network

by LA ROY TYMES
Tymshare, Inc.
Cupertino, California

INTRODUCTION

The past few years have seen many applications of the mini-computer in digital communications. They have been used as interfaces to larger computers. They have been used as terminal drivers and data multiplexors. They have been used to connect computer centers for intercomputer communication. TYMNET* is a communication net that encompasses all of these features at relatively low cost.

Although the net is very general purpose, it has been specifically oriented around the needs of the full duplex terminal in the ten to thirty character per second range. The terminal, connected to an acoustic coupler, interacts with a program in a timeshared computer on a character-by-character basis. The goal is to make this interaction as intimate as possible, even though the terminal and program are often thousands of miles apart.

THE HARDWARE

The net consists of Varian 620i's interconnected by 2400 and 4800 bit per second synchronous full duplex private leased lines. The 620i is a 16 bit machine with three working registers and a 1.8 microsecond cycle time. They are standard, off the shelf machines with 8K memory. Their only options are power failsafe and a primitive interrupt. The clocks from the synchronous modems interrupt the 620 for every bit to and from the synchronous modems. Some machines have a synchronous modem interface designed and built by Tymshare to assemble and disassemble 16 bit words, thus reducing the number of interrupts by a factor of 16.

The 620's are divided into two types, called base and remote (see Figure 1). The purpose of the base is to interface the net to the host, or timeshared computer. An assumption, which has proved to be valid, is that the base is usually up even if the host is down. Therefore, the base is useful as a node in the net even when it is not serving its host.

The remote drives the terminals. The typical remote is connected to up to 31 asynchronous full duplex modems. The data rate varies from 110 to 300 baud. Some remotes output to hard wired terminals at 600 and 1200 baud.

The remote can output on two wires and input on three wires on each modem for passing data and control information. On outputting a 16 bit word, the remote sets the voltage on 16 wires. On inputting a 16 bit word, the remote reads the voltages on 16 wires. The hardware understands the relationship between zeroes, ones, and RS232 voltage levels, and nothing more. In particular, it knows nothing about baud rate, character rate, or any other terminal characteristic. Serialization and deserialization of characters is done entirely with a software routine which is executed 1200 times a second. This simple hardware gives maximum reliability and flexibility at minimum cost.

When a user calls in to our system, he types a character to identify his terminal type. The remote analyzes this character to determine the baud rate, character rate, carriage return delay time, and other parameters. It then assigns two routines to this port, one for input and one for output (some terminals use a slower character rate for the keyboard than the printer). These routines handle all idiosyncracies of the terminal, including conversion to and from ASCII if the terminal is non-ASCII. Thus it is an easy matter to accommodate new terminal types as they become available.

If the remote has more than 8K of memory it may also have a printer, magnetic tape unit, and other equipment attached to it. The hardware interface is always the most primitive that will work.

* Trademark, Tymshare, Inc.
THE CIRCUIT

The nodes of the net are interconnected by full duplex synchronous private leased lines at 2400 and 4800 bits per second. The data is sent over these lines in the form of physical records. A physical record consists of a 16 bit header followed by several logical records followed by 32 bits of checksum. Errors are corrected by retransmission. When the error rate is nominal (less than 10 incorrect checksums per minute) about 240 data characters per second are passed on a 2400 bit per second line, or enough to handle 40 interactive terminals.

Inside each node there are a large number of character buffers, each one assigned to some routine which processes characters found in that buffer. Placing a character into a buffer is sufficient to insure that the appropriate routine will process that character at the appropriate time. In particular, some buffers are assigned to the physical record maker and placing a character into such a buffer will cause it to be assembled into a physical record with a particular virtual channel number.

Associated with each synchronous line is a permuter table (see Figure 2). This table is a list of pointers to pairs of buffers. The nth entry in the table corresponds to virtual channel n for that line. There is a matching permuter table in the node at the other end of that line.

We are now ready to trace the path of a character through the net. When the user strikes a key on a terminal, the terminal serializes the character and feeds it into an acoustic coupler. The coupler sends it over a telephone line to an asynchronous modem. The remote assembles the character and places it into the buffer assigned to virtual channel 2 and informs the physical record maker that something is ready for channel 2. The physical record maker looks up the second entry in the permuter table to find which buffer has the character. It then builds the record (possibly containing data for other channels as well). When this record arrives in the next node (the base in this example) it is torn down. Since the character came in on channel 2, the second entry in the permuter table for this line is checked to see where to put the character. In this case it goes into port 1 of the host.

I am now ready to define a circuit. A circuit is a full duplex character path through the net. It is described by entries in permuter tables. There are two buffers in each node per circuit, one buffer for each direction of character flow. Some readers will recognize this as a permutation switching network. 5, 6

A word about the efficiency is in order. Each logical record begins with one character to specify size and another character to specify virtual channel number. If data characters are sent in to the host one at a time as rapidly as a person types, there are two characters of overhead for every data character, plus the header and checksums of the physical record.

Several observations are appropriate here. First, the remote normally handles echoing on full duplex terminals. Second, while there are hundreds of terminals in use at any given instant, there are only 15 or 20 host computers clustered into three computer centers. Third, although the capacity of the synchronous lines is symmetric, the data flow into the host com-

Figure 1—A small network showing the relationships between host, base, and remote

Figure 2—A circuit passing through two nodes between terminal and host. Two buffers are used in each node, one buffer for each direction of character flow. In this example, a teletype on remote port 2 uses virtual channel 2 to reach host port 1
puters is only a small fraction of the outbound data flow. Fourth, computer programs do not usually output characters one at a time; the mean is about 40. Thus, on the inbound stream, we are “inefficient”, but it does not matter. On the outbound stream the overhead (which includes rate control and network control, see below) is about 20%.

Another problem is how does one control the rate at which characters flow. A computer program can easily output at 1000 characters per second, whereas a terminal may consume them at only 10 per second. The buffering capacity of the intermediate nodes is finite and will be exhausted unless backpressure can be generated to shut off the program.

A circuit can be low speed, meaning it can handle terminals up to 30 characters per second, or high speed, for 120 character per second links. Most circuits are low speed because they require less buffering. High speed circuits use high speed virtual channels, and low speed circuits use low speed channels. The distinction is noted by a bit in the permutation table.

Whenever node A sends characters to node B on channel n, it subtracts the number of characters sent from a counter associated with channel n. When that counter goes to zero, no more characters will be sent on that channel. Twice a second node B will send one bit per channel back to node A saying whether it has less than 32 characters (128 on a high speed channel) in the receiving buffer for that channel. If node B has less than 32 characters, node A will reset its counter to 32 (128 for high speed channels). This tends to put a vague upper bound on the number of characters a node will buffer for any given circuit. Thus, backpressure is cascaded back to the source.

The circuit may be regarded as a pair of pipes connecting two points. In each pipe the characters always flow in one direction, and they never get out of order. All interaction between terminal and program is through these pipes. All entities in these pipes are eight bit bytes, but they are not all data characters.

There are several special control characters to allow the program to control echoing modes and other properties of the circuit. All are encoded as eight bit bytes. Since a data character may be any eight bit byte, it could look like a control character. If it does, it is preceded in the pipes by an escape character which warns all appropriate machinery that the succeeding character is a data character, whichever it may look like.

Another special character is used to precede, or prefix, an eight bit parameter modifier. The parameter modifier is used to allow a user program to change one of 16 four bit fields describing the terminal in the remote. This allows the user program to alter such terminal characteristics as input baud rate, output baud rate, and the parameters in the formula to compute carriage return delay time. It can also control such things as whether an incoming line feed is echoed as a line feed or (more commonly) as a line feed, carriage return, and rubout.

The remote normally handles all echoing for full duplex terminals, but there are times when this is undesirable or impossible. For instance, if the terminal is listing output from the program and the user types ahead, the remote cannot echo the characters or they would be mixed up with the output text. Everyone types ahead on full duplex terminals. One does not realize how natural and convenient this is until one switches to half duplex after using full duplex for several weeks.) The remote stops echoing and sends a special character to the base to say that the terminal is in deferred echo mode. All echoing is now done from the host by a strategy that guarantees that the characters will come out in the right order. It is undesirable to stay in deferred echo mode because the echoing is so slow. It may take over a second for a character to make a round trip through the net if there are many nodes involved.

Before the remote can return this terminal to immediate echo, it must make sure that doing so will not cause characters to be echoed out of order. That is, it must make sure that the host has caught up with the input character stream and echoed all echoable characters. This condition is met if both pipes are empty and the program is dismissed waiting for an input character.

To test for this state, the remote places a green ball in the inbound pipe. When the green ball reaches the base, it waits until the program is dismissed waiting for an input character and the character buffers in the host associated with that program are empty. It then enters the outbound pipe and returns to the remote, possibly following some data characters. If it reaches the end of the pipe before any more data characters were sent in to the base, the terminal returns to immediate echo mode. If more unechoed data characters have been sent in during this time, then it is not safe to return to immediate echo because character echoes may follow the green ball.

A green ball can obviously take an arbitrarily long time to complete this trip. Once the remote has sent in another unechoed data character, it knows that the information that the green ball would convey if it returned is obsolete. That is, the pipes are not necessarily clear. Before the remote can make another attempt to get the terminal out of deferred echo mode, the old green ball must be flushed out. The remote places a red ball in the inbound pipe. The red ball simply goes to
the base and returns, destroying any green ball it encounters.

This same machinery is used to find out when it is time to unlock the keyboard of an I.B.M. 2741. This is interesting because it allows the remote to completely shield an ASCII, full duplex oriented host from the awkward operating characteristics of that unfortunate terminal.

When an outbound pipe is full of characters, it may take several seconds to type out. Sometimes a program wishes to clear this pipe quickly, as when it realizes that the output is not wanted by the user. To clear the pipe, the host releases a character gobbler. The character gobbler races through the pipe at top speed, gobbling all characters ahead of it. When it gets to the end of the pipe, it annihilates itself.

Even more potent than the character gobbler is the circuit zapper. When a circuit is no longer needed, its buffer pairs and permuter table entries must be released quickly and cleanly. There should be no left over characters wandering aimlessly about the net. When the user logs off and hangs up his phone, or when the host decides to release a circuit, a circuit zapper is released. As it moves through the circuit, it clears all buffer pairs and leaves a trail of null permuter table entries behind it. When a circuit is zapped, it is completely erased. There are no remnants of it to pollute the net.

THE SUPERVISOR

The supervisor is a program that runs under timesharing on an XDS 940. Its purpose is to build circuits, perform diagnostics, keep statistics, and handle all other matters of global importance to the network. There are normally several supervisors running at any moment to provide backup in case of failure, but only one is in control of the network. This one is called the supervisor in active mode, or Sam, for short. Sam has in its memory a detailed representation of the network. In particular, it has a copy of all the permuter tables.

Every node in the network has a leprechaun. The purpose of the leprechauns is to issue high priority diagnostics, pass login information to Sam, and obey all commands from Sam, such as commands to change permuter tables. Every link has one full duplex supervisory channel for leprechaun communication. In every node, one link is called upstream, or toward Sam. Some nodes also have a downstream link. Messages moving on these links go upstream or downstream. If a leprechaun wishes to send a message to Sam, it places the message in the upstream link and the message percolates up to Sam. If Sam wishes to send a message to a leprechaun, Sam sets up a downstream path and sends the message down that path. Thus the leprechaun is a very simple creature with no global knowledge of the net. In fact, it takes less than 300 words of code.

Circuit building is Sam's primary function, and can be illustrated by several examples. Suppose a user calls a remote and identifies his terminal. The next thing he types is usually his name and password. A leprechaun in the remote sends these characters to Sam. Any further characters typed by the user remain in a buffer in the remote until a circuit is built. Sam hashes the user's name into the MUD (master user directory) to find out, among other things, which host has this user's files. The user himself may not know this since an operator may have moved them the night before. Sam then checks to see if that host is up and has an available port. The next step is to select the nodes and links through which to construct the circuit. Since the representation of the network is a multilinked list of node descriptors, and since each link is tagged as being in the direction of a particular host or not in that direction, this is a trivial process requiring less than a millisecond of compute time. The only optimization done is to avoid links which are heavily loaded or which have high error rates. An unused virtual channel on each link is assigned to this circuit.

The user's name is now passed on to the host. The host hashes the name into the LUD (local user directory) to find the user's file directory, billing account, and so on. The final step is to send commands to the leprechauns in all the nodes affected to make the appropriate entries in their permuter tables.

The circuit just described is called a normal circuit. It is the only kind most programs use. Some programs use an auxiliary circuit to connect to things other than terminals. Suppose a program in host A wants information from the files of host B. It makes a supervisor request to construct an auxiliary circuit between its port on host A and a port on host B. The port on host A is multiplexed between the normal circuit and the auxiliary circuit, but only the program and Sam know this. Neither host knows that this is happening. Host B thinks the auxiliary circuit is a normal circuit with a terminal at the other end. The program in host A can interact with the program in host B just as a user would. In particular, the two programs can exchange data. Since the data rate is limited to about 150 characters per second, it is not a good way to move large files, but that is not an important restriction. Of course, the program in host B can acquire still another auxiliary circuit to host C.

Another use of the auxiliary circuit is attaching a program to some peripheral device, like a printer. The program supplies the logical name for the device, which
is unique for every device. Sam locates the device (its location may change from day to day) and makes the connection. A potential use of auxiliary circuits, not yet implemented, is remote dialout. The program supplies the telephone number. Sam selects a node with a remote dialout unit on it which is inside or close to a toll-free dialing area for that number and makes the connection.

Another type of circuit building occurs when a node or link goes down. Sam will reroute all affected circuits for which an alternate route exists.

A critical situation is created when Sam goes down. Without Sam, no new circuits can be built and no new users can be accommodated. A new Sam must be created quickly.

There are normally many supervisors running besides the supervisor in active mode. They receive a message from Sam at least once a minute. Should these messages stop, they automatically begin to take over the net.

The network takeover involves three things. First, one supervisor must be chosen to be the new Sam. Second, this supervisor must construct an up-to-date representation of the net in its own memory. Third, it must win the allegiance of all the leprechauns.

The supervisor runs in a host, and the host is connected to a base. The supervisor wins the allegiance of the leprechaun in that base by causing the upstream direction to point into the host. Once that is done, the supervisor can find out all it wants to know about the base by asking the leprechaun. It then commands the leprechaun to send a takeover message down all of its supervisory channels. When a takeover message arrives in a node from a particular link, that link becomes the upstream direction. The leprechaun identifies itself on the new upstream link and the supervisor proceeds to find out what it needs to know to build its internal network representation. The supervisor continues node by node, level by level, until there are no more unknown links. When this is done, the upstream direction has been defined for all leprechauns, the internal network representation is complete, and the supervisor becomes the new Sam.

In the process of taking over the net, a supervisor creates a steadily growing sphere of influence. Should this sphere of influence intersect that of another supervisor, a leprechaun in the intersection of the two spheres switches allegiance. Just before switching allegiance, it sends a message in the old upstream direction saying which supervisor is taking over. Thus, when supervisor A steals a node from supervisor B, supervisor B discovers that supervisor A is trying to take over the net.

The supervisors are arranged in a pecking order. If an inferior supervisor steals a node from a superior supervisor, the superior supervisor will take it right back. If a superior supervisor takes a node from an inferior supervisor, the inferior supervisor becomes quiescent.

Several points are worth stressing here. First, a leprechaun has no global knowledge. This means that the software in a given node can be treated as an independent module which obeys certain conventions. This enormously simplifies the debugging and minimizes the danger of a bug in one node clobbering another node.

Second, the supervisor has no a priori knowledge of the net when it begins to take over. This makes it convenient to make alterations to the net. The supervisor simply adapts itself to whatever configuration is presented to it.

Third, the process is entirely automatic. The consequences of such a complex process being dependent on human operators are too horrible to contemplate.

Fourth, all global information is available in one single spot. This greatly facilitates diagnostics, record keeping, and debugging.

Fifth, the most complex routines, those of the supervisor, run under timesharing. The advantages of debugging under timesharing are difficult to exaggerate.

THE FUTURE

Technology of the type described in this paper is in a rapid state of flux. The price-performance ratio of minicomputers is improving, terminals are proliferating, and commercial timesharing is becoming a big business. Therefore, the threat of obsolescence becomes a major design consideration. Before one can intelligently discuss the threat of obsolescence, two general questions must be answered. First, what are the long-range objectives of the network? Second, in what directions is technology most likely to change?

Much research is currently being done on high speed graphics terminals, sophisticated text handling terminals, and on the problems of moving large masses of data quickly. These projects are glamorous. I concede that they are useful. But they are expensive, and therefore of limited use. I feel that, given the current market and technology, the greatest social and commercial potential lay with the low cost keyboard terminal.

I would like to make the timeshared computer available to anyone who wants it. In order to do this, the cost must be low enough so that everyone can afford it. Thus, although the network is very general purpose, its performance has been optimized around the needs of the simple low speed terminal with an attempt to serve the greatest number with the fewest dollars.

In what directions is technology moving? First of all,
the variety of terminals is increasing. Since all terminal drivers are implemented entirely in software, it is easy to design a new one and make it available to all nodes of the net on short notice.

Second, the output character rates of "low cost terminals" are increasing. Keyboard CRT terminals come down in price every year and hard copy terminals are getting faster. I predict that in two or three years an acoustic coupler that operates at 1200 baud in one direction and 150 baud in the other will be in common use. Our algorithm for serializing and deserializing characters in software is well suited to outputting at 1200 baud and inputting at a much slower rate.

Third, the cost of moving data over long distance phone lines will come down, but at a slow rate compared to other aspects of the technology. 9600 bits per second is about it for voice grade lines, and although widespread availability of digital (rather than analogue) "voice grade lines" may be just around the corner on the telephone companies' time scale, it is an eternity away on the timesharing industry's time scale. This is already the dominating cost of the net, and the net is designed to optimize this resource.

Fourth, the speed of the minicomputer will continue to increase dramatically. In a system so totally committed to a software solution to all problems, the raw speed of the computer is obviously crucial. The recently available Varian 620f costs less than the 620i, yet it is more than twice as fast. Still faster computers will be available when we need them. Since the special hardware is so primitive, and since the software in any node is only about 4000 instructions, it is easy to switch to another computer. Since any node can be treated as an independent module following well defined conventions, it is easy to phase in new hardware and software (possibly not well debugged) one node at a time.

Finally, the future is guaranteed to have some surprises. One can never be absolutely certain how the future will turn out, and the only hedge against uncertainty is flexibility. Quite possibly the Tymnet of 1984 will be quite different from the Tymnet of today, but it is hoped that the evolution will be relatively painless.

REFERENCES

1 H B BURNER R MILLION O W RICHARD J S SOBOLEWSKI
   The use of a small computer as a terminal controller for a
   large computing system
   Proceedings of AFIPS SJCC 1969

2 L ROBERTS
   Computer network development to achieve resource sharing
   Proceedings of AFIPS SJCC 1970

3 S CARR S CROCKER V CERF
   Host—Host communications protocol in the ARPA network
   Proceedings of AFIPS SJCC 1970

4 K THURBER
   Programmable indexing networks
   Proceedings of AFIPS SJCC 1970

5 A WAKSMAN
   A permutation network