DATAKIT - A MODULAR NETWORK FOR SYNCHRONOUS AND ASYNCHRONOUS TRAFFIC

A. G. Fraser
Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

A set of mutually compatible modules are the elements of an experimental data network. Called Datakit, the concept being studied is an "erector set" approach to the construction of data networks that can grow gracefully and evolve to accommodate a wide variety of traffic types.

The modules, each a single circuit board, are interconnected by a hardware packet switch. The network is master clocked so that speed control can be used for stream traffic while asynchronous multiplexing can be used for bursty data.

INTRODUCTION

The incremental and adaptive nature of data network growth suggests the use of an "erector set" from which data networks can be made. A particular network would be constructed by selecting from a catalogue of plug-to-plug compatible modules. Initially one could start with a small catalogue and add to it as the market for new services expands and technology makes improvements possible.

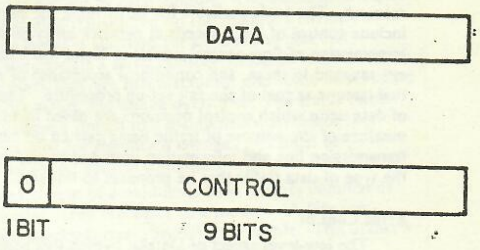
Datakit is an experimental catalogue of data network modules. At the time of writing, the modules provide communication between seven computers of various sizes and in the near future we hope to add asynchronous terminals and telephones to this small network. Support for certain types of synchronous terminal may follow later.

Data Representation

The object of a Datakit network is to permit arbitrary pairs of machines to communicate with one another in a simple and effective manner. The fact that two devices are of different types or have different natural operating speeds should not itself be a barrier to communication between them. The first step in meeting this objective is to establish a network standard for data representation.

The basic unit of data is a 9-bit *envelope* (figure 1). One of the nine bits indicates the type of data contained in the remaining 8. In one case the 8 bits are user data while the other case provides for various types of control information. Control codes exist both for user-to-user communication (e.g. end of file) and for network internal operations (e.g. flow control). One control code, called NULL, is a filler that may be inserted into, or deleted from, the data stream without altering the meaning of what is transmitted.

In many cases the user's data is presented to the network as 8-bit bytes with control information encoded separately. For example, BREAK is signaled by an ASCII terminal in the form of a temporary violation of the start/stop byte framing format. In Datakit BREAK is just another control code. Synchronous terminals that use the HDLC protocol transmit a special *flag* character to delimit messages and extra bits are stuffed into the data to ensure that the *flag* does not occur accidentally. At the interface with Datakit the *flag* becomes a SYNC control code and the extra "stuff" bits are removed from the data.



ENVELOPE STRUCTURE

Figure 1

When there is a difference of protocol between one terminal and another, the Datakit user must route his communications through a third party: A protocol conversion machine. The same may be necessary if the flow or speed control techniques employed by communicating users differ. We are currently studying how best to do this translation in certain special cases. One case is the interface between two computers that support different data link control procedures. Another case is the interface between a time-shared computer and its terminals.

Speed and Flow Control

In a synchronous digital network one can avoid congestion on an individual circuit by establishing a standard measure of time. In this way one can arrange that the rate at which data enters the circuit is exactly equal to that at which data leaves the circuit. The standard measure of time is generated by a single (reliable) master clock and is broadcast throughout the network. From that clock, all other clock speeds are derived.

CH1435-7/79/0000-0098$00.75 © 1979 IEEE

Datakit is a master clocked network so that circuits that carry bulk or synchronous data can be regulated in a way that prevents congestion within the network.

Bursty traffic from a set of independent data sources can be carried efficiently on a shared transmission line by providing some queue storage space that is used to smooth out the aggregate data flow. The maximum rate of output from the queue may be significantly less than the sum of the maximum input rates. However, there is a possibility that the queue may overflow under peak load conditions.

Datakit provides for bursty traffic and it is our aim to regulate the traffic so that the probability of queue overflow is held to an acceptably low level. This is an area of current research. The tools available for performing the regulation include control of clock speeds at network entry points, transmission of flow control signals to those machines that can respond to them, and conditional acceptance of new virtual circuits as part of the call set-up procedure. The sources of data upon which control decisions are based include a measure of the volume of traffic being carried on each transmission line and information provided by the user about the type of data traffic that he proposes to transmit.

Error Control

The low-level aspect of Datakit (which this paper primarily addresses) is designed to make data corruption improbable, but loss of transmitted data is more likely. The intent is to provide a channel with predictable error characteristics. The user, or his interface equipment, is expected to perform end-to-end error detection and, if necessary, retransmission.

Detection of error within a network node is done by including parity with every envelope. Envelopes with bad parity are discarded and status information (described later) is reported to a central point from which maintenance action can be taken.

On transmission lines data is checked by a CRC. If the CRC fails, all the associated data (48 envelopes) are discarded. Statistics are kept so that failing transmission lines can be identified and repaired, or traffic can be rerouted.

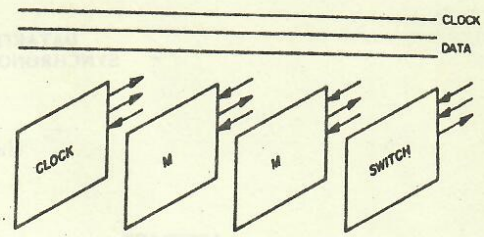
Module Interconnection

The secret of success for a modular system is a versatile set of module inter-connection standards. For Datakit this translates into the physical, electrical and procedural standards that allow data generated in one module to flow into another.

Each datakit module is a single circuit board. A connector on the board plugs into a standard back-plane and it is through the back-plane that one module communicates with another (figure 2). If the module connects to some external equipment, such as a modem or computer, that connection is by means of a second connector mounted on the circuit board.

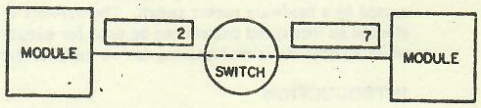
The back-plane connector positions are identical but each carries a different pre-wired connector address. A maximum of 511 modules can be connected together (in several card cages if necessary). Such an assembly of modules is called a Datakit *node*.

The modules communicate through virtual circuits as shown in figure 3. The circuits are administered by a switch



MODULE INTER-CONNECTION

Figure 2

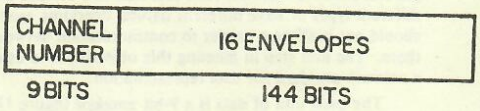


VIRTUAL CIRCUIT

Figure 3

module which is described below. For each virtual circuit, between a source module and a destination module, there is a source channel number and a destination channel number. These are not necessarily equal. The channel number is the module's means of identifying a particular virtual circuit. This 9-bit number allows one module to communicate on 512 virtual circuits at one time. The virtual circuits are established and cancelled by a procedure that will be described later.

Within a module, data is handled in packets of 16 envelopes and one 9-bit channel number (figure 4).



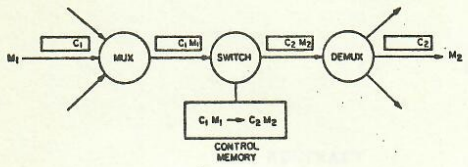
PACKET STRUCTURE

Figure 4

When the packets leave the module they have the source module address appended. The packet, after passing through a switch, arrives at the destination module with the destination module's address and channel number affixed to the 16 envelopes of data.

The Switch

There is one switch (plus an optional hot standby) in each Datakit node. Its operation is shown schematically in figure 5.



PACKET SWITCHING TECHNIQUE

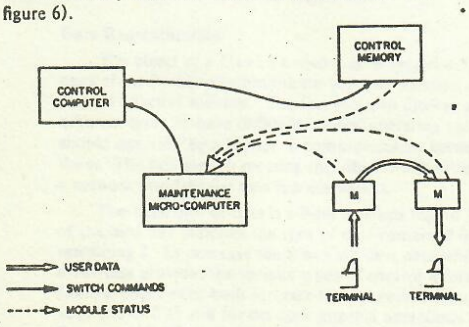
Figure 5

Packets from all modules arrive serially, each packet carrying the module address of the source and the source channel number. The switch uses these two numbers to address a control memory in which is held the module number and channel number of the destination module. These are then substituted in the packet and transmitted over a shared bus to the destination module. The switch processes data serially at 7.5 Megabits per second.

Virtual circuits are therefore defined by the content of the control memory. One word in that memory defines a single circuit joining a source module channel to a destination module channel. Circuits are established and cancelled by writing into the control memory.

The control memory itself behaves like a module. It can send and receive packets. The packets it receives are commands to read or write words in the memory and the packets it transmits are responses to those commands.

Typically, virtual circuits are established between the control memory and one or two control computers (see figure 6).



NETWORK ADMINISTRATION

Figure 6

Only one control computer should be active at one time. It is the control computer that establishes and cancels virtual circuits in response to commands transmitted to it from other modules and, ultimately, from network users.

Administration and Maintenance

A maintenance processor monitors the state of the network. There are two sources of network status information: the control memory and the modules.

One source of data is the control memory. With each word of routing information, the control memory contains a count of the number of packets transmitted on a virtual circuit. This data can be used for accounting and traffic management. The control memory also contains information about parity errors detected in data sent from the various modules.

A second source of data is status information transmitted once per second by each module. The status information is a single packet transmitted on the module's channel zero. Part of the status is standard for all modules and that includes the module's serial number (assigned individually to modules when they are manufactured) and some information about parity errors and queue overflows. The status information is usually routed to a maintenance processor that filters it looking for significant changes that must be transmitted to the control computer.

Channel zero of a module is used to carry commands from a maintenance processor to a module. A few commands are standard: reset, disable, and enable. This allows a certain degree of control in case of module failures.

All of this control and status information is carried on standard Datakit channels and all elements of the system (even the switch) are implemented as modules. Modular assembly and reconfiguration are therefore particularly easy.

Conclusion

Datakit is an on-going research project. It is of interest not only because a set of modules may provide a practical means of building new networks but also because attempts at module design sharpen one's understanding of separate or separable network functions.

The modules so far constructed are a computer interface, an asynchronous terminal interface, a transmission line interface, a switch and a clock. Modules under development include a bridge-tap module that permits one to monitor traffic on selected virtual circuits, an interface for a digital telephone, and a buffer module. The latter is a set of FIFO queues that can be switched into a transmission path when necessary to augment the smaller queue that is present on each module board.

Performance of the system is currently limited by the computer interface. We have not yet built any DMA interfaces. The switch can handle about 42,000 packets per second and the fastest transmission line can carry 10,800 packets per second. Of course, neither of these facilities can be fully loaded if bursty traffic is to be carried with reasonable probability of queue overflow.