Datakit Network Architecture

G. L. Chesson

A. G. Fraser

ABSTRACT

Datakit hardware and software components address the problems of data communications with a modular "erector set" point of view. The hardware modules comprise single circuit boards interconnected by a hardware packet switch. The software modules consist of machine-independent routines for accessing the hardware and providing process and file communication through the network.

Introduction

The Datakit network is a focal point for ongoing research in data communications. The emphasis of the work is on defining and experimenting with hardware and software modules that can be combined to build digital data networks. This paper provides an overview of some of the modules that are being studied. Additional information appears in two earlier papers[1,2].

Hardware Organization

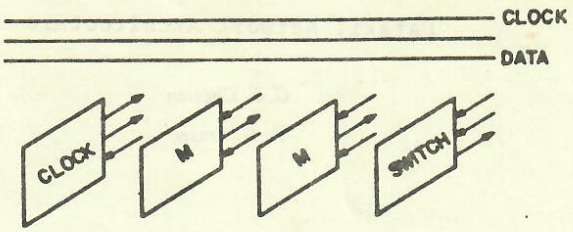
A Datakit *node* consists of a number of circuit boards, referred to as *modules*, arranged in one or more card cages, or *bins*. Each bin has a printed-circuit backplane equipped with connectors for inserting the circuit boards. The board positions on the current backplane design are numbered from zero to ten. One slot is reserved for regenerative repeater circuits that increase the logical extent of a backplane across multiple bins.

Each bin has a strappable *bin number*. A *module number* is defined as the concatenation of a bin number with a board position. The binary-encoded module number is provided on designated pins at each board connector by selective wiring on the printed-circuit backplane to power and ground.

Modules plug into a bin as depicted in Figure 1. There is a switch module at one end, clock module at the other end, and other hardware modules in between. The backplane provides signal distribution from the clock to the other boards plus two bit-serial open-collector data busses. One bus line is operated with a priority-contention scheme based on module number and controlled by the clock signals. This bus carries data from modules to the switch. The other bus moves data from the switch back to modules. A *control memory* in the switch contains routing information for virtual circuits. This arrangement results in a high-speed inter-module packet-switching capability.

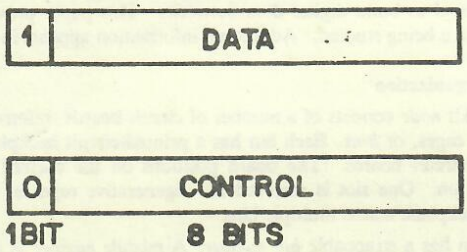
Data formats on the backplane lines are depicted in Figure 2. Transmission on the data lines is in terms of 9-bit *envelopes* each accompanied by a tenth parity bit. Envelopes are grouped into 18-element *packets* shown in Figure 3 where the first envelope carries a 9-bit module number, the second envelope carries a 9-bit *channel number*. The module and channel numbers are followed by 16 data envelopes. The module number in a packet identifies the sender on transmissions to the switch, and selects a receiver on the switch-to-module bus. The channel number similarly identifies or selects one of 512 possible logical channels on a module. Data envelopes contain either 8-bit data bytes or 8-bit control bytes according to whether the ninth bit is respectively one or zero. The 9-bit format provides for out-of-band signalling and control of the data stream.

The error control philosophy in Datakit employs error detection hardware but does not provide for error correction or retransmission at low levels in the hardware. Network data discovered



MODULE INTER-CONNECTION

Figure 1

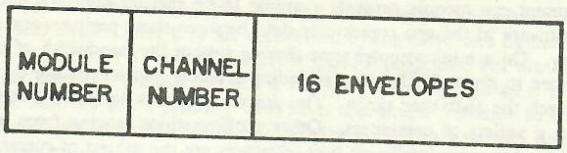


ENVELOPE STRUCTURE

Figure 2

to have errors is simply discarded, whether the problem is with parity errors on the backplane or CRC errors on modules that employ CRC generation and checking. Thus all errors appear to the end-to-end protocol level as missing data, and the mechanisms at that level are designed to recover from missing data efficiently and quickly.

A status register is incorporated into each hardware module. Included in this register are such things as the board type, its manufacturing serial number, and information about data errors recently seen by the module. The contents of this register form a *status packet* that is transmitted on channel zero once per second by each hardware module in the system. The control memory of the switch is arranged so that the various channels zero are routed to a dedicated microprocessor, called the *maintenance processor*, which continuously monitors the health of the hardware. The



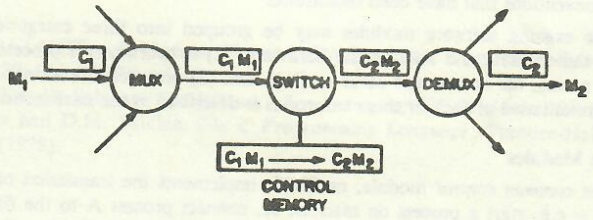
PACKET STRUCTURE

Figure 3

electrical design of the backplane is such that boards can be inserted and removed while the system is in operation. The maintenance processor(s) notice and report such configuration changes in addition to providing a means for centralized error logging.

Hardware Modules

The Datakit switch module provides virtual circuits between channels on modules as depicted in Figure 4.



PACKET SWITCHING TECHNIQUE

Data enters a module, or is generated by a module, on some channel C1. Suppose the module is plugged into board position M1. The module sends the data as one or more packets to the switch, using the board position M1 and channel number C1 in the packet format. The M1C1 header on these packets going to the switch can be thought of as *source* information. The switch replaces incoming module and channel numbers of incoming packets with new numbers (M2C2) obtained from a *control memory* on the switch module. This is done synchronously on a packet-by-packet

basis. The new header on packets leaving the switch comprises *destination* information. If a module recognizes the number M2, the remainder of the packet is copied from the bus and the 16-envelope data portion is delivered to channel C2.

The present cpu module presents a simple 16-bit register interface through ribbon cables to a host cpu. Software in the cpu copies user data between these registers and the desired locations in main memory. On a minicomputer time-sharing system the bandwidth of this arrangement is limited by software to about 50Kbits/sec, including protocol and scheduling overheads; dedicated computers can reach the 1Mbit/sec range. The simplicity of this hardware interface yields quick implementation on a variety of computers. Other configurations ranging from modules having cpu and memory on board to more intelligent host interfaces are the subject of current research.

Trunk modules operate in pairs: packets coming into a trunk module from the backplane are sent over a transmission line to a companion board in a remote node which decodes the transmission format and sends the result to the switch on its backplane. That switch routes the data to a module on its backplane in the usual way. This scheme provides for data communication between hardware modules located on different nodes subject to the bandwidth and distance characteristics of the transmission medium employed by the trunk boards. Trunks have been constructed using both twisted-pair and fiber optic technologies. These are suitable for local area networking. A trunk module interfaced to a 56Kbit/sec wide-band modem has been demonstrated and could be used for long haul traffic.

The clock module provides bit, envelope, and packet timing signals for the modules at a node. The bit rate currently used is 7.5Mbits/sec. This yields a packet switching rate of about 42000 packets/sec. When nodes are interconnected by trunks, the clock and trunk modules cooperate in propagating and synchronizing to a master clock source. This is the key to providing effective speed control in a distributed system.

Software Organization

The software modules in Datakit resemble the hardware modules in that each one implements a well-defined function that can be isolated and developed independently of other modules. Most modules are processes, and information moves between them according to the interprocess communication conventions that have been established.

The existing software modules may be grouped into three categories: (1) network control modules that operate and manage the hardware, (2) network server processes to implement remote terminal access, file access, and other services, and (3) protocol subroutines and libraries. The principal representative of each of these categories is described in the next section.

Software Modules

The *common control* module, or CMC, implements the translation of logical network service requests -- e.g. start a process on machine X, connect process A to the file server on machine Y, connect terminal 42 to an interactive command processor on machine Z, find the network mail address for John Doe, etc -- to physical addresses in the network. It is responsible for setting up and taking down virtual circuits through switches and trunks on demand from user processes. This is done by reading and writing the contents of the switch control memory and by exchanging messages with user processes and server processes in the network. The channel setup strategy and protocol is explained in some detail in [2] and need not be duplicated here. The brief description is that a user process sends a single packet service request message to the CMC which sets up a channel to the appropriate hardware module and forwards the service request to a listener process (see below) at the destination. The physical location of the CMC in the network is transparent to other processes: it may be on a time-sharing system, on a dedicated computer, or possibly built into a future switch. The number of CMC processes is also transparent to user and server programs: a single CMC can control multiple nodes, or there could be a CMC per node. The external interface to other objects in the network is the same in either case.

The *listener* module is a program that runs on each active computer in the network. It sends periodic "I-am-alive" messages to the CMC. These messages include the logical identification of the

host so that the CMC can easily tell which machines are available. The listener is actually named after its other function which is to receive service requests from the CMC. All services available on the network are reached by accessing a listener through the CMC.

The transport protocol defines how flow control, error control, and interprocess signalling are accomplished on virtual circuits after they are established by the CMC and listener. These functions correspond roughly to levels 2, 3, and 4 of current CCITT proposals, but are defined as a single coherent module on Datakit. Datakit defines a *transport level* module as a bidirectional 'filter' inserted between the network and a process. This leads to a simple i/o-based model of the interface which can be preserved across changes in the actual protocol. This has been demonstrated several times. The first transport protocol on the network, known as the *packet driver* module, was used as a test vehicle for protocol ideas and operating system interfacing for about a year. The protocol definition changed several times during this period and will soon be replaced by something completely different. In each case other modules in the system were unaffected by the changes.

All software in the system is written in the C programming language[3]. This means that the software can be executed on any machine within the purview of the portable C compiler. Portable programming techniques, plus modularization by function, and standard protocol libraries taken together form the technical basis for a so-called "reference implementation" of the network access software and protocols. The idea of the reference implementation is to have a working model of the network software that can be ported to different computers and operating systems. The task of network maintenance and proliferation across a range of machines is understandably easier than when multiple different implementations are involved.

Conclusion

A Datakit network is currently providing data communication facilities for a small number of computers. It is constructed of the modules described here and demonstrates one way of applying modularity and portability concepts to networking practice. The system 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.

References

1. A.G.Fraser, "DATAKIT - A Modular Network For Synchronous and Asynchronous Traffic," *Proceeding of ICC 79*, (June 1979).
2. G.L. Chesson, "Datakit Software Architecture", *Proceedings of ICC 79*, (June 1979).
3. B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey (1978).