Early investigation of virtual machine subsystem flexibility centered on telecommunications support and intercomputer networking and proceeded in two phases. The first phase focused on an experimental program for the virtual machine control program CP-67 that supported remote work stations and pioneered intercomputer spool communications. The results of that effort inspired a second effort in the same area with some significant redirection. This second phase ultimately led to the remote spooling communications subsystem component of VM/370, the VM/370 networking package (VNET), and a large network of interactive computer systems within IBM. These phases are discussed along with suggestions for several continuing lines of work based on current results.

Evolution of a virtual machine subsystem

by E. C. Hendricks and T. C. Hartmann

The term "hypervisor" is applied to computer systems that present a very basic user program interface—one which is so nearly identical to a particular computer machine interface that an operating system intended to support such machines may serve as a hypervisor user program without software modification. The user interface presented by a hypervisor is commonly called a *virtual machine*; the term "subsystem" may be applied to the complex of software used within a virtual machine.

The first practical hypervisor systems emerged in the mid-1960s.[1-3] CP-40 (Control Program 40) was a hypervisor system which presented virtual machines that were compatible with the IBM System/360 computers.[4,5] The first package designed specifically for use as a CP-40 virtual machine subsystem was the Cambridge Monitor System (CMS), a single-user interactive operating system. When they were completed in 1966, the two systems were combined to form a time-sharing system which established the structural basis for the current VM/370 system.

CP-40 and CMS were developed for an IBM System/360 Model 40 that had been specially modified to perform dynamic address translation.[6] This CPU hardware feature is used in conjunction with software system support to create virtual storage.[7-9] Dynamic address translation was not available on other System/360 Model 40 computers, and so CP-40 ran only on one machine. During this same period the IBM System/360 Model 67 was being developed.[7,10] This computer supported virtual storage through a different form of dynamic address translation, and was to be offered as a product by IBM. In 1967, CP-40 was modified and extended for the System/360 Model 67, and the new version of the hypervisor was named CP-67.[3,11-13] CP-67 achieved popularity as an internal IBM software development support system, was made available by IBM for use as a general-purpose time-sharing system, and is still in use at several installations today. In 1972, CP-67 was extended and adapted for use with the IBM System/370 line of computers. The result was a new main-line IBM operating system named VM/370[14] which is now in widespread use.

The original objectives of the CP-40 project were to investigate time-sharing techniques, to develop a time-sharing system for the IBM System/360, and to examine hardware requirements for time-sharing computers.[1] As the virtual machines of CP-40 and CP-67 demonstrated their usefulness in practice, the curiosity of those concerned with the development of the systems centered on two areas of interest. One of these areas dealt with performance matters in general, and focused on feedback of dynamic activity characteristics as a means of optimizing overall system resource scheduling. The other area addressed approaches to system realization through careful design for evolution and mobility.

This article will treat the latter theme, particularly in the realm of computer networking. The intent of the authors is to illuminate the process that has led to a number of concepts and functions that are now a part of VM/370. The character of this process has been one of discovery rather than invention, and in some cases the results have come as a surprise to all involved. As the evolution proceeded, its direction changed markedly, and its consequences and objectives became more clearly articulated. This article will attempt to follow that process and accurately convey its quality.

Origins of virtual machine subsystems

The designers of CP-40 intended the function of the hypervisor to be restricted to the management of real machine resources for active users.[1] Software support services such as language translators, file systems, interactive monitors, and the like were to operate at the virtual machine level wherever possible. The ratio-

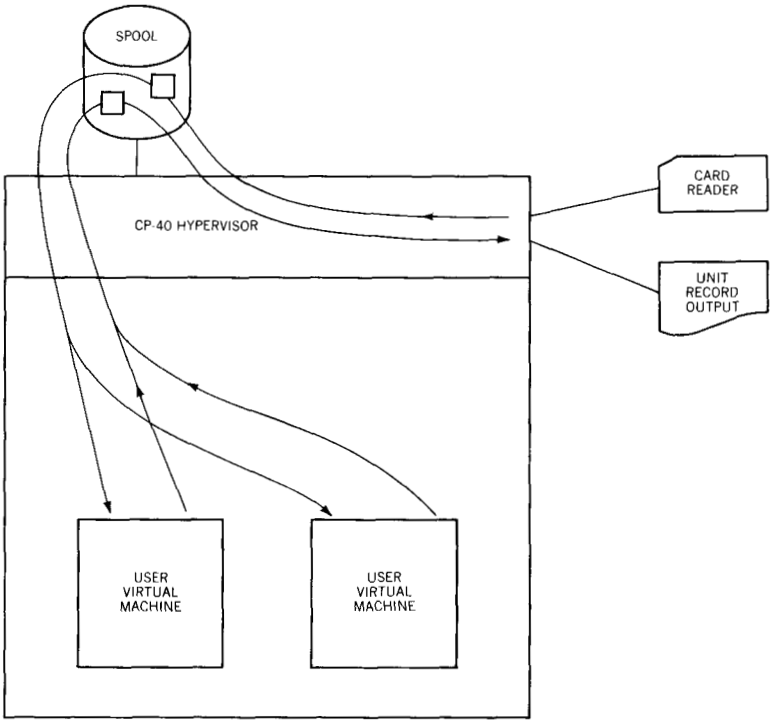
nale behind this separation of function lay in the belief that strict adherence to an interface that had been carefully designed and documented as the *IBM System/360 Principles of Operation*[5] would have the effect of limiting the size and complexity of the hypervisor to a manageable level. At the same time, the functional generality of this interface would be adequate to support any application software that could be used with a standard IBM System/360 computer. Indeed, one could use such application software together with its normal operating system in a virtual machine without requiring any program modifications.

For the most part, these objectives were met. The several restrictions imposed by the hypervisor on the standard IBM System/360 I/O interface rarely gave rise to practical difficulties in adapting systems and applications from a real to a virtual machine environment. CP-40 placed most application functions at the virtual machine level, but technical design problems caused several varieties of I/O access method support to be included in the hypervisor. Translation functions between real and virtual devices, and interactive terminal handling accounted for much of this support. A third such area was the CP-40 spool system which fulfilled the requirement for virtual machine unit-record I/O access. This spool system included unit-record device simulation, a direct access storage device (DASD) file system, and real unit-record device support.

The presence of these functions in CP-40 was recognized as being arguably contrary to the proscription of support functions from the hypervisor, but an overriding objective was to produce a practical system that could be used to evaluate the conceptual approach. Refinement of the hypervisor to the most basic functions would have delayed its availability and imposed a requirement for one or more virtual machine support subsystems for even the simplest operation. Moreover, lack of familiarity with virtual machine systems suggested a more conservative approach that sought practical results as a guide to further design. The development of CP-40 was accomplished quickly, and its transformation to CP-67 for the IBM System/360 Model 67 followed in about a year.

While CP-40 functioned without requiring virtual machine subsystems, the facilities it offered in the absence of such subsystems were scarcely more useful than those provided by a computer without any programming at all. CMS was intended to furnish most interactive user support functions. CMS emerged concurrently with CP-40 in 1966, but was developed separately using the same dedicated real computer. Although this real computer was used in its early development, CMS was intended specifically for eventual use as a virtual machine subsystem monitor. When CMS and CP-40 had both become operational, the adaptation of

Figure 1 Original CP spool

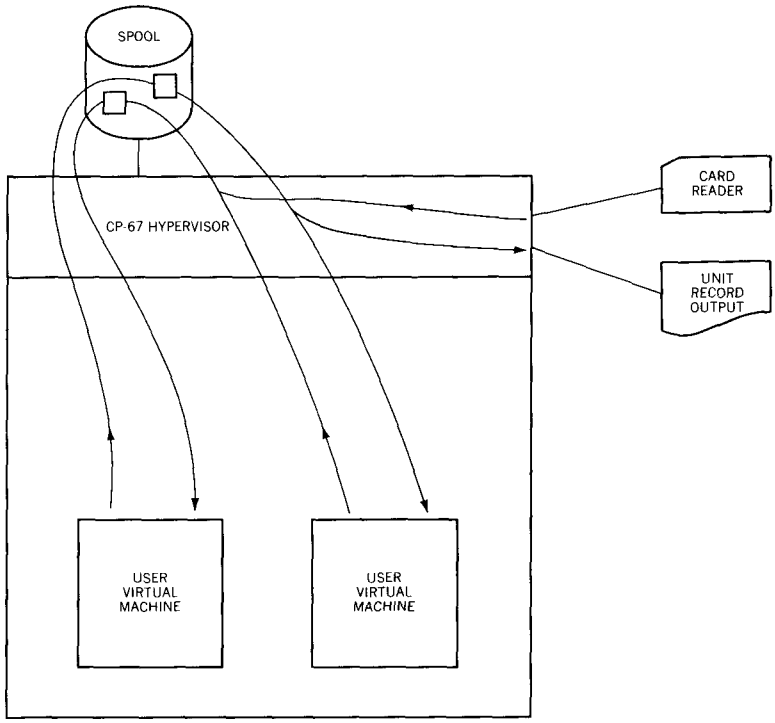


CMS to a virtual machine environment was smooth and natural. Other systems such as OS/360, which had been developed exclusively for use as real IBM System/360 native operating systems, were similarly adapted with little difficulty.

functional extensions

Operational experience with CP-67 and CMS naturally stimulated ideas for functional extensions to the hypervisor. The original CP-67 spool system shown in Figure 1 allowed users to create files only from spool representations of real card decks that had been read on real card readers, and to create spool representations of private files only for ultimate output to real printers or card punches. In 1968, the hypervisor was modified so that a virtual machine could directly access a spool file that had been created by a virtual machine as output, without involvement of any real unit-record I/O devices, card decks, or listings (see Figure 2). This simple extension amounted to the first rudimentary form of communication between virtual machine programs, and it led promptly to a new type of virtual machine application subsystem monitor. The implications of these developments have heavily influenced the evolution and use of VM/370, and the unfolding consequences continue to yield new ideas for virtual machine subsystem application and design.

Figure 2 Direct spool file exchange



Given the ability to exchange spool files, CMS users could readily employ previously existing CMS file utility functions to exchange any of their private files. The same combination of functions would allow file transfer between CMS users and special virtual machine subsystems supplying common services to interactive CP-67 users. OS/360 and a modified version of CMS had both been used as batch-processing subsystem monitors running in CP-67 virtual machines prior to the availability of direct spool file exchange between virtual machines. Input to these batch subsystems was initially limited to card decks that were read by real card readers, and output was produced exclusively on real printers and card punches. When the spool file exchange facility was installed in CP-67, these batch subsystem monitors were quickly adapted to utilize the new function, making their services much more attractive and available to interactive CMS users. Figure 3 depicts the virtual batch subsystems. These virtual machine batch subsystem monitors were most significant in that they formed a conceptual basis for the extension of CP-67/CMS user functions without requiring modification of CP-67 or CMS.

CP-67 exhibited performance characteristics that were comparable to those of any general-purpose interactive system at the time.

Figure 3 Virtual batch subsystems

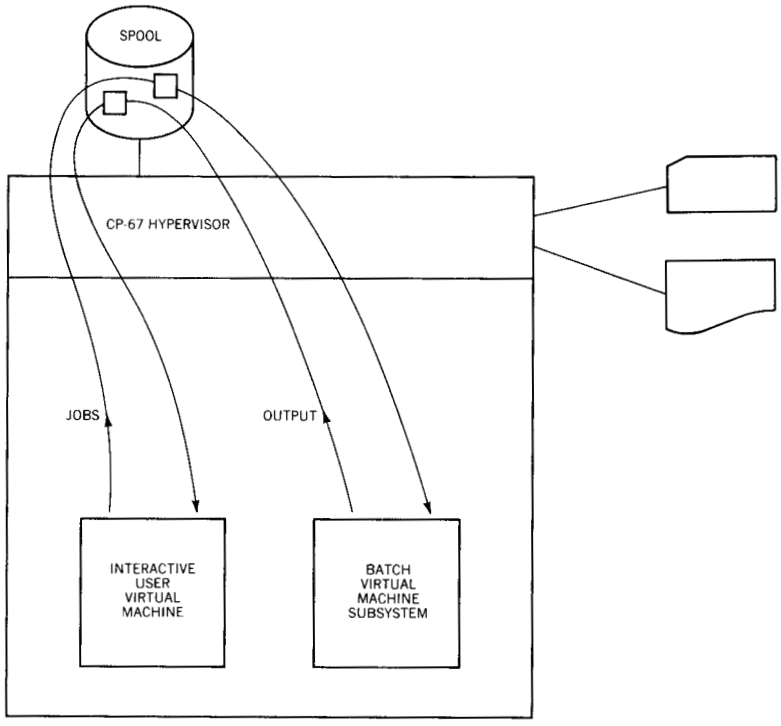
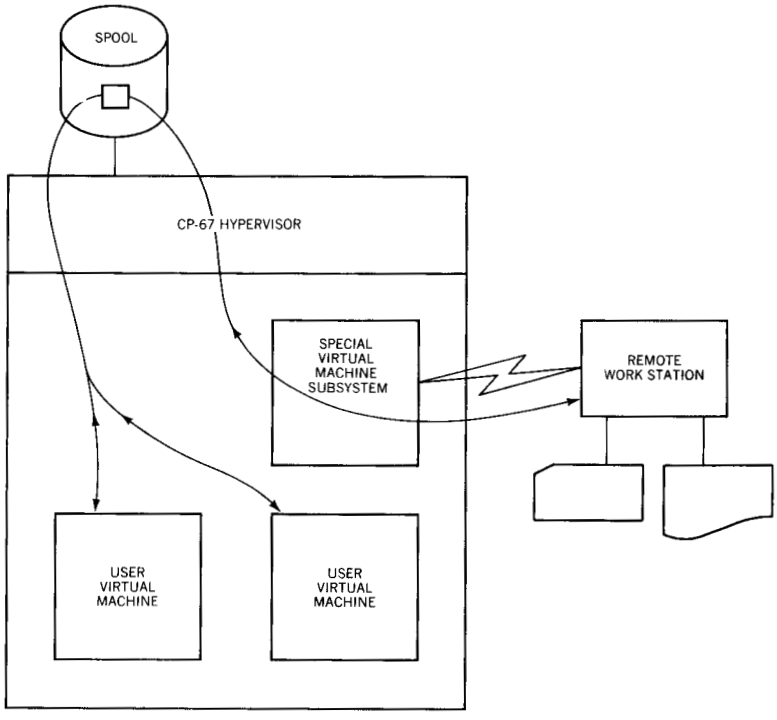


Figure 4 Virtual machine work station subsystem



Since it also seemed to multiply the capacity of the computer by creating a separate virtual machine for each terminal user, system programmers usually viewed CP-67 as highly efficient. Systems work that had previously required a large, dedicated computer, which meant contending for serially scheduled late-night sessions, could be accomplished using CP-67 at any convenient time without precluding the use of the computer by others. These advantages, combined with the general attractiveness of interactive computing, suggested a potential for virtual machines beyond the original objectives of CP-40 cited above.

From the point of view of the developers, recognition of the possibility of a long evolution of virtual machine hypervisor systems stimulated interest in development techniques that could preserve and enhance their attractiveness. One perception drawn from the early successes of CP-40 and CP-67 was that the original intent of limiting the complexity of the hypervisor by excluding as much application support function as reasonably possible had succeeded. This perception inspired a bias against implementing new function (e.g., user directory management) in the hypervisor where the need could be addressed through software at the virtual machine level.

Early objectives

By 1969, the increasing use of CP-67/CMS and dependence on it began to draw new attention to the existing functional deficiencies of the system. One such area of deficiency was bulk telecommunication. Interactive user terminals were often located at some distance from the computer installation, but CP-67 did not support spool access by remote bulk terminals or work stations. This meant that users who needed to have card decks read or punched, or listings printed, would have to do so using only the real unit-record I/O devices at the computer installation, and that arrangements for transportation of the decks and listings were required.

The CP-67 hypervisor extensions allowing virtual machines to exchange spool files offered the possibility of a virtual machine subsystem for remote work station access to the spool system without further hypervisor modifications (see Figure 4). As the direction was taken to place bulk telecommunication support in its own dedicated virtual machine subsystem, the focus of attention shifted somewhat. Functional extensions to CP-67 had never before been attempted through development of specialized virtual machine subsystems, and so the feasibility of the technique was uncertain. The project came to be viewed as an experimental effort to shed some light on these questions, rather than to produce a functionally useful package for distribution. It was hoped that

further work along similar lines with heavier emphasis on practical utility would follow if the approach showed promise.

design problems

The first design problem to arise addressed the choice of a virtual machine supervisor system on which software could be built as an interface between the CP-67 spool system and remote work stations. At the time, experience with virtual machine subsystems was limited to CMS and systems that had been designed for use as native real machine supervisors, mainly OS/360. A second initial design problem concerned establishment of a means for interactive users to direct their spool output to particular remote work stations. The original CP-67 spool system included no file attributes such as the VM/370 spool class and distribution code which could serve the purpose, and the desire to retain complete transparency of spool data content argued against the use of imbedded data control records resembling job control language (JCL) or identifier (ID) cards.

CMS had been designed to be a single-user monitor system, featuring extensive support for interactive applications and a fixed-block DASD file system. Likewise, OS/360 was intended as a multiprogrammed batch job execution system. The several other operating systems that were used under CP-67 had evolved on real machines to meet their particular objectives as well. None of these systems was designed for use as a base for a virtual machine subsystem of the kind envisioned, and each of them included extensive support for functions that would be of no use in the intended environment.

performance concerns

The first assessment of difficulties likely to arise in an attempt to develop basic system support using a virtual machine subsystem placed primary concern on questions of performance. An earlier experimental telecommunication system named CLMON had been developed in late 1968 to support data exchange between an IBM 1130 system and OS/360 running in a virtual machine.[15] CLMON had demonstrated that telecommunication timing problems arising from possibly delayed response of CP-67 in servicing a virtual machine could be conveniently managed without modifications to the hypervisor. Performance concerns at that time dealt with overall system implications and divided into three areas: potential for excessive virtual storage demands, additive execution overhead due to cascaded supervisors, and unproductive hypervisor loading associated with spool device simulation.

It was observed that a simple means of directing spool output to particular remote work stations could be afforded by driving each of the remote work stations from a different and independent virtual machine subsystem. Users could then implicitly select destination work stations by explicitly directing their spool output to the particular virtual machine subsystem responsible for manage-

ment of the desired work station. This approach seemed attractive because it imposed no requirement for control records in user spool data, and no modifications to the hypervisor were required. Furthermore, the approach seemed natural because the virtual machines serving as work station support subsystems could each be given standard one-to-eight-character virtual machine IDs suggesting their associated geographical destination, making the users' specification of file routing somewhat self-descriptive.

While replication of subsystem software in multiple virtual machines allowed easy file addressing and software tailoring to individual requirements of particular remote work stations, it also threatened to impose an unduly heavy paging load on CP-67 resulting from the addition of multiple virtual storage spaces. Recognition of this possibility discouraged the use of an existing supervisor as a base for the new virtual machine subsystem. Most of the execution overhead and storage requirements associated with these supervisors were related to support for unneeded functions. These effects could obscure experimental results by introducing extraneous overhead that would be difficult to isolate and measure. Consequently, an entirely new and highly specialized virtual machine subsystem would be developed to serve as its own virtual supervisor and to require as little virtual storage as possible.

experimental effort

The initial experimental effort was to support a locally situated IBM 1130 system as a remote spool work station since it was readily available and much of the required software support for the work station system was already completed. The aforementioned CLMON system for the 1130 computer presented a nonstandard protocol interface that was compatible with Binary Synchronous Communication (BSC)[16-18] controllers and adapters, but incompatible with BSC terminals and other application interfaces. The protocol was designed to quiesce line activity when message exchange was idle; it was symmetrical in that it included no master-slave distinction between the communicating machines, and would normally communicate with an identical copy of itself.

The CP-67 spool system used a compressed data format to increase effective DASD capacity. The same kind of data compression would increase effective telecommunication line throughput. Data blocks could be moved directly from the CP-67 spool through the telecommunication interface to the remote work station in compressed format, avoiding intermediate decompression and recompression overhead. Going in the opposite direction, however, direct entry of packed data from a remote work station into the CP-67 spool system could impact compatibility by establishing an interprocessor spool format standard other than the universally accepted unit-record image. Since CP-67 logic treated spool data blocks as internally generated items, format errors would be in-

terpreted as internal system errors. Therefore, any new provision for user-level software generation of such blocks would either compromise system integrity or require new verification and error-handling logic in the hypervisor.

After consideration of all the factors, it seemed worthwhile to modify the CP-67 hypervisor to allow virtual machines to read, but not to write, compressed data blocks directly from the spool system. This interface appeared to the virtual machine as a normal virtual card reader, except that compressed spool data blocks rather than ordinary card images were transferred to virtual storage. This approach minimized modifications to the hypervisor and significantly reduced potential execution overhead. The use of a symmetrical protocol implied that compressed data blocks would be transmitted in both directions and decompressed by virtual machine subsystem software on reception. The ultimate result of this design approach was that the CP-67 spool data block format was adopted as the communication data format as well.

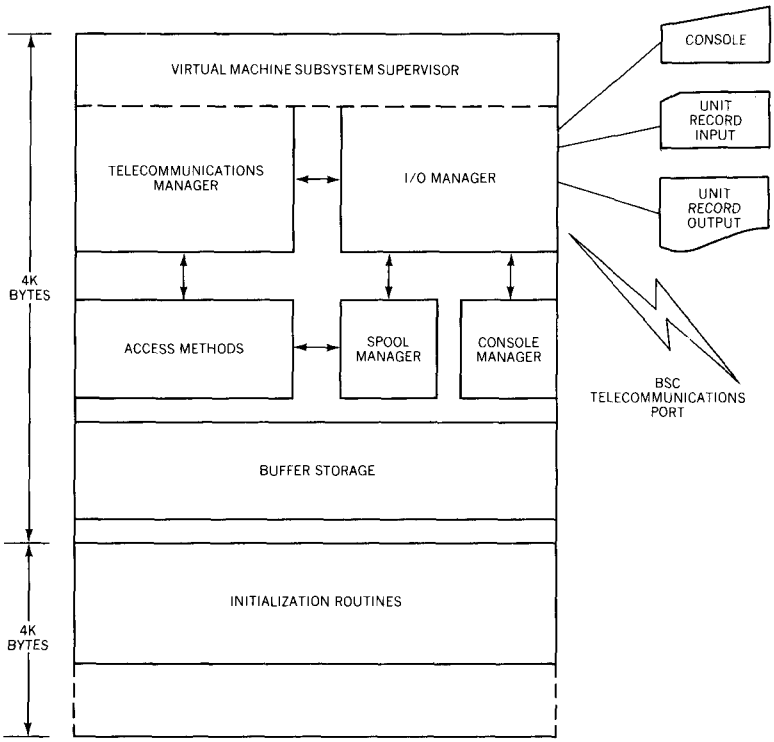
CPREMOTE

These design considerations led to the CP-67 virtual machine subsystem named CPREMOTE, a single assembler language module requiring only the 8K-byte minimum virtual storage size for CP-67 virtual machines and using a single 4K page after initialization. CPREMOTE included its own rudimentary supervisor which was partly integrated with the functional support logic. I/O operations were performed at the channel program level with virtual Start I/O (SIO) instructions, and the necessary error recovery logic was included within the body of the program. Figure 5 illustrates the CPREMOTE structure.

CPREMOTE presented clear evidence that the common virtual machine furnished a healthy environment for the growth of specialized subsystems. The modification to the hypervisor might have been avoided at some cost in the form of increased unproductive overhead, but it pointed out the fact that development of new types of functional support subsystems could place new requirements on the virtual machine interface. It was later to become evident that some of the design compromises necessary to avoid further modifications to CP-67 significantly impaired the practical usefulness of CPREMOTE.

Notwithstanding these qualifications, the initial results were encouraging. Development and debugging were facilitated by the isolation of the virtual machine environment, the relatively small virtual storage dumps, the simplicity of the virtual subsystem, and the utility of the CP-67 user console functions. The use of a symmetrical protocol facilitated testing by permitting concurrent op-

Figure 5 CPREMOTE structure



eration of two independent CPREMOTE subsystems within the same CP-67 system connected by an external telecommunication link. Performance of CPREMOTE was most encouraging in that its virtual machine was given no special scheduling or dispatching priority by CP-67. The system load imposed by CPREMOTE was relatively modest, throughput was only slightly below line speed limitations, and response was quick. All these factors were comparable to those one might have expected had the support been integrated directly into the hypervisor.

operational utility

Despite its success as an experimental prototype, the operational utility of CPREMOTE was limited. The requirement for separate virtual machine management of separate lines effectively prevented convenient operator control. The absence of commands and messages proved to be a more serious detriment than had been anticipated; operators couldn't independently determine whether a line was working, and could do little to identify and correct problems even when they were known to exist. While most local users preferred to submit card decks and to retrieve output directly from the computer room rather than to use the work station facilities of CPREMOTE, the alternatives available to physically distant users were less attractive. In response to requests from these users, CPREMOTE was modified to scan card

input data from work stations for ID cards and automatically route files to the specified virtual machine users. The rest of the deficiencies were tolerated, and CPREMOTE quickly found some limited use.

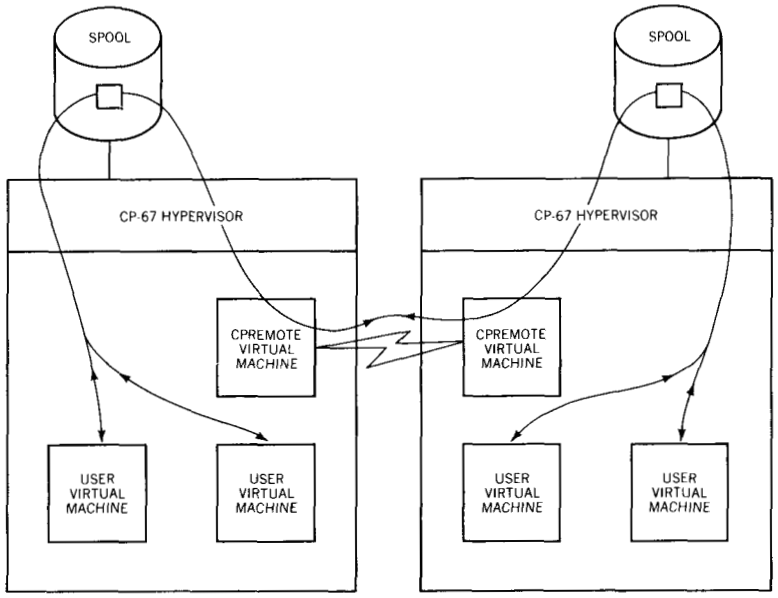
When CPREMOTE was completed in mid-1969, the use of CP-67 was only beginning to spread within IBM. As more CP-67 installations appeared during 1970, some unexpected trends in the use of CPREMOTE began to materialize. An increasing number of CP-67 users began to rely on the remote work station support, and its use became part of regular system operations at some locations. The availability of work station facilities naturally suggested further possibilities. As an initial response to requests for OS/360 compatible support, CPREMOTE was modified for use as an ordinary task under OS/360. This supported some limited communication between CP-67 and OS/360, but there was no facility for job entry from CP-67 or for job output transmission from OS/360.

improved use

Many of these operational shortcomings were removed through independent development work by system programmers at the locations where the functions were needed. CPREMOTE was integrated into a HASP system so that CP-67 could serve as a HASP work station for OS/360 job entry and job output processing. Most installations that used the package added commands and messages to CPREMOTE and developed a number of other functions to improve usability. When VM/370 became available in late 1972, CPREMOTE was adapted to the new VM/370 spool data format without changing its telecommunication interface. CPREMOTE was used as a base for the development of a number of different telecommunication support packages for CP-67 and VM/370, including several for the IBM 2780 bulk telecommunications terminal. One of these was released with VM/370 under the name SRP2780 shortly after the original release of the system.

The most interesting use of CPREMOTE was in the transfer of CMS files between users of different CP-67 operating systems as shown in Figure 6. The possibility of employing CPREMOTE for communication between two CP-67 systems had been foreseen, but its usefulness had seemed to center on testing of work station support without requiring actual work stations. In order to accomplish file transfer between CMS users of different systems, the sending user was required to insert the standard CP-67 ID card specifying the destination user ID into the spool representation of each file to be sent. This process could be accomplished using the CMS context editor, but it was awkward and could not be applied to listing image files. In spite of these reservations, the transfer of CMS files using CPREMOTE often proved to be faster and more convenient than any available alternative. The importance of this result was in pointing the way to the user file exchange facility that was to emerge.

Figure 6 CPREMOTE user file exchange



In the two years following its completion, the character of CPREMOTE had shifted almost spontaneously from experiment to utility. This change in direction was unexpected and not entirely welcome; the operational restrictions of CPREMOTE had been justified on the basis of its experimental nature, but they were seriously hampering the use and growth of the function. The popularity of CPREMOTE was attributed in part to an unforeseen demand for intercomputer spool-based communication. Virtual machine residence and symmetrical protocol appeared to have contributed enough flexibility and portability to offset the serious practical flaws of CPREMOTE. When these observations were combined with ideas offered by users, it began to appear that a CP-67-based spool network connecting multiple computers might be realized. Finally, the limitations of CPREMOTE were so fundamental and severe that it was discarded in 1971 in favor of an entirely new subsystem serving the purpose.

SCNODE

The increasing popularity of CP-67 did not give rise to the development of many specialized virtual machine subsystems as the designers had hoped and expected. To the contrary, the demand for functional extensions was being met by repeated modification and enlargement of the CP-67 hypervisor and CMS, and the structural separation of the two was becoming less distinct. In rejecting specialized virtual machine subsystems for system functional exten-

sions, users and system programmers often cited doubts regarding efficiency and performance, the functional limitations of the supervisor base of CPREMOTE, the need for CMS file system functions, and an intuitive feeling of awkwardness toward the approach. As design work began in mid-1971 on a new network support subsystem to replace CPREMOTE, the intent was to produce a package which would effectively counter these objections, offer a base for other kinds of virtual machine subsystems, and ultimately encourage others to exploit the technique for system development work of their own.

The main features of the design strategy adopted for the new subsystem were to avoid requirements for special hardware or extensive modifications to CP-67, and to build a structure that could be easily adapted to changing interfaces as the needs arose. The objective was to install a prototype spool-based network that would generate some operational experience. The resulting feedback would help direct decisions concerning functional extension and modification to the network support. Because of their common availability, voice-grade telephone lines and standard BSC controllers were to be used for data exchange between locations. The name SCNET (for Scientific Center Network) was applied to this planned network as a whole, and the first version of the new virtual machine subsystem was named SCNODE.

message switching

The five years prior to the start of work on SCNET and SCNODE had seen the emergence of some general-purpose "resource-sharing" computer networks offering integrated support for both interactive terminal sessions and bulk data transmission.[19-23] These networks employed variations of a design technique known as "message switching" in which any form of communication between computers would be formatted into standard data blocks called *messages* for presentation to the network interface. These messages might or might not be automatically broken into smaller units called *packets* for transmission to their destinations as relatively independent entities.[24-26] Upon the assessment of the results of these efforts, it seemed that some of the thorniest problems that were encountered could be avoided by exclusion of support for interactive terminal sessions.

There was little apparent reason to expect to provide good interactive network terminal response without using high-speed lines and dedicated real controllers which could maintain reliable timing characteristics, and there was no reason to believe that such equipment would become available. Interactive terminal support through a virtual machine subsystem interface would have meant extensive and unprecedented modifications to the CP-67 hypervisor. Besides, no practical means could be foreseen for interactive computer users to maintain multiple accounts permitting access to various systems administered by independent computer

operations organizations. To increase the likelihood of success by simplifying the problem, the objectives of SCNODE were limited to file and message exchange, and interactive terminal session management was left to other efforts that were under way at the time.[27]

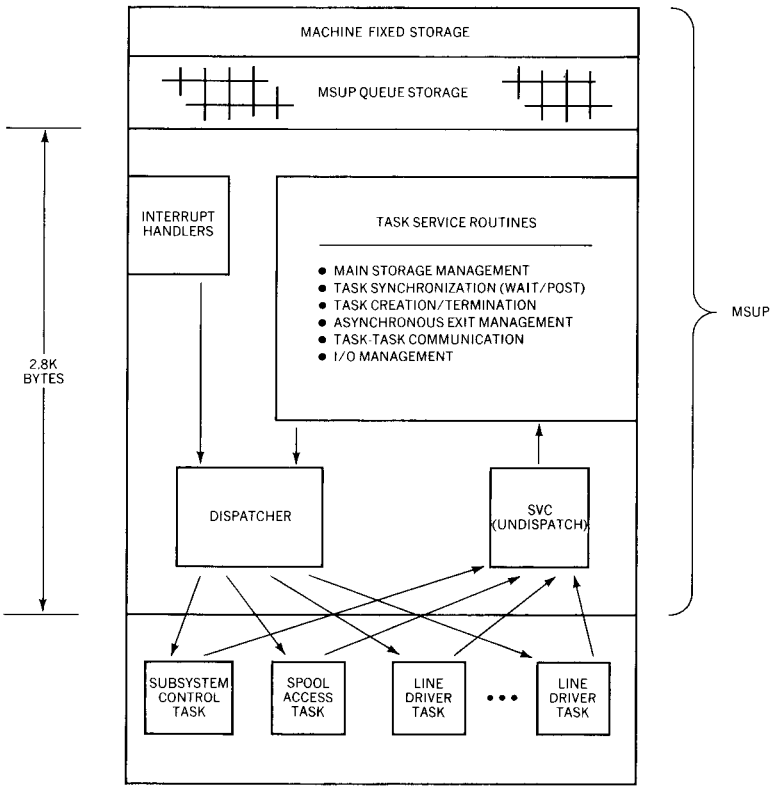
subsystem supervisor

The first step was to design a virtual machine subsystem supervisor base to replace CPREMOTE. Managing individual lines from separate virtual machines was abandoned because of the difficulty of coordinating store-and-forward exchange among multiple virtual machines within a single system, and to alleviate the operational awkwardness of CPREMOTE. The resulting need to concurrently manage multiple remote interfaces using a single virtual machine subsystem meant that the new subsystem supervisor would have to support multiprogramming and multiple concurrent I/O operations. To meet the need for natural adaptation to dissimilar remote interfaces, all functions related to such interfaces were isolated as separate monitor tasks called *line drivers* which would present a standard program interface to the inboard system. To produce a basic supervisor applicable to a variety of possible subsystems, all application-related function was excluded from the supervisor. In particular, all functions that would depend on special CP-67 interfaces would be placed at the task level so that the supervisor could be used on a real IBM System/360 compatible computer.

The name MSUP was given to the resulting subsystem supervisor.[28] MSUP was developed as a collection of small assembly language modules, each of which manages a particular supervisor function. MSUP supplies basic I/O management, task creation and deletion, main storage allocation, asynchronous exits, and two varieties of intertask communication. Task synchronization with events external to the task is organized through a basic WAIT and POST function analogous to that function of OS/VS,[29,30] and dynamic task status is maintained by a simple dispatcher. By distilling all of these functions to their basic elements, supervisor execution path length was minimized, and main storage residency requirements for MSUP code were limited to less than 3000 bytes. System support functions for the SCNODE subsystem were developed as two independent monitor tasks identical to line driver tasks for purposes of MSUP management (see Figure 7).

SCNODE would use the CP-67 spool system to retain files from local users, or files in transit from remote locations, prior to their further transmission. CP-67 virtual machines could concurrently read multiple spool files, but there was no means for file addressing to remote locations, nor any facility for inspection and reordering of input spool file queues. These functions were required so that appropriate files could be selected for reading and transmission on particular lines.

Figure 7 SCNODE internal structure

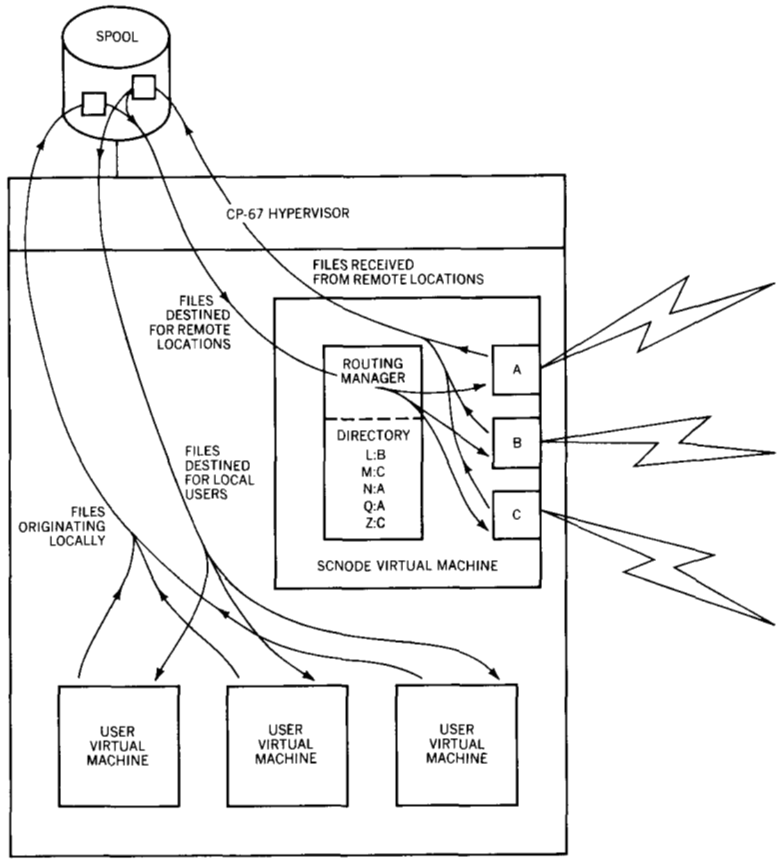


network routing

The CP-67 spool system had presented a kind of communication network among the local virtual machine users of a single system. CPREMOTE had suggested the possibility of logically extending that network to encompass the users of multiple systems. The addressing scheme employed by CPREMOTE had evolved into two separate levels: the system location address which had been implied by the virtual machine ID of the CPREMOTE subsystem that managed its remote line, and the destination user ID at that system which had been specified by an ID card imbedded in the file data. After careful analysis, it was determined that this kind of two-level address would suffice for the purposes of SCNODE. Each physical system on the network would have to be assigned a unique one-to-eight-character EBCDIC identifier. SCNODE would need to maintain a simple directory of all defined destination location identifiers specifying the next location on the path for each entry. In this manner a fixed path between each ordered pair of network locations would be defined collectively among the locations on the path, and routing control would be distributed as shown in Figure 8.

After adoption of this network routing scheme, some mechanism was sought for users to address their files conveniently and for

Figure 8 SCNODE spool store-and-forward



SCNODE to inspect the addresses and to sort its input file queue efficiently. A number of requests for user-specifiable file attributes and new spool functions had arisen in other contexts, and the need for a number of file attributes could be readily foreseen in addition to the need for the destination address for use by spool support subsystems. To meet these requirements, CP-67 was extended to include a string of character data called a "tag" to be logically attached to each spool file. The tag would be set by a user and interpreted by a subsystem but not by the hypervisor, permitting the tag to be used differently by different subsystems operating concurrently within a single CP-67 system.

The need for more extensive facilities for inspection and manipulation of the input spool file queue was not limited to SCNODE. Similar functions had been requested by virtual machine terminal users for various applications. A natural means of supplying these functions would have been through extensions to the CP-67 user console functions. Since console functions had for some time been executable from virtual machine programs, this approach

would have satisfied the requirements for SCNODE as well. After careful consideration, significant extensions to the CP-67 command language at that time were rejected. It was feared that once those extensions were made they would gain wide use quickly and would be impossible to withdraw should experience prove them to be undesirable. The preferred approach was to seek a means to gain such experience at less risk to the healthy development of the CP-67 user interface.

As an alternative, an imaginary spool controller device simulation was experimentally added to the CP-67 hypervisor in 1972. The virtual spool controller behaved as a hybrid device which had some DASD and some unit-record device characteristics, but which had no real machine equivalent. The controller presented a set of I/O commands to a virtual machine program for input file queue inspection and manipulation. Using these commands, a virtual machine program could read the tag for any of its input spool files, search its queue for files with particular tag settings, reorder the queue, and selectively purge files.

network prototype

With this much of the design in place, a working prototype of the new networking subsystem was developed. CPREMOTE was modified to operate as an SCNODE line driver and automatically construct ID card images from the destination user IDs specified in the tags of files to be transmitted. At this point the package was sufficiently complete to meet its first objective as a replacement for the original CPREMOTE program, and a small network of CP-67 systems and remote work stations was established using SCNODE in mid-1973.

Even before this combination was put into regular operation, SCNODE was in use within IBM in a different configuration. With the imposition of some interim restrictions to the user interface, the spool access logic in SCNODE was altered to operate without the spool modifications to CP-67 to improve its portability. The System/360 stand-alone HASP work station was modified to serve as an SCNODE line driver, and was first put into operation with SCNODE in March 1973. This package was used at several IBM installations to supply CP-67/CMS users with remote job entry access to separate OS/360 systems using the HASP MULTI-LEAVING protocol.[31]

Remote spooling communications subsystem

As these developments began to yield some useful operational experience with the new subsystem design, most CP-67 installations were converting to VM/370. Many of the spool input queue manipulation functions that had been missing from CP-67 were available as VM/370 user commands, so no serious consideration

was ever given to propagating a CP-67-style virtual controller device into the new system. VM/370 introduced a number of changes to the spool interface presented by CP-67, including the addition of some access and queue manipulation functions required by SCNODE. Adaptation of SCNODE to these new spool interfaces presented no serious obstacles.

functional extensions

Functional extensions to the VM/370 hypervisor to support the subsystem became necessary once again. A major objective was to retain transparency of spool-file attributes through network transmission. In other words, all original VM/370 spool-file attribute settings would be recreated with the file upon delivery at the destination. The use of one of the existing VM/370 spool-file attributes that could be set by the originating user as a network destination address was rejected because of potential impact to existing applications. Instead, the VM/370 spool system was extended to include a general-purpose tag function similar to that of the experimental CP-67 extensions.

The result was the VM/370 TAG command for user setting and inspection of tags associated with virtual output devices and spool files.[32] The virtual output file tag record is entered as the first record of a spool file using a no-operation (NOP) command code. When NOP records are encountered during the reading of a file by an ordinary virtual card reader, they are discarded by the hypervisor and are not presented to the virtual machine program. Real unit-record control units similarly discard NOP command data on file-output processing.

Since VM/370 supported virtual machine program access to uninterpreted spool data blocks for the telecommunication support package SRP2780, NOP data records could be detected and processed through the same interface. Attempts to design a virtual machine subsystem for spool queue management using only this interface for file tag access yielded logic that was inefficient and awkward. This problem was addressed through an extension to the VM/370 Diagnose Read interface called *Successor File Descriptor*[33] for selective program scanning of sections of a virtual machine's input spool file queue without requiring each file to be opened and read. An "all-class" virtual reader function was introduced to VM/370 to facilitate asynchronous notification to the virtual machine subsystem of new spool-file availability.

development of RSCS

The prototype work on SCNODE and the addition of these hypervisor functions led to the development of the Remote Spooling Communications Subsystem (RSCS) component of VM/370.[14,33,34] The intent of RSCS was twofold. On the one hand, it was to replace the remote bulk terminal support of SRP2780 with functional enhancements and extensions for additional types of bulk terminals and HASP MULTI-LEAVING interfaces. On the other hand, it

was to serve as a base for further functional extensions into the intercomputer networking subsystem intended by SCNODE.

The RSCS command language was developed to resemble the existing VM/370 operator command language as closely as possible while supporting the requirements for remote bulk terminal management and spool networking extensions. The existing SCNODE software was used as the starting point for RSCS program development. New logic was developed at the subsystem control task level for RSCS command and message management, system generation and initiation, system error handling, read access to CMS files, common reentrant subroutines, and VM/370 spool system compatibility. A third subsystem control task was added for RSCS, and a CMS-based installation and maintenance system was established using VM/370-standard techniques.

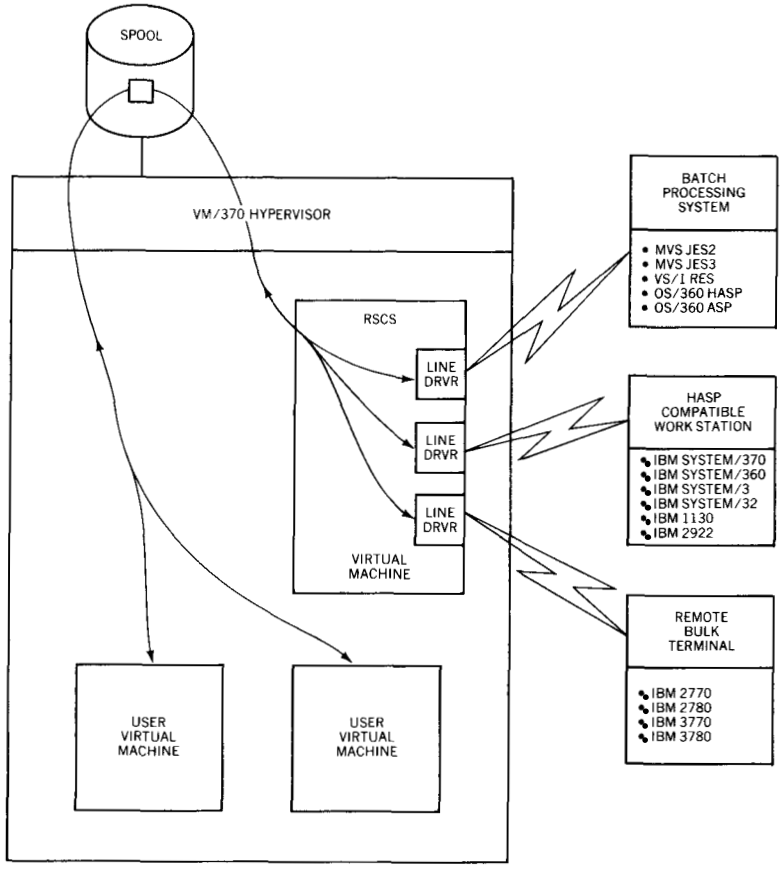
The SCNODE line driver for HASP MULTI-LEAVING was upgraded to handle RSCS commands and messages, and extended to present an interface as either a remote HASP main host or a HASP work station. A CPREMOTE-based virtual subsystem for IBM 2780 remote bulk terminal support, different from SRP2780, had previously been adapted to operate within SCNODE. This line driver was similarly upgraded to handle RSCS commands and messages, and support for IBM 2770, 3770, and 3780 remote bulk terminals was added. These two line drivers, combined with the SCNODE-derived subsystem monitor described above, constitute RSCS as it was originally released with VM/370 in January 1975, and as it is currently available today. Figure 9 is an illustration of RSCS functions.

VNET development

The RSCS development work was done during 1974 by the authors at their respective locations which were separated by several hundred miles. This situation gave rise to a need for quick and convenient exchange of programs, documentation and messages between the two locations. Even before RSCS was ready for internal distribution within IBM, an early version of a CPREMOTE-derived line driver for intercomputer communication was developed and put into operation with the new subsystem using a dialed telephone line connection. To maintain throughput and compatibility with CPREMOTE, the CP-67 spool data-block format was retained for communication between VM/370 systems, rather than the VM/370 spool data-block format which excluded data compression.

The intercomputer communication facility dramatically increased the efficiency of the joint effort, created a convenient test environment for the new software under development, and generated immediate feedback regarding the usability of RSCS as it emerged.

Figure 9 RSCS functions



The advantages of the new communication functions were so valuable that dependence on their availability quickly followed. Demand increased as others learned of the existence of the link and began to use it, and establishment of the dialed telephone connection became a regular operational procedure. The dialed connection was ultimately replaced by a permanent leased line to reduce telephone costs and free the operators from having to regularly dial the connection. In retrospect, this was an early instance of a pattern of interacting demand and availability which was to reappear many times in different circumstances and which has generated the impetus behind the rapid growth of what is now a very large computer network.

the "SUN" network

An independent project at another IBM location was undertaken in 1972 to develop a different spool-based network. This network employed a symmetrical computer-to-computer variety of the HASP MULTI-LEAVING protocol similar to that of the TUCC/IOWA Network[35] for automatic forwarding of jobs, output, and console messages in a manner analogous to that of SCNODE. In early 1974,

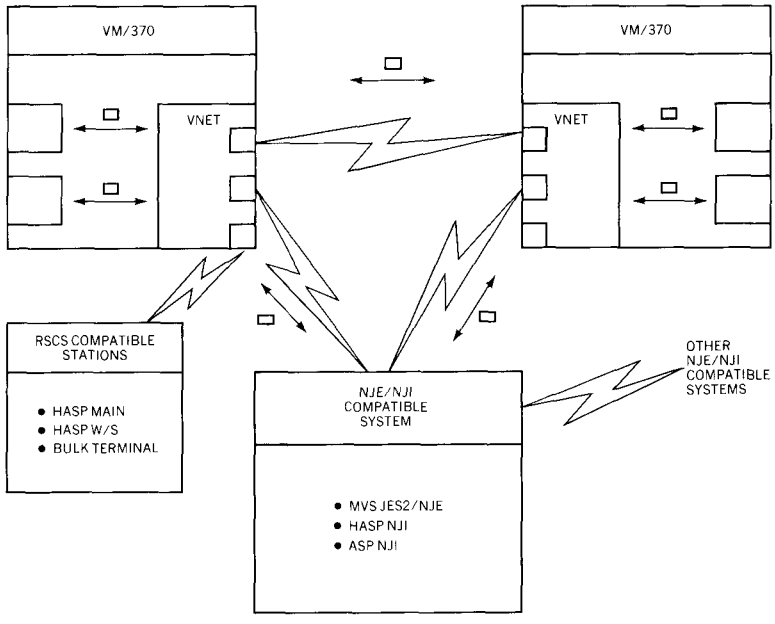
the HASP MULTI-LEAVING line driver for SCNODE was modified to support this new HASP networking interface. The new line driver was immediately put into operational use, and practical experience with a hybrid VM/370 and OS/360 spool network was gained before RSCS development was finished. The functional concepts and interface requirements for this network of dissimilar systems were jointly defined and formalized by the developers, and the emerging internal IBM network was given the name "SUN" (for Subsystem Unified Network).

When RSCS was completed for distribution with VM/370, attention turned to developing network support on an RSCS base for use within IBM. Rudimentary modifications to RSCS implemented indirect store-and-forward routing for spool files, console commands, and messages. The CPREMOTE-derived line driver that supported RSCS development was upgraded for networking interconnection between VM/370 systems and used as a base for a similar channel-to-channel adapter line driver. A modest network connecting several VM/370 systems located mainly in the eastern U.S.A. was installed using the new RSCS extensions in the spring of 1975. Connection with the HASP- and ASP-based job network centered in the west followed shortly thereafter, yielding a nationwide hybrid network of about a dozen mixed VM/370 and OS/360 systems.

This composite network attracted a broad range of users,[36] its installation membership grew steadily, many projects became dependent on its services, and requests for added function increased. In response, the SUN interface was extended and redefined, the Network Job Entry (NJE) facility for MVS/JES2[37,38] was developed to the new interface, and support was upgraded in all of the participating systems. As internal IBM installations converted from OS/360, an increasing proportion of MVS systems began to populate the network through JES2/NJE connections. The combination of VM/370 and MVS in a job networking environment offered users and system programmers the attractive potential of efficiently combining the two systems to develop new approaches to their application problems.

Support of the NJE protocol posed a new technical problem to VM/370 that required a further modification to the hypervisor. The NJE protocol introduced extensive new attribute information for each OS/VS data set imbedded within a single job's output. This meant that in order to avoid separating the data sets of a network job into independent files within the VM/370 spool system, the attribute control information would have to be imbedded within the VM/370 spool file data. To address the problem, the VM/370 spool system was modified to accept NOP commands with data from virtual output devices for entry into spool files being created, similar to the tag records that already existed. The ability to insert

Figure 10 VNET functions



NOP data records into output spool files enables the NJE compatible line driver to encode control information within VM/370 spool files in a manner transparent to users and communication interfaces.

From experience and the feedback from network users, new functions and improved human factors were developed for the VM/370 spool networking subsystem on a continuing basis. Within a year the package of networking updates to RSCS grew very large and unwieldy, and network users became increasingly confused over the differences between the standard RSCS component of VM/370 and the extended version which had largely displaced it within IBM. In an attempt to alleviate these problems, the networking updates to RSCS were merged into a new source program base, and the new package was renamed VNET.[28,39] The HASP and ASP job networking packages were combined with VNET to form NJI (for Network Job Interface[40]), a set of software providing mutual compatibility with one another and with NJE for JES2.[38] VNET was offered as IBM product software with NJE and NJI in late 1976, and is available as such today (see Figure 10).

The IBM Corporate Job Network

The small networks within IBM that led to NJE and NJI have now grown and connected to form the IBM Corporate Job Network. (The term "job" is not used here with precision; in fact, most of the systems participating in this network use VM/370 which has no

job concept in the batch-processing sense, and much of the network traffic represents file and message transfer rather than batch job entry.) Today this network interconnects more than 200 different computers throughout North America and Western Europe, and it extends to several installations in Australia and Japan. The participating systems include practically every model of the IBM System/370 computer line and use a wide variety of communication links from voice-grade telephone connections to high-speed data links, channel-to-channel adapters, and MVS/JES2 multiaccess spool devices. The intercomputer protocols used within the network are an even mix of the extended CPREMOTE protocol, which is used between VM/370 systems, and the NJE protocol, which is used by OS/VS systems for communication with each other and with VM/370. The number of connected work stations, bulk telecommunication terminals, and interactive terminals is such that computer system users at most company locations can access the network.

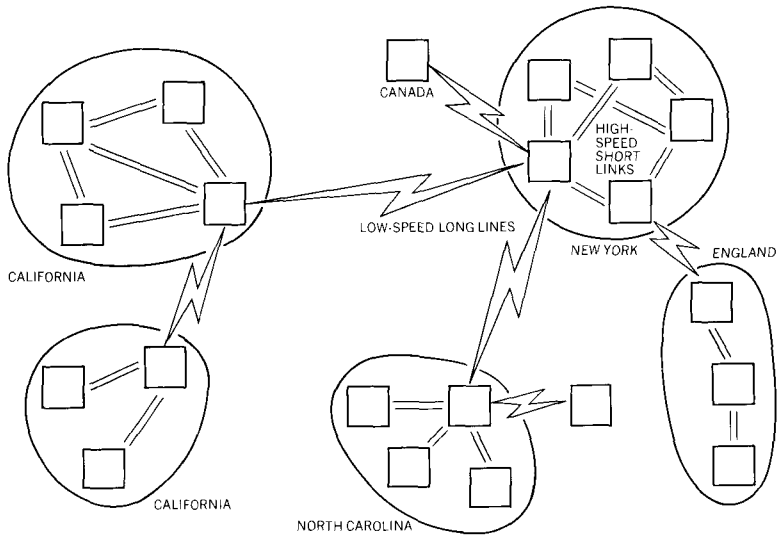
The growth rate of the Corporate Job Network remained almost constant at roughly one computer a week through 1977, but that rate doubled in 1978. The ubiquity of the network is most surprising in that it has materialized quite spontaneously without any explicit mandate or governing organization. In many cases new system connections to the network are made through communication links that had already been installed for use by other kinds of support. Initial justification for connection of a computer system to the network is likely to be made on the basis of experimentation, particular application needs, or contingencies. When a new connection is made, the availability of network communication tends to lead rapidly to its utilization and dependence by diverse projects and applications that had not previously recognized the usefulness of the facility.

hierarchical characteristics

The evolved structure of the Corporate Job Network exhibits some hierarchical characteristics as shown in Figure 11. Local groupings of several computers in close physical proximity are likely to be internally linked by high-speed telecommunication lines or channel-to-channel adapters. These links tend to be employed largely for real batch-job entry and for management of real unit-record output. Slower-speed, long-line telecommunication links interconnect these local groups and tend to be used primarily for file and message transfer. Optimization of network connectivity is usually addressed at the local grouping level, whereas alternate path routing is sometimes used to balance long-distance traffic between the groups. These patterns are curious in that they have formed without explicit design provisions. As such, they may offer useful clues for future design directions.

The administrative organization of the Corporate Job Network bears a striking resemblance to its logical structure. Just as con-

Figure 11 IBM Corporate Job Network hierarchical structure



trol of routing and traffic are distributed across the participating systems on an equal basis, coordination of network maintenance is accomplished cooperatively among the participating organizations. Each installation chooses location IDs for its own connected computer systems and work stations, and these are communicated to the network at large by means of a machine-readable network connectivity map which is distributed using the network. Where more than one routing possibility exists between one location and another indirectly connected location, the choice is made by the local installation according to its own criteria. Such choices can result in looping network paths, so routing decisions must be coordinated with other locations on the potential loops. The network itself allows system programmers to query the routing status of any remote location, so routing decisions impose little difficulty in practice.

network utilization

Network utilization now encompasses practically all areas of IBM internal computer use to the point where the network has become an integrated and indispensable part of normal computer operations. Unfortunately, there is no satisfying answer to the natural question, "What do people use the network for?" One can reasonably expect that any computer application involving a spool system is a potential networking application. Accounting records are kept for all files shipped and are used to prepare periodic network utilization reports. These reports typically show steady increases in file traffic. Communication links that are initially established as dial telephone connections tend to be replaced by leased lines, and leased-line bandwidth tends to expand to accommodate increasing traffic load.

The availability of network communication has spawned an array of application support packages that improve the human factors of the basic network interfaces. The CMS automatic command execution facility (EXEC) is commonly employed to simplify file shipping and job entry for interactive users. A growing selection of application programs for automatic memo and mail composition, delivery, and logging via the network are available within IBM. Another package employs a CMS-based virtual machine subsystem to receive and execute requests from remote users for automatic file retrieval. Several aperiodic newsletters, describing the status of these and other application packages and reporting computer system and language activity in general, are prepared in machine-readable form and distributed to interactive users on the network.

Conclusions

The rapid spontaneous growth of the IBM Corporate Job Network is probably attributable to the design philosophy employed by its software support. The lack of dependence on uncommon computer machinery and VM/370 hypervisor modifications, the ease of installation requiring no system generation or unusual virtual machine specification, the familiarity of the operator command language and procedures for subsystem loading and maintenance, the relative operational independence afforded by the peer relationship of the interconnected systems, and the simplicity of the user interface have encouraged computer installations to invest some effort to give VNET a trial. The simulated unit-record device interface affords operating systems running under VM/370 some access to the network without any software modifications at all. Initial experiences with an unfamiliar computer system interface appear to generate strong lasting effects. A crucial factor in the success or failure of new system software is its cooperativeness with naive users.

The continuing attractiveness of accepted software over a long term hinges on its adaptability to a changing environment. The internal structure of VNET that separates line management functions into disjoint line drivers eases conversion and naturally accommodates individual local modifications. Requirements for coordinating software changes across more than two directly connected network systems would have limited network growth to small groups of installations that could be brought under the control of a single computer operations organization. Adaptability to HASP MULTI-LEAVING, SUN, and, finally, NJE dramatically broadened the appeal of VNET and created new application possibilities for VM/370. Aside from the internal structure of VNET, the isolation of the virtual machine environment and the interactive sup-

port facilities of CMS have streamlined development and testing activities. This in turn facilitated quick incorporation of ideas for improvements as they emerged.

Virtual machine subsystems supplying functional extensions to VM/370 have become common. Such subsystems are widely used to maintain tape libraries and user directories, to help operators manage volume set-up requests, to provide unit-record utility functions, to schedule jobs for virtual machine batch subsystems, to monitor VM/370 system performance, and to adapt VM/370 to a number of interactive networks. Designers of these subsystems have regularly chosen either CMS or OS/VS to serve as a base, apparently to facilitate development. Despite the intent to offer a widely applicable base for virtual machine subsystem programming, the authors are aware of only several cases in which MSUP is used with subsystems other than RSCS and VNET. The unfamiliarity of the internal structure of MSUP and the lack of specific provisions for compatibility with CMS and OS/VS seem to have discouraged its acceptance.

The cost of added system overhead imposed by moving functions to virtual machine subsystems has proven to be far less significant than had been intuitively feared. In general, performance problems that have arisen with VNET have been manageable through minor redesign of virtual machine subsystem logic. Further performance improvements could probably be achieved through modification of VNET and VM/370 hypervisor logic to remove observed bottlenecks, but the bulk of user concern seems to center on functional extensions instead. Performance concerns appear not to be a necessarily good reason for integrating new function into the VM/370 hypervisor rather than implementing it at the virtual machine level.

Constraints on logical coordination imposed by the interface between the virtual machine and the hypervisor are a different matter. The very same interface restrictions that isolate the virtual machine environment and impede the natural increase of overall system complexity[41] give rise to limitations that can hinder functional improvements. It is instructive to observe that nearly all of the modifications to the VM/370 hypervisor that arose from the evolution of VNET have extended and enriched the virtual machine interface; none have been necessitated by poor performance.

The future

Some major areas of unrealized development potential remain with VNET itself. Communication logic could be extended to manage parallel trunks between two locations and multiple logical file

streams on a single trunk. Developing logic to support these functions would be straightforward enough, but devising techniques for scheduling file transmission to take advantage of such capability is not as simple. There is no question that the ongoing transmission of a very long file should not block the transmission of a very short file as it does now. Also, within the context of a single trunk, it is clearly undesirable to multiplex concurrent transmission of identical length files. Logic that could make intelligent scheduling decisions for multiple streams, trunks, and files of various lengths would present some intriguing design problems.

VNET permits operators to change routing manually during network operations in response to failure of computer systems or communication links. Correct decisions concerning rerouting have proven very difficult for operators to make reliably. This situation has stimulated many requests for some form of automatic path selection function in VNET. A clear possibility would be to adopt the Network Path Manager function currently employed by NJE for JES2,[37] and a number of other possibilities exist as well. Logic that could choose among multiple paths to route files toward their destinations presents a significant design challenge in its own right, and it would further complicate the scheduling questions described above.

virtual machine subsystems

Possibilities for continuing work in the broader realm of virtual machine subsystems are even more intriguing. Some work has been done to develop VM/370 spool support for real unit-record devices using a VNET base. A comprehensive virtual machine subsystem to support real spool devices could produce an environment that would encourage speedy evolution of new spool functions. Such a subsystem could even lead to the ultimate removal of real spooling functions from the VM/370 hypervisor altogether, thereby reducing the complexity and extending the useful life of the entire system.

Similarly, some preliminary work has been done in supporting VM/370 interactive user terminals through virtual machine subsystems. While this kind of subsystem function requires a significant new extension to the interface between the hypervisor and the virtual machine, there is every reason to believe that the concept is feasible. Terminal support constitutes a large part of the programming in the VM/370 hypervisor; its removal from the hypervisor and development of techniques at the virtual machine level for selection of installation-specific terminal management would be most beneficial.

The concept of an interface that would allow a virtual machine program to simulate an arbitrary I/O device to another virtual machine (a generalized virtual channel-to-channel adapter) has been discussed for some time. If such an interface were realized, a

virtual machine subsystem could support virtual devices for other virtual machines. As one result, the entire VM/370 spooling function might be made to reside in a virtual machine subsystem. The practical potential of such a proposal can surely be challenged today, but there is no way to guarantee that it would not lead to useful developments of some kind. Given the System/370 I/O architecture, the technical design of the required interface between virtual machines presents serious practical difficulties. Should further study in the area shed more light on the exact nature of these difficulties, the result might introduce new concepts of I/O architecture that would benefit all areas of system design.

independent VNET

There remains the unfulfilled potential to move software originally developed as a virtual machine subsystem entirely out of the VM/370 system. If one were to develop a file system for VNET, this could be readily accomplished without difficulty. A spool system similar to the MVS/JES2 multiaccess spool that could be shared between VM/370 and VNET running on an independent processor would allow VNET to function as a front-end processor, and could greatly improve network reliability as a result. The realization of VNET as an independent system could reopen some lines of development that had been abandoned in the past for reasons of performance and timing dependencies—to wit, integrated bulk and interactive network support.

The prospect of eventually moving virtual machine subsystems outboard to operate in stand-alone mode on real computers refocuses attention on the special hypervisor interfaces required for the subsystem functions. In the case of VNET, these interfaces are mainly the virtual side of the VM/370 spool system. In the general case, the interfaces used by a virtual machine subsystem to interact through the hypervisor with VM/370 users and their virtual machine programs can effectively bind the subsystem to its virtual machine environment. It is useful to consider the hypothetical displacement of a subsystem to a separate real machine environment in the design of these interfaces.

Finally, the various support functions now included in CMS might be broken into separate building blocks and reassembled to form a family of specialized virtual machine monitor subsystems. A common shareable file system similar to the current CMS file system might establish mutual data compatibility among all the members of the subsystem family. Specialized interactive language subsystems, graphics subsystems, data base subsystems, administrative support subsystems, and basic system support subsystems similar to VNET are some possibilities. An environment in which these could be independently developed without losing mutual cooperativeness would realize a long-standing goal of the virtual machine concept.

Virtual machines and the subsystems that evolved to make them useful have already produced some fascinating developments. Further possibilities now seem more promising than ever, and the end is nowhere in sight.

ACKNOWLEDGMENTS

The authors wish to thank the many users, operators, and programmers who tried out unfamiliar systems and who contributed their own ideas and efforts. The following individuals made major contributions to our work: Bob Adair and Charlie Salisbury for their knowledge of system architecture and virtual machines, Norman Rasmussen and Rip Parmelee for their understanding of the innovative process, Craig Johnson and Curt Deegan for managerial direction and support, Jim Sullivan and Harry Von Borstel for their organizational accomplishments, and the creators of CP-40 and CMS, Bob Adair, Dick Bayles, Les Comeau, Bob Creasy, and John Harmon, for their pioneering of the concepts on which our work is based.

CITED REFERENCES

1. R. J. Creasy, *Research Time-Sharing Computer*, IBM Corporation, Cambridge Systems Research and Development Center, Cambridge, MA, Memorandum No. 1 (unpublished) (January 1965).
2. R. J. Adair, R. U. Bayles, L. W. Comeau, and R. J. Creasy, *A Virtual Machine System for the 360/40*, IBM Corporation, Cambridge Scientific Center, Cambridge, MA, Report No. 320-2007 (May 1966).
3. M. S. Field, *Multi-Access Systems—The Virtual Machine Approach*, IBM Corporation, Cambridge Scientific Center, Cambridge, MA, Report No. 320-2033 (September 1968). (ITIRC 68A06950).
4. G. A. Blaauw and F. P. Brooks, Jr., "The structure of SYSTEM/360, Part I—Outline of the logical structure," *IBM Systems Journal* **3,** No. 2, 119–135 (1964).
5. *IBM System/360 Principles of Operation*, A22-6821-7, IBM Corporation, Poughkeepsie, NY (September 1968).
6. A. B. Lindquist, R. R. Seeber, and L. W. Comeau, "A time-sharing system using an associative memory," *Proceedings of the IEEE* **54,** No. 12, 1774–1779 (December 1966).
7. W. C. McGee, "On dynamic program relocation," *IBM Systems Journal* **4,** No. 3, 184–199 (1965).
8. J. B. Dennis, "Segmentation and the design of multi-programmed computer systems," *Journal of the ACM* **12,** No. 4, 589–602 (October 1965).
9. B. W. Arden, B. A. Galler, T. C. O'Brien, and F. H. Westervelt, "Program and addressing structure in a time-sharing environment," *Journal of the ACM* **13,** No. 1, 1–16 (January 1966).
10. *IBM System/360 Model 67 Functional Characteristics*, GA27-2719, IBM Corporation, Kingston, NY (1967).
11. *CP-67/CMS Version 3 System Description Manual*, GH20-0802-1, IBM Corporation, Data Processing Division, White Plains, NY (1970).
12. *CP-67 Program Logic Manual*, GY20-0590, IBM Corporation, Data Processing Division, White Plains, NY (1970).
13. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts," *IBM Systems Journal* **11,** No. 2, 99–130 (1972). This article is particularly useful for its comprehensive bibliography listing many sources of material concerning virtual storage and virtual machines.
14. *IBM Virtual Machine Facility/370: Introduction*, GC20-1800-8, IBM Corporation, Poughkeepsie, NY (December 1977).

15. E. C. Hendricks and D. B. Tuttle, *Notes on Design Objectives and Implementation Under OS/360 of a General Purpose Binary Synchronous Telecommunications Package for Multi-Programmed Applications in OS/360*, IBM Corporation, Cambridge Scientific Center, Cambridge, MA, Report No. 320-2047 (August 1969).
16. J. L. Eisenbies, "Conventions for digital data communication link design," *IBM Systems Journal* **6,** No. 4, 267–302 (1967).
17. *General Information—Binary Synchronous Communications*, GA27-3004-2, IBM Corporation, Research Triangle Park, NC (October 1970).
18. *IBM 2701 Data Adapter Unit Original Equipment Manufacturers' Information*, A22-6844-3, IBM Corporation, Research Triangle Park, NC (June 1969).
19. L. G. Roberts and B. D. Wessler, "Computer network development to achieve resource sharing," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **36,** 543–549 (1970).
20. L. Tymes, "TYMNET—A terminal oriented communication network," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **38,** 211–216 (1971).
21. M. P. Beere and N. C. Sullivan, "TYMNET—A serendipitous evolution," *IEEE Transactions on Communications* **COM-20,** No. 3, 511–515 (June 1972).
22. A. B. Cocanower, "MERIT computer network: software considerations," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 65–77 (1972).
23. M. Schwartz, R. R. Boorstyn, and R. L. Pickholtz, "Terminal-oriented computer-communication networks," *Proceedings of the IEEE* **60,** No. 11, 1408–1423 (November 1972).
24. F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The interface message processor for the ARPA computer network," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **36,** 551–567 (1970).
25. D. W. Davies and D. L. A. Barber, *Communication Networks for Computers*, Wiley-Interscience, New York (1973).
26. D. C. Wood, "A survey of the capabilities of 8 packet switching networks," *IEEE Proceedings of the 1975 Symposium, Computer Networks: Trends and Applications*, 1–7 (1975).
27. R. M. Rutledge, A. L. Vareha, L. C. Varian, A. H. Weis, S. F. Seroussi, J. W. Meyer, J. F. Jaffe, and M. K. Angell, "An interactive network of time-sharing computers," *Proceedings of the 24th National Conference, Association for Computing Machinery* **P-69,** 431–441 (1969).
28. *VM/370 Networking (Programming RPQ P09007) Logic Manual*, LY20-2342, IBM Corporation, White Plains, NY (April 1977).
29. B. I. Witt, "The functional structure of OS/360, Part II, Job and task management," *IBM Systems Journal* **5,** No. 1, 12–29 (1966).
30. *OS/VS2 Supervisor Services and Macro Instructions*, GC28-0683-2, IBM Corporation, Poughkeepsie, NY (April 1978).
31. *OS/VS2 HASP II Version 4 Logic*, GY27-7255-0, IBM Corporation, Kingston, NY (March 1973).
32. *IBM Virtual Machine Facility/370: CP Command Reference for General Users*, GC20-1820-2, IBM Corporation, Poughkeepsie, NY (August 1977).
33. *IBM Virtual Machine Facility/370: System Programmer's Guide*, GC20-1807-6, IBM Corporation, Poughkeepsie, NY (August 1977).
34. *IBM Virtual Machine Facility/370: Remote Spooling Communications Subsystem (RSCS) User's Guide*, GC20-1816-2, IBM Corporation, Poughkeepsie, NY (August 1977).
35. J. F. Walker and W. F. Decker, *TUCC/IOWA Remote HASP to HASP System*, Triangle Universities Computation Center, P.O. Box 12175, Research Triangle Park, NC 27709 (December 1971).
36. R. P. Crabtree, "Job networking," *IBM Systems Journal* **17,** No. 3, 206–220 (1978).
37. R. O. Simpson and G. H. Phillips, "Network job entry facility for JES2," *IBM Systems Journal* **17,** No. 3, 221–240 (1978).

38. *Network Job Entry Facility for JES2, General Information*, GC23-0010, IBM Corporation, Gaithersburg, MD (July 1976).
39. *VM/370 Networking (Programming RPQ P09007) Program Reference and Operations Manual*, SH20-1977-0, IBM Corporation, White Plains, NY (April 1977).
40. *Network Job Interface Programming RPQ P09007 (VM/370), P09008 (ASP), P09009 (HASP), General Information Manual*, GH20-1941-1, IBM Corporation, White Plains, NY (April 1977).
41. L. A. Belady and M. M. Lehman, *Programming System Dynamics or the Meta-Dynamics of Systems in Maintenance and Growth*, Research Report RC 3546, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (September 1971).

Reprint Order No. G321-5089.