RC 9027 (#39545) 9/9/81
Computer Science 17 pages

Research Report

Distributed Processing in a Large Systems Environment

Barry Goldstein
Gene Trivett
Irving Wladawsky-Berger

Computer Sciences Department
IBM T.J.Watson Research Center
P.O.Box 218
Yorktown Heights, N.Y. 10598

IBM Research Division
San Jose · Yorktown · Zurich

RC 9027 (#39545) 9/9/81
Computer Science 17 pages

Distributed Processing in a Large Systems Environment

Barry Goldstein
Gene Trivett
Irving Wladawsky-Berger

Computer Sciences Department
IBM T.J.Watson Research Center
P.O.Box 218
Yorktown Heights, N.Y. 10598

Abstract

This paper is about multiprocessing within an MVS local network environment. Specifically, it describes a multiprocessing architecture that enables address spaces running on any processor of the local network to request and receive service from any other address space running on that processor or on any other processor of the network. In application, a processing hierarchy is proposed wherein the requesting processors are small scale "personal" computers and the service processors are those of large-scale, data base oriented computing systems. The key difference between the architecture presented in this paper and that associated with standard network based systems is that here the application and service appear to reside in the same (requesting) processor. This feature, which gives the small processor the appearance of a large system and provides a basis for migration, is achieved through an extension of MVS herein called the **Virtual Service Interface (VSI).** VSI integrates the set of otherwise independent MVS operating systems of the local network into a single operating system having a single system image.

PREFACE

The following paper does not in any way imply the potential of an IBM product. Rather this paper represents simply the initial observations of an ongoing research effort.

Before we begin, we would especially like to thank Lynn Trivett, George Case, and Andy Heller for their overall advice and guidance.

We would also like to thank Lee Hoevel and Jean Voldman for their help in the initial design of the VSI architecture.

INTRODUCTION

Advancements in technology have provided us not only with the availability of high performance processors with growth potentials to approximately 50 MIPS per uniprocessor (within the next 10 years), but have also provided us with the ability to produce a S/370 on a chip, performing at approximately 200KIPS with 256K to 1 megabyte of storage, **and** at a relatively small cost. Similar developments are occurring with our mid-range processors.

So, while we are seeing a tremendous growth in the processing power at the high end of the spectrum, we are also witnessing the development of mini-370's and faster mid-range machines. One natural set of questions that arises is: what should the operating systems be for this range of processors; and how can a given installation grow to either large numbers of mini's or mid-range processors or in fact to large high end machines (i.e. are there any migration inhibitors)? It will be the intent of this paper to address these concerns.

It was once thought that large numbers of mini's would one day replace the high end machine. It is clear that while processing power is getting cheaper, the cost of peripherals (such as DASD, tape pools, MSS, 3800s) is not proportionately decreasing. The implications are that even if a large number of mini's were purchased the installation could still neither afford to allocate the peripherals per mini nor have effective processing power to efficiently utilize the peripherals. The organization to support the operation of a large operating center is not something that a typical user desires to be confronted with. The high end machine **cannot** be totally **replaced.**

But, one must then ask: why do our installations, which can afford high end machines, also want mini's? There are a number of reasons:

1. Function Isolation - i.e. the loss of a single machine does not effect any other machine. There are no longer fears that a program in development will destroy another application due to unforeseen errors.

2. Reduced Complexity - Given that a mini is intended to support only a single user then the human interface can be made very simple, e.g. no complicated JCL, etc. Furthermore, the user need not negotiate with computer center personnel in the requirements of his applications. In addition, there is a relatively small education time required to learn how to use the mini. Furthermore, system maintenance costs are relatively small.

3. Better performance of the high end machine - By **off-loading** trivial user transactions (e.g. short path length with few I/O requests) the high end machine could devote its resources to more efficient support of its non-trivial transactions, such as IMS. We have seen, for example, on a storage rich system such as the 16meg 3033MP VM system at the IBM T.J. Watson Research Center, that:

   27% of the CPU is devoted solely to trivial transactions

   59% of all page reads,

14% of the data I/O, and

78% of the scheduler work is on behalf of these transactions.

Hypothesis

The operating system currently pays a heavy price for work that could easily be off-loaded.

While mini computers tend to satisfy these concerns they clearly have their limitations. As has been said they are primarily DASD limited and thus preclude very large data base support or large application support. In general, their limitations are those which are easily addressed by high end machines.

The approach taken in this paper is to build a continuum from the mini to the high end machine. The mini will serve as a personal computer and the high end machine (or a conglomeration of high and midrange machines - the computer center) will serve as the **service processor.** What will make this approach viable will be the:

1. the software interface between the mini and the high end

2. the physical connection between the mini and the high end.

In architecting the software protocol between the mini and the high end, one can perceive an analogy between this architecture and the one presented by VM/370 to its virtual machines[1]. VM/370 presents an interface which is the S/370 architecture. This interface enables guest operating systems to believe that they are actually running on real S/370's. Analagously, we will be developing a service architecture that will enable applications, running on mini's, to believe that they are running on a real MVS high end machine, issuing typical high end machine service requests. This philosophy is completely different from the ones standardly chosen for teleprocessing network architectures, which provide software protocols for communication between heterogeneous operating systems. What we are providing is not an architecture for operating system to operating system communication but rather a general architecture for service processing. We view the services of the operating system (MVS) to be extant on all processors in the network providing the appearance of a single system image. For the remainder of this paper, this architecture will be referred to as the Virtual Service Interface (VSI). (Though the actual description of the VSI will be deferred to a subsequent paper).

With respect to the physical connection between processors, we are looking at the linkage being serial links made either of optical fibers (50-400 megabaud) or high bandwidth coaxial cable (10MB seems to be the standard today). Thus, we will not be addressing the global networking application that supports, for example, processors in New York that want to conduct dialogues with processors in Los Angeles. Rather, we will be addressing the local network environment. By employing direct linkages having low error rates, and relatively low noise attenuation, we can assume low software overhead in the processor communication path. Furthermore, we will be designing the connection so that the linkage will appear to be from memory to memory. This will enable us to use memory access as the mode of communication rather than employing synthetic protocols that utilize channels and channel programs. An additional benefit is that two proc-

esses communicating through memory to memory protocols, will run efficiently even when on the same processor or even on two or more processors sharing a common memory.

The logical network structure will be referred to as a **S/370 processing hierarchy** which is based upon the presumption that the user will use the mini for the majority of his needs and be as independent of the high end machine as desired depending upon how many of the services needed cannot be satisfied by the mini. Similarly, the user can either use the mini or directly use the high end machine without any program (or command language) changes. Without a high speed connection between processors such an approach would not be viable as the performance degradation would be exceedingly evident. The mini we are targeting for will have such characteristics as being a 200 KIP S/370 with a minimum of 512K, at least two floppy disks, 24 line display, and 10 MHZ links.

OPERATING SYSTEMS

The basic operating system for the high end machine will be MVS [2], while for the mini there will ultimately be a variety of local control systems but initially the primary emphasis will be on a CMS machine. This choice is based upon the simplicity and user friendliness of CMS.

The objective that we will have in using CMS as an operating system for the mini will be not only to take advantage of the friendly user command interface (and application packages already written for CMS such as APL, EDGAR, XEDIT, QBE, ...) but also to allow applications that were developed for MVS to run on this system (i.e. to have the best of both worlds).

In order to effect this, CMS will have to be modified in order to support MVS application interfaces. The SSI (SubSystem Interface), in particular, will have to be added. The SSI, which is essentially an indirect branch mechanism[3], is the basic interface from applications to such IBM subsystems as IMS, the JES's, and OS data management. By having this mechanism in place, we will be able to have, for example, IMS applications run in the mini with the IMS data base management in the high end machine.

With respect to virtual memory support, CMS is a non-relocate operating system. To achieve relocation, we will either have to modify CMS to support relocation, or presume that VM can fit in a mini, or in fact develop our own small hypervisor for the mini. While we would prefer using VM, pragmatics dictate that we will have to favor the latter choice. The implications of not using VM would imply that we will also have to provide terminal and data physical access method support as CMS does not directly talk with devices but uses a DIAGNOSE interface to VM for said services.

In addition to providing the user with the CMS command set, we will also provide a TSO passthru capability; i.e. allow the user to enter TSO commands which will be passed on to the MVS service processor to be executed. In essence, any command that CMS does not recognize as either a CMS command or exec will be considered to be a TSO command and forwarded on. We will, alternatively, recognize TSO commands through usage of an escape character.

The mini could also support an office product system with the MVS service processor serving both as a repository for archived files and as the focal point for a user's data base.

Similarly, new operating systems could be developed for the mini and connected to the MVS service processor for full service support.

Finally, while we have chosen MVS as the operating system for the service processor, there is no reason why one couldn't develop specialized **offload** engines for specific service requests, such as data base management, printer subsystems, a personal computer, a programmer workstation, etc. The key ingredient is the provision of a clean separation between application and service. It is the intent of the VSI to provide such a separation. Thus operating system restructure would become a relatively simple task.

VIRTUAL SERVICE INTERFACE

Looking Through a Mirror

Before we describe the architecture for the VSI we will briefly describe the application's perceived interface to MVS service.

As has been said, the interface between applications and such MVS services as IMS, is through an indirection object, the SSI. Thus, in order to provide the service to the mini from the remote MVS high end machine we must implement a service surrogate on the mini to receive the application requests and pass them on to the MVS service engine through the VSI. When the request is sent over to the MVS service engine, the request must be **call reflected;** i.e. reissued to the appropriate service subsystem. This requires the surrogate to translate the requestor's parameter list into a standardly defined service subsystem model. Clearly, new applications could go directly to the service subsystem (via the VSI). The service surrogate merely exists as support for the current application inventory.

In providing a virtual service interface there are two basic pieces that are required: the actual VSI, which can be implemented in either software or firmware depending upon price/performance considerations; and the high speed interconnection.

THE VSI

For completeness then, there are basically three parts to the VSI:

1. The basic call mechanism from the user/surrogate subsystem to a specific service subsystem.

2. The VSI services provided to allow users and service subsystems to broadcast interest and availability of selected services.

3. The hardware/firmware needed for physical transmission of requests and data between memories in the local network.

This paper will address the first two parts of the VSI. In a subsequent paper we will address the architecture/design of the physical inter-processor communication linkage.

The intent of the architecture between surrogate subsystem (such as the user's perceived variant of IMS or an auxiliary storage manager) and the real subsystem, is to make the interface work with minimal overhead. The rationale being that the surrogate to service subsystem interface could be used for shared memory processor to processor systems or even when they both reside within the same processor (such as in peer coupled subsystems like IMS/MSC). The following is a brief description of this architecture, including the rationale for its original inception.

VIRTUAL SERVICE REQUEST MECHANISM

This mechanism provides the ability to define a mapping of arbitrary (software and hardware) interrupts into action routines for each distinct program environment. A queued interface is used, thus avoiding the need to **disable** interrupts. It also avoids the need for first and second level interrupt handlers (FLIHs and SLIHs in the OS venacular). This implies the potential for higher performance and better CACHE hit ratios.

Equally important, this mechanism is used as a means for inter-processor communication, where the requester is not aware of whether the receiver is on the same processor or a different one. Thus the requester does not have to code alternative logic in its mainline paths depending upon whether a call is to be made on the same processor or on a different one.

Rationale

With the advent of the VS class of operating systems, IBM acquired the ability to utilize **virtual** memory, giving the appearance of a large amount of storage.

While providing enhanced capability, however, we maintained the old absolute form of interrupt handling (i.e. calls from hardware to software when asynchronous activity completed). Thus we still had from our old System/360 architecture the five specific interrupts: I/O, SVC, External Interrupt, Machine Check, and Program Check. While the number of interrupt exits remained invariant in the transition to the VS systems, the number of possible interrupt types increased (e.g. Page Faults, Monitor Calls, PERs, etc). Since the interrupt architecture was still oriented towards System/360, the implication was that the hardware was required to know only the 5 interrupt handlers and since the interrupt handlers were constrained to having to be defined through 5 explicit architected addresses in low real storage, there was no way to define additional interrupt handlers for the new types of emerging interrupts. Thus the new interrupts had to be handled by the old interrupt handlers (e.g. Page Fault was handled by the Program Check Interrupt handler, etc). This required the interrupt handlers to determine the type of interrupt and call a second level of interrupt handler in order to process the request (for certain interrupts this determination of which routine to call can

amount to approximately 100 instructions). Furthermore, while the interrupt handler is making this determination and while the actual processing is being performed the code tends to be hp2.disabled, i.e. not allowing any other interrupts of at least this interrupt handler. When the interrupt handler has completed handling the interrupt, processing is enabled and control is returned to the Dispatcher to find some work to do. If interrupts had been deferred while the interrupt handler had been disabled then these interrupts now serially occur and the interrupt process is repeated. This process of disabling and enabling can potentially cause a poorer cache hit ratio as well.

Furthermore, real time processes, which cannot afford this cost of the multi-level interrupt handler, tend to overlay the location of the interrupt handler with the address of the real time interrupt handler. When an interrupt occurs, the real time interrupt handler determines whether the interrupt was generated by the real time process or by the . VS system. If the interrupt was caused by the VS system, the real time interrupt handler invokes the regular VS interrupt handler at obvious performance degradation to both.

The problems described above increase as we increase the number of possible interrupts, i.e. as we move to more and more asynchronous processing of major functions of VS systems newer types of interrupts will be defined, where interrupt now means "return information for an activity that was performed asynchronously". Given the existing interrupt structure this will result in greater complexity in attempting to retrofit these new interrupts into the 5 existing classes and performance degradation in adding logic to these interrupt handlers to process the interrupts.

Furthermore, with the possibility that a given function can either be performed on the same processor or on an asynchronous processor the need exists for the requester of the function to be transparent to the physical location of the function rather than suffer the performance degradation (and possible migration inhibitors) in invoking the function.

Similarly a function request need not always result in generating an interrupt. In some instances having the request merely queued would enable a software polling function to sort these requests for greater performance.

Therefore the following mechanism. This mechanism provides us with a means for defining to the system an arbitrary number of service interactions (therefore labeled as Virtual) and a queued interface that avoids requiring interrupt handlers to disablement.

Description

The following new additions are made to the system:

1. New services to define new interrupts (called SERVICE REQUESTs, which are similar in nature to remote procedure calls), and to either invoke, queue, receive, or purge service requests.

2. An Indirection Array (IA) which is maintained and used by the system to access service subsystem built structures.

The following is a brief summary of the VSI services:

These additions are used in the following manner:

Eight new services are defined for service processing:

1. DEFINE_SERVICE - which establishes or destroys system recognition of a service subsystem environment.

2. RECEIVE_REQUEST - used by a service routine to acquire queued service requests.

3. SERVICE_REQUEST - used by software to generate a service request or to send a response to a service request.

4. SERVICE_RETURN - used by either a service routine or its surrogate to indicate completion of a service request.

5. TEST_RETURN - used by a requester to determine the status of an asynchronous service request (polling).

6. XP_MOVE (Cross Processor Move) - used by software to move data from one processor to another

7. PURGE_REQUEST - used by software to reject any queued service requests for a given service.

8. QUERY_CONNECT - used by software to determine which processors are connected to the requesting processor

9. ACTIVATE_SERVICE - used by software to indicate availability to receive service requests from other processors or to remove availability.

Again, the VSI will be described in full detail in a subsequent paper.

Note: The VSI, summarized above, can be implemented as new instructions. In the absence of these instructions, we will simulate them in software by treating SERVICE_REQUESTs in a manner similar to the way SSI calls are supported in MVS. Thus, these new instructions could be treated as macros until instructions are deemed a good performance enhancement. The decision to implement the VSI as instructions is a price/performance consideration. We hope, through prototypical activity, to determine if new instructions are warranted.

SUBSYSTEM INITIALIZATION

The **Define Service** request is used to initialize the remote processor to enable access to the service processor. This includes both the initialization of the service subsystem surrogate (if the service subsystem is on a different processor in the local network), the implied (and user/application transparent) processor to processor connection, and the identification of the location of the service subsystem. Initialization of the service subsystem, in support of the remote processor is the responsibility of that subsystem, i.e. the subsystem should provide a service, through the VSI to enable remote users access of its services.

This, though, is usually performed through the service subsystem issuance of a **Define Service** request followed by an **Activate Service** to inform other processors of the service subsystem's availability. **Define Service** also includes mechanisms for notification of abnormal termination of a service processor. Notification of the failure of a service subsystem is, again, the responsibility of that subsystem (other than processor failure). Similarly, mechanisms are defined to notify service subsystems of requester (process or processor) failure.

The VSI keeps track of the location of all services through symbolic identification (such as "IMS"). Thus, while there are fast forms of VSI requests that imploy simple indirect paths to service subsystems, logical subsystem names can also be used as path arguments.

SUBSYSTEM DIRECTED SERVICE REQUESTS

The definition of subsystem directed **SERVICE REQUESTs** is the responsibility of the service subsystem. Specifically identified subsystems are:

1. IMS (e.g. for services such as: IDENTIFY, SIGNON, CREATE_THREAD, DL/I call, SUBMIT_TRANSACTION, GET_OUTPUT, COMMIT, TERMINATE_THREAD)

2. Paging

3. Data Management

4. TSO Command Processor

5. JES printer and job submittal subsystems

This list is not intended to be all inclusive but rather names the service subsystems that seem reasonable for prototypical effort.

DATA MOVEMENT COMMANDS

The VSI also provides mechanisms to:

1. Move data from the user's memory to a given service subsystem's memory and vice versa. This can be done explicitly via **XP_Move** or performed implicitly as part of the **Service Request/Return** mechanisms.

2. Enqueue a request to a given service subsystem A **Service Request** can be either queued to or directly handled by a service subsystem. The decision is purely one of service subsystem predilection and is transparent to the requester.

3. Dequeue any requests directed to the user. The requester can specify whether or not he is willing to wait for the service subsystem to complete processing or indeed would desire asynchronous notification either through VSI notification or explicit polling (**Test Return**).

GLOBAL NETWORK SUPPORT

Current Network Environment

In extending our architecture to localize currently global functions we still have to assume the existence of relatively fast interaction capabilities with the host. Without this assumption, data access, whether for block or bulk data transfer, would be exceedingly slow, noticeable, and in general unacceptable. The availability of a high speed local interconnect is a **prerequisite in being able to provide an MVS Distributed Subsystem** in general and the Virtual Service Interface in particular.

It appears that no matter what the capacity of the composite system complex, in general, it will not satisfy all the user MIP requirements. This is because the perception by individuals for personal and and responsive systems is and will continue to escalate rapidly. The perception that one's TRS80 is better than a display terminal is real, dangerous, and a challenge, but unfortunately not entirely wrong. The secretary has on his desk star wars, high function text editors, record management, and information retrieval. When RAS requirements, responsiveness, and personal requirements are taken into consideration the total system complex appears to be headed for more trouble than we have today. It sounds like we are in a position similar to that of Major Chesty Springer, who, when his division was surrounded by the enemy in Korea, in 1954, said "... They will **NOT** get away from me now."

It appears reasonable that by 1985, workstations will run well upwards of .5MIPS 370 problem state (probably a conservatively low number). If the complex had 50 MIPS at central and supported 400 terminals then we would have an aggregate total of over 250 MIPS -- of which the central computation complex is a **small part.** From a MIP only point of view one gets the positive effect of having each new user who connects to the system bringing along .5MIPS which locally is probably more than one needs most of the time.

The topic of networking is currently a very hotly debated issue; it is a significant technology project effort within the industry. The primary debates seem to be whether the 10 MHZ ETHERNET (a reality) is better than the a token ring) or a multiband approach such as WANGNET. The debate appears to have an evolving theme of whose local nets are faster, cheaper, and more reliable.

The arguments seem to focus on the differences and not the similarities of the systems. Functionally, they are both acceptable for performing useful work while offering capabilities far in excess of what we have available today. The availability of high speed, cheap networks is complimenting the availability of the microprocessor to accelerate the distribution of function. This, of course, is occurring not only outside the computer room complex, but it is also manifested within the central machine room with the evolution of backend system networks such as Hyperchannel. The evolution of functional components as isolatable, moveable, and customizable entities is underway. This paper explores ways to isolate, physically move, and customize function for the user. All of this will be done in a manner that should take advantage of the superior management

facilities of the central complex, and the personalized nature of the local intelligent terminal.

Networks

The following is a technical discussion of

- Our requirements for a high speed local network facility
- A summary of what is currently available and what will be probably available by 1985.

Requirements of a High Speed local network facility

First, a brief discussion of the local workstation is appropriate. (It is assumed that the reader has a reasonable knowledge of MVS.) The workstation is assumed to be reasonably **intelligent.** This, at a minimum, implies that if MVS is unavailable, then the local workstation can perform some function on its available local data such as edit, assemble, communicate with others, and run star wars.

Such a workstation could be characterized as follows:

- A small and quiet physical box
- Has 25-64 line display (APA capabilities)
- Has a minimum storage of 512K
- Has optional local print capabilities (15,30,45 cps)
- Has local random access store, 2-4MB minimum
- Optionally, a fixed file on a shared file capability
- High speed, cheap networking capability
- Ability to run in standalone mode
- Has a 3270 Passthru capability
- Capability to IPL from local diskette (or disk) or network.

All of this should be available in the 1980's for under $10K. Note, the little thing should be a good editor too. Just look at what your secretary is doing with his today. Without going into detail, this thing should at a minimum have the standalone capability to execute the following software services.

- Run CMS editors
- Run Assemblers
- Run COBOL compilers

- Run PASCAL compiler and execute
- Run application programs
- Provide message/Mail services
- Run BASIC compiler and execute
- and whatever else you have cooked up on your diskette (or disk).

All of this should run at reasonable speed. Reasonable?? Well, this is tough to define but it should be a lot faster than the TRS80 or ALTO. Well, this sort of outlines our meaning of **intelligent.** Of course, the item that really makes this thing cook is the virtual system facilities that are available when this thing is hooked into the high end machine and is getting those good MVS subsystem services - which appear to be local.

Just a few words on speed. It is obvious for example that editing of local files will be faster than editing of remote files - but not by much. However, there will be reasons to download one's data in bulk form from the high end machine to the local system, e.g. a data set, or a set of indices. Paging could be done locally or optionally done over the link with little noticeable performance difference in many environments such as editing of large data sets.

The data that is envisioned to be transferred over the link are:

- Bulk data such as:
  - Data sets or files
  - Large query response
  - Another user's data set or file
  - IPL data
  - Any other transfer under user direction
- Block data
  - IMS transaction & response
  - Transaction Search either user or central program
  - Program and application responses
  - Message - user or system initiated
  - Indices - here certainly the end user would like to be able to see his disk. It is impractical and ill-advised to keep all of this local. However, a copy of the indices can be kept local and as the user scans the index, the files that are not local will be tagged. Lets face it, that big system is great at managing those hordes of files, and keeping them secure and consistent.

- Paging - There is no reason that one cannot have a local clustered file serve as high level paging store with the central computer's paging store acting as a lower level backing store. The link should be able to handle this for a reasonable level of paging.

The above are just a few of the types of data that will be transported across the link. There will be much more such as mail, memo distribution, presentations, security data, operating system error corrections, accounting, new system components, and game information.

The following is a discussion of the environment of the local link itself. Just one word of caution, local links may be a misnomer because one of the local links may in fact be a gateway to either another local link or some other network. The physical characteristics of a network follows:

Minimum number of wires. The problem of having a 3270 size wire from central to each workstation is unacceptable, at least undesirable, and certainly unnecessary. One wire should be able to service a reasonable number of workstations (e.g. 16).

If a workstation should fail then it should not bring down or negatively affect others

Since terminal configurations are constantly changing, this link should be easily tapped. Other workstations may notice a few seconds outage, but no substantial down time.

To perform the tap, it is desirable that one's office need not be entered

The tap should also be easily removed/bypassed.

The link should be capable of carrying audio and non-coded information as well

Speed considerations:

typical encoded data 3 MHZ

one channel of audio 4MHZ

NCI data at 20MHZ

Since the link is common to more than one station then data should be able to be encrypted/decrypted on the link.

The data link should be able to be made SECURE

Data transported on the link should be checked. It is not a requirement that if the link makes an error or loses/inserts/modifies something that it even knows it. It is not a requirement that the link detect its own errors.

It is a requirement that the link have extremely high reliability.

This then puts the burden on higher levels of protocol to assure data transmission accuracy and drive proper recovery procedures.

Mechanisms such as Parc Universal Packet (PUP) or SNA should work in a link.

The addressing structure for such a link should be very flexible. That is a station should be able to have multiple and unique addresses.

Network Summary

The important point here is that just as microprocessors began their evolution, the evolution of local networks has begun. One should note that the MC68000 and the Intel 8086 seem to be the dominant force in micros. The use of microprocessors is going to continue to accelerate. Clearly, it is the need for the best microprocessor capability that is the driving force and not a particular logo. As it is with microprocessors it seems to be with local nets - even more so. The underlying technology to achieve high speed networking is the bottom layer, and the most easily isolated/replaced component of the system. The success of this project assumes that cheap high speed networking is available.

The first name that generally comes to mind when one mentions networks is ARPA. Indeed, it is still going strong today. ARPA is essentially a packetized distributed network. It has proven its worth of a packetized system since a wide variety of computer types use the ARPA network. Another major characterization of the ARPA network is the Interface Message Processors (IMP's). These standalone computers provide a standardized interface to the network so that all nodes communicate in the same manner. On the back end of the IMP - the part of the IMP interfacing to the host computer - functions vary enormously. The physical interface - e.g. the architected packet however is the same; the back end computers are from a variety of manufacturers and all work against this interface. Today in essence, the ARPA network functions that are in the IMP has been specialized and put on a chip to handle specialized local transmission.

ARPA, however, is basically a distributed network system. Local networks have much the same logical structure as ARPA but are characterized by two significantly different aspects. First, the attachment to the local media is done by an integrated, cheap, high speed adapter that does the IMP functions serially. The local computer networks (LCN) interface is very specific in that it only interfaces to one specific media on one side and has a single non queued fixed packet interface to its host on the back side.

There are many types of local computer networks that generally have a specific framing and line protocol associated with them. They basically are:

- Token Ring
- Passive Bus (ETHERNET and Wangnet)
- Loop (R-Loop)
- Multiple Bus (Hyper channel)
- STAR
- MULTI BAND (Wangnet)

Each of the above have strengths and weaknesses but each has the effect of implementing the goal of the customer - to get a low cost, high speed transport mechanism that can handle a set of heterogeneous hosts, fail soft, have high reliability, and be easily reconfigured. The use of the local network implies the set of attached machines or devices are interconnected by compatible Communication Interface Units (CIU's) Network topologies are implemented so that the attached CIU/Host/Device pairs can communicate with any other CIU/Host/Device pair over the network media. Generally, local networks employ twisted pair or coax cables but fiber optics seems to be emerging.

In each of the above mentioned types of local networks, the CIU generally consists of three identifiable sections- see Figure T1- a communications device interface, interval logic and buffers, and a host/device interface. Following is a brief discussion of the content of function of each of these sections.

Host/Device interface

The function here is to interface to the host with some system wide architected packet. From above this point the Host/Device will exercise higher level protocols that give meaning to the received packet. Note the Frame/Packet as seen in Figure T2 is not all visible at this level. It is the function of this interface to present Fields B-F to the higher level system. Fields A, G, and H are usually stripped off by the lower level interfaces. In any case it is certainly a desirable feature that the top level interface to this section be in digitalized form and in the specific architected frame format. The real worth of the system and 90 percent of the work is in the higher level protocol that utilize the data in these packets. The lower two levels are indeed buildable cheaply and simply with the knowledge that higher level protocols will recover from errors.

Interval logic and buffers

This section of the CIU can vary significantly depending upon the topology selected, the protocols, and distribution of functions. Basically, this component contains the physical store buffers to provide the necessary isolation of real time dependencies from the host interface. The buffers on a simple topology (star and passive bus) may require only one or two frame buffers. The Ring could require a larger set of buffers to assure a sequential stream at the host interface.

The logic involved generally involves the checking of the packet for viability and accuracy. Some lower level protocols could be performed checking and retransmission protocols. Also, the logic may contain a low level knowledge of the network nodes and names- certainly it knows its node name.

Communication Device Interface

Here the physical send and receive functions take place. This generally involves the serialization/deserialization of the data going to/from the physical media. The checking of addresses on the network is done here. The physical transmission specifics are a technology question such as electronic signal type for coax vs. fiber optics.

FIGURE T1 COMMUNICATION INTERFACE UNIT- CIU

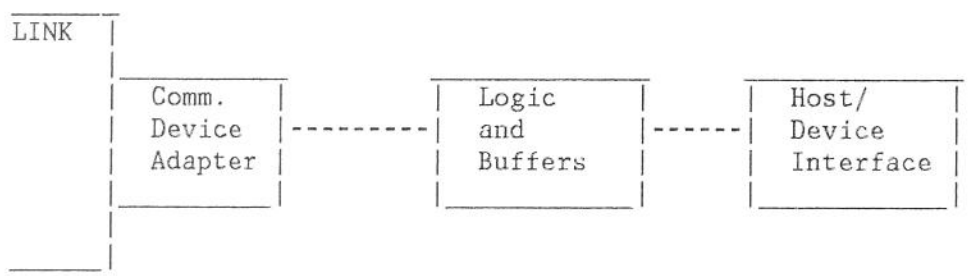

FIGURE T2 FRAME FORMAT

A	B	C	D	E	F	G	H
BEGIN FRAME	DEST. ADDR.	SOURCE ADDR.	LINK INFO.	DEVICE DATA	DATA	CHECK ECC/ CRC	END FRAME

The VSI Use of the High Speed Networks

Now where does all of this take us in relationship to the VSI? The answer is that the VSI is really thought of as being implemented in an environment that has a big system backing it up and CIU's of the type described are readily available. The real benefits of the VSI will be realized when the capabilities of the big system can be utilized in a reasonable manner at the local workstation. Granted the local station only needs, at an instance, a very small subset of the larger system resources. However, these resources and the ability to interact with these resources in a meaningful way is really what the VSI is all about. Hence, each user of an intelligent terminal will bring with him varying degrees of workload and responsiveness demands upon "CENTRAL".

The scenario is now outlined. The fact there are lots of intelligent terminals around does not alter the problem - if anything it accentuates the problem. The fact there are lots of high speed local nets around does not alter the problem - if anything it accentuates the problem. The fact that there are lots of big, powerful "CENTRALS" around also accentuates the problem. The problem of course being that we are unable to connect these intelligent terminals to **services** of the central via high speed local nets in a meaningful manner.

An understanding of the problem is very important, so we'll restate it in other words. The problem is that we are unable to provide meaningful services via the VSI at the host for use by hundreds and thousands of intelligent terminals because the high speed local nets do not connect to the "CENTRAL" in a reasonable manner. The obvious question then seems to be - where is the break in the line? We believe this can best be addressed by initially addressing what the problem is **not.**

The problem is NOT:

- Intelligent terminal, there are an abundance of these.

- The availability of local nets, there are an abundance of these.

- The existence of big centrals, there are lots of these.

- The lack of data management capability at central and locally - there is an abundance of these and lots of improvement is rapidly appearing.

- The lack of an adequate language, there are too many of these.

- The lack of adequate technology for professionals and non-professionals to communicate with the system while there is much room for improvement here people learn to use lots of good and lousy interfaces.

- The problem is not the lack of another interconnection protocol, there are lots (and lots) of these.

The problem is:

- We are unable to connect all these intelligent terminals and high speed local nets together with "CENTRAL".

  - If one connects the terminals or high speed local nets to "CENTRAL" via channels they eat the central alive with interrupts and connections.

  - If one connects them via a cluster controller type system they are much, much too slow.

- Local nets can communicate at high speed with other local nets, and high function cheap terminals but not the big central - incredible but true. High performance gateways do **not** connect to central.

One final comment. The solution is obvious if one accepts the problem as articulated and is left as an exercise to the interested reader.

SUMMARY

In extending our architecture to support a global network we still have to assume the existence of relatively fast wide bandwidth linkages (such as microwave or satellite). Without this assumption, data access, whether for block data access or bulk data transfer, would be exceedingly slow, noticeable, and in general unacceptable.

The architecture presented above would still be appropriate for such connections with the addition that network recovery constraints would have to be added to the underlying transmission layer but clearly would not be added to the application/subsystem layer.

It is also clear that such a network would not be used to connect a mini in New York to a service processor in Los Angeles. The global network would be used to connect service processors to service processors, whether they

be high end machines or amalgamations of midrange processors. An example of such a connection would be in support of a partitioned data base, use of a controlled (classified

REFERENCES.

1. IBM VM/SP: Introduction, IBM Form No.GC19-6200 (Sept.1980)

2. A.L.Scherr, "Functional structure of IBM virtual storage operating systems, Part II: OS/VS2-2 concepts and philosophies," **IBM Systems Journal** 12, No.4, 382-400 (1973).

3. J.A.Cannavino,B.C.Goldstein, T.W.Scrutchin, "OS/VS Release 2 Job Management Structure, **Proceedings of Guide 48** (May 1979)