SNA Perspective On Internetworking

Volume 14, Number 6
June, 1993
ISSN 0270-7284

The single source, objective monthly newsletter covering internetworking with IBM's Systems Network Architecture

IBM's Messaging and Queuing Architecture and Products

The deployment of new, distributed applications that take advantage of the power of personal computers and intelligent workstations has resulted in a wide variety of new networking requirements. Replacing the legacy applications that were typically based on simple interactions between dumb terminals and mainframe-based applications are new applications based on models such as client-server computing. These types of applications potentially require complex interactions among software elements running on a variety of platforms throughout the network.

In a distributed computing environment the networks become a key part of the application platform. The role of the network is to facilitate communications among the software components of applications without regard for their physical location within the network. In addition to the basic connectivity provided by the network's transport services, standard protocols for program-to-program interaction must also be provided. These program-to-program communications services usually involve two interrelated elements:

- An application programming interface
- A set of end-to-end services to support interactions among distributed applications

A variety of program-to-program communications standards exist in the networking industry. Some examples would include SNA's Advanced Program-to-Program Communications (APPC), various types of remote procedure calls, and named pipes. IBM's networking blueprint includes three distinct styles of program-to-program communications, including messaging and queuing.

(continued on page 2)

Frame Relay Meets The Network Control Program

SNA Communications Controllers running the Network Control Program (NCP) have made up the backbone of SNA networks since the orginal SNA announcement in 1974. Communications Controllers have evolved from their original wide area-only role to support LAN technologies such as Token-Ring and Ethernet.

(continued on page 11)

In This Issue:

(continued from page 1)

Program-to-Program Communications Within the Networking Blueprint

IBM's networking blueprint provides a framework for mixing and matching networking protocols and services within enterprise networks. Figure 1 shows the overall structure of the blueprint. Messaging and queuing resides within the application support layer of the blueprint. The application support layer provides a variety of end-to-end services that can be used to connect network users. These end-to-end services include generalized program-to-program communications services, as well as services such as file transfer and system services such as directory services.

The three types of program-to-program communications services defined within the blueprint are:

- Conversational
- Remote procedure call
- Messaging and queuing

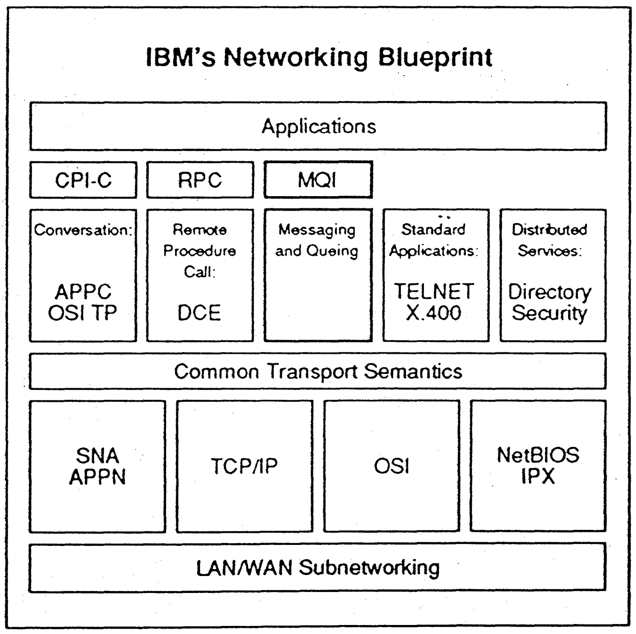

IBM's Networking Blueprint
Applications
CPI-C
RPC
MQI
Conversation: APPC OSI TP
Remote Procedure Call: DCE
Messaging and Queing
Standard Applications: TELNET X.400
Distributed Services: Directory Security
Common Transport Semantics
SNA APPN
TCP/IP
OSI
NetBIOS IPX
LAN/WAN Subnetworking

Figure 1

Each type of interface supports a different style of interaction between distributed applications and each has its own application programming interface which is used to access the program-to-program communications services.

As its name implies, the conversational style of program-to-program communications supports a structured dialog between the communicating applications. The conversational interface that is defined within the blueprint is the Common Programming Interface for Communications (CPI-C) and the Advanced Program-to-Program Communications (APPC) protocols that support structured interactions between distributed applications. This conversational interface not only provides a method for exchanging data between distributed applications, but also provides a defined set of protocols for synchronizing the processing of information between the applications.

Remote procedure calls (RPCs) are a category of program-to-program communications interfaces which are characterized by their transparency to the communicating applications programs. From the point-of-view of the application program, remote procedure calls appear to be ordinary subroutine calls to local subroutines, but RPC software intercepts the calls and transparently provides the communications required to transport the call to the remote subroutines. IBM has adopted the Distributed Computing Environment's (DCE) remote procedure calls for inclusion within the networking blueprint.

The blueprint's messaging and queuing interface (MQI), which is the subject of this article, supports a style of program-to-program communications which is fundamentally different from the conversational interface and remote procedure calls because it does not require a real-time connection to be established between the communicating applications. The MQ architecture defines an environment which supports the asynchronous delivery of messages among distributed applications. In situations where the target application is either not active or not reachable due to a network outage, the MQ software will queue messages and attempt delivery when the required resources become available.

Asynchronous Networking

This asynchronous, store-and-forward style of networking is certainly not new in the IBM networking world. The earliest example was probably the original versions of the TCAM access method which supported the queuing of data on the mainframe. TCAM is designed to support a host-terminal environment where the mainframe-based applications can output information destined for terminals which may not be connected while the application is running. The data is, in these cases, written to queues on the mainframe and then sent by TCAM to the terminal when it connects.

A more recently introduced type of asynchronous networking is SNA/Distribution Services (SNA/DS). SNA/DS is a distributed store-and-forward networking service which is usually implemented over an SNA transport service. SNA/DS is widely used in the IBM world to support file distribution, electronic mail, and other types of applications.

Messaging and queuing provides asynchronous networking services to a wide range of distributed applications including client-server applications and mission-critical transaction processing applications.

Asynchronous communications is only one feature of the MQ architecture. One of the key features of Messaging and Queuing is a very high-level application programming interface, called the Messaging and Queuing Interface (MQI), which makes applications completely independent of communications processing. Messaging and Queuing applications simply put messages to and get messages from a local queue. All communications processing is external to the application programs.

The Messaging and Queuing architecture, and the MQseries products which implement the architecture, are designed to support mission-critical transaction processing across multiple platforms. These types of applications frequently require that updates to multiple, distributed resources such as databases be synchronized so that either all updates are accomplished or the entire unit of work is rolled back to its original state. To support this environment the Messaging and Queuing message queues can be included within units of work. This means that the delivery of messages can be made a part of critical units of work.

MQ networks can also guarantee delivery of messages to remote applications. Critical messages can be declared to be persistent which means that they will be queued to nonvolatile storage until delivery can be completed. When queue managers are restarted, due to either failures or standard operational procedures, persistent messages are preserved and their delivery is guaranteed.

Another Messaging and Queuing feature which is targeted at transaction processing is the execution of transaction programs triggered by the arrival of messages on selected queues. System resources can be conserved through the use of triggering because applications don't have to be executing continuously. This is particularly important in situations where applications are used infrequently. Triggering can also speed up distributed transaction processing by ensuring that all required transaction programs are activated immediately as needed.

The Structure of MQ Networks

Messaging and Queuing networks are made up of three major elements:

- Application programs
- Queue managers
- Queues

The application programs communicate with one another via queues which are controlled by message queue managers. Queue managers are middleware products which reside between application programs and the underlying communications networks and protocols. The queue managers communicate with one another to deliver messages to remote systems. The actual transport protocols used for communication among the queue managers could be SNA, TCP/IP, or any other protocol. In fact, different transport protocols could be used on each hop. Regardless of the transport protocol being used, the queue managers communicate with one another via an MQ-defined set of higher level protocols.

One of the key features of Messaging and Queuing is that it completely isolates application programs from the network. The only parts of the Messaging and Queuing network that application programs are aware of are the local queues to which they read and write messages. The queue managers are responsible for the actual delivery of the messages.

Figure 2 shows the logical structure and operation of a very simple messaging and queuing network. Application program A resides on one system and Application B on another. Each system must be running a message queue manager and each queue manager will have a set of first-in-first-out queues defined. The applications in the network communicate with one another by agreeing to pass messages through particular named queues. In our example, Application Program B is set up to read messages from Queue2. When Application program A wants to send a message to B, it simply puts the message into the queue named Queue2.

An important feature of messaging and queuing networks is that the application programs need not be aware of the physical location of the message queues. In our example, Application Program A does not have to know the location of Queue2. It is the responsibility of the message queue managers to deliver the messages to the proper location. While it appears to Application Program A that it is directly putting a message on Queue2, the interactions are actually more complex.

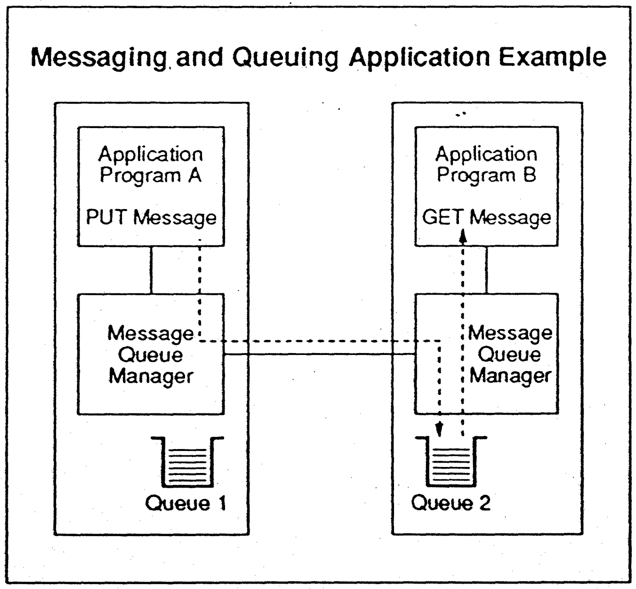


Figure 2

For example, Application Program A's queue manager will put the message on a local queue which is transparent to Application Program A. The reason for putting the message on this local queue before it is transmitted is to ensure that it is safely stored to prevent loss in the case of a failure either within the queue manager or on the network being used to deliver the message to the remote system. This intermediate queueing also allows Application A to send messages even when the remote queue manager is not running or when the communications facilities connecting the two systems are not active or available.

A Messaging and Queuing Example

The operation of a Messaging and Queuing network can be demonstrated with a simple example. This application involves processing at three locations. Orders are initially entered via AS/400 systems located in sales offices. The orders are then transmitted to the corporate headquarters where a mainframe-based order entry application validates the order, and then sends the required information to a mainframe-based billing program and also sends inventory and shipping information to the warehouse which is in another location.

Note that this example describes the logical operation of a messaging and queuing network from the perspective of the application programs that are interconnected by the network. Later in this feature we will describe the message delivery process in more detail.

Figure 3 (see page 5) shows the structure of this application when implemented using the Messaging and Queuing architecture. The value of using Messaging and Queuing rather than synchronous styles of program-to-program communications such as APPC or remote procedure calls is that the participating applications programs are not required to interact with one another in real time. In fact, these applications programs do not even have to be executing at the same time. This means that processing can be scheduled in each location to optimize resource utilization.

Now let's look at how Messaging and Queuing supports the order entry application.

1. A clerk in the sales office uses an AS/400-based application to initially enter and validate the order. A message which contains the order is placed on a queue, called QOE, which resides on the mainframe and represents the mainframe-based order entry application. Note that this delivery involves a remote connection between queue managers, but this is transparent to the applications which are unaware of the physical location of any of the queues in the network.
2. The order entry application gets the message from its local queue, called QOE, and processes the order.
3. The order entry application sends billing information to a local queue, called QBILL, which represents the billing program.
4. Order processing information is put on a remote queue, called QPROC, which represents the order processing application at the warehouse. The queue managers, again, handle the remote delivery.
5. The billing application removes the message containing billing formation from its queue, called QBILL, and creates an invoice.
6. The order processing application gets the message describing the order to be shipped from its local queue called QPROC.

Note that any of the applications which sent messages could have requested a reply to those messages. Also, if any of the applications or data links had not been available, the messages would have been stored in message queues until the required resources became available.

The Messaging and Queuing Application Interface

The MQ architecture defines a standard application programming interface which is called the Message Queuing Interface (MQI). The MQI is not an interface between the application program and the network, rather it defines an interface between the application program an its local queue manager. The message queues which are used to exchange messages reside within the queue manager and are accessed via the queue manager.

Since the MQI does not deal with any networking or communications issues, it is relatively simple for application programmers to use. This, in fact, is one

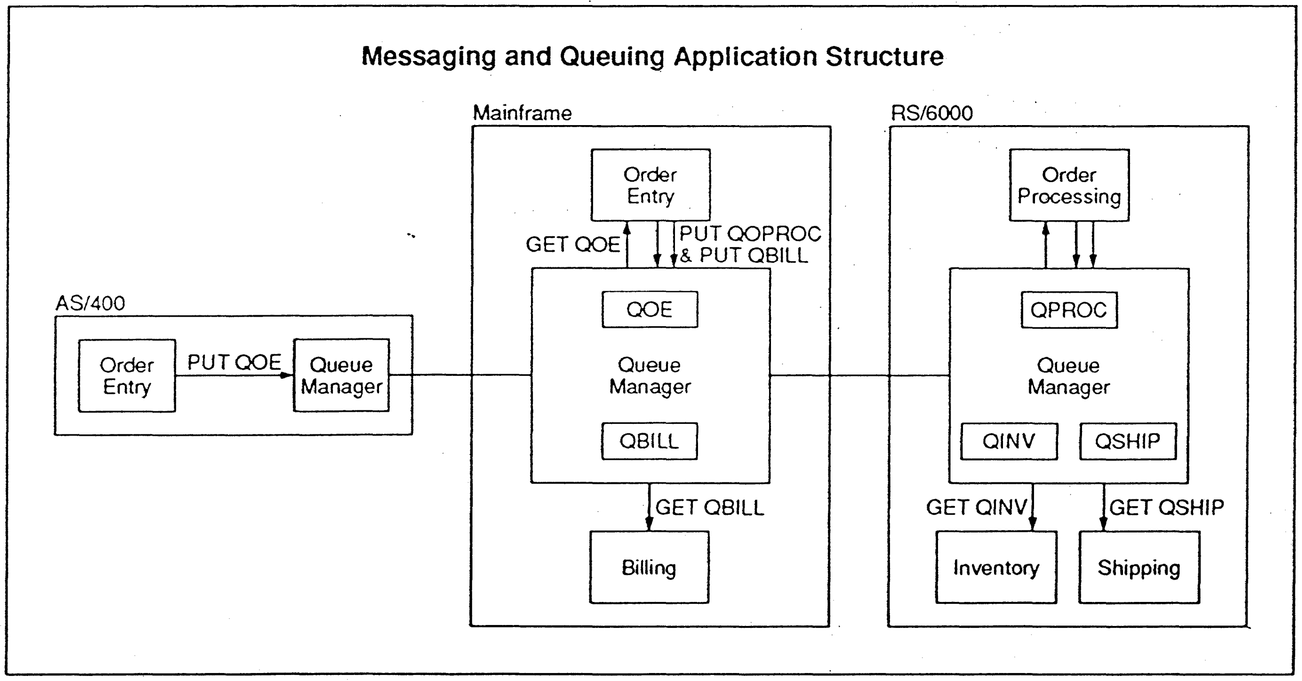


Figure 3

of the key benefits of Messaging and Queuing. Programmer training is an obstacle to the use of many program-to-program communications APIs.

Functionally, the MQI interface defines a relatively small number of calls which are used to manage the logical connection between application programs and queue managers, and to send and receive messages across the network. The following functions must be performed in sequence when application programs communicate with one another across an MQ network.

- Establish a logical connection with the local MQ software which is called the queue manager
- Open a message queue to which messages will be sent and received
- Get and put messages on a queue
- Close a message queue
- Terminate connection to the local queue manager

These functions are implemented by a set of calls which constitute the MQI definition. The calls are issued by applications to their local queue manager. There is no direct interaction between applications and remote queue managers. The following is a list of MQI calls:

CALL MQCONN—establishes a connection between the calling application and a local queue manager. No interaction with any of the message queues in the network is possible until this connection is established.

CALL MQOPEN—opens a queue for getting or putting messages. Local queues (residing in the local queue manager) can be opened for either input or output. Remote queues can only be opened for output. Multiple applications can open a single queue.

CALL MQPUT—puts a message on a local or remote queue. The queue must have been previously opened by an MQOPEN call.

CALL MQPUT1—combines the functions of MQOPEN, MQPUT, and MQCLOSE into a single call. Designed to be used in situations where messages are infrequently written to a queue.

CALL MQGET—gets a message from a local queue. The queue must have been previously opened by an MQOPEN call.

CALL MQCLOSE—closes a queue for access by the calling application. Other applications can continue to interact with the queue. A queue does not have to be empty at the time that it is closed. If the queue is a permanent queue messages will be retained for access by this or other applications.

CALL MQDISC—terminates the logical connection between the calling application and the local queue manager. No other MQI calls can be issued by this application until the logical queue connection is reestablished via an MQCONN call.

CALL MQINQ—allows an application to retrieve certain information about the properties of a queue or other messaging and queuing objects such as queue managers or process definitions. Queue properties which can be retrieved include queue depth, maximum message length, and the number of applications which have the queue open for input or output.

CALL MQSET—allows applications to set certain queue attributes including triggering parameters and whether message put operations are allowed.

All communication between applications within the MQ architecture occurs by the exchange of messages. The MQ architecture defines four types of messages which are handled by the network:

- Datagram—a message which does not require that any reply be returned to the sending application
- Request—a message which requires a reply from the receiving application
- Reply—a message sent when an application receives a request message
- Report—a message which is used to report exception conditions which relate to other messages.

The messages which are sent and received by applications are made up of two major parts—application data and control information. The MQ architecture does not impose any constraints on the types of data that can be included within the application data portion of the message. The message-queuing service does not make any changes to the application data. Maximum message sizes which can be supported are determined by the maximum sizes configured for the queues through which the message will pass as it moves through the network.

The control information portion of a message is used to convey information between the application programs and the queue managers which provide the message-queuing services. The control information is passed across the MQI interface within a data structure that is called the message descriptor. Some of the more important fields within the message descriptor include:

Message identifier—a 24-byte field which uniquely identifies a message. The message identifier is usually generated by the queue manager to ensure that it is unique, but the sending application has the option of forcing a specific identifier to be used.

Correlation identifier—a 24-byte field which usually contains the message identifier of another message. The correlation identifier can be used to match up related messages such as a request and its associated reply.

Priority indicator—specifies the relative priority of the message and can be used to alter the normal first-in-first-out queuing sequences for high priority messages.

Persistence indicator—identifies messages which must be protected when queue managers are restarted. Persistent messages are written to non-volatile storage.

Format name—an 8-byte string which can be used to indicate the format of the application portion of a message. The format name is interpreted by the receiving application.

Character-set identifier and encoding—this field is set by the sender of a message to indicate the character set being used to represent application data and the encoding of numeric data fields.

Reply-to queue name—contains the 48-byte name of a queue. The named queue will be the recipient of reply messages sent in response to a request message. Any report messages which are generated will also be sent to the named queue.

Reply-to queue manager name—contains the 48-byte name of the queue manager which supports the reply-to queue.

Report options indicator—determines whether or not reply messages should be generated when exception conditions occur.

Queues and Queue Managers

Message queues always reside within queue managers. Queue managers would typically be implemented in middleware products which act as a buffer between the application programs and the underlying network. This positioning of the queue managers contributes to one of the key benefits of Messaging and Queuing—application program independence from communications issues and platform independence.

Some of the queues used with the Messaging and Queuing network are visible to the application programs while others are used by the queue managers and are transparent to the communicating application programs. There are two major categories of queues:

- Local queues—queues which reside on the queue manager to which an application is connected. Local queues can be opened for input or for output
- Remote queues—queues which reside within other queue managers. Remote queues can only be opened for output.

Before an application can send or receive messages in a Messaging and Queuing network it must establish a logical connection with a queue manager. This connection is established when the application issues an MQCONN call across the MQI interface. All of the queues which reside on the connected queue manager are called local queues and all other queues in the network are remote queues from the application's point-of-view.

Alias queues

It is sometimes advantageous to redirect messages to queues other than the one to which they were originally addressed. This is accomplished through the use of alias queues. Alias queues can be particularly useful when resources are moved around a Messaging and Queuing network. A good example would be a client-server application where many client systems are accessing a central server. If the location of the server changes, possibly to take advantage of an underutilized application platform, there are two ways to configure the network with the new location. The first option would be to reconfigure each of the queue managers which support client applications. This would be very difficult to coordinate in a network with a large number of client systems.

The alternative configuration solution would be to define an alias queue on the original server to indicate the new location of the server application. The client systems, in this case, would continue to direct messages to the original location which would then direct them to the new server location.

Alias queues can also be used to control access to queues. For example, a queue might be defined to allow both read and write access. Access can be limited to read-only for certain applications by defining an alias queue with only read access. The applications with read-only access will interact with the target queue through the alias queue, while other applications requiring both read and write access would access the target queue directly.

Dynamic Queues and Model Queues

Some queues are statically defined by system adminstrators. These are called permanent queues. There is also another class of queues, called dynamic queues, which don't have to be predefined. Dynamic queues are created only when their use is required by an application. System administrators define model queues which specify the characteristics of the dynamic queues when they are created.

Dynamic queues can be either temporary or permanent. Persistent messages must be stored in permanent dynamic queues to ensure that they will be recoverable in the event of a queue manager failure. Messages remaining on temporary dynamic queues will be lost when the queue is closed.

Transmission Queues

A special type of local queue, called a transmission queue, is employed when an application sends a message to a remote queue. Figure 4 shows how transmission queues are used to deliver messages to remote queues. If the application puts a message that is addressed to a local queue, the queue manager simply puts the message into the appropriate queue. When messages are addressed to a remote queue the local queue manager places the message on a transmission queue which contains messages addressed to the target queue manager. A transmission program will then read messages from this transmission queue and deliver them to the appropriate remote queue manager. In some cases there might be more than one transmission queue associated with a single remote queue manager. This might be done in situations where messages will travel over different paths to a single destination.

Communications Between Queue Managers

The actual location of message queues within a messaging and queuing network is transparent to the

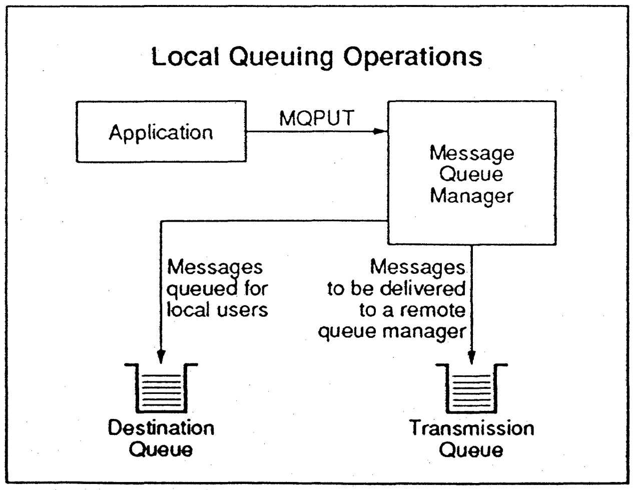


Figure 4

communicating applications. It is the queue managers which have to be configured with information about the actual locations of message queues and the communications resources which are available to deliver messages to those queues.

The Messaging and Queuing architecture defines a high level networking protocol called the Message Channel Protocol (MCP) which is used to reliably deliver messages between queue managers. IBM has not yet released the details of this new protocol, but we will look at the overall structure of the communications software implemented within queue managers.

Whenever a message is to be sent from one queue manager to another queue manager it is put onto a transmission queue. The software which is responsible for transporting messages from one queue manager to another is called a Message Channel Agent (MCA). Figure 5 shows the relationship between MCAs, queue managers, and the network transport connections used to deliver messages.

A connection between a pair of queue managers is supported by an MCA at each end of the transport connection. Two pairs of MCAs provide bi-directional communications between the queue managers. The MCA sender processes get messages from a transmission queue and uses a transport connection to send them to the remote queue manager. Note that the MCA sender process issues MQGET calls to the local queue manager to gain access to the messages. Channel definitions which are set up by system administrators map transmission queues to the appropriate MCA sender processes and transport connections.

Each system also has an MCA receiver process which receives messages from remote queue managers and puts them on the appropriate destination queues where the messages can then be retrieved by local application programs. In this case the MCA receiver process issues MQPUT calls to put the messages on the appropriate local destination queues. In cases where the destination queue is more than one hop away from the original transmission queue, the message would be put on another transmission queue rather than a destination queue. If the next transport connection is not available, the message will remain safely stored in the transmission queue until the needed communications resources become available.

The high level protocol used between the two systems is the Message Channel Protocol which is independent of the transport protocol being used. The transport protocols could be SNA LU6.2, TCP/IP, or almost any other protocol. IBM's initial messaging and queuing products use LU 6.2 for communications, but support for other transport protocols can be expected. This independence of upper layer protocols from transport services is a key design feature of IBM's networking blueprint.

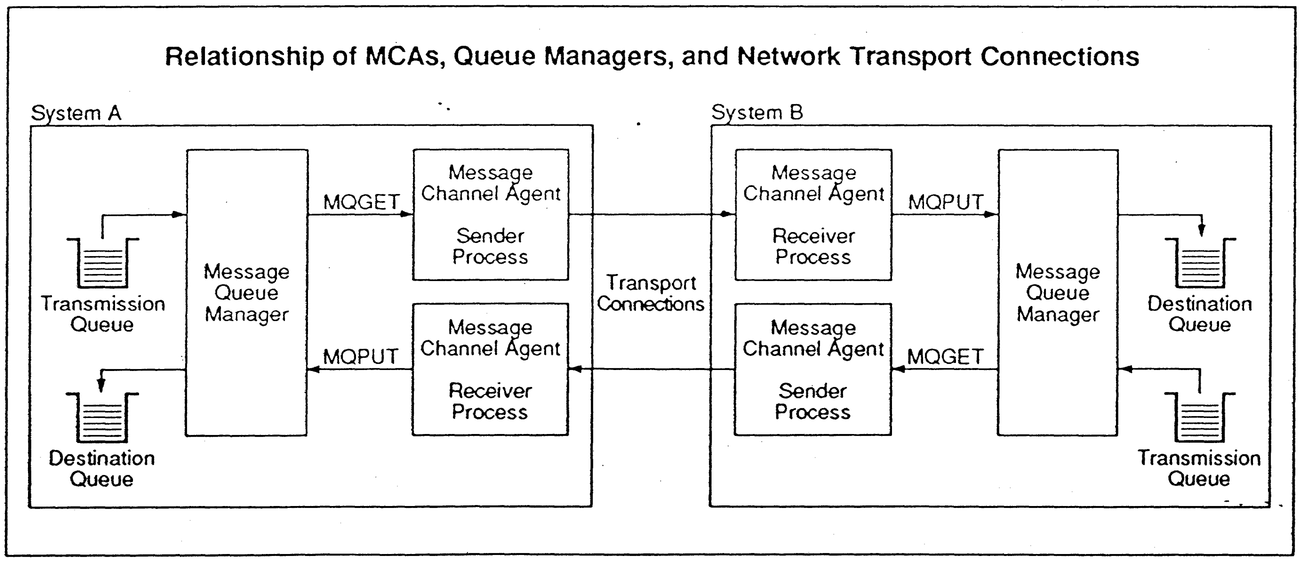

Relationship of MCAs, Queue Managers, and Network Transport Connections
System A
System B
Transmission Queue
Destination Queue
Message Queue Manager
MQGET
MQPUT
Message Channel Agent Sender Process
Message Channel Agent Receiver Process
Transport Connections
Message Channel Agent Receiver Process
Message Channel Agent Sender Process
MQPUT
MQGET
Message Queue Manager
Destination Queue
Transmission Queue

Figure 5

Messaging and Queuing Products

In December of last year IBM and Systems Strategies, a networking software company based in New York, announced that they would jointly develop and market products based on the MQI application programming interface. At the time, Systems Strategies was already marketing its exBridge Transact line of middleware products which provided interoperability across multiple platforms from various vendors.

On March 16 IBM announced that it would begin marketing the Systems Strategies ezBridge Transact product line under its Cooperative Software Program.

On March 30 IBM and Systems Strategies jointly announced the MQseries of middleware products. The MQseries products implement the MQI application programming interface as well as other elements of the Messaging and Queuing architecture. Most of the announced MQseries products are MQ-ized versions of Systems Strategies' ezBridge Transact product line. The main enhancement from an application programming point-of-view is the implementation of the Messaging and Queuing MQI application programming interface. The following is a list of the platforms for which MQseries products were announced.

Hardware	Operating System
System/390	MVS/ESA/CICS MVS/ESA/IMS MVS/ESA/TSO MVS/ESA/Batch
System/390	CICS/VSE
System/88	VOS
DEC VAX	VMS
Tandem	Guardian
PC	DOS OS/2 Windows
AS/400	OS/400
RS/6000	AIX/6000

Migration aids will also be provided for users of the original ezBridge Transact products (Release 2.4) who migrate to the newer MQseries ezBridge Transact products. Run-time software will be provided to map Release 2.4 calls into MQI calls. This will preserve current customer investments in ezBridge Transact applications.

System-to-system communications protocols also change with the MQseries products. A software tool which allows messages to be exchanged between the two environments will be available in conjunction with IBM's Message Queue Manager MVS/ESA.

Impact of Messaging and Queuing

Clearly, not all applications can be supported by the style of networking provided by the Messaging and Queuing architecture. Applications which require real-time interactions across a network will continue to employ synchronous styles of program-to-program communications such as APPC or remote procedure calls.

For applications than don't require real-time interactions, Messaging and Queuing offers some real adavantages. First, MQ applications are completely isolated from any knowledge of the underlying network. This can simplify applications and allow networked applications to be developed by programmers with no networking experience. The network interactions are the responsibility of the middleware queue managers.

MQ networks also provide a robust delivery system that is capable of guaranteeing delivery of messages with no application intervention. The application simply puts the messages on the appropriate queues and then delivery becomes the network's responsibility.

The third key advantage of Messaging and Queuing is that asynchronous delivery can be used to optimize the use of application processors within the network. Since messages do not have to be processed in real time, they can safely be kept on queues within the network until processor capacity becomes available. Communications resource utilization can also be optimized by the same queuing technique. Messages can be delivered when communications lines are lightly utilized.

It is also important to keep in mind that applications will most likely mix-and-match their program-to-program communications techniques. In many distributed applications there are some interactions that must be carried out in real time while others are much less time dependent. In these situations synchronous communications, such as APPC, could be used to support the time-critical interactions while Messaging and Queuing could provide the links between less time-critical processes.

In summary,the Messaging and Queuing architecture, and the products which implement it, will give designers of distributed applications a new set of tools optimizing the use of resources in mission-critical applications. ■

(continued from page 1)

The Communications Controller's wide area support has also changed radically. Early Communications Controllers handled only low speed dedicated SDLC links. Early in the evolution of SNA, support for X.25 packet switching networks was added to the NCP. Last year IBM added NCP support for one of today's hottest WAN technologies—frame relay networks.

Frame relay networks have many of the same chacteristics as X.25 networks. Both are packet switching networks which allow users to share a common mesh network where any user can be connected to any other user. Both can be implemented as either public or private networking services.

Frame relay departs from X.25 by providing a combination of more and less service to network users. More service is provided in terms of higher bandwidth and lower network latency, but less error detection and no error recovery is provided by frame relay services.

Frame Relay in the SNA Backbone

Support for frame relay networks connecting SNA Communications Controllers becomes more important as new LAN-based applications are deployed. These new applications frequently require more bandwidth than older host-terminal applications. The data traffic that they produce also tends to be burstier and traffic patterns are less predictable because the SNA host is no longer the focal point for all communications.

Frame relay services excel at handling bursty traffic by providing bandwidth on demand. When wide area bandwidth is shared among a large number of users who are producing bursts of data, network resources be shared on a demand basis to optimize the use of data links.

Frame relay services are not limited to the low speeds of older X.25 networks which don't operate efficiently at speeds much higher than 64 Kbps. Frame relay services are capable of supporting users at T1 (1.544 Mbps) and higher data rates.

In this article we'll review frame relay technology and then show how it is being exploited in SNA networks by the NCP.

What is Frame Relay?

Frame relay is an industry standard for an interface between computer systems and networks which provide packet switching services. These can be either public or private packet networks. Packet switching networks which provide frame relay interfaces are frequently called fast packet switching networks to distinguish them from their lower speed predecessors which supported the industry standard X.25 packet switching interface.

While there are a number of similarities between frame relay and X.25, there are also some significant differences. Both X.25 and frame relay define connection-oriented services. This means that explicit connections must be set up between pairs of users who will communicate across the network. The X.25 standard calls these logical connections virtual circuits while the frame relay specifications refer to them as virtual connections.

Even though both X.25 and frame relay define connection-oriented services, the connections are achieved in very different ways. X.25 employs the protocols defined by the lower three layers of the OSI Reference Model—the Physical Layer, the Data Link Layer, and the Network Layer. The implementation of these three layers of protocols by X.25 results in reliable connections between end users which includes full end-to-end error checking and error recovery. In the 1970s, when X.25 was defined, it made sense to include all of this error checking and recovery because the transmission facilities were very unreliable and subject to frequent errors. But the price that is paid for this error checking and recovery is a high level of protocol overhead. The result of this overhead is that X.25 interfaces do not perform well at speeds exceeding about 64 kbps.

Frame relay, on the other hand, takes advantage of the high speeds and low error rates of modern networks to define an interface that performs at higher data rates. This is achieved by streamlining the protocols that are used across the frame relay interface. Frame relay makes use of only Physical Layer and Data Link Layer protocols. In fact, the frame relay interface provides only partial data link level support.

Frame Relay Characteristics

In order to determine whether frame relay services are an appropriate networking solution, network designers must understand the characteristics of a frame relay environment. The following is a summary of the key characteristics which can be used to describe frame relay services.

Frame Relay Reliability

Reliability involves two elements—error detection and error correction. Frame relay services will detect errors in the data link level frames, but will not perform any type of error recovery. Frames which are in error are simply discarded by the network. The basic design philosophy is that the end systems which are communicating should have responsibility for recovering from errors. Modern wide area networks have very low error rates which allows the shift of recovery responsibility from the network to the communicating end systems. The result is higher network performance because of the elimination of the error recovery protocols within the network. Reliability is provided by higher level protocols.

Performance in Frame Relay Networks

The performance of frame relay services compares favorably with that of earlier packet switching technologies such as X.25. The aspects of frame relay performance that we will examine are overall throughput and the transit delays which are introduced by frame relay services.

The throughput efficiency of the frame relay interface in comparison with X.25 is a direct result of its lack of error recovery protocol overhead in frame relay services. Frame relay interfaces and services which operate at T1/E1 speeds are common while X.25 interfaces which must bear the burden of X.25's error recovery protocols generally don't operate at speeds greater than 64 kbps.

The other aspect of frame relay performance is the transit delay that is associated with the use of frame relay services. Again, frame relay has an advantage over its predecessor, X.25, and the advantage is due to the streamlined protocols used by frame relay. In earlier packet switching networks error checking and correction were usually performed on a hop-by-hop basis as the data traversed the packet switching network. This resulted in a cumulative transit delay which most often impacted the performance of interactive sessions.

In frame relay networks there is much less handshaking between the computers which attach to the network, and between intermediate nodes within the network. This results in reduced transit delays.

Another important characteristic of frame relay services is the ability to share hardware communications ports among multiple sessions. This ability to multiplex and demultiplex sessions on a single communications port is similar to that of earlier X.25 packet switching networks. The benefit to network designers is a potential reduction in the number of wide area communications ports required to connect users across the network.

In frame relay networks, full mesh connectivity among a large number of users can be achieved through a single access port on each user's system. The industry trend from hierarchical to decentralized applications increases the value of port sharing to reduce the hardware costs of end user systems and internetworking equipment. In SNA networks, the migration to APPN will make frame relay's port sharing an increasingly valuable feature.

Multiprotocol Frame Relay Support

Few enterprise networks today use a single protocol suite to support all network users. Network designers are using every tool available to create backbone networks that are capable of carrying mixed protocols. Frame relay is one more tool that can be used to support mixed protocols. Traditionally, dedicated wide area circuits have been used to create SNA backbone networks. These circuits were only capable of carrying a single protocol—SNA. LAN-based users had the advantage of being able to run multiple protocols across a single LAN. The only industry standard multiprotocol WAN solution was X.25 compatible packet switching networks, but their bandwidth limitations often limited their usefulness.

Frame relay provides a higher bandwidth mixed protocol WAN solution. There are, in fact, two ways in which frame relay networks can provide mixed protocol transport services to attached users. The first approach is simply to exploit frame relay's port sharing capability and to use separate logical connections to support mixed protocols through a single port. In this case, a separate data link level address will be assigned to each protocol suite.

There is also an industry standard method for carrying mixed protocols on a single logical connection. This standard has been defined by the Internet Engineering Task Force (IETF). These are the folks who define TCP/IP and other Internet standards. The standard is designated as Request For Comments (RFC) 1294. RFC 1294 defines a header which is added to each packet of data going across the frame relay interface. This header contains a Network Level Protocol Identifier (NLPI) which assigns a unique number to each of several industry standard protocol suites so that they can be identified and demultiplexed by the recipient. RFC 1294 does not currently support any identifiers for SNA traffic but IBM has assigned a set of SNA identifiers and has submitted them to the IETF for inclusion in RFC 1294.

Bandwidth on Demand

Probably the most frequently cited benefit of frame relay services is the ability to obtain wide area bandwidth on demand. This is a requirement that increases in importance as a greater percentage of wide area data traffic results from LAN-to-LAN communications. This is due to the fact that LAN-to-LAN traffic tends to be very bursty in nature. Large amounts of wide area bandwidth are needed to handle the bursts of traffic when they occur, but much less bandwidth is needed between the bursts.

When dedicated circuits are used to support LAN-to-LAN communications, the wide area circuits must be capable of carrying the maximum amount of bandwidth required. This expensive bandwidth is wasted between bursts of traffic. As more and more LAN traffic is handled by SNA Communications Controllers, frame relay support in NCP should become an increasingly attractive wide area networking solution.

Frame relay provides a way of supporting bursty traffic without dedicating bandwidth to individual LAN-to-LAN transfers. Computers and internetworking devices are attached to frame relay networks through a physical wide area port. The bandwidth of this port represents the maximum rate at which bursts of traffic can be transferred across the frame relay interface.

Frame relay users are usually assigned a Committed Information Rate (CIR) which is defined as a specific amount of bandwidth. The CIR is usually less than the total bandwidth of the frame relay access port. The CIR is an amount of bandwidth that will be virtually guaranteed to be available to the attached user. The CIR usually cannot be guaranteed with absolute certainty due to the statistical multiplexing of data that goes on in a packet switching network.

Bursts of traffic can be sent at the maximum bandwidth of the frame relay access port even if it exceeds the CIR. The network will make a best efforts attempt to deliver the bursts of traffic. If the network becomes congested, the data packets which were sent in excess of the CIR will be the first to be discarded by the network.

Managing Frame Relay Networks

Current frame relay standards define a management interface which can be used by Terminating Equipment to determine some basic information about the status of connections across the frame relay network. This management interface was originally called the Local Management Interface (LMI), but recent ANSI and CCITT standards now use the term Link Integrity Verification.

This management interface gives Terminating Equipment the ability to gather the following types of management information.

- Currently active virtual connections
- Notification when virtual connections are activated or deactivated
- Indication of whether current connections are congested or not
- Verification of the integrity of the physical link between the Terminating Equipment and the adjacent Frame Handler

In addition to these basic management functions, the Internet Engineering Task Force (IETF) has developed a frame relay Management Information Base (MIB) which will standardize frame relay management via the Simple Network Management Protocol (SNMP). This frame relay MIB will allow monitoring of information such as the number of bytes and frames sent on each virtual connection, and information about the times at which virtual connections are created.

The Structure of Frame Relay Networks

All frame relay networks, whether they carry SNA traffic or not, share a similar structure. Note that the frame relay standards only define the operation of the nodes which directly interface with the computer systems and internetworking equipment attached to the network. The internal operation of the network and the protocols used between the nodes "inside the cloud" are not defined by the frame relay standard. The packet switching network must simply deliver the transport services defined by the frame relay standards.The generic frame relay network model is shown in Figure 6 (see page 15).

There are three basic types of nodes which are found in a frame relay environment. The end user devices which communicate using the services of a frame relay network are called Terminating

Equipment. These nodes send data frames which are formatted according to the frame relay interface specifications and they are also capable of receiving such frames. The role of Terminating Equipment is similar to that of Data Terminating Equipment (DTE) in X.25 packet switching networks.

The nodes which exist within the fast packet switching network "cloud" and provide the interface to Terminating Equipment and packet switching services are called Frame Handlers. Frame Handlers are the frame relay counterparts of X.25's Data Communications Equipment (DCE). The Frame Handlers support an interface which accepts frames from Terminating Equipment and delivers frames to the Terminating Equipment.

The frame relay environment also defines a category of nodes which are used to provide an interface to the fast packet switching network for devices which do not support the frame relay interface directly. The nodes are called Frame Relay Assemblers/ Disassemblers (FRADs). These FRADs are similar in purpose to the Packet Assembler/Disassemblers PADs) which are used in X.25 compatible packet switching networks.

Note that the frame relay standards define the syntax and symantics of the data frames that are exchanged between Terminating Equipment and Frame Handlers. The standards specify the general levels of service which should be provided by the fast packet switching network, but they do not specify the protocols or data streams used within the fast packet switching networking.

Frame Relay Data Formats

All data which is passed across a frame relay interface must carry a header whose format is specified by the frame relay standards. The general packet format is similar to that used by SDLC and HDLC and is shown in Figure 7.

The packets are delimited by leading and trailing flag characters. An address and control field follows the beginning flag. The address field, which is usually two bytes long, contains the Data Link Connection Identifier (DLCI) which is used to identify the individual connections associated with a physical access port.

The command/response field is not used by frame relay and is passed through the network transparently. The extended address fields can be set to indicate the use of a two, three, or four byte address field. The use of a two-byte address field is currently typical.

The next two indicators are used to deal with congestion control within the network. The Forward Explicit Congestion Notification (FECN) is set to identify congestion in the direction that the frame is being sent. This informs the receiving Terminating Equipment that the packet encountered congestion in the network. The Backward Explicit Congestion Notification (BECN) is sent to the sending Terminating Equipment to notify it that frames that it sends are likely to encounter congestion and have a high probability of being discarded by the network.

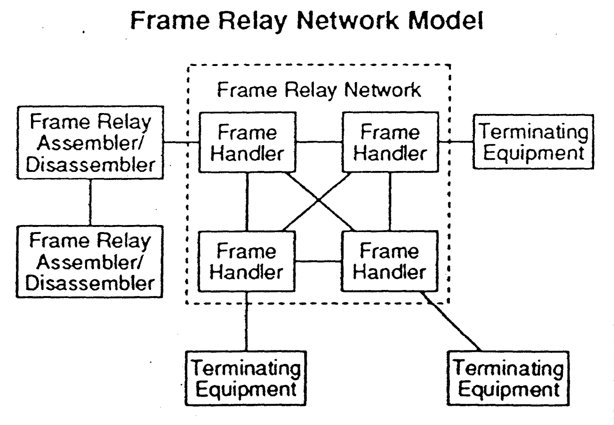
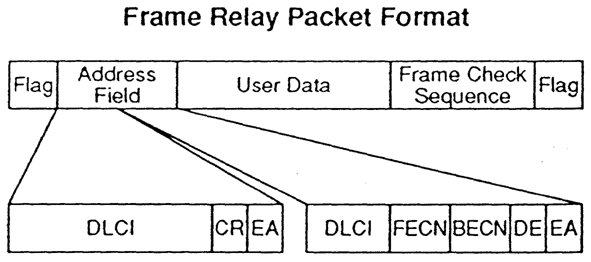


Figure 6



DLCI	=	Data Link Connection Identifier
CR	=	Command/Response
EA	=	Extended Address
FECN	=	Forward Explicit Congestion Notification
BECN	=	Backward Explicit Congestion Notification
DE	=	Discard Eligibility

Figure 7

Addressing In A Frame Relay Environment

Frame relay addressing is accomplished by a mechanism which is frequently called label swapping or ID swapping. This technique does not actually use true addresses to identify data flowing on various connections. Instead, it relies on the use of identifiers which have only local significance at each end of the connection. This ID swapping technique is similar to that used in X.25 packet switching networks and it is also used within SNA Advanced Peer-to-Peer Networking (APPN) networks.

Each of the virtual circuits supported by a frame relay interface is identified by unique identifiers called Data Link Connection Identifiers (DLCIs). The DLCIs are numbers which only need to be unique within a single network access port. The DLCI identifying a virtual circuit is likely to be different at each end of the connection because they are assigned locally at each frame relay interface. The DLCIs are assigned at the time that the connection is set up. The Frame Handlers at each end of the connection map the data traffic to and from their local DLCIs.

NCP Frame Relay Support

The first NCP frame relay support was delivered as a part of NCP Version 6 Release 1 in August of 1992. NCP V6R1 allows NCPs to connect with one another across frame relay networks. The NCP functions as frame relay Terminating Equipment (TE). The frame relay support is entirely implemented in software. There is no additional 3745 hardware required. This ability to implement frame relay interfaces in software is an important characteristic which distinguishes frame relay from technologies such as Asynchronous Transfer Mode (ATM) cell relay networking. ATM implementation requires either internal or external hardware interfaces.

NCP/Frame Relay Congestion Control

Congestion control in the SNA backbone network has always been a major NCP design objective. Traffic on the virtual routes which tranverse the backbone is controlled through the use of virtual route pacing. When frame relay networks begin to replace dedicated SDLC data links connecting Communications Controllers, additional congestion control problems arise. This is due to the fact that frame relay networks themselves can become congested due to traffic which may or may not have been generated by the SNA users.

.In our discussion of the frame relay packet formats we discussed the Forward Explicit Congestion Notification (FECN) and the Backward Explicit Congestion Notification (BECN) fields. The NCP uses these fields to manage congestion which can ocurr within the frame relay network. Figure 8 shows the use of these indicators between NCPs.

NCP A sends a packet of data into the frame relay network. Frame Handler 2 is congested which causes it to turn on the FECN indicator in the packet which it forwards to NCP B. The FECN indicator notifies NCP B that congestion has ocurred on the virtual circuit. In order to notify NCP A that it is causing congestion, NCP B sends NCP A an early acknowledgement of the data that it has received. Frame Handler 2 sets the BECN indicator on this packet which is being sent to NCP A. When NCP A receives the packet with the BECN indicator turned on, it reduces its sending rate until it receives a packet with the BECN indicator turned off.

During very severe congestion conditions the frame relay network may be forced to begin discarding

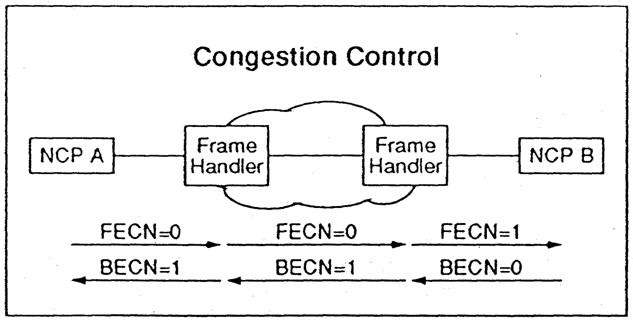


Figure 8

frames. Frame relay provides a prioritization indicator which is called Discard Eligibility to prevent important packets from being discarded. NCP exploits this capability by giving session control commands priority over ordinary data packets. This reduces the chance that session control commands will be discarded during severe congestion.

NCP as a Frame Handler

NCP Version 6 Release 2 can be configured as either frame relay Terminal Equipment or as a Frame Handler. This means that frame relay TEs can communicate with one another through a network backbone made up of interconnected NCPs. NCP V6R2 supports up to 239 virtual circuits per line. The maximum line speed is T1/E1.

Using NCP V6R2 allows the network designer to create networks of the type shown in Figure 9.

Note that the NCPs can support both frame relay and pure SNA traffic simultaneously. The data traffic is carried by a combination of dedicated data links and a public frame relay network. From the point-of-view of the frame relay TEs this is a single frame relay network. The SNA users view the network as a standard SNA subarea backbone network.

Enhanced Subarea Routing

NCPs can be configured as both frame relay FHs and TEs. This dual configuration gives the NCP some interesting new routing capabilities. Figure 10 shows a subarea backbone network which can exploit these capabilities by using frame relay protocols rather than traditional SNA subarea routing to forward packets across the backbone network. The use of frame realy makes the NCPs appear to be logically adjacent to one another.

By replacing SNA routing with the frame relay routing the backbone is enhanced in two ways. First, frame relay routing is more efficient than SNA subarea routing. The subarea routing is perfomed at layer 3—the Path Control Layer. As we discussed earlier, frame relay acheives higher performance by operating at layer 2—the Data Link Control layer. There has been much attention paid lately to an enhanced routing scheme for APPN which is being called APPN+. This new routing technique acheives higher performance by pushing the routing function from Path Control down into an enhanced Data Link Control layer. The combination of NCPs with frame relay might be said to result in Subarea+ routing.

Another important benefit that is gained by using the FHs to perform the routing function is improved reliability. The FHs dynamically reroute virtual circuits in the event of network failures. These combined enhancements give the SNA backbone a new look and new capabilities by combining the traditional SNA environment with the new frame relay technology.

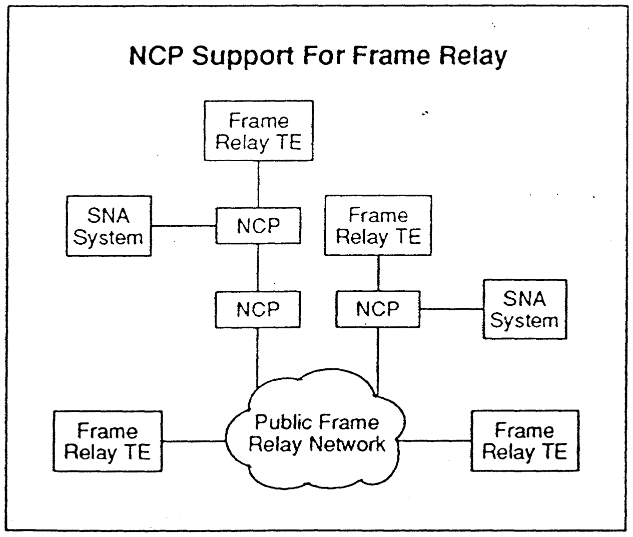


Figure 9

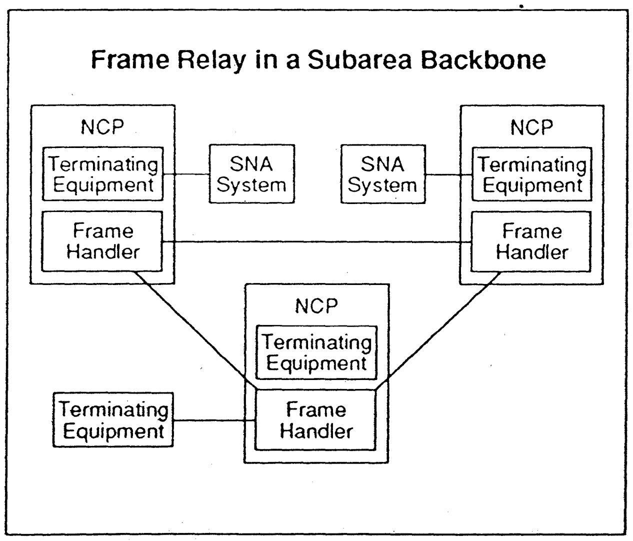


Figure 10

IBM's Other Frame Relay Products

NCP V6R1 and V6R2 are not IBM's only announced frame relay products. IBM has announced or is delivering various levels of frame relay support in its networking product line. The following IBM products have announced or delivered frame relay support.

- 6611 multiprotocol router
- Integrated Digital Network Exchange (IDNX)
- RouteXpander/2

The following table summarizes the frame relay support that is currently announced or being delivered in IBM networking products.

	NCP	6611	IDNX	RX/2
FR Terminating Equipment Support	Yes	Yes	Yes	Yes
FR Frame Handler Support	Yes	No	Yes	Yes
LMI Management	Yes	Yes	Yes	Yes
SNA or SNMP Management	SNA	SNMP	SNMP	Either
Maximum Frame Size	8,250	2,108	4,096	4,484
Max # of Virtual Connections/Port	218	254	1,024	200

Beyond Frame Relay Services

Frame relay represents the first major enhancement to packet switching technology since the introduction of X.25 in the mid-1970s. The main advantage of frame relay over X.25 can be summed up in one word—bandwidth. While X.25 networks generally don't provide connections at speeds higher than 64 Kbps., frame relay services are already being offered at T1/E1 speeds and speeds could potentially be increased in the future. But frame relay is not the technology which will offer the leap in bandwidth that is required to support services such as full motion video. Frame relay is also unsuitable for supporting real time voice communications. For these, and other reasons many industry observers are calling frame relay an interim technology.

With the current rapid changes in networking technologies, just about any technology can be considered interim, but we believe that frame relay will have a long useful life. There are several reasons for this. First, the frame relay interface can be implemented on a wide range of computers and internetworking products with little more than software upgrades. Newer high speed networking technologies such as Switched Multi-megabit Data Service (SMDS) will require both hardware and software upgrades.

The second reason that we believe frame relay will have a relatively long life is the fact that relatively few users will have a requirement for bandwidths in excess of T1/E1 rates for the foreseeable future. It is true that leading edge users will require higher bandwidths to support multimedia applications and some image processing applications, but these users will remain in the minority for at least the next few years.

Frame relay will also provide a migration path to the use of higher speed cell relay technologies in the future. Internetworking devices such as bridges and routers which today support frame relay interfaces will generally be upgradable to use high speed cell relay networks via a combination of hardware and software upgrades to those products.

In summary, frame relay is a good current solution for users who require very flexible wide area connectivity at speeds in the T1/E1 range while we wait for the applications which will require greater bandwidth and the private and public networks which will deliver those services. ■

Architect's Corner

DLS Not Panacea

by Dr. John R. Pickens

A great deal of excitement is being generated within the multi-vendor community about extending and standardizing IBM's Data Link Switching protocol (DLSw). I share that excitement of course. I, too, would like to see DLS refined and published as a multivendor standard. But, are not the expectations for DLS getting out of hand? Or, to use a metaphor from the medical community, doesn't it appear that DLS is being considered as the cure (aka panacea), rather than the ointment?

Increasingly I am seeing DLS being used loosely as a synonym for "SNA routing"—in the press, in marketing positioning statements, and in prognostications from the analyst community. As though DLS were the horizon of the journey—even the destination.

Of greater concern to me is the medicine cabinet filled to overflowing with DLS enhancement proposals:

- internodal flow control (between DLS routers)
- NT 2.1 XID
- Ethernet
- capabilities exchange negotiation
- SDLC link
- security
- DLS MIB
- Network Node architectural integration
- Directory services based configuration (!)
- Subarea transmission groups (!)
- Mainframe channel DLC (!)
- Transparent bridging—spanning tree (!)
- Priority (!)
- Subarea COS enhancements (!)

Wow!

Certainly some of the enhancements are required. Especially standardization of internodal flow control and certain data link support options. But support of spanning tree? Subarea? Priority?

I am concerned that by raising expectations too highly for the requirements addressed by DLS, that the vendor and customer community may defocus from the real objective—SNA routing.

DLS is, despite all the fanfare, a "bridging" technology—nothing more or less. DLS extends the properties of the datalink layer across routed backbones. Actually DLS supports a subset of bridging functions—i.e., for LLC Type 2 connection services—NetBIOS and SNA stacks only. It uses TCP as a virtual layer two pipe. Yes, DLS does a little more than bridging. It terminates layer two source routing (for token ring), and in the process serendipitiously provides a token ring to ethernet "SRTB" translation function (quite by luck). And, it terminates LLC2 connections locally, spoofing the end-to-end data link connection timers in the process. Also, since DLS is not a router, it does make some simplifying tradeoffs—such as requiring manual definition of partner DLS routers.

Some of the proposed enhancements—priority, subarea properties, directory services, and security seem to be more in the realm of "routing" functions, not "bridging" functions. I do not question that they cannot or should not be addressed by the DLS AIW working group, but rather that by striving to solve the problems "correctly", redundant work will be created to that already being performed robustly by the SNA "routing" layer. Other functions such as spanning tree protocol support (which I among others suggested—in a weak moment), are very tough problems to solve, and have tied up bridging experts in IEEE 802.1G for several years already. Of course APPN addresses all of the above (except the subarea properties—a good candidate for future APPN extensions).

Actually, DLS as it currently exists, limited though it may be, is not too far off the mark for a migration solution from bridging to routing. DLS offers a way to cross multiprotocol backbones via a pragmatic, simplified but extended bridging technology, with limitations. For the users that want to solve the problem completely, use APPN routing.

My preference would be to tidy up a few details for DLS, but be careful to constrain the requirements so that the effort doesn't get dragged down into what is essentially the invention of another routing protocol. Then, with the tidied up DLS, turn the efforts in two directions:

1. Work with the IEEE 802.1 G bridging work-group to add IP as a virtual datalink type for remote bridging.

2. Implement APPN—multiple datalink support, mapping to IP backbones, class of service, prioritization, security, etc. etc. Work to define more refined data link mappings—APPN to IP, APPN to PPP. Especially work for widespread adoption of APPN/HPR routing—whose RTP sublayer will be 3-10 times faster than TCP anyway (TCP does not make all that powerful of a tunnel anyway, with its slow start characteristics, lack of support for message block boundaries, and lack of prioritization).

So, what is the disease? Inadequate support of SNA sessions across multi protocol backbones.

What is the ointment? DLS-based bridging (with extensions) of SNA and Netbios traffic across TCP/IP backbones.

And what is the cure? APPN routing, especially APPN/HPR (not bridging). And, of course, do not forget APPN Dependent LU Requestor for 3270 (and other non LU6.2) session support. ■

Copyright © 1993 The Saratoga Group, all rights reserved. Reproduction is prohibited. • Subscription rates: U.S. - one year $395, two years $595. International - one year $445, two years $695 • *SNA Perspective* is published monthly by The Saratoga Group, 12930 Saratoga Avenue, Suite A-1, Saratoga, CA 95070 • Telephone (408) 446-9115 • Fax (408) 446-9134 • Managing Editor: Donald H. Czubek • Associate Editor: Stephen J. Randesi • Circulation Coordinator: Karen Stangel • Contributor: Dr. John R. Pickens • Typesetting and illustration: Aaron Lyon at dSIGN
• The information and opinions within are based on the best information available, but completeness and accuracy cannot be guaranteed.