IBM System z9 Open Systems Adapter for Communication Controller for Linux

M. Zee
J. W. Stevens
B. L. Thompson
J. A. Fowler
J. Goldman
P. T. Chan
T. P. McSweeney

The IBM 374x Communication Controllers, and the NCP (network control program) software that runs on them, have been at the center of the IBM SNA (Systems Network Architecture) for many years. However, the 374x hardware is no longer being produced. In order to continue to offer IBM customers various functions provided by the NCP product, IBM has developed a Communication Controller for Linux® (CCL) for the IBM System z™. CCL is a software program that emulates the 374x hardware, enabling the NCP to function in Linux. IBM customers now have the ability to migrate their NCP product to a Linux partition on System z. The current NCP product, running on an IBM 374x Communication Controller, supports both host channel and network attachment. The channel protocol used for the host-channel support is referred to as channel data link control (CDLC). In order to provide the System z9™ host operating systems with the ability to attach to the new CCL NCP over a channel interface, a new channel adapter is required. The new innovative Open Systems Adapter for NCP (OSN) channel support provided by the OSA-Express2 allows various operating systems on the same System z9 to attach "internally" to the CCL without using any external network or channel fabric.

Introduction

A communication controller [1] manages data input and output to a host computer or computer network. Such a device converts parallel computer data to serial data for transmission over communications lines. In 1973, IBM introduced the IBM 3705 Communication Controller. The IBM communication controllers serve as the control point for multiplexing various types of telecommunications equipment, and for providing host connectivity for that equipment. The controllers have served as a front-end processor for several generations of IBM mainframes, and the technology has evolved significantly over the last three decades, with important additions in functionality, support for new communication protocols, and various hardware performance enhancements concerned with memory, processor, and the adapter interface. This hardware controller is managed by the network control program (NCP). The NCP serves as the "operating system" for the controllers.

For more than 30 years, several generations of the IBM communication controllers have carried the bulk of the world's business traffic. After many years of service, IBM shipped its last 374x Communication Controller on December 31, 2002. A significant era in enterprise systems and Systems Network Architecture (SNA) communications, on which many large corporations have based their information technology, may have seemed to be approaching an end [2].

Although most IBM customers have migrated from SNA-based solutions to solutions based on Internet Protocol (IP), many corporations still depend on SNA and NCP functions to run their critical business applications [3]. One such key SNA function is SNA Network Interconnect (SNI). The SNI function is jointly provided by a Virtual Telecommunications Access Method

0018-8646/07/$5.00 © 2007 IBM

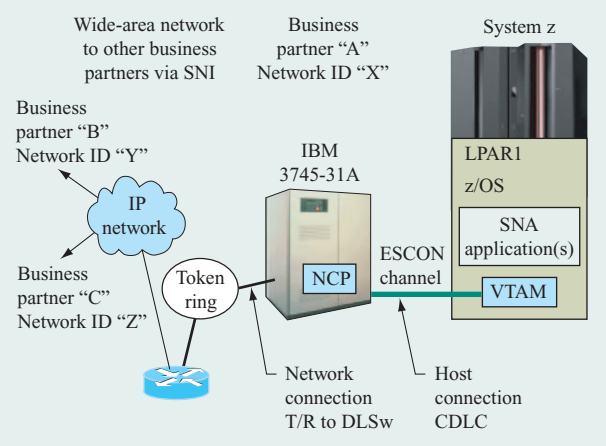

Figure 1

Typical channel-attached communication controller providing SNI. (T/R: token ring.)

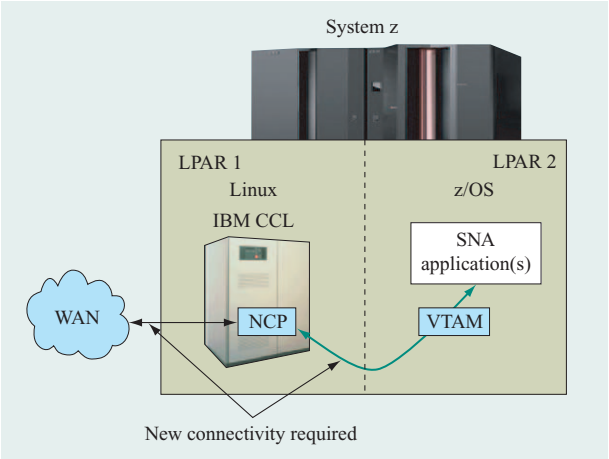

Figure 2

New CCL operating environment.

(VTAM*) host and an NCP. Together, the two products provide the SNI function, which allows two independent business partners, such as a retailer and a supplier, to communicate with each other and yet also remain autonomous to each other. Many business partners built their backbone networks on the basis of this technology.

For many years, AT&T provided a worldwide communications service that provided SNI connectivity to many independent business partners [4].

Figure 1 illustrates a typical example of how SNI is exploited. Here, an IBM System z* host (at right) is running VTAM with a channel-attached communication controller running NCP. In this example, the NCP is connected to the network using DLSw (data link switching). Various business partners are connected using the SNI technology, so that each partner is within its unique network, identified by a network ID.

While most NCP functions could be replaced with similar IP solutions, this was not true for SNI. Although several contemporary alternatives to SNI exist [3], attempts to replace SNI presented a unique set of business challenges. Business partners had to agree on the replacement technology and then carefully coordinate the migration of those changes.

Although IBM had announced an end of production for the 374x Communication Controllers, a clear business need began to emerge to provide various NCP functions, such as SNI, beyond the life of the hardware platform provided by the communication controller. Our team was challenged to solve this in such a way as to avoid producing an entirely new communication controller hardware platform.

The IBM Enterprise Networking Solutions (ENS) team in Research Triangle Park, North Carolina, evaluated the problem and proposed the creation of a new software control unit. The software control unit would transparently emulate the hardware control unit by providing a new operating environment for the NCP. Given that SNI required both a host and a control unit, this new control unit would be tightly integrated with the IBM System z host operating systems, which exploit NCP. The new emulated control unit would reside inside the System z within a Linux** operating system image. It would leverage key strengths of System z, such as virtualization (via z/VM* and logical partitions), the Integrated Facility for Linux (IFL) processors, and all of the critical quality-of-service attributes of z/OS* and the System z platform.

In May 2005, the Enterprise Networking Solutions team produced a new product called the IBM Communication Controller for Linux (CCL) [5], which serves as the new communication controller, emulating the 374x architecture and instruction set [6]. The NCP operates transparently in the new CCL environment. That is, the NCP can be loaded into the new CCL and then execute in the CCL environment without requiring NCP software product changes. Given that the NCP is a very large and complex product, this was a critical objective [7]. Most NCP users will be required to make some NCP configuration (e.g., definition) changes. **Figure 2** illustrates the new CCL operating environment. With the communication controller moved inside the System z, new host and network connectivity is required.

CCL host and network connectivity

Creating the CCL product required us to solve many unique design problems. One general set of problems

related to CCL connectivity. The hardware controllers had two main types of connectivity—host and network. Because the controller was to be used in the System z9, the design team had to ensure that the host and network connectivity of the controller could be provided in this new hardware environment. As we have mentioned, the solutions had to meet the key objective of avoiding product changes to the NCP and the numerous host operating systems that connect to the CCL NCP.

For host connectivity, the existing 374x controllers supported both LAN (token ring) and ESCON* channel connectivity. ESCON (Enterprise Systems Connection) is an optical serial interface between IBM mainframe computers and peripheral devices such as storage drives. With channel connectivity being the most prevalent type, we concluded that both forms of connectivity would be required, with each type presenting a unique set of challenges. Both forms of connectivity would require either new System z hardware or changes to the existing System z hardware. The LAN connectivity could be provided with only minor changes to the existing OSA (Open Systems Adapter), and a minor change to the host networking subsystem (VTAM). However, if channel support was deemed necessary, this support would require a new version of old technology.

Channel connectivity offered various operational advantages over LAN connectivity and was the only form of connectivity that was supported by the TPF operating system. Transactional Processing Facility (TPF) is the IBM high-performance transactional processing monitor. The combination of TPF and NCP products was still critical for various IBM enterprise customers.

We concluded that LAN connectivity alone would not be sufficient. In order to make CCL a viable solution, the channel connectivity would ultimately be required. Therefore, the host connectivity would have to be incorporated in the product. The initial release of CCL provided only LAN connectivity. The follow-on release of CCL (October 2005) provided the channel support. Designers concerned themselves with the type of hardware that could be exploited to provide this new version of "channel" support, and how this function could be provided quickly for the System z.

When evaluating the host and CCL requirements along with the critical time frame, we decided that the solution must be based on current System z technology. The channel solution presented a challenging set of requirements:

1. The solution must remain transparent to the host operating systems and NCP, where each host must continue to use the existing channel protocol called channel data link control (CDLC).
2. The solution cannot require adding new external hardware such as adapters, cables, or fabric, and switches.
3. The performance characteristics of the solution are critical in terms of both throughput and instructions per second.

A new OSN channel to support CDLC

We created a new channel type to enable all System z9 operating systems to communicate with NCP on the Linux image by using the channel-attach (CDLC) method. It was important to maintain the same CDLC support for those operating systems that are currently attached to IBM 374x Communication Controllers. This channel support is necessary in order to enable those operating systems that require management of loading, dumping, controlling, and operating an NCP. These functions are complicated and can only be supported over the channel-attached CDLC interface. Our objective was to maintain the current level of the software, along with the use of this new channel type, so that neither the NCP nor the host interface software has to change. Offering this feature on the System z9 as a completely new hardware interface would not be feasible. The new channel type is called Open Systems Adapter for NCP (OSN); its main function is to bridge an operating system image using CDLC protocol to the CCL in the Linux image using the queued direct I/O (QDIO) architecture [8].

The OSN channel provides the CDLC channel with connectivity between logical partitions (LPARs) or virtual machines in the same mainframe system. OSN CDLC support was critical for TPF users. TPF supports only channel connectivity to an NCP. Therefore, as discussed, TPF systems can connect to a CCL NCP only through the new OSN. The communications server (VTAM) can also leverage the OSN channel, which provides a CDLC interface for *internal* connectivity to CCLs. When CCLs are running in the same mainframe system as a VTAM, OSN connectivity between that VTAM and the CCLs is considerably more efficient than using VTAM external communications adapter (XCA) communications through an external LAN. One additional advantage of using OSN for VTAM connectivity to CCLs is that the CCL NCP is configured and managed as a local, channel-attached NCP rather than as a remote NCP. As a local NCP, the CCL NCP can be loaded and managed in the same way as local 3745 Communication Controllers, further reducing the changes required to migrate from 3745s to CCLs.

The bridging function must transport SNA frames from a host operating system (e.g., z/OS) image internally to the Linux image running CCL NCP on the same mainframe using CDLC. In practice, multiple

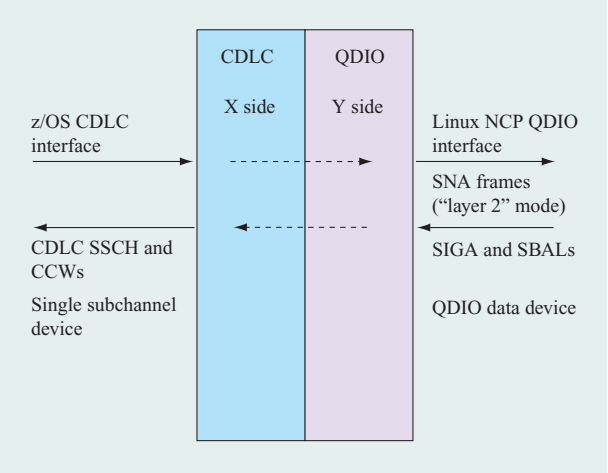

Figure 3

Concepts underlying the OSN CDLC architecture. (CCW: channel command word; SIGA: signal adapter; SSCH: start subchannel instruction; SBALs: storage buffer access lists.)

connections and instances of both the host OS and Linux can be supported. Again, one of the major focuses was to provide an identical CDLC interface to those operating systems that use the channel-attach method to communicate with the NCP, thereby avoiding updating numerous operating systems software products. This can provide a transparent and functionally equivalent option that allows customers to migrate from the current IBM 3745 Communication Controllers.

Figure 3 illustrates the concepts that underlie an OSN adapter, which appears to have two functional "sides" (interfaces)—the X side and the Y side. The two-part description which follows refers to the CDLC (host) side as the X side and the QDIO (CCL) side as the Y side. The X- and Y-side roles are not negotiated; they are permanently associated with the CDLC host image (X) and QDIO Linux image (Y). The two sides are described as follows:

1. *CDLC X side:* Here, the host (e.g., z/OS) image, using the CDLC architecture, communicates with the OSN CDLC "control unit." The system device support appears as a 374x-type device.
2. *QDIO Y side:* Here, the CCL image, using the QDIO architecture [8], communicates with the OSN, which connects to the host OS. The system device support appears as an OSN device.

OSN forwards or "bridges" the SNA frames from the source half of the interface to the target half. For each I/O event, the OSN bridging functions do the following:

1. Find the corresponding X or Y half of the connection.
2. Add or remove the 32-byte QDIO header.
3. Insert the proper identifier in the header when adding (i.e., building) the 32-byte header that is inbound to CCL.
4. Forward the frame or frames to the target host image when necessary.
5. Respond to the originator of the I/O event.

The CDLC half (X side) performs both read and write operations over the CDLC interface using the standard CDLC channel programs. Each CDLC interface to the host operating system requires the allocation of a single 3745 device. The QDIO half (Y side) uses the QDIO interface for I/O operations between the OSN channel and CCL. Each QDIO interface with the Linux image requires the allocation of three OSN devices (two for control and one for data devices). Because OSN performs only a bridging function, it does not examine the SNA frames and provides no SNA protocol logic or functionalities.

Before bridging can take place between the host operating system and the Linux NCP image, a logical connection must be established between the X side and Y side. Each logical connection is uniquely associated with an OSN data device and the corresponding CDLC device. The association is represented by a unique identifier called a channel connection identifier (CCID), which associates each host and the corresponding CCL instance as a single logical connection. The complete CCID value consists of the subsystem CSS (channel subsystem) identifier, multiple image facility (MIF) identifier, and unit address to the CDLC device. The unique pair of devices can be correlated by OSN using the CCID.

To create a single logical connection, a minimum of four devices are needed (three OSN devices and one CDLC device). Each OSN can support a maximum of 180 CDLC (3745-type) devices and 480 OSN (QDIO) control/data devices.

Inside CCL

The Communication Controller for Linux on the System z is a software emulation of the IBM 3745/3746 M900 Communication Controller model 31 A. The CCL runs as a Linux application in user space. The CCL allows a network control program (NCP) to be executed within the CCL as though the NCP were running on actual 374x hardware. In order to provide continued NCP support, an emulator solution had to be used in order to move the NCP function to the System z. The actual 374x is a specialized processor, with a specialized instruction set, that prohibits the porting of the NCP code directly to the System z.

The CCL provides most, but not all, of the functions provided by the 3745 hardware. Functions that are not supported are the following: CCU (central control unit) cycle utilization, 374x base frame adapters, start/stop, BiSynch (binary synchronous), SDLC (synchronous data link control), Frame Relay, EP (emulation program), Ethernet, and 374x type-6 and type-7 channel adapters.

Multiple CCL NCPs can be run in a single Linux partition. System resources, such as the amount of CPU processing and memory consumed by each CCL, can become limiting factors when the deployment of multiple CCLs within the same Linux image is under consideration. The option of running multiple CCLs within the same Linux image permits the possible migration of multiple 3745/NCP control units to the new CCL environment.

Network communications to the CCL are provided using emulated 3745 token-ring interface coupler (TIC) 2 or 3746 M900 TIC 3-type token-ring adapters. The underlying physical media can be either a token ring or an Ethernet. Network communications between CCLs can also be achieved using the Internet Protocol transmission group (IPTG) function of CCL, which allows connections between CCLs to be established over a TCP/IP network. For this function, the connection to NCP appears to the NCP as a token-ring connection through the use of an emulated IBM 3746 M900 TIC3-type adapter.

Network communications and CDLC connections are provided by the network device handler (NDH). The CCL opens a socket pair to the NDH for each physical connection, and data subsequently passes over the CCL to the NDH socket interface. For token-ring connections, complete token-ring frames are passed. For CDLC connections, the data consists of OSN primitives.

The IPTG function does not use the NDH for connectivity, but instead opens a Linux TCP/IP socket. The data is exchanged using standard API socket calls, and then the data travels over the TCP/IP network.

As in the actual IBM 3745 hardware, the CCL provides 16 megabytes of memory for the NCP, as well as eight general registers, 127 external registers, and five program-interrupt levels. Many of the hardware functions are emulated by the CCL. Most of the CCL is written in C code, with the exception of the core CCU function, which had to be written in assembly language to achieve high-performance functioning. Because the CCL performs its functions by emulating the 3745 instruction set, performance would be degraded if the core CCU functions were written in C.

The CCL is a multithreaded application, with the main thread providing the CCU function, which is responsible for executing the IBM 3745 instruction set. A thread also exists for the maintenance and operator subsystem (MOSS), as well as at least two threads for each adapter [TIC2, TIC3, ESCON, X.25 NPSI (NCP Packet Switching Interface), IPTG] that is activated. Most of the time consumed by the CCL is consumed within the CCU thread, which executes IBM 3745 instructions. As each instruction is fetched, it is decoded and then executed as 390 instructions to perform the desired operation; hence, the CCL is an emulator and interpreter. When a 3745 instruction is directed for an adapter operation, the adapter thread is signaled and control is passed to the appropriate threads in order to finish processing the requests. The adapter thread operations are typically done in parallel with the CCU thread operations.

For CCL-to-OSN communications, data and control information is passed between the CCU and the adapter threads in dynamic parameter/status areas (DPSAs), which are structures that were defined for the IBM 3745/3746 interface based on the OSI (Open Systems Interconnection) interface. (The OSI model is a layered description for communications and computer network protocol design.) The adapter thread code is then responsible for translating DPSAs to OSN primitives, or OSN primitives to DPSAs. Similar processing is performed for network communications, except that instead of involving OSN primitives, the DPSA translation takes place between DPSAs and token-ring frames.

CCL with CDLC support allows the loading or dumping of an NCP over the CDLC interface. If the CCL does not have CDLC connectivity, the loading and dumping of an NCP can be performed only by using the emulated 3745 disk functions. This means that when the CCL is first used, the load module must be placed on the Linux system using an FTP-type program in order to transfer the load module from the host operating system to the Linux system, and the CCL is then started with a particular NCP name. With CDLC, however, the NCP can be loaded over the CDLC connection. Unlike the actual 3745, which can have multiple channel adapters assigned for loading or dumping, the CCL provides only one adapter interface. This adapter interface is defined in both the NCP generation deck [9] and the CCL configuration file [10].

Loading with CDLC has an obvious advantage because the extra step of manually transferring the NCP load module from the host operating system to the Linux system can be avoided. Instead, the CCL can be started with no load module, and then the load module can be transferred over the CDLC connection and started automatically.

The CCL provides an HTTP server that is used to emulate some of the functions provided by the IBM 3745 MOSS. The MOSS is an easy-to-use interface that permits a variety of functions to be performed by the

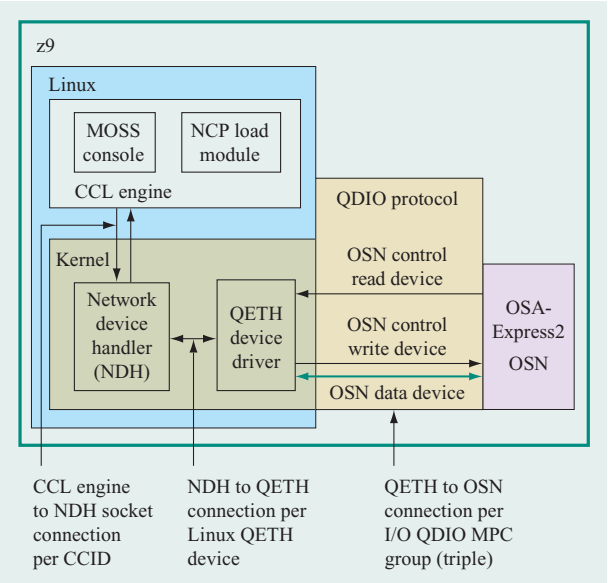

Figure 4

NDH interfaces with CCL and QETH. The term "triple" indicates a group of three devices. (MPC: multipath channel.)

operator, such as storage and register alteration, starting and stopping of NCP and CCL traces, restarting the NCP or CCL, or displaying various logs related to the CCL and NCP. The MOSS also provides multiple load-module support and timed IPL support functions that are currently provided by the 3745 hardware. The MOSS interface can also permit an IBM support service to access a particular CCL if needed in order to provide remote diagnostic support.

CCL network device handler (NDH)

The path traveled by a frame between the OSN device and the CCL NCP within the Linux image consists of multiple steps. The term *steps* here refers to the transfer of the frame or control of the frame from one system component to another, and this transfer may not be a physical copying of the data. The direct connection between OSN and Linux is implemented using the existing Linux OSA-Express QETH (QDIO Ethernet) device driver. The standard QDIO architecture serves as the basis for OSN, which utilizes a single input (read-control) device, a single output (write-control) device, and a single data device. The Linux QETH device driver is the enabler of the OSN communications to the Linux image. We created the network device handler (NDH) to serve as a transport layer between the CCL engine and the device driver. **Figure 4** illustrates how the NDH fits into the CCL solution and the relationship of the NDH to the other components.

With respect to QETH support, the role of the QETH device driver is somewhat different in the OSN device case. Because QDIO headers and OSN primitives are based on CCID-level information, tasks directly associated with the manipulation and creation of these are not performed by the QETH Linux device driver. The Linux driver maintains device-level information, and multiple CCIDs can be associated with a single OSN device. The QETH driver does not maintain any individual sessions or CCID associations. Therefore, the work required to manage the QDIO headers was displaced from the QETH driver up a level to the NDH. The NDH is a kernel module and has implemented a direct interface with the QETH driver. This interaction between the NDH and QETH is one of the multiple steps that a frame traverses between OSN and the CCL NCP.

The NDH is an external loadable kernel module that provides the interface between the CCL engine and the Linux QETH device driver. The NDH maintains the association between the Linux device, of which the QETH driver has knowledge, and the CCID, of which the CCL engine has knowledge. This association is enabled by defining the OSN read-device subchannel address within the NCP generation deck. The NDH registers a callback function with QETH that is invoked to pass all frames received over an individual OSN device. This registration occurs on a per-Linux device basis. The NDH has a connection on a per-CCID basis with the CCL engine via AF_NDH (address family, network device handler) sockets. This maintained association allows the NDH to build and interpret the OSN QDIO headers. The termination of this per-CCID socket connection with the CCL engine causes the NDH to send the DEL_CCID (delete CCID) primitive in order to terminate the session connection. (A primitive may be considered as a command or signal.) Neither the session level nor CCID state data is maintained by the NDH. The CCID-level state information is maintained only by the two farthest endpoints (the CCL engine and the OSA) that interact with the Linux part of this system. The passing of the frame to and from the CCL engine via the AF_NDH sockets is another one of the multiple steps traversed by a frame in its path between OSN and the CCL NCP.

Inside the Open Systems Adapter for NCP (OSN)

The Open Systems Adapter-Express2 (OSA-Express2) is used to provide connectivity to local area networks (LANs). In order to configure an OSA-Express2 as an OSN channel path ID (CHPID), the user must configure the CHPID type and devices (e.g., CDLC and OSN) by using the System z I/O configuration tools, HCD (hardware configuration definition), or IOCP (I/O control program) [11] input statements. (HCD serves as a tool used to define I/O devices, and IOCP is part of the S/390*

microcode that uses the output of those definitions.) In order for the OSN to function properly, both 3745 devices (X-side CDLC) and OSN devices (Y-side QDIO/Linux) must be configured. When the OSA is initialized during power-on-reset, the I/O configuration definitions (IOCDs) are loaded into the adapter. When an OSA-Express2 channel is defined as an OSN CHPID type, the new OSN code is loaded in the OSN memory, and the OSA-Express2 is viewed by the system as an OSN channel. Once configured online, OSN dynamically learns about the 3745 devices in the I/O configuration and builds an internal table entry for each one.

Figure 5 illustrates the dataflow between the CDLC interface and the QDIO interface. To create an association between a CCL instance and a 3745 device, a registration process is required to exchange information between the CCL instance and the VTAM or TPF in the host image. During the registration process, the CCL uses the configured values of NCP parameters and forms a CCID (comprising the CSS ID, MIF ID, and unit address [12]). The CCL registers with OSN to establish a logical point-to-point connection using a SET_CCID primitive. The SET_CCID primitive contains information that identifies the target 3745 device that is to bind with the CCL to subsequently form a connection. [Set, Delete, and Modify are the three types of primitives (i.e., commands) that we created for the OSN support. These are new commands created to register (set) a CCID, deregister (delete), and modify (mod) the CCID.] OSN examines the request and validates the target 3745 device through its internal table entries. If the target 3745 device is not defined or it is currently in use, the SET_CCID primitive is rejected with a negative reply by OSN. This is considered to be a configuration error.

If the target 3745 device is found to be valid, a table entry for this unique CCID is created to form a relationship between the CCL and the 3745 device specified by the unique CCID. Each table entry represents a "bridged connection" between a 3745 device and an OSN device. Negotiations continue between the CCL in the Linux image and VTAM in the host OS image. A pre-determined set of CCW commands—for example, Write IPL, Dump/Load, Write/Read Start, Contact/Discontact, Restart-Reset and Write/Read XID (exchange identifier)—flow over the 3745 device in a specified sequence, and the information is repackaged and sent to the CCL in the Linux image using the MOD_CCID primitive. The information flows from CCL to host OS in a similar fashion.

The OSN has the responsibility to maintain a basic "state machine" for each connection, and a table entry (CCID) is created for each connection. The initial connection state is RESET and does not become ACTIVE until XID parameters are successfully

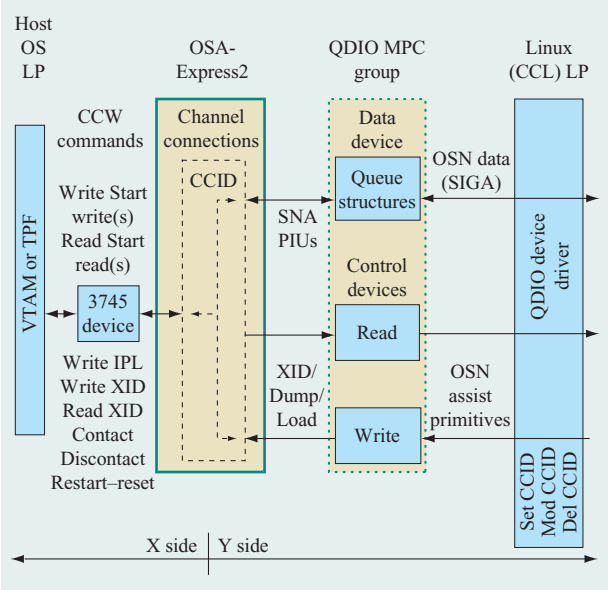

Figure 5

Inside OSA-Express2 OSN. (CCW: channel command word; SIGA: signal adapter; XID: exchange ID.)

negotiated by both sides. Once the registration process is completed between CCL and OSN via the SET_CCID, the control-type CCW commands (e.g., XID exchange or dump/load) can flow from the host OS via the 3745 device to reach the OSN CHPID. The OSN validates the CCID table entry for every control CCW command sent and received by the 3745 device. If the table entry is valid, CCW commands and data are sent by the OSN control devices to the CCL in the Linux image using the MOD_CCID primitive.

Once the XID sequence completes successfully and the CCL transfers the parameters to OSN, the connection is considered to be ACTIVE. When the connection state is ACTIVE, read and write I/O operations are permitted through the OSN data device. For a 3745 write operation, OSN has transferred all of the SNA PIUs (path information units, which describe SNA messages) for the entire write channel program into OSN memory from the host OS via the 3745 device. OSN then repackages the data (adding the QDIO header) and transfers it to the Linux image using the OSN data device via QDIO queue structures [8].

For a QDIO write operation, OSN receives a signal from the Linux image. OSN copies all of the data frames from Linux memory space into OSN memory space. If the targeted 3745 device is not currently receiving input data, OSN issues an attention to alert the operating system in the targeted host image. As part of a

disassembling process, QDIO headers are removed and the data frames are disassembled into individual PIUs and transferred to the VTAM/TPF in the host OS via a read channel program. This process continues until data is exhausted. When the read program is completed, OSN is required to hold the data until the host issues the next read I/O instruction or reissues the previous read I/O instruction. If the previous read I/O instruction is reissued, OSN resends all of the PIUs that were buffered; otherwise, all of the data is purged at the start of the new read channel program. Any queuing logic managed by OSN must preserve the original order of frames and not allow the frames to be dropped.

In some networking environments, such as an environment with TCP/IP, it is acceptable for packets to be "dropped," for example in instances in which there are no internal buffers available to data coming from the LAN or from the sending host image. The higher-level architecture forces the packets to be retransmitted, and eventually the packets should reach the intended destination.

In the OSN design, packet dropping is not acceptable. A channel connection for an SNA network is considered a "reliable transport" in which packet loss or dropped frames cannot be tolerated. Any lost frames over a channel connection in an SNA network result in the SNA connection being disconnected. To avoid this, a channel protocol known as "slowdown" exists. This provides a way of controlling dataflow between the host, OSN, and CCL. Each side has the ability to indicate to the other side that it is reaching its maximum utilization of buffer resources. An "enter-slowdown" indicator is reported to the sender of data when its buffer resources are about to be depleted. When this indicator is received, the sender is required to stop transmitting packets until the receiver of data indicates that the out-of-buffer condition has been resolved.

A delay may exist when the receiver realizes that it is too backed up and before the sender can be notified of this condition. We instituted an algorithm in the OSN such that when its buffer pool is at 80% capacity, OSN enters the abovementioned "slowdown" state. This state tells the sending CCL that OSN is backed up and that it should stop sending data frames. The sending CCL then "holds back" and waits until an "exit slowdown" indicator is sent by OSN before it resumes sending data. The exit slowdown is triggered when the OSN buffers have dropped to 25% usage.

When an active connection must be terminated, CCL sends a DEL_CCID primitive to OSN. Upon receiving the DEL_CCID, OSN unbinds the X side and the Y side and releases all of the associated resources for that connection. The DEL_CCID primitive is designed to be used for normal termination or abnormal termination of a connection.

OSN—the final product

CDLC over OSN has been designed for high-performance functioning. CDLC connections between CCL NCP and an SNA host over OSN have a natural advantage over LAN and even ESCON connections: Data flowing through OSN is not limited by the speed of the external physical media (e.g., because there is no external channel fabric or wire). Data flows at bus speed from the SNA host into the OSA, and again at bus speed from the OSA into the Linux image in which CCL resides. The SNA host uses CDLC to communicate with the OSA. Despite certain inefficiencies of the CDLC protocol compared with more modern technologies such as QDIO, VTAM can communicate with adjacent NCPs more efficiently over CDLC than over an LSA (link services architecture) LAN. (LSA is an SNA-oriented channel protocol.) Therefore, customers who deploy the OSN support instead of LSA will reduce z/OS cycles and gain efficiency.

On the Linux side, the CDLC data flows over a QDIO interface to the Linux image, which is more efficient than the process of LAN data flowing over LCS (LAN channel station, an IP-oriented channel protocol). However, the biggest performance savings come from the fact that CCL NCP does not have to manage the CDLC protocols that deal with such items as timers and channel status. The NCP allows the (emulated) ESCON adapter to handle such protocols, thus minimizing the number of emulated 374x instructions that have to be executed for data transfer to and from the SNA host. In addition, the OSN support has been designed so that the OSA adapter manages the details of the CDLC protocols, minimizing the processing in the Linux image. In contrast, for NTRI (NCP token-ring interface) LAN connections, the NCP manages all of the details of every LLC2 (link layer control-2) connection, using 374x instructions. (The term *LLC2* refers to the upper portion of the OSI layer-2 data-link control layer and is concerned with connection-oriented traffic.) This is processor-intensive work in an actual 3745, and the burden on the processor is magnified because CCL has to emulate the 3745 instructions.

To determine the relative performance of CDLC over OSN compared with 374x ESCON TIC3 adapters and also with LAN connections over LSA and LCS, we took a number of performance measurements in a controlled environment, using dedicated processors and isolated networking equipment. We set up two network configurations that are representative of most of the SNA traffic running through NCPs today: an SNA Network Interconnect (SNI) back-to-back configuration with two VTAMs and two NCPs, and a boundary network node (BNN) configuration with one VTAM, one NCP, and a number of peripheral (BNN) nodes.

For each of the two network configurations, we set up the following three SNI hardware configurations:

1. NCPs running in 3745s (with TIC3) that were attached to VTAMs via ESCON channels [**Figure 6(a)**].
2. NCPs running in CCLs that were attached via LAN (100M Fast Ethernet) to their VTAMs [**Figure 6(b)**].
3. NCPs running in CCLs that were attached via OSN to their VTAMs [**Figure 7(a)**].

We then performed a number of measurements in each configuration, varying the number of sessions and the think time (i.e., delay time) between transactions.

Figure 7(b) illustrates the traditional hardware BNN configuration with one VTAM using an ESCON-attached 3745 NCP, and a number of peripheral (BNN) nodes.

In a manner similar to the SNI test environment, we then replicated this BNN environment twice more. We began by replacing the 3745 NCP with CCL NCP using LAN connectivity between VTAM and CCL. Finally, we replaced the LAN connectivity with the OSN connectivity between VTAM and CCL. Note that for the BNN environment we have provided only a single illustration of the initial hardware configuration.

Next, we performed a number of measurements using each configuration, varying the number of sessions and the think time between transactions. **Figure 8** illustrates the average zLinux (or 374x) processor utilization and throughput measurements for the different configurations. Figure 8(a) provides the SNI comparisons, and Figure 8(b) shows the BNN comparisons.

Performance conclusions and summary

In both network configurations, the number of transactions per second for the 3745 peaked at slightly more than 400 transactions per second, and was far outrun by CCL on the System z9. The most interesting comparison is between the FE (Fast Ethernet) and CDLC graphs in Figures 8(a) and 8(b). The only difference between those two configurations is the connection between the CCL NCP and the VTAM. The CDLC configuration achieves more throughput using less CPU than FE. For the purposes of comparing the computational cost of the two solutions, we use these graphs to calculate the CPU cost on a per-transaction basis for each configuration. Using CDLC instead of FE can reduce the CPU cost per transaction by 18% on average, and as much as 30% in the Linux image running CCL. In addition, using CDLC instead of FE results in a z/OS CPU cost per transaction savings of 6% on average, with savings as high as 18% in some cases [13].

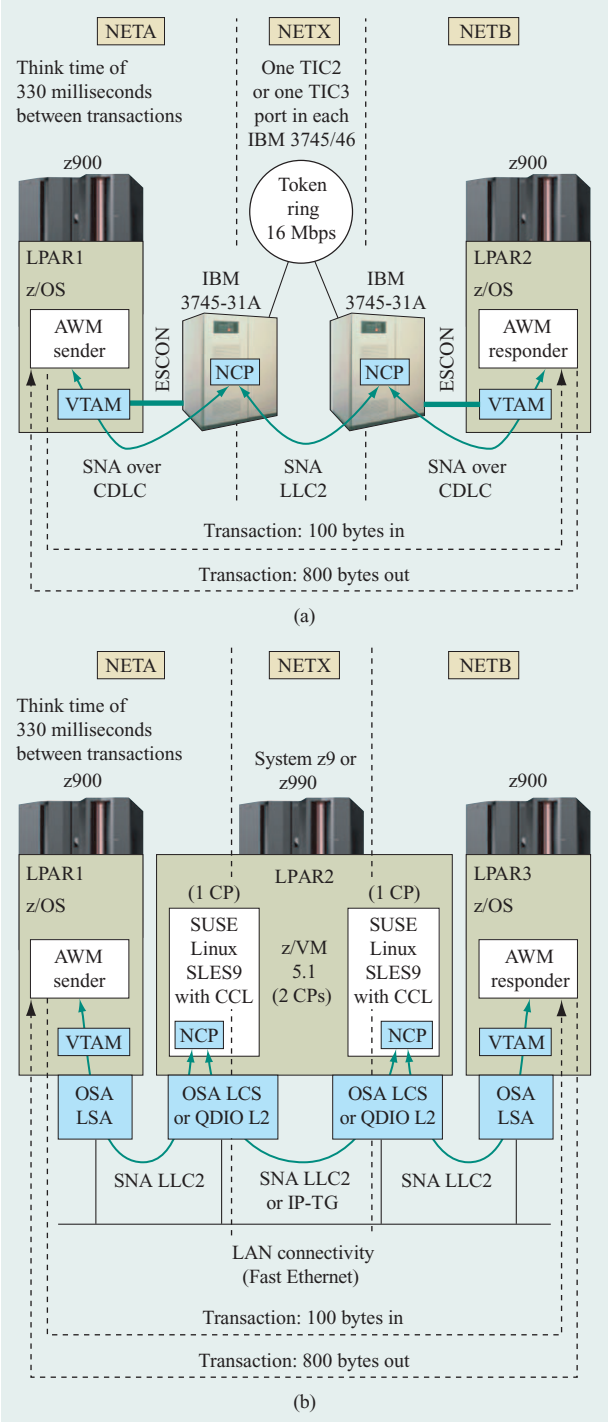

Figure 6

(a) IBM 3745/46 SNI test environment. The term "transaction" refers to a transactional workload or test case. (AWM: application workload modeler, a test tool that is used to generate and simulate the application workload; LLC2: link layer control-2.) (b) IBM CCL SNI test environment, using a shared LAN between VTAM and CCL. (SUSE** is a major retail Linux distribution, and SLES9 is a particular Linux release. CP: central processor.)

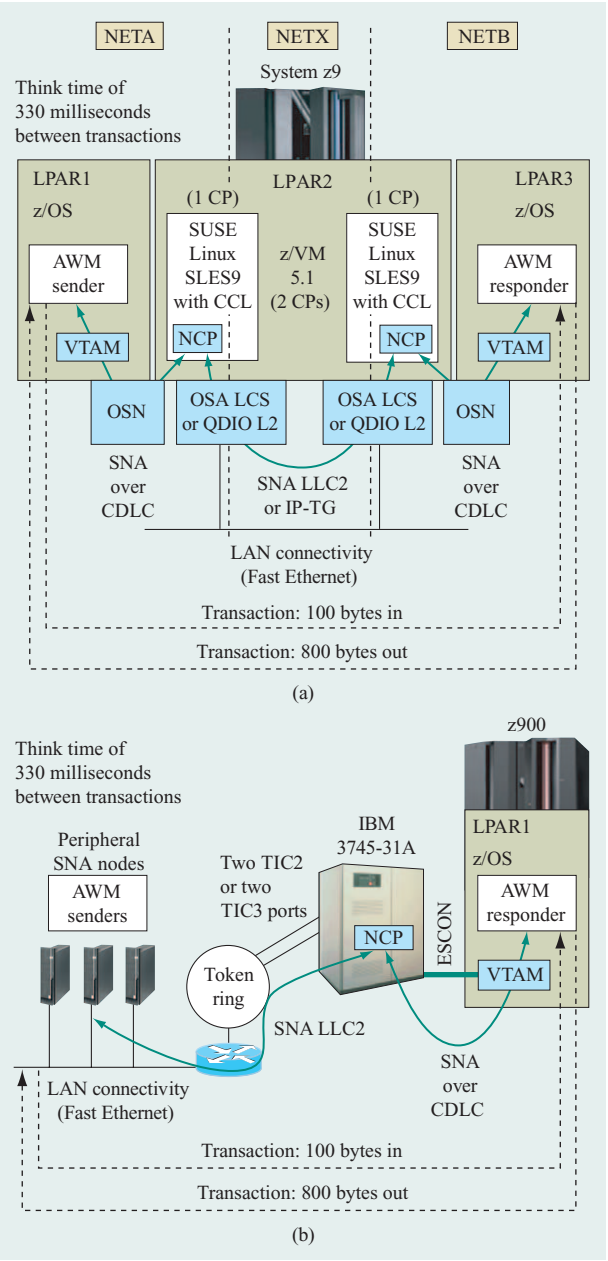

Figure 7

(a) IBM CCL SNI test environment using OSN between VTAM and CCL. (b) IBM traditional BNN test environment using actual 3745 NCP.

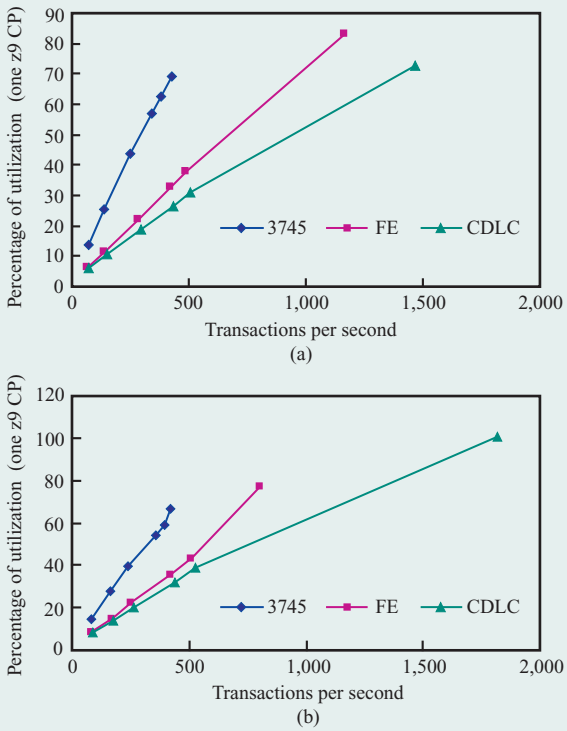

Figure 8

(a) SNI FE vs. CDLC performance graph. (FE: Fast Ethernet; CP: central processor.) (b) BNN FE vs. CDLC performance graph.

System operations using OSN

The Open Systems Adapter for NCP (OSN) functions as a CDLC-attached 3745 device to the SNA host systems in the System z9 CPC (central processing complex). For the 3745 device, the OSA microcode supports the same CDLC protocol used by the 3746 Model 900 ESCON adapter. This means that there are no required software changes in the SNA (VTAM or TPF) host to exploit the OSN support to a CCL NCP [14]. In other words, VTAM operating in a z/OS environment, or TPF, or VTAM operating in a VM or VSE environment can communicate and operate with an OSN-attached CCL NCP as if it were connected to a 3745 NCP over an ESCON CDLC connection.

Both subarea and peripheral CDLC connections are supported through this OSN connection to CCL NCP. The VTAM and TPF definitions for channel attachments through OSN to CCL NCP are configured in the same manner as VTAM and TPF definitions for channel attachments through ESCON to a 3745 NCP. Similarly, each SNA host OSN CDLC connection is defined in the CCL NCP definitions as an ESCON logical PU (processor unit). There is one ESCON logical PU definition in NCP for each OSN CDLC connection to an SNA host, just as there is for each 3745 ESCON CDLC-attached SNA host.

CCL NCP load and dump operations over an OSN-based CDLC connection are supported in the same manner as they are for an ESCON-attached 3745 NCP. The same VTAM commands and host utilities can be used to manage CCL NCP load and dump operations.

These operations over the OSN CDLC connection take less than one third of the time that the corresponding action takes over a 3745 ESCON CDLC connection.

In summary, our new System z9 OSN functionality achieved all of the critical functional design and performance objectives that were necessary in making the CCL a complete solution. The OSN support is tightly integrated into the CCL, complementing the communications solutions provided by the CCL. The innovative System z9 technology, such as IFLs, LPAR, and OSN when combined with the CCL, creates a powerful overall solution, allowing our customers to continue to exploit NCP functionality for their mission-critical applications for years to come [15].

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark service mark, or registered trademark of Linus Torvalds, Novell, Inc., The Open Group, or Microsoft in the United States, other countries, or both.

References

1. T. Sheldon, "Communication Controller: Linktionary Term," *Encyclopedia of Networking and Telecommunications*; see *http://www.linktionary.com/c/comm_controller.html*.
2. IBM Corporation, "IBM Networking Hardware: 3745 Communications Controller and NCP Network Control Program"; see *http://www.networking.ibm.com/nhd/webnav.nsf/pages/375:375prod.html*.
3. B. Louden, IBM Corporation, "IBM Communication Controller Migration Guide," IBM Redbooks; see *http://www.redbooks.ibm.com/abstracts/sg246298.html*.
4. AT&T Service Guide,"AT&T Global Network Services AT&T Managed Data Network Services – Connectivity Services Withdrawn from Marketing"; see *http://www.att.com/abs/serviceguide/docs/agnscs_sg.doc*.
5. IBM Corporation, "IBM Communication Controller for Linux on System z9 and zSeries V1.2 Extends NCP Connectivity in the Linux Environment," IBM United States Software Announcement 205–267; see *http://www-306.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=an&subtype=ca&appname=GPA&htmlfid=897/ENUS205–267*.
6. J. Stevens, L. Napoli, and E. Lewis, IBM Corporation, "Back to the Future: New Environment for SNA Applications," *z/OS Hot Topics Newsletter*, Issue 12, February 2005 (GA22-7501-08); see *http://publibz.boulder.ibm.com/epubs/pdf/e0z2n151.pdf*.
7. A. Gurugé, "IBM Communication Controller for Linux on System z9 and zSeries V1R2: Extending the IBM 3745/46 Legacy for Yet Another Decade," *IT In-Depth*, October 2005; see *http://www.itindepth.com/AnuDocs/CCL_WP1.pdf*.
8. M. E. Baskey, M. Eder, D. A. Elko, B. H. Ratcliff, and D. W. Schmidt, "zSeries Feature for Optimized Sockets-Based Messaging: HiperSockets and OSA-Express," *IBM J. Res. & Dev.* **46**, No. 4/5, 475–485 (2002).
9. IBM Corporation, "IBM Networking: ACF/NCP, ACF/SSP, EP, NPSI, and NTuneMon Product Documentation"; see *http://www.networking.ibm.com/nhd/webnav.nsf/pages/375:public.html*.
10. IBM Corporation, "Communication Controller for Linux on System z"; see *http://www-306.ibm.com/software/network/ccl/library/*.
11. P. Rogers, A. Salla, and L. Sousa, "ABCs of z/OS System Programming Volume 10," September 2006, IBM Redbooks; see *http://www.redbooks.ibm.com/redbooks/pdfs/sg246990.pdf*.
12. B. White, W. Fries, D. Jorna, H. Kordmann, J. Nesbitt, F. Packheiser, and E. Palacio, "IBM System z Connectivity Handbook," May 2006, IBM Redbooks; see *http://www.redbooks.ibm.com/redpieces/pdfs/sg245444.pdf*.
13. B. Perrone and A. Christensen, IBM Corporation, "IBM Communication Controller for Linux on System z V1R2.1 (CCL) and System z CPU Capacity Planning Information for SNI and Boundary Function Workload," September 2006; see *http://www-1.ibm.com/support/docview.wss?uid=swg27006207&aid=1*.
14. B. White, D. Di Casoli, O. Ferreira, W. Porschen, and M. Riches, "IBM Communication Controller for Linux on System z V1R2.1 Implementation Guide," November 2006, IBM Redbooks; see *http://www.redbooks.ibm.com/abstracts/sg247223.html?Open*.
15. IBM Corporation, "Communication Controller for Linux on System z"; see *http://www-306.ibm.com/software/network/ccl/*.

Received March 22, 2006; accepted for publication April 25, 2006; Internet publication December 5, 2006

Mooheng Zee *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (mzee@us.ibm.com).* In 1988, Mr. Zee received a B.S. degree in electrical engineering from Polytechnic University, joining IBM that same year in Poughkeepsie, New York. He started in the ESCON channel diagnostic group and then moved to the networking I/O development team to help develop the ATM networking products for the IBM eServer*. In his fifteen years in networking development, Mr. Zee has received several IBM Outstanding Technical Achievement Awards and two patents. He is currently involved in the development and continued enhancement of the networking channel functions and OSN CDLC development.

Jerry W. Stevens *IBM Software Group, P.O. Box 12195, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (sjerry@us.ibm.com).* Mr. Stevens is a Senior Software Engineer in Application and Integration Middleware working in Raleigh, North Carolina, in the Enterprise Platform Solutions Architecture Strategy and Design Group for z/OS communications servers. He has more than twenty years of software development and design experience, primarily in networking. His primary focus is associated with the architecture and design of transport layer interfaces and device drivers.

Belinda L. Thompson *IBM Software Group, P.O. Box 12195, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (belindat@us.ibm.com).* Mrs. Thompson is an Advisory Engineer in application and integration middleware software development. She received a B.S. degree in computer science from Old Dominion University. She joined the IBM z/OS Communications Server team in Research Triangle Park, North Carolina, in 1997 after a career as a UNIX** and Linux system administrator and programmer, network administrator, and application programmer. Mrs. Thompson was awarded an IBM Development Excellence Award. She is currently responsible for development of kernel device drivers for the Communications Server for Linux and Communication Controller for Linux on zSeries projects.

Joel A. Fowler *IBM Software Group, P.O. Box 12195, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (fowlerja@us.ibm.com).* In 1979, Mr. Fowler joined IBM as a customer engineer in Moline, Illinois. In 1982, he became a program support representative, moving to Raleigh, North Carolina, in order to join the Network Control Program Level 2 Support and Change Team. Mr. Fowler joined the Network Control Program development team in 1998. In 2003, he designed and developed an IBM 3745 Communication Controller emulator to run on the Windows** platform, which was to be used for NCP software problem diagnosis. The emulator was later converted to a Linux application, which is used as the base for the Communication Controller for Linux product. In 2003, Mr. Fowler joined the Communication Controller for Linux design and development team, where he continues to be a lead developer for the CCL product.

Joel Goldman *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (jgoldman@us.ibm.com).* Mr. Goldman received a B.E. degree in computer science from Stevens Institute of Technology in 1978. He joined IBM that same year in Kingston, New York. He spent 14 years at IBM Kingston before moving to Poughkeepsie, New York, where he joined the 8100 adapter development laboratory, working on hard-drive firmware and subsequently moving to a networking I/O development team. He helped to develop the OSA Support Facility and QDIO networking products for the IBM eServer. Mr. Goldman has received one patent. He is currently involved in the development and continued enhancement of the Networking QDIO functions.

Ping T. Chan *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (ping@us.ibm.com).* Mr. Chan is an Advisory Engineer working on the OSA development team. He graduated with a B.E.E.E. degree from City College of New York in 1984 and joined IBM at Poughkeepsie, New York, that same year. He has held various technical positions in the IBM eServer diagnostics and I/O areas. Mr. Chan has received several IBM Outstanding Technical Achievement Awards for his contributions in OSA development.

Thomas P. McSweeney *IBM Software Group, P.O. Box 12195, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (tommcs@us.ibm.com).* Mr. McSweeney joined IBM after receiving a B.S. degree in computer science from Louisiana State University in 1982. He worked for 13 years in Network Control Program (NCP) development and design, implementing numerous SNA networking functions in NCP, from SNA Network Interconnect (SNI) to APPN (Advanced Peer-to-Peer Networking) performance routing. He continued his SNA networking career for several more years, enhancing and supporting the SNA networking functions for the IBM 2216/2210 router family and for the 3746 NNP. After a brief time working in the area of storage networking, he rejoined the SNA networking development area. He is currently the technical team leader of the Communication Controller for Linux (CCL) development team.