zSeries features for optimized sockets-based messaging: HiperSockets and OSA-Express

by M. E. Baskey
M. Eder
D. A. Elko
B. H. Ratcliff
D. W. Schmidt

In recent years the capacity of mainframe-class servers has grown, and the quantity of data they are required to handle has grown with them. As a result, the existing S/390® I/O architecture required modifications to support an order of magnitude increase in the bandwidth. In addition, new Internet applications increased the demand for improved latency. Adapters were needed to support more users and a larger number of connections to consolidate the external network interfaces. The combination of all of the above requirements presented a unique challenge to server I/O subsystems. With the introduction of the zSeries™ comes an enhanced version of a new I/O architecture for the mainframe called queued direct I/O (QDIO). The architecture was initially exploited for Gigabit and Fast Ethernet adapters. More recently the architecture was exploited by the OSA-Express network adapter for Asynchronous Transfer Mode (ATM) and high-speed Token Ring connections, and it was exploited by HiperSockets for internal LPAR-to-LPAR connections. In each of these features, the TCP/IP stack is changed to tightly integrate the new I/O interface and to offload key TCP/IP functions to hardware facilities. For external communications, the offloaded functions are performed by the OSA-Express hardware microcode; for internal communications, the offloaded functions are performed in the zSeries Licensed Internal Code (LIC). The result is a significant improvement in both latency and bandwidth for sockets-based messaging which is transparent to the exploiting applications.

1. Introduction

The IBM G5 processor family introduced a new networking I/O adapter, the Open Systems Adapter, or OSA-Express, which provides direct connectivity between the transmission control protocol/Internet protocol (TCP/IP) stack running in the S/390* operating system and an Ethernet (network) adapter card. This connection bypasses most areas of the two key components in the path, the I/O supervisor (IOS) component of the operating system and the channel subsystem component of the system assist processor (SAP). It represents a new approach for

0018-8646/02/$5.00 © 2002 IBM

mainframe I/O architecture, providing a significant performance improvement for sockets-based applications running on the mainframe. It required a major architectural extension to the ESA/390 architecture called queued direct I/O (QDIO).

The zSeries* processor family extends the functionality of the OSA with the next generation of the OSA-Express product, including improved hardware and optimized microcode. An excellent description of the OSA-Express feature is provided in an IBM eServer zSeries 900 OSA-Express overview [1].

The zSeries also introduces a second major enhancement for sockets-based applications that run on separate logical partitions (LPARs) residing in a single zSeries mainframe called HiperSockets. Like OSA-Express, HiperSockets is based on the QDIO architectural facility, but with key enhancements which include CPU-based data movements and adapter interruptions. The main idea behind HiperSockets is to take advantage of the close proximity of logical partitions that share the same memory system in order to minimize latencies and to exploit the fast data transfer capabilities of the zSeries memory subsystem. HiperSockets is complementary to OSA-Express: HiperSockets optimizes communications among distributed applications on a single mainframe, while OSA-Express is targeted to multisystem communications. As with OSA-Express, an excellent overview of HiperSockets is available [2].

References [1, 2] provide a basic overview of the performance characteristics of the OSA-Express and HiperSockets features and give a detailed view of the externals and configuration options associated with each feature. Taken separately, each feature has inherent strengths and weaknesses; taken together, they complement each other and provide a very robust set of configuration options for the customer. These options are demonstrated in the concluding section of the paper, in which the advantage of combining these two features is shown for a typical two-tiered server configuration.

The purpose of this paper is to describe in greater detail than in [1, 2] how these two features are implemented in the zSeries and, in particular, how they are based on a common architectural framework. Their strengths and weaknesses are then discussed from this point of view.

Section 2 provides an overview of the QDIO data queues and describes the key extensions to the architecture for HiperSockets. The queue state machine, adapter interruptions, and polling bytes are included in the QDIO data queue description.

Section 3 discusses OSA-Express and focuses on the offloading of functions from the Internet protocol (IP) stack to the OSA adapter. This set of functions is defined by the IP assist architecture. These enhancements provide a significantly improved network attachment interface. OSA-Express more completely exploits the capabilities of the QDIO architecture to meet its requirements for a high-bandwidth, low-overhead attachment to remote systems. However, OSA-Express was not designed to provide optimal communication between logical partitions.

Section 4 provides a description of the HiperSockets hardware implementation, which was designed specifically for the cross-LPAR environment. Careful consideration was given to the placement of function within the various Licensed Internal Code (millicode/microcode) components in the hardware, and to the tradeoff among message latency, throughput, and effects on the memory subsystem. The development of the architecture was closely tied to the implementation issues depicted in this section. In particular, it will become clear that the objective of exploiting the close proximity of LPARs is achieved with the implementation of HiperSockets, but with some obvious limitations on connectivity and scale.

The paper concludes with an application of OSA-Express and HiperSockets to a two-tiered server configuration. This provides an excellent example of how the strengths of the two features complement each other, resulting in advantages for the customer that exceed the capabilities of each feature taken separately.

2. Queued direct I/O data queues

This section provides an overview of the basic concepts of the QDIO data queues which form the interface between the IP stack and both the OSA-Express physical adapter and HiperSockets logical adapter. These two adapters rely on the same basic architecture for the data queue formats and queue state machines, but they differ in the design of the interruption mechanism. OSA-Express relies on a combination of queue state polling and intelligent-stack-controlled program-controlled interruptions (PCIs). Program-controlled interruptions are described in [3]. The HiperSockets implementation had restrictions that made PCI interruptions difficult to support, so a new function known as *adapter interruptions* was defined. It is described below. In the following, the IP stack is referred to as the *program* and the QDIO adapter as the *adapter*.

The central component of the QDIO interface is a collection of data queue structures located in main storage. Each collection of data queues is associated with a particular adapter and consists of one QDIO input queue and four QDIO output queues. The input queue is used to receive data from the adapter, and the output queues are used to send data to the adapter. Typically, the data placed into such input queues originates from an I/O device or network of devices to which the adapter is connected. Correspondingly, when QDIO output queues

are provided, the program can transmit data directly to the adapter by placing data into the appropriate output queues. Depending on the adapter, the data placed into such output queues may be used internally by the adapter or may be transmitted to one or more I/O devices to which the adapter is connected.

For both QDIO input and output queues, main storage is used as the medium by which data is exchanged between the program and the adapter. Additionally, these queues provide the ability for the program and the adapter to communicate directly with each other in an asynchronous manner which is both predictable and efficient. This communication does not require the services of a centralized controlling mechanism, such as an operating system input/output supervisor, and the resulting overhead such a control mechanism implies. Both input and output queues are constructed in main storage by the program and are initialized and activated at the QDIO adapter as described below. Each queue consists of multiple separate data structures, called queue components, which collectively define the characteristics of the queue and provide the necessary controls to allow the exchange of data between the program and the adapter. The associated data structures, once allocated, are fixed in memory for the life of the queue. These locations are communicated to the associated adapter and allow the adapter to pre-build the direct memory access (DMA) structures necessary to extract the queue data structures from main storage.

Queue state machine and exchanging data

The IP stack and the QDIO adapter use a state-change-signaling protocol in order to facilitate the exchange of data. This protocol is applied to each of the 128 input and output data buffers[1] associated with each of the active input and output queues. This section describes the logic associated with the queue state machine.

Both input and output buffers are managed and exchanged between the program and the adapter by placing the buffer into one of the following buffer states. Additionally, each data buffer also has an ownership state which identifies either the program or the adapter as the controlling element of the buffer for the period of time that element is responsible for managing and processing the buffer. The initial ownership state of all input and output buffers is assigned to the program. Once the ownership state is changed from the program to the adapter, the program can no longer make modifications to any of the queue structures associated with the specific buffer. The same holds true for the adapter once it transfers ownership back to the program.

[1] *Buffer*: A portion of storage used to hold input or output data temporarily.

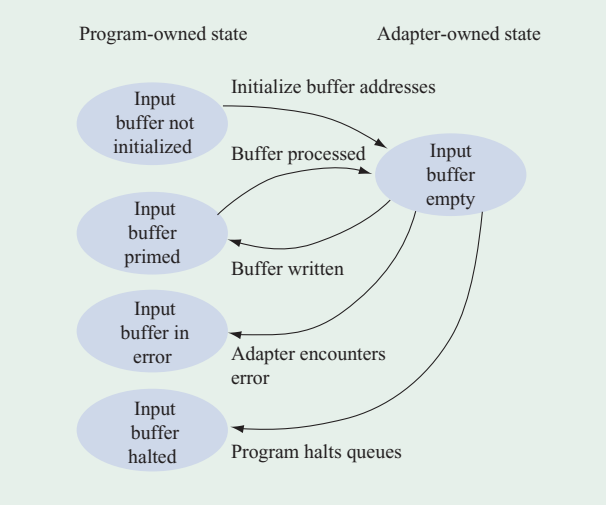

Figure 1

Input buffer state diagram.

QDIO input queue data exchange

The input queue is used by the program to receive data from the adapter or from devices controlled by the adapter. The state machine for the input queue is shown in **Figure 1**.

The program initially allocates data buffers in main storage and places their addresses in the buffer address list associated with each buffer. Each buffer address list normally contains 16 4KB buffer addresses. The program then transfers ownership of the buffers assigned to the address lists by changing the ownership state to the adapter-owned state of input buffer empty. As the QDIO adapter receives data packets from the associated I/O device, it "blocks" these packets into the main storage locations extracted from the input buffers that are in the input buffer empty state. Once all of the buffer space associated with a specific buffer has been used or a specific event occurs (i.e., the I/O device goes idle or the timer expires), the adapter transfers ownership back to the program by changing the state from input buffer empty to input buffer primed.

At this point, the adapter may also generate a program-controlled interruption (PCI) or an adapter interruption to prompt the program to interrogate the input queues.

The program examines the state of the input buffer lists associated with the QDIO input queue and processes the data in each input buffer list that is in the input buffer primed state. Upon completion of input buffer processing, the program may change the state of the buffer to input buffer empty in order to make the buffer available for reuse by the adapter for subsequent input data from the attached I/O device.

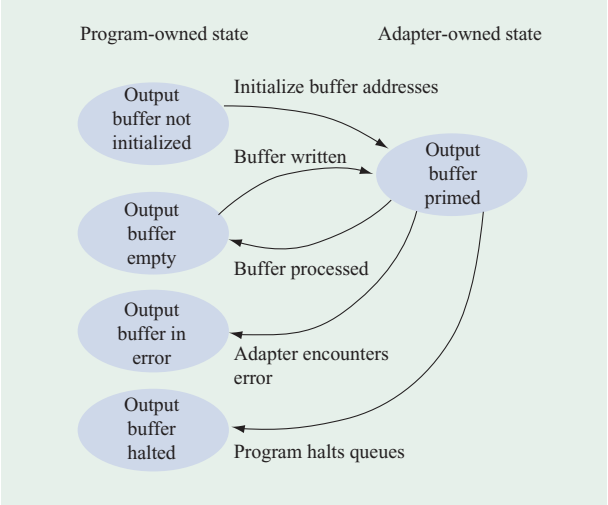

Figure 2

Output buffer state diagram.

If the adapter is receiving data from the I/O device and no input buffers are in the input buffer empty state, the adapter queues the packet. The adapter periodically interrogates the program input queue to check for available buffers. These steps are repeated as necessary for subsequent exchanges of input data.

QDIO output queues data exchange

The output queues are used by the program to send data to the adapter or to devices controlled by the adapter. The state machine for output queues is shown in **Figure 2**.

The program places output data in one or more output buffers, which are located in main storage. The placing of the data in the output queue does not require that the data be copied; instead, the address list associated with a buffer that is in either the output buffer empty state or output buffer not initialized state is updated to point to the data location. Once the address lists are updated, the associated buffer is placed in the output buffer primed state. To reduce processing, the program uses a signal adapter (SIGA) reduction algorithm to determine whether a SIGA instruction should be issued in order to signal the adapter that one or more output queues now have data to be transmitted to the I/O device attached to the adapter.

The adapter transmits to the attached I/O device the data in all of the buffers that are in the output buffer primed state. The program and the adapter each cycle sequentially through the complete set of buffer address lists in round-robin fashion. The program maintains a pointer to its current buffer being filled or to be filled with data to be transmitted, and the adapter maintains a pointer to its current buffer with data being received or to be transmitted.

After processing all of the output buffer-primed entries in the queue that were in the outbound queue at the time the SIGA instruction was issued, the adapter interrogates the program output queue again to check for any new buffers in the output buffer primed state. This action allows the program to continue adding entries to the output queue without having to execute the SIGA instruction for each entry placed in the output buffer primed state.

Upon completion of transmission, the adapter changes the state of each processed buffer to the output buffer empty state in order to make the buffer available for reuse by the program. These steps are repeated as necessary for subsequent exchanges of output data.

Special considerations for OSA-Express and HiperSockets adapters

Both OSA-Express and HiperSockets adapters use the QDIO queues for both IP unicast and IP multicast queues. (See Reference [4] for a definition of IP unicast and IP multicast queues.)

For queues associated with the OSA-Express adapter or for HiperSockets IP multicast queues, the adapter asynchronously transmits the data in each output buffer that is in the output buffer primed state, from the adapter's current buffer to the first buffer that is not in the output buffer primed state. Initiative is generally maintained for OSA-Express adapters by the adapter itself, and the SIGA instruction is needed only if errors occur that cause the associated subchannel to leave the subchannel active state. However, for HiperSockets IP multicast queues, a SIGA instruction must be issued after each output buffer or group of output buffers is placed in the output buffer primed state. This is because the processing for IP multicast queues for HiperSockets is performed by the system assist processor (SAP), which is also used for standard I/O processing and is not continuously monitoring the QDIO data queues.

For HiperSockets IP unicast queues, the QDIO adapter synchronously transmits the data in the adapter's current buffer, which is in the output buffer primed state. As with HiperSockets IP multicast queues, the initiative to the HiperSockets adapter is generated by the SIGA instruction, which must be issued for each output buffer. In this case, the adapter is logically embedded in the CPU millicode that performs the data movement as part of the instruction execution. When the SIGA instruction is completed, the entire contents of the output buffer have been moved to the target input buffers, and the data transmission is complete.

For both IP unicast and IP multicast queues, the HiperSockets data movements are performed using the

data-move hardware in the zSeries memory subsystem. This is the same hardware that is used for *move character long* or *move page* instructions executed within a single logical partition. Therefore, with HiperSockets the data-move performance that exists within a logical partition has been extended for data moves between partitions.

Virtualization

QDIO also supports virtualization, in which the virtual machine (VM) hypervisor provides translation between guest memory and the actual host memory for communicating with the adapter. Through use of the *signal adapter synchronization* (SIGA SYNC) option, the state information that is shadowed by VM is reflected back to the guest to ensure consistent operation with total synchronization between the guest and the adapter. There is some performance degradation as the number of guests increases, but the value is the ability to share the adapter across a number of guest systems. These could be Linux** guests or z/OS* guests, or a combination of the two.

Adapter interruptions and polling bytes

The original architecture for QDIO data queues provided for standard PCI interruptions to be generated by the adapter after data was placed into input queues. The PCI interruptions were generated for the QDIO subchannel that it associated with the data queues and were processed by the operating system as standard I/O interruptions. To lower the number of PCIs which had to be issued by the adapter, the input buffer processing state was added to the architecture. When an input buffer primed state was detected by the program, the state was changed to the input buffer processing state. The program then processed the inbound data. Once the program had completed processing the inbound data, the program changed the state to either input buffer not initialized or input buffer empty. The program then made one more check for any new entries in the input primed state. The adapter relied on the input processing state to determine whether the program was going to interrogate the input queues at least one more time, so that a PCI was not needed.

With the introduction of HiperSockets, a new class of I/O interruptions has been defined that is not associated with any subchannel. Instead, the interruption is associated with the collection of configured adapters of a particular type containing polling bytes that identify a particular QDIO data queue. Thus, the information previously provided by the interruption parameter is now located in a main storage location that can be periodically tested, or polled, by the operating system.

A separate device state change (DSC) indicator located in program storage identifies the specific combination of adapter and device with which the interruption is associated. The cause of the interruption and the intended program action when a DSC indicator is made active are dependent on the adapter type.

The adapter interruption mechanism is a combination of polling by the OS and interruption operations generated by the machine Licensed Internal Code (LIC). Associated with the adapter interruption facility is an adapter global summary (GS) indicator that can be tested by means of a *test vector summary* (TVS) instruction. The adapter GS indicator is set whenever a DSC indicator is set for any device. The adapter GS indicator is located in the hardware system area (HSA), with one indicator defined per logical partition.

A second summary indicator, the adapter local summary (LS) indicator, is also associated with each device. The adapter LS indicator is located in program storage and is set whenever a DSC indicator is set for an associated device. The OS may define multiple adapter LS indicators, but a device is associated with at most one adapter LS indicator.

Setting a DSC indicator initiates the adapter interruption facility to set the adapter LS indicator, to set the adapter GS indicator, and to raise an adapter interruption, in that order. However, the actual generation of an adapter interruption is conditionally based on a timing algorithm. The adapter interruption facility tracks the time interval from the time at which the adapter GS indicator was changed from the not-active state to the active state and has not yet been reset to the not-active state. If this time interval exceeds a program-specified time delay value and a DSC indicator is set for some device, an adapter interruption is generated. Otherwise, the adapter interruption is suppressed.

The collection of controls associated with adapter interruptions is depicted in **Figure 3**. A collection of input queues are shown, along with their associated DSC indicators. These are arranged in two groups, with each group having an LS indicator. The single GS indicator is shown in the hardware system area, together with the current time stamp and time delay value.

In Figure 3, the adapter has written in step 0 to the rightmost input queue and changed its queue state to input buffer primed. The adapter stores a nonzero value in the associated DSC indicator in step 1 and sets the LS indicator in step 2. In step 3 the GS indicator is set, and then the time values are interrogated in step 4. If the difference between the current time and the time stamp is less than the time delay value, no interrupt is generated. It is assumed that the operating system is being responsive in its testing of the GS byte. However, if the difference is greater than the time delay value, an adapter interruption is raised in step 5.

The adapter interruption facility is enabled by an explicit action of the operating system and remains active until an I/O system reset or image reset occurs. Once the

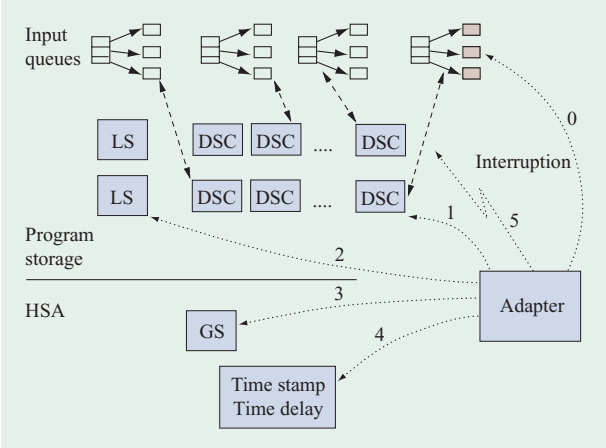

Figure 3

Adapter interruption sequence.

facility is enabled, the time delay value is set by means of the *set channel subsystem characteristics fast* command.

A significant aspect of the design of adapter interruptions is the balance between system overhead and system responsiveness. An actively running program can detect state changes in a responsive manner through polling alone, thereby avoiding the system disruption caused by an interruption. However, periods of intense workload activity may create long suspension intervals where polling is less effective and interruptions are required to achieve the desired system responsiveness to state changes. Adapter interruptions, designed with this careful balance in mind, give maximum flexibility to the QDIO device driver to determine the optimal balance and to adjust the mechanism according to current workload needs. This self-adjusting quality of the completion process is among the more innovative aspects of the HiperSockets design.

3. IP stack changes and OSA-Express

OSA-Express exploits the QDIO data queues to provide a highly optimized interface for transporting data to and from S/390 storage. The optimizations are tightly integrated into the TCP/IP protocol stack and provide the following:

1. *Improved dispatching of network tasks*. Pre-allocation of system request blocks (SRBs) coupled with use of the *perform lock operation* (PLO) instruction minimizes the scheduling and dispatch of network I/O tasks. This design allows QDIO to reduce the number of task switches necessary to accept new work and to intelligently dispatch the correct number of requests based on the number of processors and the workload.
2. *More efficient storage management*. Extensions to the communications storage manager (CSM) in support of QDIO and HiperSockets enable the elimination of extraneous data moves and the optimization of I/O storage pools. One key extension is a buffer expansion feature that allows faster reaction to increased traffic.
3. *Optimized I/O supervisor interfaces*. A dynamic interface has been introduced to vary the number of PCI interruptions on the basis of current traffic flow, resulting in a reduction of I/O interruptions without sacrificing latency. By communicating with the adapter more directly through the dynamic interface, the SAP is avoided for channel command word (CCW) translation, direct memory access (DMA) is exploited, and, in most cases, interrupts are avoided. This further improves system performance by reducing demands on the SAP and thereby freeing up additional SAP cycles for other forms of I/O processing.

IP assist architecture

The IP assist (IPA) architecture was developed to enable the dynamic sharing of a single local area network (LAN) adapter among multiple host operating systems in the S/390 LPAR environment. It provides for a programmatic way of configuring an integrated networking adapter to the owning stacks, reducing external configuration, and providing for a single point of configuration in a multiply partitioned environment. It also provides a way of configuring offloaded functions from the stack to the adapter on a function-by-function base across a family of adapters.

IP assist functions provided include the following:

- *Address resolution protocol (ARP) offload.* ARP is a TCP/IP protocol used to convert an IP address into a physical address, such as an Ethernet address. Previously resolved addresses are maintained in a cache known as the ARP cache. New ARP requests test the cache before performing the full translation. ARP offload reduces the host cycles involved in managing the ARP cache. OSA-Express responds to ARP requests received from the network, as well as issuing ARP requests on behalf of the TCP/IP stack. The OSA-Express adapter maintains the entire ARP cache in the adapter memory. Functions such as query ARP cache and purge ARP cache are available to the host TCP/IP stack for manipulating the OSA-Express ARP cache.
- *Media access control (MAC) handling.* A MAC is a hardware address that uniquely identifies each node of a local area network. Previous S/390 LAN gateway interfaces required the channel subsystem (CS) stack to build the entire LAN header prior to transmission to the

gateway. OSA-Express constructs the appropriate MAC header on behalf of the TCP/IP stack. This does not require the host to identify the LAN interface (i.e., Ethernet, Token Ring, ATM, etc.) and it can just send IP datagrams to the OSA-Express adapter.

- *IP filtering.* Prior to inclusion of this support, all network traffic would always be passed to the stack regardless of the network protocol that generated the packet (IPX, DecNet, etc.). OSA-Express now filters out all unsupported protocols. Broadcast and multicast packets are also filtered unless the stack disables these filters.
- *IP addressing.* Dynamic assignment (e.g., addition/deletion of IP addresses) provides base support for the virtual IP address (VIPA) takeover during recovery of failed applications. The stack in which the failure occurred deletes the address, and the recovery stack adds the address, all without human intervention. This enables the VIPA address of a failed application to "follow" the application to the recovery stack. This function is exploited by the virtual IP address takeover function in z/OS TCP/IP for high availability/recovery support.
- *Query/set IP assists*. This function provides an interface to the TCP/IP stack so that it can query the OSA-Express adapter to determine which TCP/IP offload assists are supported. Once the specific offload assists have been determined, the TCP/IP stack can then "set" the offload assists which are required for the specific stack. This interface allows the stack to enable new hardware features when stack support is available. It also enables the stacks to use different levels of OSA-Express hardware in the field.

Performance considerations

One of the main advantages of the QDIO architecture with respect to the existing S/390 I/O architecture is its simplification of the data transfer operation process, during both initiation and completion. Rather than constructing channel programs in which CCWs point to data buffers that consist of both TCP/IP headers and data, QDIO uses data buffers that are directly addressable by the adapter with access to the shared state machine. Use of the *signal adapter* operation replaces the *start subchannel* operation, which bypasses the system assist processor (SAP) for initiating the operation. When the operation is complete, rather than a normal I/O operation, a program-controlled interruption is provided. However, through some additional algorithms, the number of SIGA instructions and program-controlled interruptions can be reduced dramatically. This results in almost no additional overhead at high data transfer rates, with minimal SAP utilization compared with today's traditional channel-attached devices.

OSA-Express provides some advantage when the source and destination IP addresses reside in separate logical partitions on the same z900 processor. In this case, the adapter recognizes that the target IP address is local to the processor and moves the data between the two logical partitions via its local storage buffers. This avoids sending the data on the network itself, but still requires that the data move from main storage to adapter storage and back again. The data path is not as optimal as it would be if the memory subsystem were used for performing the data transfer. HiperSockets directly addresses this weakness in OSA-Express.

4. HiperSockets

HiperSockets (also referred to as internal queued direct I/O, or iQDIO) provides an intra-LPAR communication fabric for the zSeries that allows efficient and low-latency message passing among logically distributed server images. This has been realized by designing a virtual iQDIO transport layer underlying the software TCP/IP stack which enables memory-to-memory message delivery across logical partitions in the zSeries server. Key to this design point is that it does not introduce any application changes in order to gain benefit, and therefore is transparent to applications at the Sockets interface. Further, in order to be exploited by Linux for zSeries without requiring modifications to the TCP/IP stack within the Linux kernel, exploitation of the virtual iQDIO transport by Linux is being insulated to the device driver layer underneath the TCP/IP stack.

Implementation considerations

The virtual iQDIO transport layer has been realized by integrating the QDIO channel and the main control unit functions into the system microcode without requiring a network adapter. The channel and the main control unit functions are emulated by the system assist processor (SAP). In order to keep the development effort for the entire project as small as possible, HiperSockets uses the same multipath channel (MPC) and IP assist (IPA) communication architecture for the TCP/IP stack configuration as is used by OSA-Express. (See the previous section for a description of the IP assist architecture.) To the operating system, the emulated control unit appears just like another model of a regular QDIO networking control unit because the CCW interface was kept very similar. Thus, the operating system changes to support HiperSockets are kept to a minimum.

The new input queued direct (IQD) channel has been defined to integrate this virtual adapter into the system. The CCW interface for the network configuration and management (MPC and IPA) and the IP multicast function are executed by the main control unit code on the system assist processor. Hardware system area (HSA) space has been reserved for holding the IP

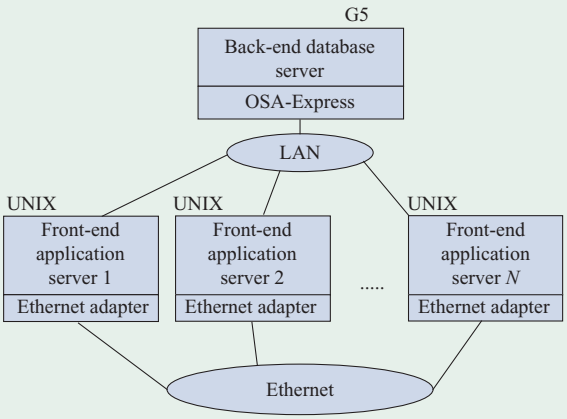

Figure 4

Two-tiered server with UNIX, G5, and OSA-2.

lookup tables that are maintained by this control unit code.

For optimization of latency, the performance-critical data transfer between operating system images for the message-passing functionality has been implemented in millicode. That code executes synchronously on the processor that initiates the network traffic using the SIGA instruction. Since the data transfer is executed synchronously on the issuing processor, the number of executed instructions must be kept small for this implementation. Therefore, the data transfer is limited to a maximum of 64 KB per SIGA instruction, and all data associated with a single SIGA instruction must be directed toward the same target operating system image. That way only IP unicast operations are executed synchronously on the CP, while IP multicast operations are asynchronously executed on the SAP. The SIGA millicode uses the next-hop IP address in the header of the first data segment to look up the target operating system using the IP lookup tables mentioned above.

For latency optimization, the interruption of the target operating system image at the end of the message-passing data transfer was changed from subchannel-based program-controlled interrupts (PCIs) to adapter interruptions. As described in an earlier section, the design of the adapter interruptions permits the operating system to suppress the interruptions if it is successfully servicing its inbound queues.

The MPC architecture requires two subchannels (control paths) that can be used to flow the MPC and IPA protocol between operating system and control unit. One subchannel is used for streaming control data from the operating system to the control unit (write path), while the other subchannel is used for the opposite direction (read path). On the read path, a long-running channel program is set up by the operating system. This way, the control unit can send the response over the read path in reply to a prior request sent by the program over the write path. Also, the control unit can send requests to the program on its own initiative, such as termination of connections. The MPC and IPA protocols are used to define lower-level data link control (DLC) interfaces which may be used by multiple TCP/IP connections. Along with the two control paths, one to eight subchannels can be defined by the MPC protocol for data connections (data paths or data devices). The data paths are used to flow the TCP/IP protocol over the QDIO transport mechanism. Thus, the data devices represent QDIO queues defined by the operating system and residing in the program storage. Each data device is dynamically assigned and represents a TCP/IP stack.

The group of two control subchannels and one to eight data subchannels is called an MPC group; thus, a single MPC group can support one to eight TCP/IP stacks.

Depending on the configuration, a maximum number of 1024 . . . 2457 TCP/IP stacks can be supported system-wide, as the maximum number of HiperSockets subchannels is limited to 3072. This was done to limit the needed HSA resources to the necessary maximum.

Four HiperSockets channel path identifiers, or CHPIDs (type "IQD"), can be defined to permit the configuration of four independent subnets. Logical partitions not sharing the same IQD CHPID cannot communicate with each other using the HiperSockets interface. This was done to fulfill a customer requirement to allow one set of logical partitions (LPARs) to communicate with one another but disallow communication with another set of LPARs resident on a different IQD CHPID. This is similar to the concept of virtual LANs. In this case, two CHPIDs must be defined, one per HiperSockets LAN, with each logical partition set defined in the associated CHPID candidate lists.

Additionally, the maximum frame size (MFS) can be defined along with an IQD CHPID. The MFS can be set to one of four values: 16 KB, 24 KB, 40 KB, or 64 KB. These values are reflected back to the operating system by the control unit code during the MPC initialization. The outbound data buffer associated with a SIGA instruction must not exceed that MFS. Inbound data buffers for all QDIO queues on the same CHPID can be optimized for that MFS, thereby saving storage capacity. This means that outbound data buffers cannot be larger than available inbound buffers if the programs obey these rules; otherwise, an error is returned to the sending program.

The IQD channel can be configured on and off by the service element. Since there is no physical adapter

associated with the virtual adapter, the IQD channel cannot be set in service mode for maintenance. Licensed Internal Code (millicode/microcode) is maintained using the already existing system features for LIC maintenance (LIC concurrent/disruptive patch).

Performance considerations

The whitepaper cited in Footnote 2 describes the performance capabilities of HiperSockets. It is evident that the design goal of exploiting the capabilities of the memory subsystem of the zSeries to gain bandwidth and reduce latency for IP data transfers between system images on a single processor has been achieved. HiperSockets provides a significant improvement over previous communication paths, as well as over OSA-Express. However, HiperSockets is limited to communications within the processor itself and can achieve the higher performance levels only when both the source and target images are executing as first-level guests. The same performance cannot be achieved for images running as VM guests.

The next section shows how the strengths of OSA-Express and HiperSockets complement each other.

5. Configuration example: Two-tiered servers

Figure 4 depicts a very common server configuration that is referred to as a two-tiered server. In this configuration, a number of application servers are running on separate UNIX** servers. In a typical example, the application servers are connected to client machines by a network using TCP/IP protocols. (However, other types of local connection protocols are also used.) The application servers are connected to a back-end database server (in this case, a G5 processor). The application servers are connected to the G5 via a LAN connection, and the G5 is connected to the LAN with a previous-generation OSA-Express adapter.

Figure 5 shows the same logical configuration as it might be deployed on a zSeries processor running LPAR. In this case, HiperSockets and the newer OSA-Express adapter are both used to achieve what is now a physical one-tiered server configuration. The application servers, running on Linux for zSeries, are running inside logical partitions. Each represents what had previously run in a standalone UNIX server. The database server is also running in a logical partition.

The application servers communicate with the database server using HiperSockets connections. This communication path is a significant improvement over that previously available with the LAN connection.

The application servers are connected to the Ethernet via the OSA-Express adapter, which now carries the combined data load that was previously supported by the collection of Ethernet adapters on the UNIX servers.

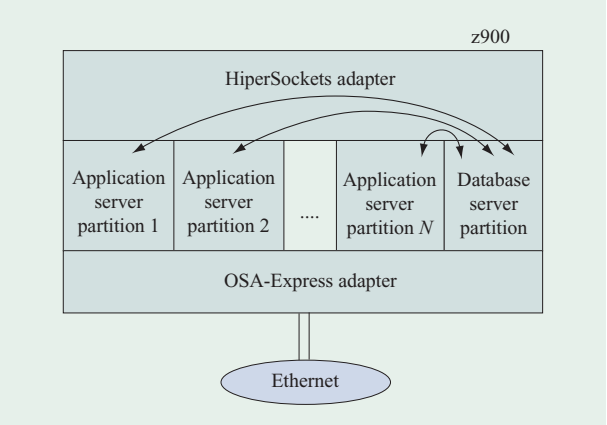

Figure 5

Logical two-tiered server with z900, HiperSockets, and OSA-Express.

This ability to collapse a physical two-tiered structure into a single one-tiered structure while maintaining the same logical two-tiered application structure provides better performance at less cost than the physical two-tiered server offered. It also shows the inherent strengths of HiperSockets and OSA-Express and their complementary nature. Additionally, they are both achieved through the exploitation of a single common architecture, QDIO.

6. Summary

This paper has described two new features for network attachments in the zSeries: HiperSockets and OSA-Express. Based on the highly unique QDIO architecture, these features continue the evolution of the mainframe capabilities for network communications. Moreover, these two features work in concert to provide significant performance improvements and cost savings for the customer. The example illustrated in the previous section shows how these new capabilities can be applied to actual customer situations. The ability to collapse a physical two-tiered server into a single processor provides tremendous savings in cost and complexity in the hardware infrastructure while preserving the customer application suite unchanged. At the same time, the ability to leverage the close proximity of the logical partitions running in the zSeries through HiperSockets significantly improves the performance of the critical message path between the application servers and the database server. Because of these capabilities, the zSeries is unique in the industry.

[2] Chris Panetta and Donna Von Dehsen, *HiperSockets Performance* (web whitepaper), 2002.

Acknowledgments

The authors wish to recognize the original QDIO architecture work of Les Wyman and Eugene Hefferon, which provided the critical foundation for the ideas presented here. We would also like to thank Arthur Stagg for his technical insights and advice. Finally, we wish to thank the referees for their comments, which helped us to improve the paper significantly.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Linus Torvalds or The Open Group.

References

1. IBM Corporation, *IBM eServer zSeries 900 OSA-Express Overview*, October 2001; see *www-1.ibm.com/servers/eserver/zseries/networking/osax.html*.
2. Bill White and August Kaltenmark, *zSeries HiperSockets* (Redpaper), December 2001; see *http://redbooks.ibm.com/redpapers/pdfs/redp0160.pdf*.
3. IBM Corporation, *z/Architecture Principles of Operation*, Order No. SA22-7832-00, December 2000; available through IBM branch offices.
4. Sidnie Feit, *TCP/IP Architecture, Protocols, and Implementation*, McGraw-Hill Book Co., Inc., New York, 1993.

Received November 21, 2001; accepted for publication April 11, 2002

Michael E. Baskey *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (mbaskey@us.ibm.com).* Mr. Baskey is a Senior Technical Staff Member in the eServer Software Design group. He is the leader of the eServer Networking team and was one of the original designers of the OSA-Express and QDIO architecture. He is also a key member of the TCP/IP Leadership Team. Mr. Baskey joined IBM in the TPF organization in 1978 after receiving a bachelor's degree in computer science/mathematics at SUNY Buffalo. He joined the OS/390 group in 1992 and was most recently the Core Technology (operating system and hardware interfaces) leader for z/OS. He has received an IBM Outstanding Technical Achievement Award for design/development of the coupling facility emulator and an IBM Outstanding Innovation Award for the design of the Gbit Ethernet/QDIO architecture. Mr. Baskey has reached the fifth plateau in patents/invention disclosures.

Marcus Eder *IBM Development Laboratory, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (meder@de.ibm.com).* Mr. Eder received his Diploma (M.S.) in aerospace engineering from the University of Stuttgart in 1993. In 1994 he joined IBM, where he worked on the design and implementation of the Multiprise internal disk subsystem, with focus on the emulation of ECKD disk control units. For HiperSockets he introduced the channel and control unit emulation technology, worked on the HiperSockets design and architecture, and leads the HiperSockets microcode team.

David A. Elko *IBM Server Group, 11501 Burnet Road, Austin, Texas 78758 (elko@us.ibm.com).* Dr. Elko is a Senior Technical Staff Member in the Advanced Systems Architecture Department. Prior to that assignment, he worked in the z/Architecture Department, with responsibility for the zSeries Parallel Sysplex architecture. Dr. Elko received a B.S. degree from Indiana University of Pennsylvania in 1976, an M.S. degree from the University of Notre Dame in 1978, and a Ph.D. degree from the University of Notre Dame in 1984. He joined IBM in 1980, working first in the MVS development group and later in the System/390 Architecture Department. He moved to IBM Austin in 1995. Dr. Elko received an IBM Corporate Award and an IBM Outstanding Innovation Award for work on the Parallel Sysplex architecture. He is an author or coauthor of 24 patents and has published two articles.

Bruce H. Ratcliff *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (brucer@us.ibm.com).* Mr. Ratcliff is a Senior Technical Staff Member in the Connectivity Solutions Department, developing networking solutions. He received a B.S. degree in computer engineering from Ohio State University in 1981, joining IBM that same year in the Field Engineering Division in Kingston, New York. In 1989, he moved to interconnect products and helped develop the first networking products for the mainframe class machines. In his ten years in networking, Mr. Ratcliff has received two IBM Outstanding Technical Achievement Awards and two IBM Outstanding Technical Innovation Awards. He has also authored or co-authored 42 patents, four technical disclosures, and two published articles.

Donald W. Schmidt *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (donws@us.ibm.com).*
Mr. Schmidt graduated from Shippensburg University, Pennsylvania, in 1982 with a B.S. degree in computer science. He has recently worked on varying I/O projects that require the optimum throughput and lowest communications delay possible. These efforts have included designing an optimized prototype TCP/IP stack for z/OS and being the lead architect for the HiperSockets project.