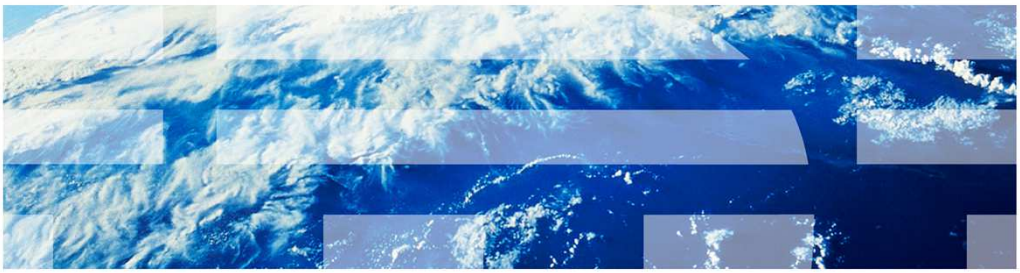
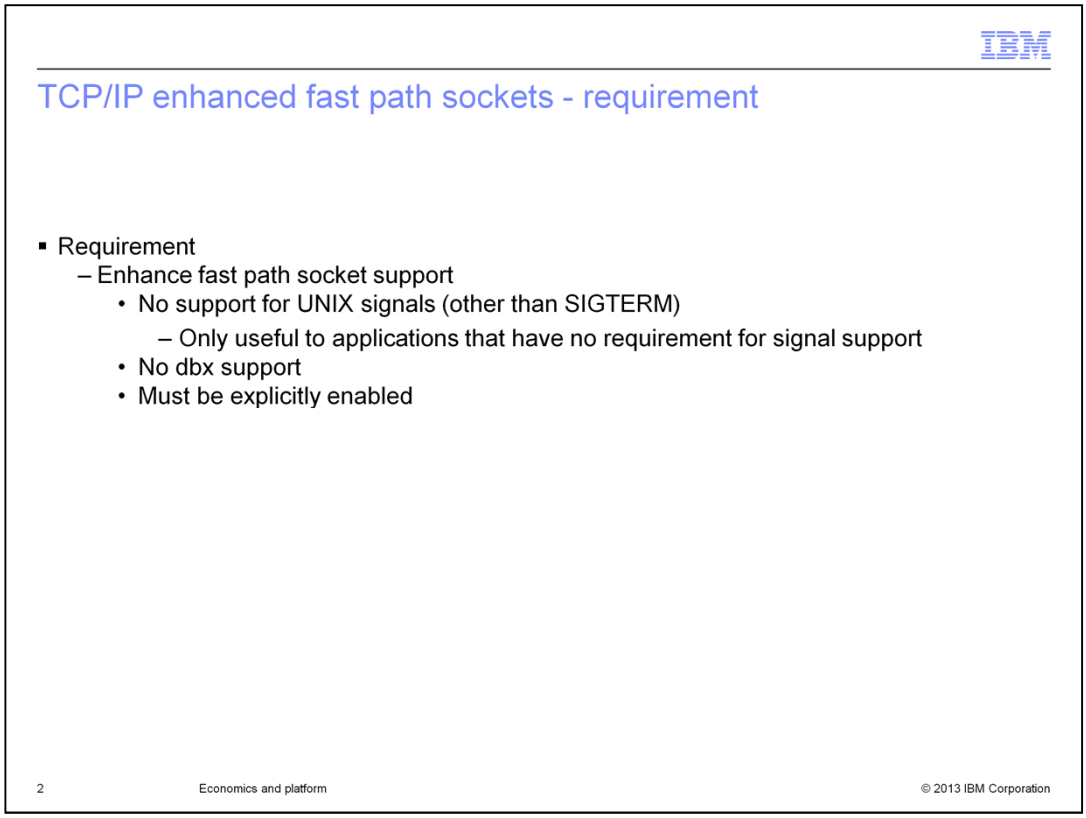
z/OS V2R1 Communications Server

Economics and platform efficiency



This presentation provides an overview of the enhancements for economics and platform efficiency in z/OS® V2R1 Communications Server.



In previous releases, the TCP/IP fast path sockets function has a reduced path length for selected socket APIs that send and receive data. This provides a significant optimization for socket send() and recv() API calls on UNIX Callable Services or Language Environment C/C++ socket APIs. Some Request-Response benchmarks have shown a 20% reduction in processor cost when comparing fast path sockets with conventional sockets.

This reduction is accomplished by two key optimizations. The first optimization is to take a much more efficient code path through the Logical File System bypass on the way to the TCP/IP Physical File System layer. The second optimization is to not invoke the UNIX System Services service to suspend and resume user threads during blocking conditions, such as waiting for data from the network. Instead, TCP/IP implemented its own logic for wait and post, which is Transmission Control Block mode, and its own logic for suspend and resume, which is Service Request Block mode. This enhancement yields the bulk of the savings, an estimated 15% out of the 20% savings just mentioned.

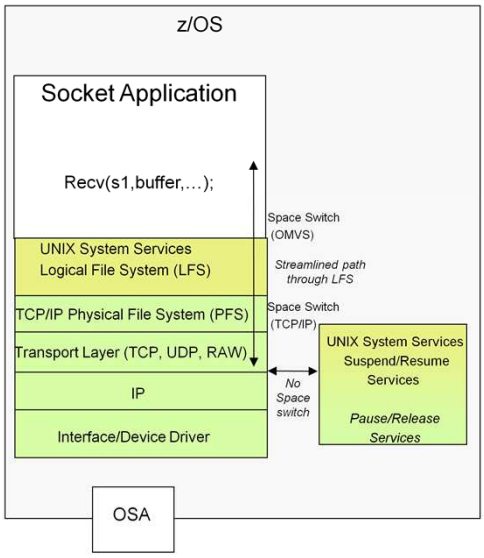
The optimization provided by the existing fast path sockets support eliminates support for POSIX signals for callers blocked in socket-related waits. This is because UNIX System Services has no means of resuming threads that have been directly suspended by TCP/IP logic. As a result, its usefulness is confined to applications that have no requirement for signal support.

Also, the fast path socket support does not allow the use of the dbx debugger for socket applications that exploit the fast path. These applications can be debugged by using the non-fast path support in z/OS UNIX and TCP/IP.

Another limitation of the fast path socket support in previous releases is that the output from some DISPLAY OMVS command options does not show the correct work unit status.

The fast path sockets support can be enabled for an entire z/OS UNIX process by using the z/OS UNIX environment variable _BPXK_INET_FASTPATH. Alternatively, the application can enable the fast path processing for a single socket by issuing Iocc#FastPath IOCTL.

TCP/IP enhanced fast path sockets – solution (1 of 2)

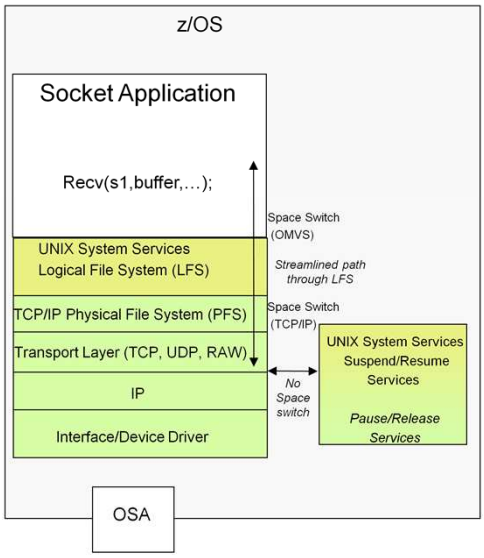


- Provide performance similar to fast path sockets for all sockets applications
  - Without requiring explicit enablement by the application or the administrator
  - With Portable Operating System Interface (POSIX) compliance, signals support and dbx support
  - Valid on most socket APIs
  - Streamlined path through UNIX System Services Logical File System (LFS) for receive and send set of APIs

3 Economics and platform

Here you see the processing that occurs for the enhanced fast path sockets support in V2R1. It does not have to be enabled and it is available to all the socket APIs, with the exception of the Pascal API. It performs better than the existing fast path sockets support.

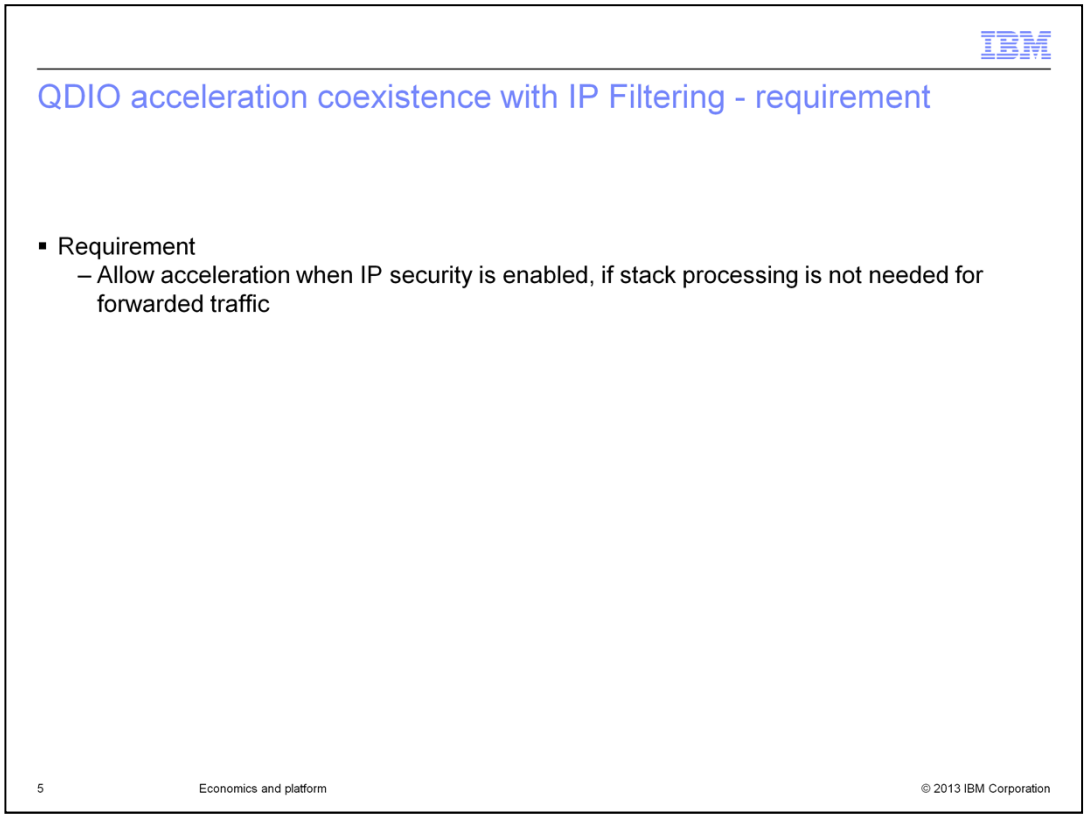
TCP/IP enhanced fast path sockets – solution (2 of 2)



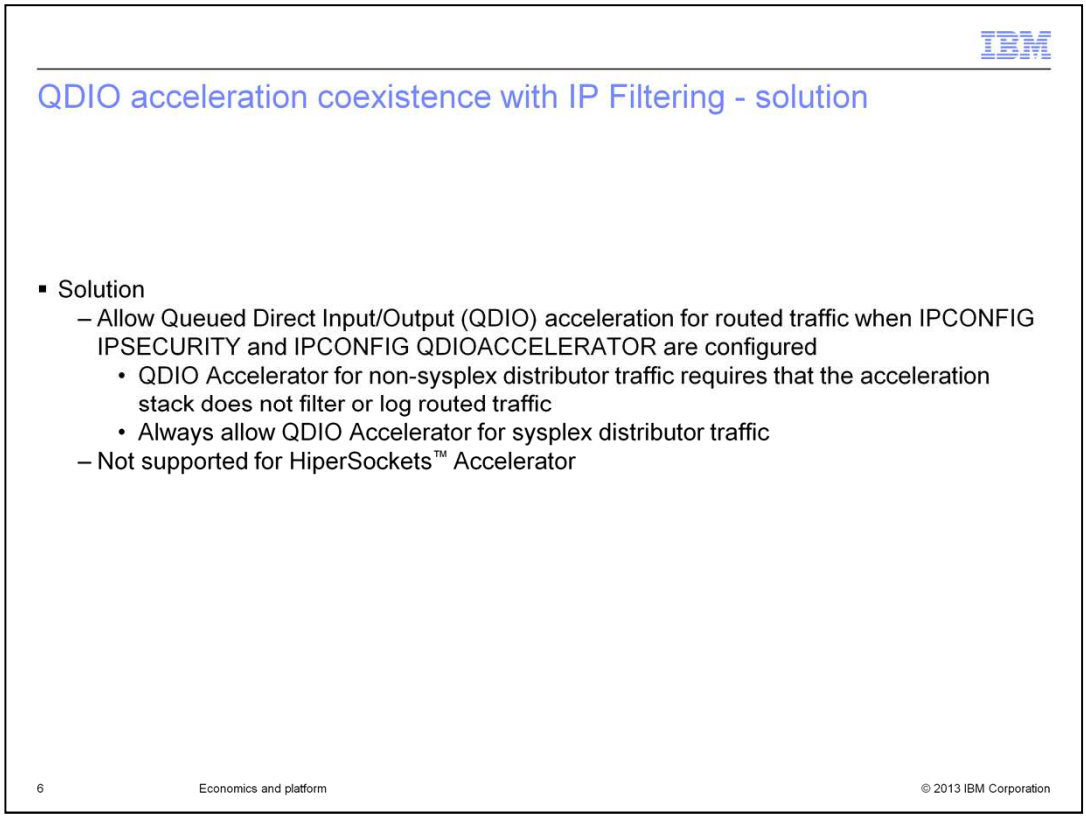
- New efficient UNIX Services for suspend/resume processing
  - Runs as an extension of TCP/IP stack
  - No need to space-switch into OMVS address space
  - Exploits MVS Pause/Release services
  - Applicable to these socket APIs:
    - send, receive, sendmsg, recvmsg, sendto and recvfrom

4 Economics and platform

The enhanced fast path sockets support exploits the new efficient UNIX Services suspend and resume processing. These services run as an extension of TCP/IP and so the space switching into OMVS address space is not done.



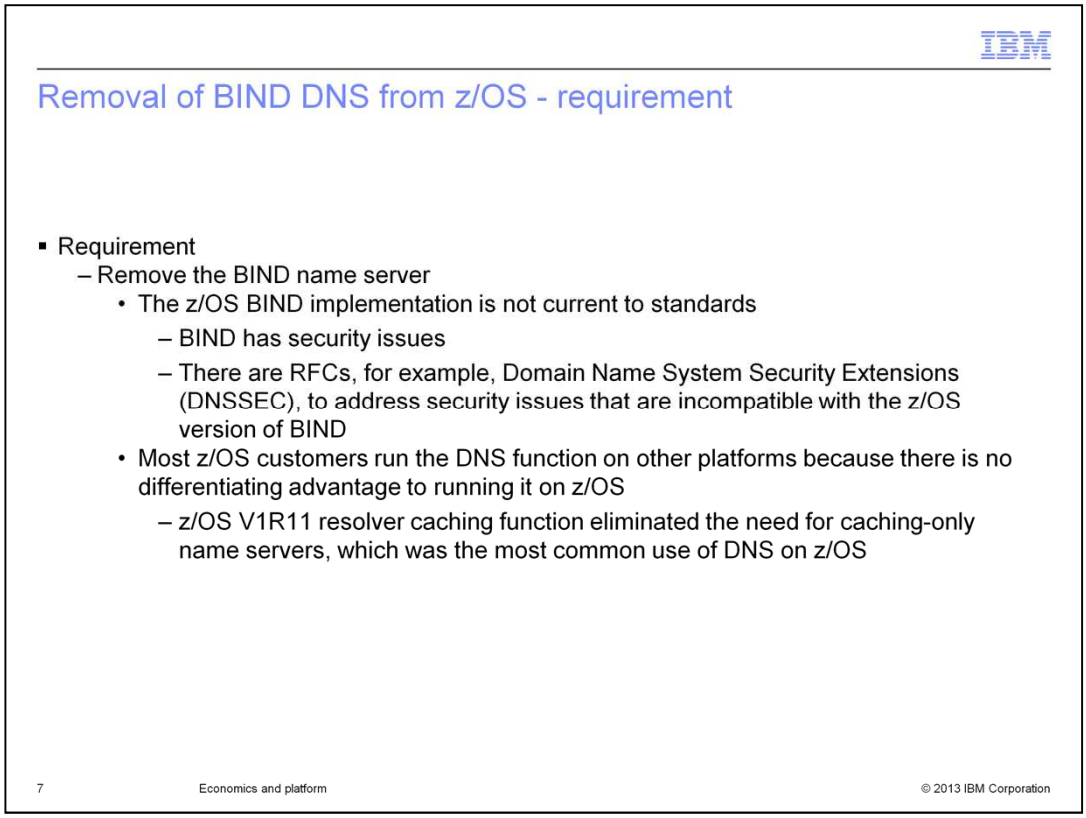
In previous releases, Queued Direct Input/Output Accelerator is disabled when Integrated IP Security is enabled, in other words, when IPCONFIG IPSECURITY is configured. The reason for this is that, in many cases, stack processing of inbound packets is required. There are, however, situations where stack processing of forwarded packets is not required – specifically when IP filtering is either to be performed at the target stack or not at all. In these cases, it would be nice if forwarded packets could realize the benefits of Queued Direct Input/Output Accelerator.



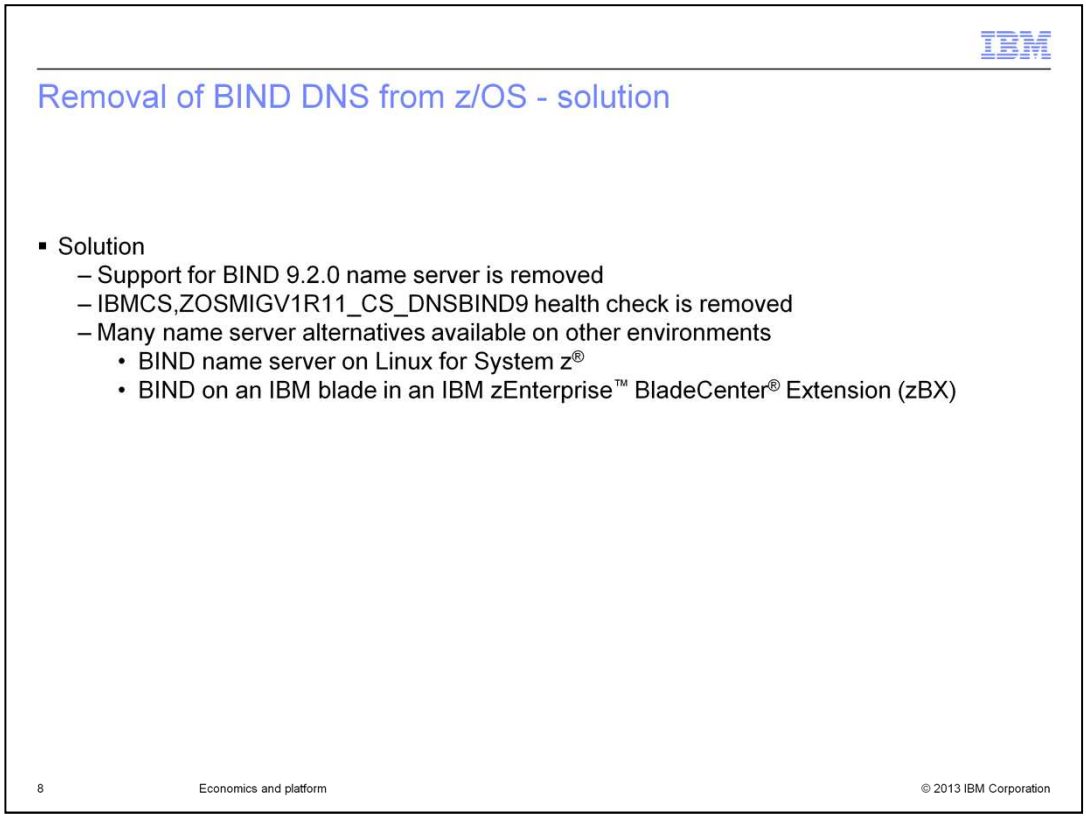
z/OS V2R1 Communications Server provides support for coexistence of the Queued Direct Input/Output Accelerator and IP filtering functions. IP packets can now be accelerated when the forwarding stack does not need to perform IP filtering or logging of routed traffic. IP filtering and logging is enabled in the TCP/IP profile, the Policy Agent configuration, or the Defense Manager Daemon defensive filters. With this new support, sysplex distributor traffic is always eligible for acceleration when Queued Direct Input/Output Accelerator and IP Security are enabled.

This function is not supported for HiperSockets accelerator.

Because sysplex distributor traffic that is forwarded by Queued Direct Input/Output Accelerator is always filtered at the target rather than the distributor, sysplex distributor acceleration will always be enabled, regardless of the current TCP/IP filter rules at the distributor.

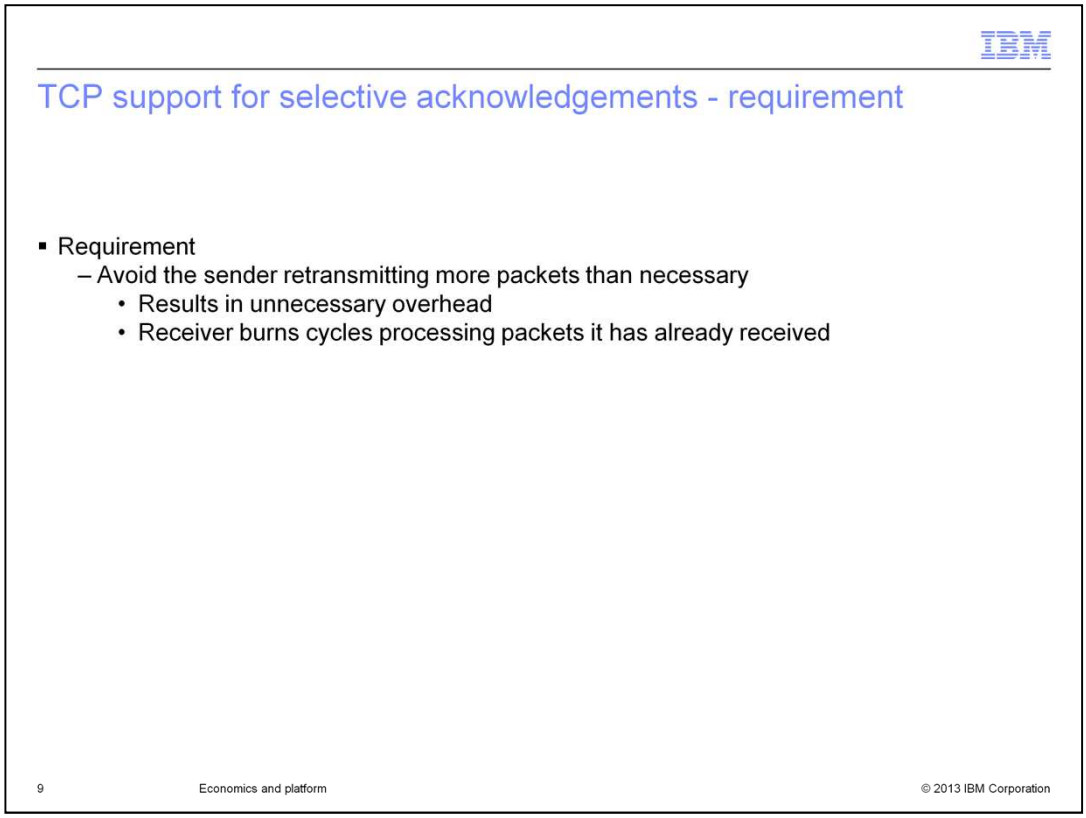


IBM no longer supports the BIND 9.2.0 name server in V2R1. This support was removed for several reasons. One reason is that there is limited use or strategic value in running the BIND 9.2.0 name server on the z/OS platform versus other platforms. Resolver caching support, which was added in z/OS V1R11, can replace a BIND 9.2.0 name server that is running in caching-only mode.



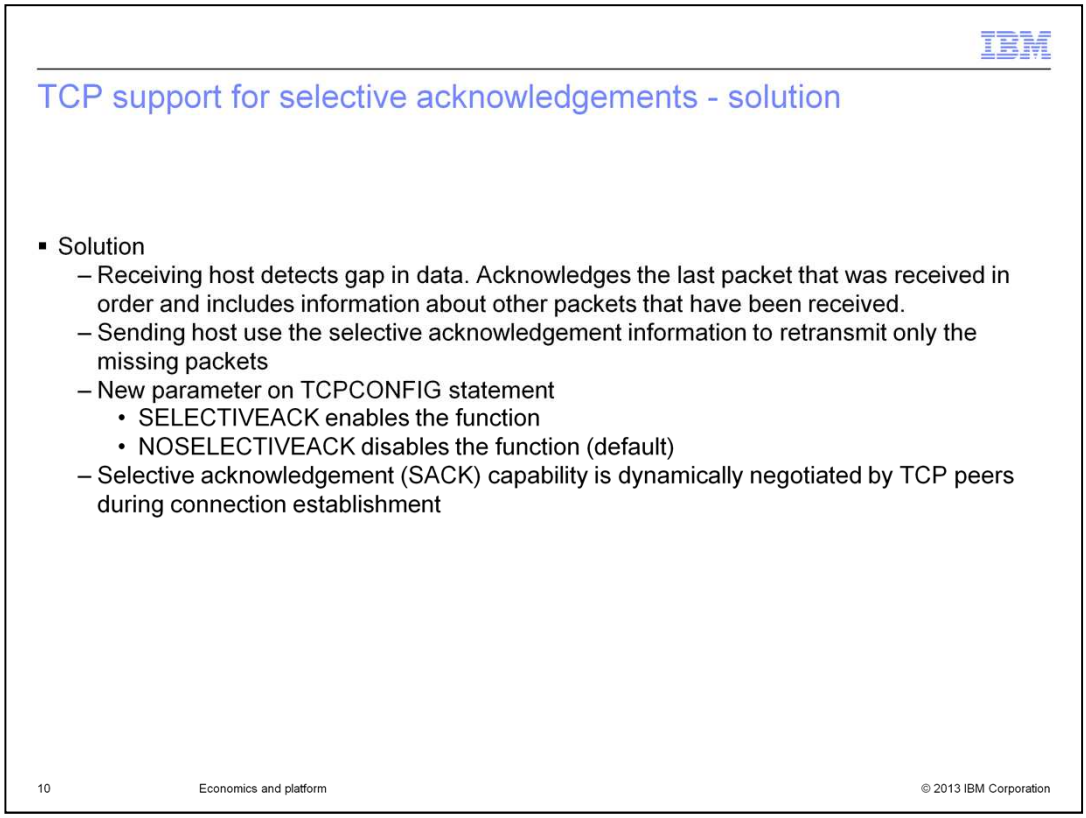
There are many alternative environments in which to run a name server. For example, a BIND name server can be run on Linux on System z or on an IBM blade in an IBM zEnterprise BladeCenter Extension.

The health check shown here has been removed from V2R1. In previous versions, it checks whether a BIND9 DNS is in use on this system.



In previous releases, the TCP retransmit and acknowledgement logic is structured such that, when an out-of-order data packet arrives, the receiver responds with a duplicate acknowledgement. This signals to the sender that out-of-order data has been received and a packet has possibly been lost. When the sender receives three duplicate acknowledgements, it assumes that data has been lost in the network and resends the missing packet and all the packets that were sent after the missing packet.

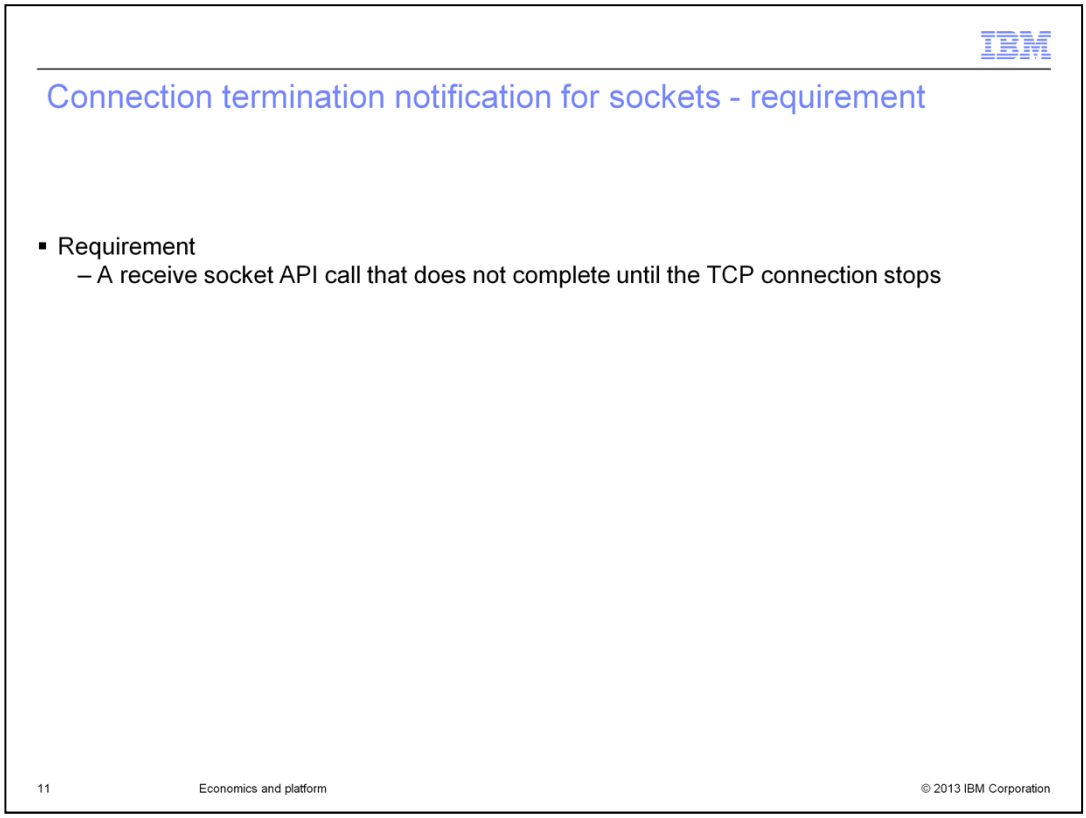
Sending all data, when only a small amount of data is missing, causes unnecessary packets to be sent. This hurts performance.



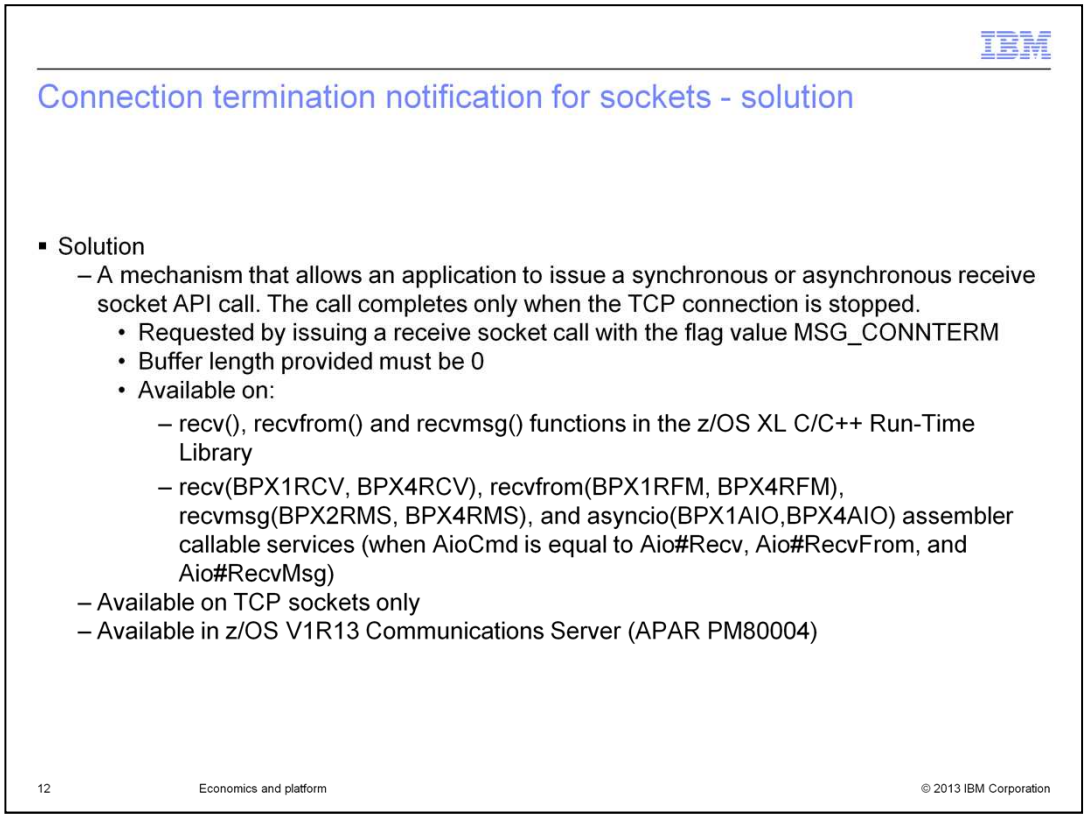
RFC 2018 TCP Selective Acknowledgements defines a solution. Two new TCP header options are supported: selective acknowledgement permit and selective acknowledgement option. Selective acknowledgement permit is used to determine if both sides support selective acknowledgements. Selective acknowledgement option is used by the data receiver to tell the sender which segments of data have been received, when out-of-order segments are received. Selective acknowledgement option is used by the data sender to determine which segments to resend. Only the segments that are missing are resent.

The RFC does not change when acknowledgement or retransmitted segments are sent. The acknowledgement number in the TCP header is not changed. The sender still waits for three duplicate acknowledgements or retransmission timeout to retransmit data.

RFC 3517 defines how a sender should use selective acknowledgement information to retransmit lost packets for fast retransmit. The sender will resend missing packets as long as there are at least three maximum segment sizes-worth of missing data beyond the segment being sent and only resend missing segments.



Some applications would find it beneficial to be able to make a receive socket API call that only completes when the TCP connection is stopped. This notification would allow the applications to cancel any back-end processing related to that connection.

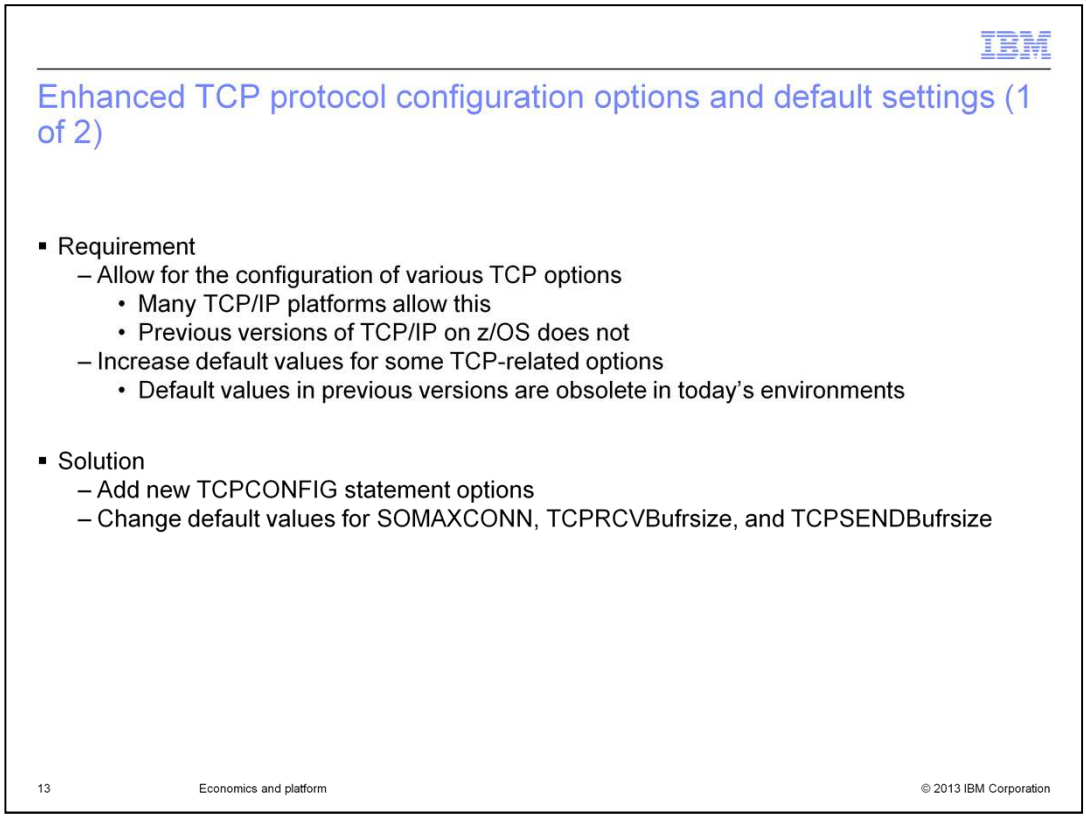


z/OS Communications Server provides a mechanism that allows an application to issue either a synchronous or asynchronous receive socket API call that completes only when a TCP connection is stopped.

A new flag MSG_CONNTERM is added on recv(), recvfrom() and recvmsg() in z/OS Language Environment C/C++ as shown here.

This new flag is mutually exclusive with other flag values such as MSG_OOB, MSG_PEEK, and MSG_WAITALL and is supported only on TCP sockets. When used, the length of the buffer provided on the call must be zero.

This function is also available for z/OS V1R13 Communications Server in an APAR.



Many TCP/IP platforms allow for the configuration of various TCP options. For example, it is possible to specify how long connections remain in the TIMEWAIT state and the amount of time a TCP packet is retransmitted before aborting the connection. It is also possible to specify the number of times a TCP packet is retransmitted, the maximum number of keepalive packet probes sent, and whether Nagle's algorithm is globally enabled.

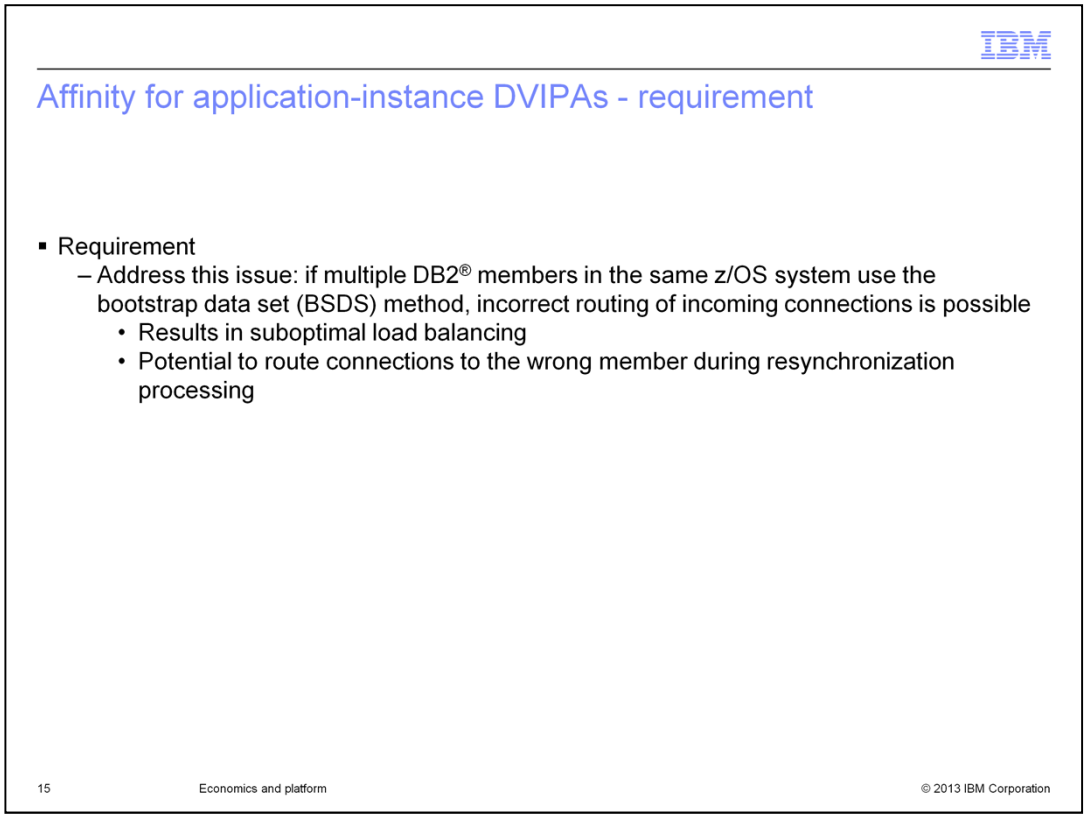
In z/OS V2R1, new TCPCONFIG statement options have been added to allow the configuration of these options. In addition, the default values for the receive and send buffer sizes were changed from 16 kilobytes to 64 kilobytes. These values can be overridden on a per-socket basis with a setsockopt() call. Most applications will achieve better throughput with larger send and receive buffer sizes.

Also, the default value for SOMAXCONN was changed from 10 to 1024. The SOMAXCONN statement sets a global limit on the TCP listen backlog. The backlog parameter on the listen() call sets the limit for the number of half-open connections that can wait in the backlog queue to reach the established state. If a connection request comes in and the backlog is full, the new connection request is typically dropped. Therefore, setting the backlog value too low can cause busy servers to drop many connections.

Enhanced TCP protocol configuration options and default settings (2 of 2)

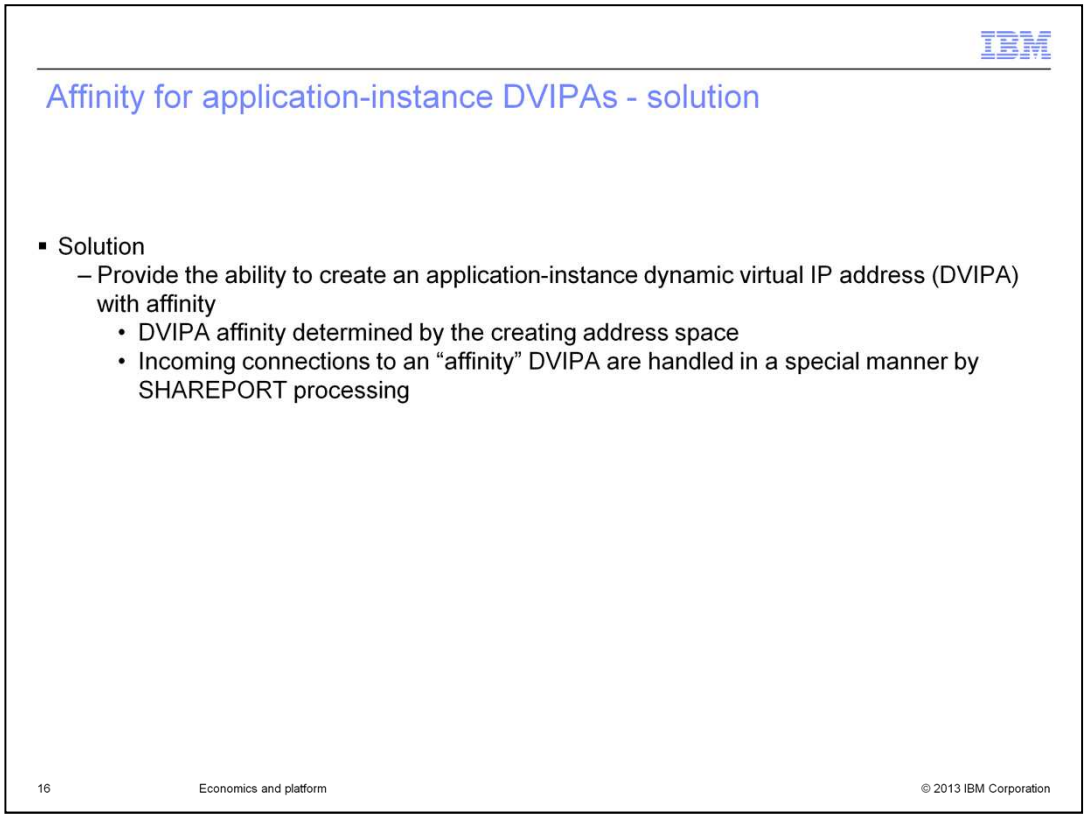
- Configure any of these new parameters on the TCPCONFIG statement
  - TIMEWAITINTERVAL
  - MAXIMUMRETRANSMITTIME
  - RETRANSMITATTEMPTS
  - CONNECTTIMEOUT
  - CONNECTINTERVAL
  - NONAGLE
  - NAGLE
  - KEEPALIVEPROBES
  - KEEPALIVEPROBEINTERVAL
  - QUEUEDRTT
  - FRRTHRESHOLD
  - TCPMAXSENDBUFRSIZE
- See the IP Configuration Reference for detailed descriptions

This slide contains the list of new parameters that can be configured on the TCPCONFIG statement. See the IP Configuration Reference for additional details about each parameter.

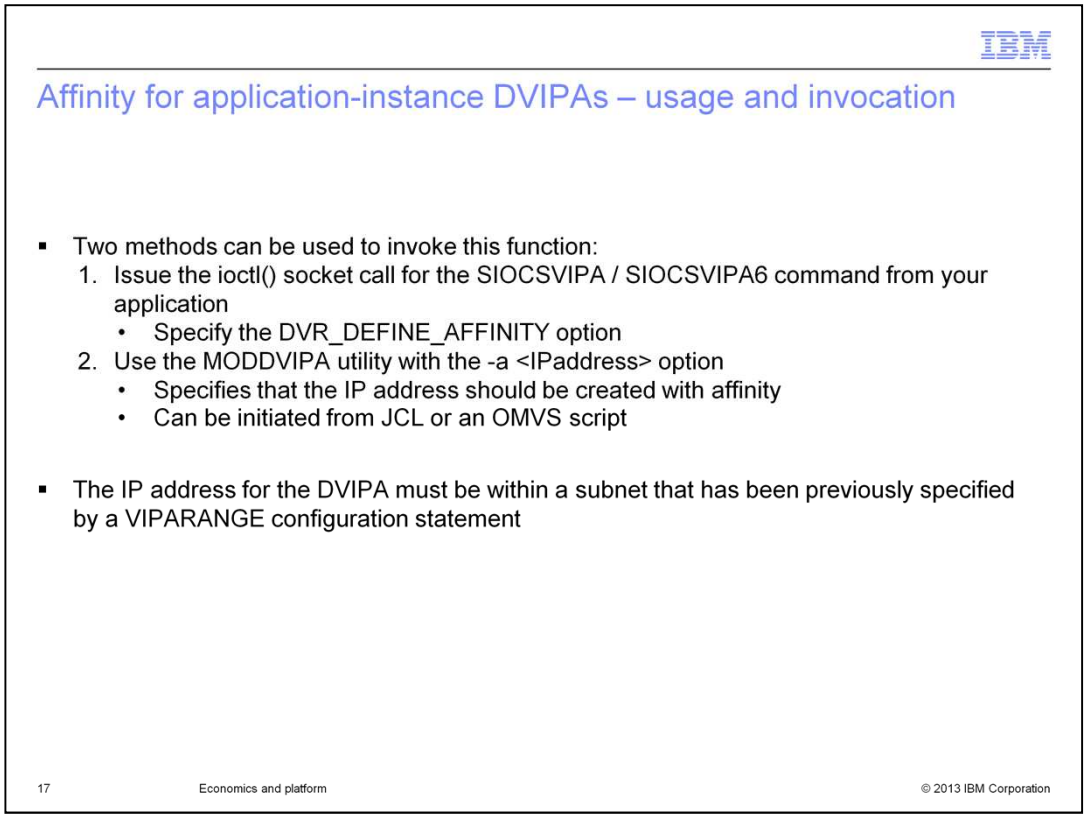


The DB2 bootstrap data set approach can result in incorrect behavior if more than one DB2 application is active on the same stack. Because each server uses inaddr_any and the distributed dynamic virtual IP address port, there is no specific association of a server to an application-instance dynamic virtual IP address.

The client sends its initial connection request to the distributed dynamic virtual IP address and port. The sysplex distributor chooses a server. The selected server responds with a list of all available servers that are application-specific dynamic virtual IP addresses and each server's Workload Manager weight available. The client chooses an application-specific dynamic virtual IP address and sends a connection request to that dynamic virtual IP address and port for its transaction. But there are two listeners on the TCP/IP stack and no way to determine which listener created that application-specific dynamic virtual IP address. The TCP/IP stack might pick the wrong listener. This results in suboptimal load balancing. Worse, if a resynchronization port is not being used, a reconnect might get routed to the wrong listener. If clients are using XA protocols for transaction coordination, the distributed port is used for resynchronization. A resynchronization port listener is not used.



The solution is to provide the ability to create a VIPARANGE dynamic virtual IP address with affinity to an address space instance that is a DB2 instance. When a connection request for an application-instance dynamic virtual IP address is received, if it was created with affinity, it is sent to the address space of the server that created it. When multiple listening sockets for the target port are available, then the listening socket owned by the address space that created the dynamic virtual IP address is found. If an address space with affinity is not found with a listening socket on the target port, then existing shareport load balancing is used to route the connection to a listening socket that can accept it. That is bound to INADDR_ANY. This allows the dynamic virtual IP address to be used by non-DB2 applications, such as incoming FTP connections to the member-specific dynamic virtual IP address. This works only when the dynamic virtual IP address is still active.

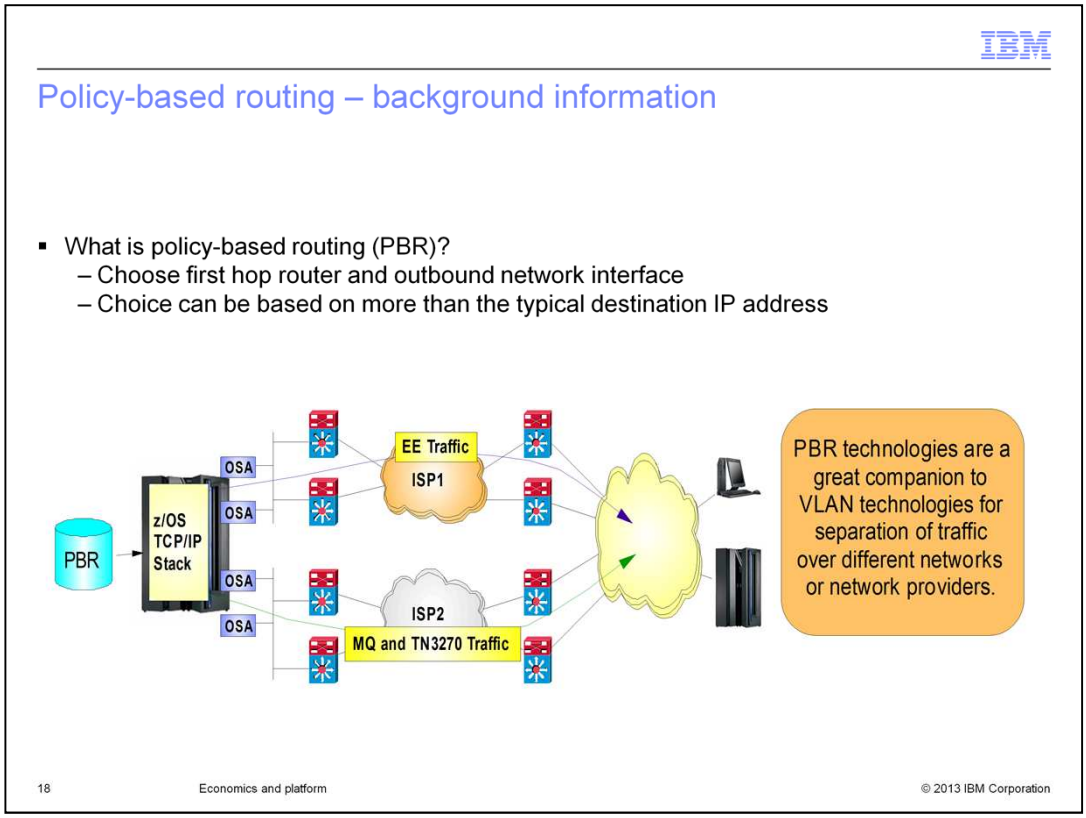


An ioctl command, either SIOCSVIPA or SIOCSVIPA6, allows an application to create or delete a dynamic virtual IP address on the stack where the application is running. The application can issue the IOCTL call by using the XL C/C++ run-time or by using the UNIX System Services Callable Services BPX1IOC API. A dynamic virtual IP address can be created by using either the DVR_DEFINE option or the DVR_DEFINE_AFFINITY option of the SIOCSVIPA or SIOCSVIPA6 ioctl. The DVR_DEFINE_AFFINITY option creates the dynamic virtual IP address with affinity to the address space that created it.

You can use the MODDVIPA utility to activate or delete a dynamic virtual IP address. This utility can be initiated from JCL or an OMVS script.

For more information about the use of the SIOCSVIPA or SIOCSVIPA6 ioctl() or the MODDVIPA utility, see the Configuring the unique application-instance scenario section in the IP Configuration Guide.

When creating a new dynamic virtual IP address, the requested IP address must be within a subnet that has been previously specified by a VIPARANGE configuration statement in the PROFILE.TCPIP data set for the TCP/IP stack.

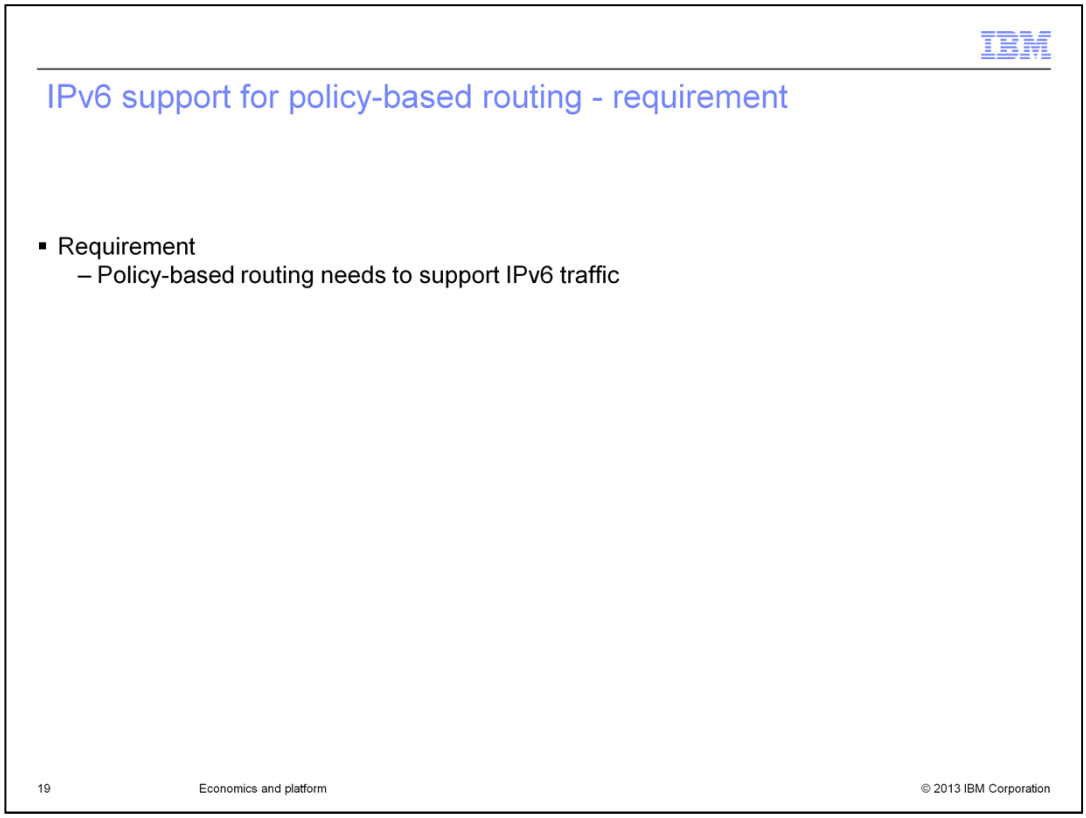


Policy-based routing allows IP routing to use additional route selectors. It is made possible through the use of multiple route tables. In addition to the main route table, the TCP/IP stack can have multiple policy-based route tables. Policy-based route tables have many of the same characteristics as the main route table. They can contain both static and dynamic routes and the static routes can be configured as both replaceable and non-replaceable.

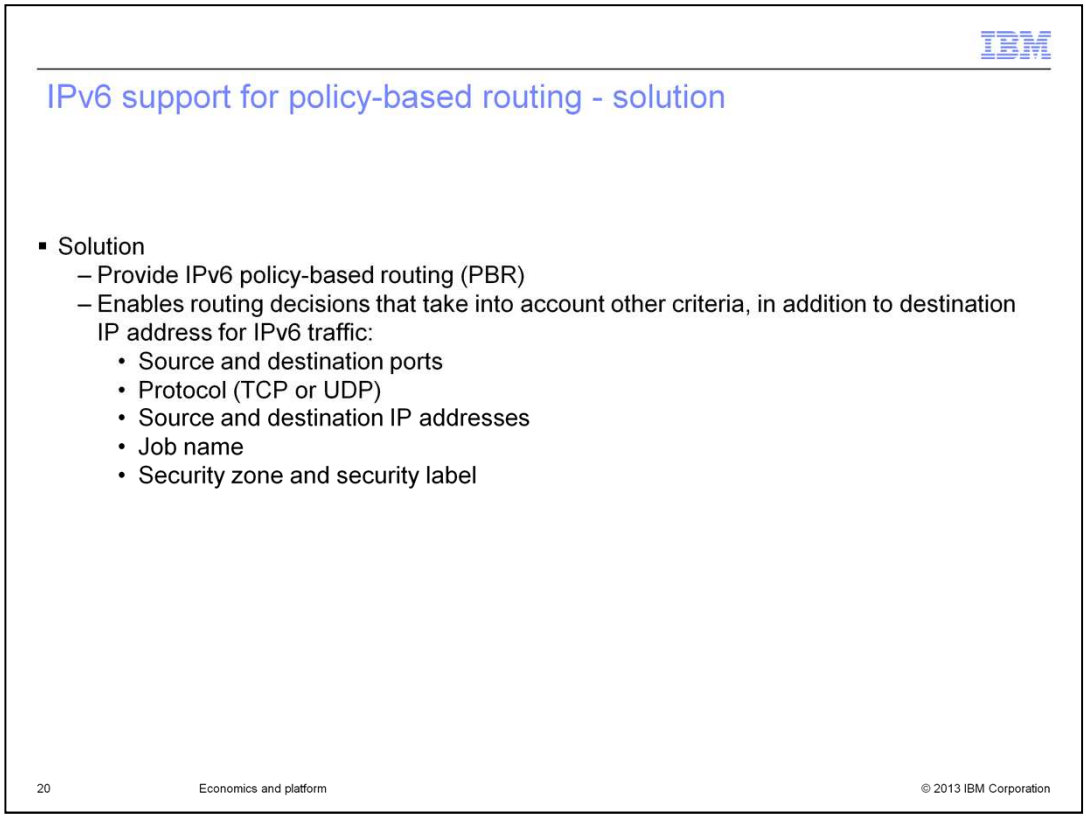
Policy-based routing allows an installation to separate outbound traffic for specific applications to specific network interfaces and first-hop routers. The separation can be based on security, choice of network provider, or isolation of certain applications.

Most often there is one policy-based route table defined to be used for the traffic; but there can be as many as eight. Each of the policy-based route tables is searched, in the order defined, for a route to the destination. If any active route to the destination is found in a route table, the search is stopped and that route is used for the traffic. This route might be a host route, a subnet, network, or supernet route, or a default route. If no active route to the destination is found in a route table, the search continues in the next route table. If all policy-based route tables are searched without success, the main route table might also be searched, if the policy indicates that the main route table can be used as a backup.

Policy-based routing is not supported for all types of IP traffic. The support is limited to locally-originated TCP and UDP traffic. All forwarded traffic and all traffic using protocols other than TCP and UDP is not processed by policy-based routing and continues to be routed by using only the main routing table.



The policy-based routing support that was added in V1R9 provides support for IPv4 only. Similar support for IPv6 is needed.



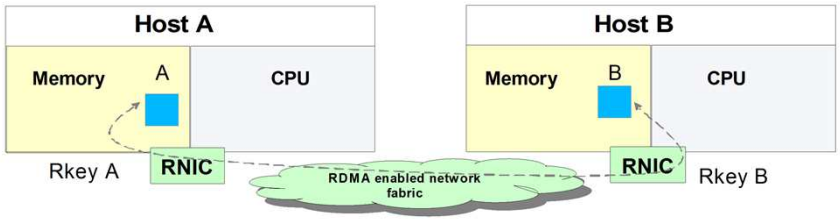
IPv6 policy-based routing enables the TCP/IP stack to make IPv6 routing decisions that take into account criteria other than just the destination IP address. The additional criteria can include job name, source port, destination port, protocol type, source IP address, NetAccess security zone, and security label. With policy-based routing, you can define a policy that selects the network to be used for outbound traffic, based on the application that originated the traffic.

Here are a couple of scenarios in which policy-based routing might be useful. You might want to favor high-bandwidth links for batch IPv6 traffic, while preferring low-latency links for interactive IPv6 traffic. If so, you can define a policy such that Telnet traffic can be routed over the low-latency links, while FTP traffic can be routed over the high-bandwidth links. As another example, you might want to define a policy to ensure that IPv6 traffic that is tagged with a security label or zone is routed to a secured network through an appropriate outbound interface.

IPv6 policy-based routing applies only to TCP and UDP traffic that originates at the TCP/IP stack. IPv6 traffic that is using protocols other than TCP and UDP and all IPv6 traffic being forwarded by the TCP/IP stack will always be routed using the main route table. This happens even when IPv6 policy-based routing is in use.

Remote Direct Memory Access (RDMA) - background information

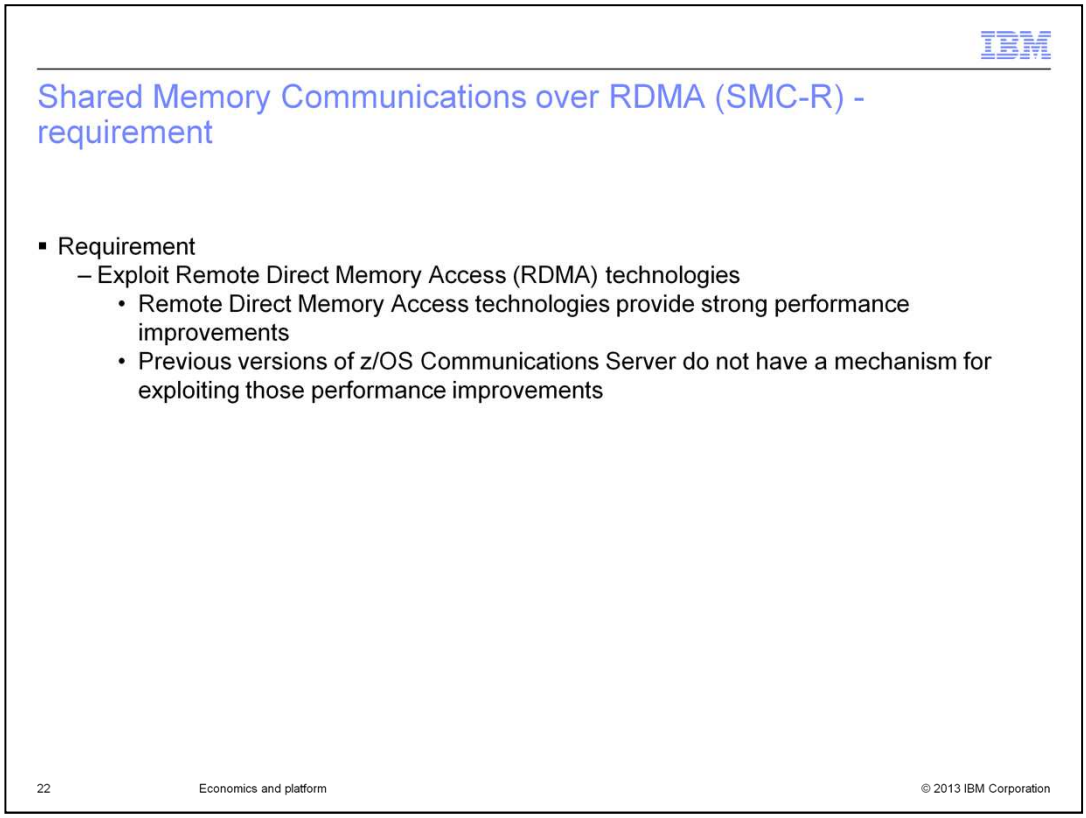
- Enables a host to read directly from or write directly to a remote peer's memory
  - Peer's processor and operating system not involved in transfer
  - Host registers specific memory for RDMA partner's use
  - Interrupts are still required for notification
- RDMA reduces networking stack overhead by using streamlined, low-level interfaces



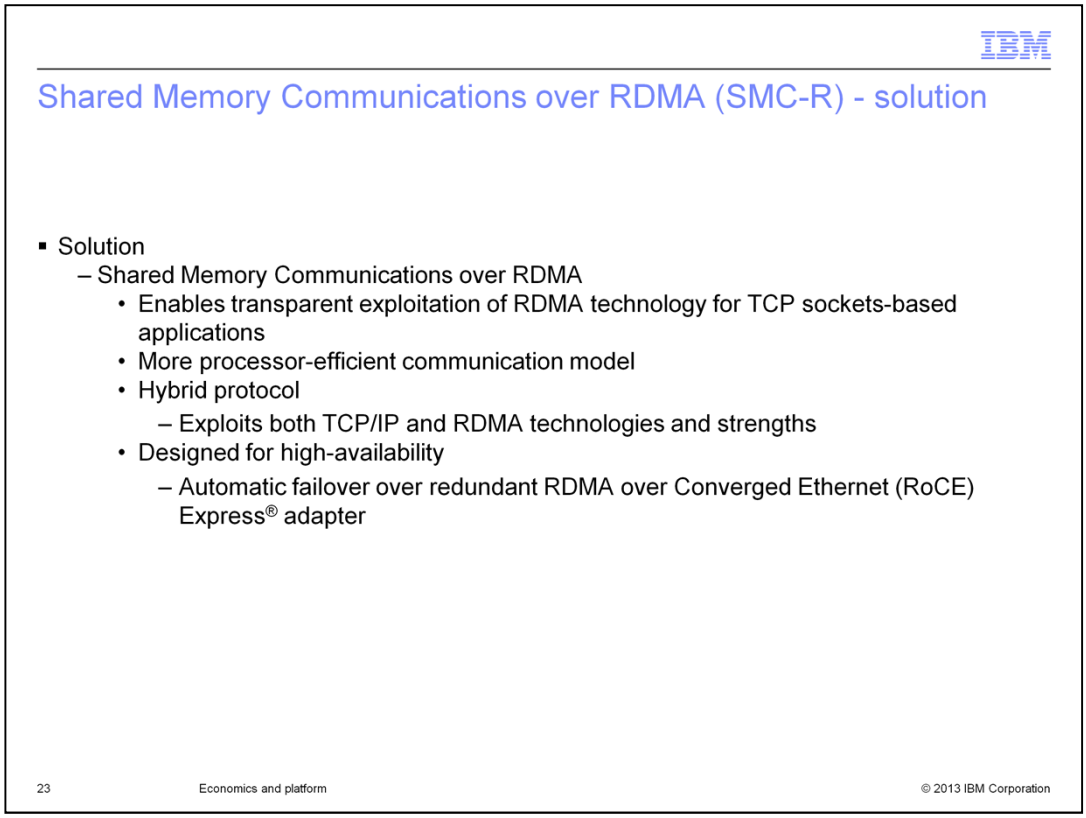
21 Economics and platform

Remote Direct Memory Access allows a host to access memory on a remote peer that is connected to the same Remote Direct Memory Access-capable Ethernet fabric. Remote Direct Memory Access protocols allow a host to read from or write into memory that the peer has allocated specifically for use by that host.

Although interrupts are still required to alert the host that data has been received, significant performance improvements can be achieved by using RDMA. Some performance gains are achieved because the processor and operating system at the receiving host do not participate in the RDMA transfer. Other gains are achieved because the TCP stacks can use simpler processing to send data by using Remote Direct Memory Access.



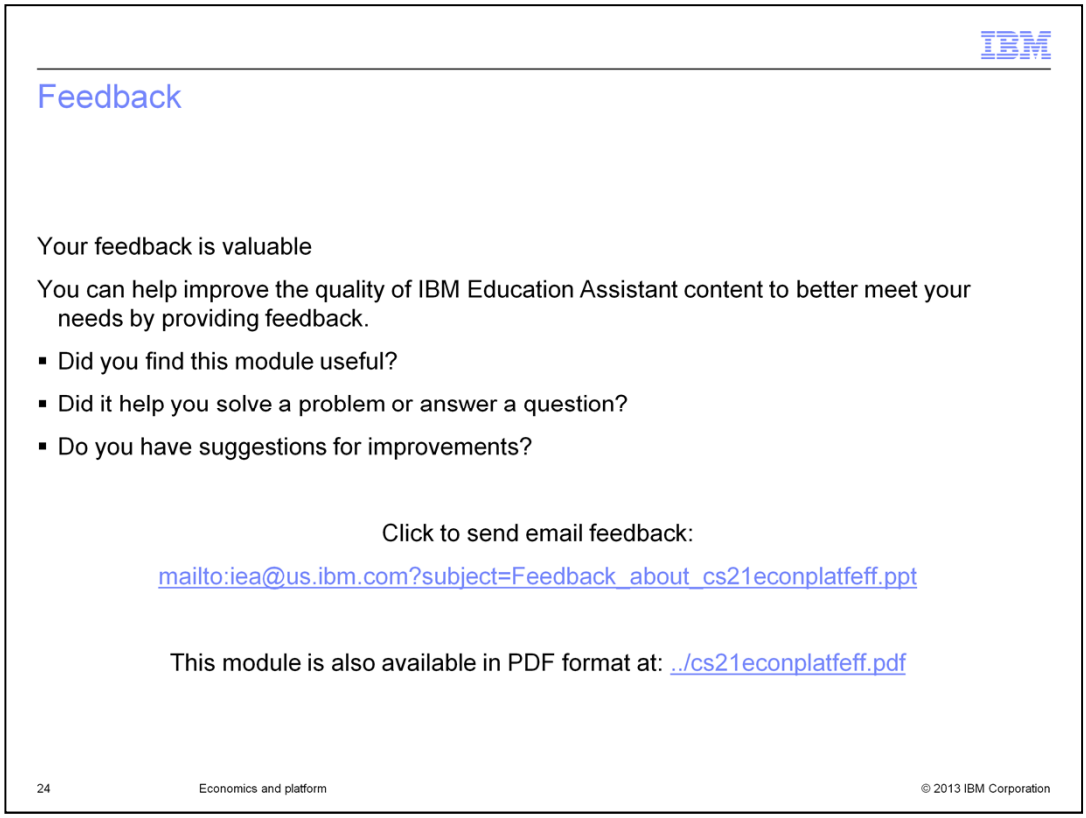
In versions before V2R1, z/OS Communications Server cannot exploit Remote Direct Memory Access because the necessary RDMA over Converged Ethernet hardware is not available on the platform.



z/OS V2R1 Communications Server exploits RDMA technology by using Shared Memory Communications over RDMA protocols. These protocols provide a transparent socket-based exploitation model for RDMA, allowing existing TCP applications to benefit without change.

The decision to use or not use Shared Memory Communications over RDMA for a given TCP connection is made during TCP connection establishment. The protocols for making that determination are called rendezvous processing. If rendezvous processing successfully selects or creates a Shared Memory Communications over RDMA link, then subsequent application socket data is exchanged “out-of-band” by using this link. After the choice is made to use Shared Memory Communications over RDMA protocols, the TCP connection cannot revert to using traditional IP protocols, even if the link or RDMA over Converged Ethernet Express interface encounters errors later. z/OS Communications Server prevents any application socket data from being exchanged until the choice of Shared Memory Communications over RDMA or IP protocols has been made. The TCP connection stays active for control flow and connection termination processing, but otherwise remains idle.

Significant performance savings are achieved by switching to “out-of-band” RDMA protocols. One advantage is that the TCP/IP stack does not have to break the application socket data into smaller packets to be transported across the IP fabric. Instead, the data is moved as larger chunks of data, up to the size of the remote memory buffer that was made available by the peer. z/OS Communications Server selects a buffer size for the peer, based on the receive buffer size specified by the local application. The RDMA over Converged Ethernet Express adapter is designed to guarantee delivery of the RDMA data in order to the peer. This means that traditional TCP-layer processing for retransmitting lost packets is not necessary with Shared Memory Communications over RDMA. It also means that the TCP layer does not have to exchange acknowledgements to verify that the data has been received properly. This greatly streamlines the TCP-layer processing for Shared Memory Communications over RDMA, which provides additional performance gains. Additional gains are achieved because the entire IP layer is bypassed in favor of a new, more streamlined Shared Memory Communications over RDMA processing layer.



You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, BladeCenter, DB2, Express, HiperSockets, System z, z/OS, and zEnterprise are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.