May 2010

Validation of OSPF on IBM Linux on System z at Scale

E. M. Dow, S. Loveland, G. Markos,
{emdow, d10swl1, markos}@us.ibm.com
IBM Platform Evaluation Test Laboratory
Poughkeepsie, NY

Table of Contents

Introduction and Motivation

This paper documents observations about a moderate scale out of the Open Shortest Path First (OSPF) dynamic routing environment architecture presented in "A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF" [Arch]. The small scale investigation in the aforementioned document served only to validate the architecture. This document is presented as an experience report of an attempt at moderate scaling, and is not intended to be a rigorous technical performance characterization. CPU consumption was the parameter measured. We did not examine the effects on system memory utilization. Although our scaling endeavors do not achieve the theoretical maximum, we feel we have selected a practical size suitable for widespread result applicability [Scaling].

The motivation behind this scale up effort is based on perceived OSPF overhead. OSPF routers on the same broadcast domains form adjacencies with one another, in order to build their topology mapping. When an adjacency is formed, the neighboring OSPF routers exchange Link State Advertisements (LSAs). LSAs are used to create and maintain a view of the network topology, as well as calculate the shortest path for every route in each respective OSPF area that it belongs to. Any changes to the network topology (due to link-state changes) are propagated throughout the OSPF area. Therefore, every time that a router receives a link state update, the router will need to re-compute both its view of the network, and update its list of shortest paths to each destination in the network. Depending on the size of the network, as well as the respective link stability within those networks, OSPF can theoretically become both processor and memory intensive due to the required re-convergence calculations.

The following sections of this paper include a discussion of our environment configuration, test scenarios performed with their associated empirical observations, and the conclusion that OSPF at this scale is a favorable availability option in our environment.

Environment Configuration

Each of the Linux instances used in this investigation was running on an IBM System z9® mainframe. The logical partition (LPAR) supporting this investigation was running z/VM® release 5.3 and has been configured with 8 CPU's.

For this investigation, we created 66 SUSE Linux Enterprise Server (SLES) version 11 Linux® guests, each with a single processor, and 512 MB of memory [Opsys][Cpu][Mem]. Each guest was created as a default installation [Disks]. We disabled the same services that we typically do in our test environment, including the CUPS print daemon and the Sendmail MTA service [Procs].

Each guest was given:

- One IBM VSWITCH connection

- One HiperSockets™ connection
- One highly available Static Virtual IP address (VIPA) bound to a dummy0 interface provided by the dummy kernel module [Nics].

We configured our Zebra settings to convey information for each of these devices [Zebra]. Note that though each guest is running OSPF, they are not configured to forward packets on behalf of one another. Rather, each Linux guest utilized its respective OSPF routing daemon to receive routing information to be used only by the local system. For example, each Linux instance has a routing entry to reach each of its peers via their respective static VIPA, as well as have dynamically received a default gateway entry pointing to one of the two ABRs [Routes]. Martian packets are discarded as usual.

A special OSPF router called a Designated Router (DR) performs most of the OSPF protocol activities on each shared multi-access medium, such as synchronizing database information and informing members of the broadcast network about changes to the network. Because a HiperSockets network is also a broadcast network, and because only z/OS®, z/VM, and Linux for IBM System z® nodes can participate in a HiperSockets network, one IBM System z node must be elected as the OSPF DR and optionally, another one as the Backup DR (BDR) on a HiperSockets LAN. To accommodate the required DR and optional BDR election, a unique non-zero router priority was explicitly set for the HiperSockets interface on three of the 66 Linux guests.

For the VSWITCH network, we configured two OEM hardware routers as Area Border Routers (ABRs) to serve as the DR and BDR. This was accomplished by setting a non-zero OSPF Router Priority value on each of the ABRs for their respective interfaces connecting to the VSWITCH, while setting a router priority value of zero on each of the IBM System z servers for those same multi-access network connections. Altering the priority involved editing the ospf.conf file such that the line “ip ospf priority 0” was changed to “ip ospf priority X” where X is the non-zero, non-negative router priority value. The priority value indicates the order in which routers are elected. Selecting a priority value of '0' indicates a that the host machine cannot become a DR or BDR. The highest number which can be assigned is 255. If there is a priority tie, the router IDs are used to break the tie. Higher numbers will be elected designated router before routers with lower priority values.

We created a very simple OSPF topology, which behaved very much like a Totally Stubby area would, as is illustrated in Figure 1, Networking Topology. Specifically, our OSPF network topology consisted of a single fully meshed Stub Area with only one pair of ABRs straddling the only OSPF area in that topology. Thus, the ABRs never had an opportunity to receive, nor had the need to send, any inter-area summary LSAs (type 3 and type 4) into our Stub area. Therefore, the lack of inter-area summaries being presented in our topology made our Stub Area behave very much like a Totally Stubby area.

Each Linux instance, along with the two OEM hardware routers, used the default values for the OSPF Hello and Dead timer values: 10 and 40 seconds, respectively. Sample Zebra and OSPFD configuration files for the Linux instances are shown in the Appendix of this document [Zebra][Ospf].

All CPU consumption observations mentioned in this document were taken from our installation of the z/VM performance toolkit, occasionally supplemented with observations from the Linux "top" command.

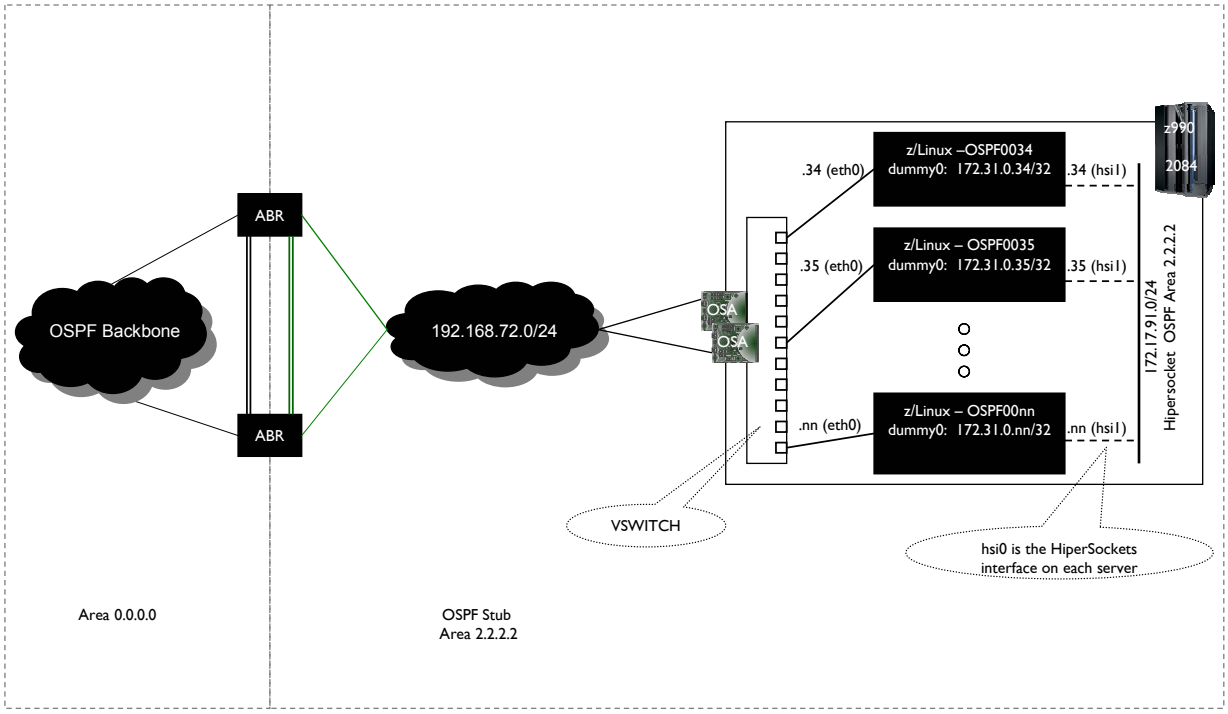

Figure 1: Networking Topology

Scenarios Executed

We began our experimental observation by measuring the CPU consumption of each of our newly provisioned guests with both the OSPF and the Zebra processes stopped. This measurement is considered our base line performance measurement. We then started the Zebra and OSPF processes and measured individual guest consumption again to arrive at a per machine delta. As expected, there was a small increase in per machine CPU consumption.

We observed favorable results over this portion of the study where OSPF neighbor adjacencies were neither formed nor lost. In other words, in periods of normal operation where all established OSPF adjacencies were being maintained by each of the 66 Linux guests, such that no additional systems were being added or removed from the environment, the measured CPU overhead was negligible in our environment. This measurement was intended to compare systems running a quiesced OSPF to the identical systems when OSPF had been started. As a point of comparison, logging into each guest via SSH causes spikes that overshadowed the overhead of OSPF running as an additional "mostly idle" service. We say mostly idle here because each of the OSPF systems were periodically exchanging OSPF Hello packets every 10 seconds.

We next executed a number of scenarios to examine the effects associated with not only losing, but with also establishing OSPF neighbor adjacencies in order to characterize the CPU overhead of each of these operations. These scenarios were split into two major groups:

- Service scenarios
- Failure and recover scenario*s*

The first group of scenarios, Service scenarios were designed to emulate performing routine service and maintenance to the Linux instances, in order to understand the OSPF overhead associated with both losing and establishing an OSPF neighbor adjacency. Scenarios executed successfully include:

1) Halting a single Linux instance via the "shutdown -h" command, and subsequently restarting that same guest after a period of time had elapsed. This scenario simulates applying maintenance on the directory entry, or some other aspect of system maintenance that requires an individual guest to be shut down cleanly. The details and expected results of these scenarios are as follows:
   A) The first half of this scenario was designed to quantify the OSPF overhead associated with losing an OSPF adjacency on a cumulative level (across 65 of the 66 Linux guests on one Central Processor Complex (CPC)). Specifically, during the graceful shutdown procedure, the OSPF process continues to transmit OSPF Hello packets across all of its configured interfaces (eth0 and hsi0) until the OSPF process has actually been halted. Then, when the OSPF process has completely stopped, such that the guest is no longer able to send OSPF Hello packets, all of the remaining 65 OSPF neighbors, as well as the two ABRs, will recognize the lost OSPF neighbor adjacency when they have not received a Hello packet from the Linux guest that was shutdown within the prescribed Dead Timer interval. An

OSPF adjacency loss occurs for each OSPF neighbor across each of the physical interfaces (hsi0 and eth0). It is important to point out that after an adjacency has been lost, each OSPF instance (all Linux guests and ABRs) will need to update their respective Link State Database to accurately reflect the changes in the network topology.
B) The second variation of this same scenario was designed to quantify the OSPF overhead associated with forming OSPF neighbor adjacencies at both an individual Linux guest level, as well as on a cumulative level (with 65 Linux guests and two ABRs). To form an OSPF adjacency across each network interface, a complex series of packet exchanges occurs. For example, each Linux guest, along with the ABRs, steps through a series of link state changes (down, attempt, init, 2-way) with the new host for each interface, in order to form an OSPF neighbor adjacency. In addition, each OSPF designated router (the three Linux guests acting as a DR or BDR on the HiperSockets LAN, along with the two ABRs) and the new host being added step through an additional set of link state changes (exchange, loading, and full) for each interface. This set of link state changes is done to exchange full topology information and ensure that their link state databases are fully in sync. Specifically, when a Linux guest in this environment was restarted, it had to systematically establish an OSPF adjacency with each of the other Linux guests (65 in this case) and with the two ABRs across the VSWITCH network, in addition to establishing OSPF adjacencies with each of the 65 Linux Guests across the HiperSockets Interface. No measurable CPU consumption increase was observed on any of the Linux instances during the shutdown and restart. We observed that the processor use for the shutdown and restart procedures dwarfed the OSPF overhead by an order of magnitude.

2) Halting five guests concurrently via the "SIGNAL" command from z/VM, and the subsequent restart of those same guests after a simulated maintenance window with the "XAUTOLOG" command. As with the single host scenario above, this scenario was designed to quantify the OSPF overhead associated with losing and establishing five OSPF adjacencies on a cumulative level (across 61 of the 66 Linux Guests on one CPC). We observed little measurable CPU overhead during the shutdown and restart portion of the procedure for each of the Linux instances. Once again, the shutdown and restart procedures of the five guests dwarfed the increase in CPU consumption of OSPF link state updates across the remaining Linux instances by an order of magnitude.

The second group of scenarios focused on Failure and recovery. These scenarios were designed to emulate the types of unexpected outages that our OSPF architecture is intended to accommodate.

1) The first scenario was to drop the VSWITCH connection (eth0 device) on a single guest. This was followed by reactivation of that network interface a few minutes later. Note, as mentioned in the OSPF reference architecture paper, when recovering a networking device such as a HiperSockets, or in this case the eth0 device, a running Zebra service does not detect the "ifup eth0" command activity. It is strictly necessary to restart zebra and OSPF so that both the Zebra and OSPF processes recognize the interface. You may choose to work around this behavioral quirk by scripting a solution in your environment to restart both the Zebra and OSPF processes on these events. As expected, the Linux guest losing the VSWITCH connection immediately sent an LSA out across its HiperSockets interface to

inform its neighbors on that broadcast medium that it no longer has a link to the VSWITCH network. Furthermore, the Linux guest also lost the default route that it had dynamically acquired from the ABRs, because it lost its only connection to the ABRs. The two ABRs recognized that they had lost an OSPF adjacency. They performed the necessary link state re-convergence calculations when they did not received a Hello packet from the Linux guest that had lost its VSWITCH connection within the prescribed Dead Timer interval. Finally, the resulting routing tables on the ABRs correctly reflected that the shortest path to the Linux guest with the failed VSWITCH connection was across the HiperSockets network from any one of the remaining Linux guests that were still actively connected to the VSWITCH.

2) The second pair of related scenarios is the logical extension of the first. During this scenario, we disconnected the entire VSWITCH, paused for a few minutes to simulate maintenance or outage detection time, then recovered the VSWITCH. We observed the expected loss of each of the dropped VSWITCH links on the ABRs, as well as a small but measurable increase in CPU utilization on the Linux instances elected as the DR and BDR on the HiperSockets network, while these Linux instances updated their routing tables and relayed the re-converged routes to the HiperSockets addresses of each Linux instance. Once again, as expected, all of the Linux guests immediately sent an LSA out across the Hipersockets interface to inform all the OSPF neighbors that each guest no longer had a link to the VSWITCH network. Furthermore, each Linux guest also lost the default route that it dynamically acquired from the ABRs, because it lost its only connection to the ABRs. The two ABRs recognized that they had lost OSPF adjacencies with each Linux Guest, and they performed the necessary link state re-convergence calculations when they did not received Hello packets from each Linux guest within the prescribed Dead Timer interval. Finally, the resulting routing tables on the ABRs correctly reflected that there was no longer a route available to reach any of the Linux guests.

Observations and Conclusion

During periods of failure we saw small spikes in CPU consumption, as expected. These spikes were very moderate and were no more than 150% of the baseline utilization. These values were observed on the order of 1.4% of a single CPU at peak, as compared to a baseline of 0.8% of a single CPU at idle, as measured with the Linux “top” command. Attempts at observing overhead of these non-router Linux instances running OSPF as seen from z/VM Performance Toolkit indicated that the individual overhead was so small as to be not reliably measurable. The guests operating as the designated and backup designated routers for the HiperSockets connection experienced marginally more CPU consumption than the peers who were not designated routers. These routers were measured between 3-4% of a single CPU according to the output of the top command from those Linux guests. Attempts at observing overhead of these Linux OSPF routers as seen from z/VM Performance Toolkit indicated the individual overhead was so small as to be unobservable reliably on a system with approximately 100 virtual servers running the various routine workloads that occur daily in our testing environment. Processor utilization of the LPAR was observed to be approximately 60% utilization during the study. The paging rate was low, at approximately 185 pages per second.

In general, we find the results to be quite positive. It is our position that OSPF on Linux on IBM System z has been shown to be an effective use of the computational expense for our environment at a moderate scale. We have found the results to be favorable enough to proceed with a roll out across our test systems, however our findings should not be taken as a recommendation, or strict scientific measurements under controlled conditions. Rather, these observations should be seen as an endorsement of feasibility based on empirical observations in our own dynamic environment.

Our Stub Area configuration for this study, while simple, was quite optimal in that it behaved very much like a Totally Stubby Area. This behavior can be attributed to only having a single pair of ABRs which never had an opportunity to send, or to receive inter-area routes from other ABRs. Therefore, the ABRs never sent summary LSAs (type 3 and type 4) into our Stub Area. Additionally, this configuration also greatly reduced overhead for every server that was not configured to be elected as either the DR or BDR, which in turn tends to minimize the cumulative overhead.

It is worth noting that for OSPF topologies with a large number of fully meshed ABRs, a Totally Stubby Area would be the preferred configuration. For this situation, configuring a Totally Stubby Area would prevent the importing of hundreds, or even thousands, of inter-area routes in the form of summary link state advertisements that are accessible via the ABRs. A Totally Stubby Area deployment would effectively prevent each Linux guest from having to recalculate hundreds or thousands of routes if an ABR is ever bounced up and down, with the resulting overhead likely being quite significant.

Another issue to consider is complexity. An OSPF routing daemon must be properly configured on each Linux guest, which is a non-trivial operation. If the OSPF areas are not configured optimally, the kind of route convergence overhead just discussed is a real possibility. There also exists the potential for future maintenance issues. For example, if network MTU sizes must be adjusted at a later date, OSPF sensitivity to MTU size might require that change to be implemented on all Linux guests at the same time. Such complexity issues are probably manageable for network administrators who are highly skilled with OSPF, but for the novice they could present some unique challenges.

As with any recommendation, careful consideration should be given to your own operational environment and business-resilience needs, when evaluating OSPF for production use. We recommend performing your own measurements before placing this solution in production.

Appendix

[Cpu] CPU Information as observed on a representative Linux instance:

```
OSPF0036:~ # cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 1
```

```
bogomips per cpu: 6868.00
features        : esan3 zarch stfle msa ldisp eimm dfp
processor 0: version = FF,  identification = 017456,  machine = 2094
```

[Mem] Memory as observed on a representative Linux instance:

```
OSPF0036:~ # free -m

             total       used       free     shared    buffers     cached
Mem:           494        403         91          0        142        175
-/+ buffers/cache:         85        409
Swap:            0          0          0
```

[Disks] Disk as observed on a representative Linux instance:

```
OSPF0036:~ # df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/dasda1     6.8G  4.7G  1.9G  72% /
udev            248M  128K  248M   1% /dev
```

[Nics] Network configuration as observed on a representative Linux instance:

```
OSPF0036:~ # ifconfig

dummy0    Link encap:Ethernet  HWaddr 76:93:38:0D:9B:5D
          inet addr:10.20.50.36  Bcast:0.0.0.0  Mask:255.255.255.255
          inet6 addr: fe80::7493:38ff:fe0d:9b5d/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14245 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:1110966 (1.0 Mb)

eth0      Link encap:Ethernet  HWaddr 02:06:00:00:01:5C
          inet addr:192.168.72.36  Bcast:192.168.72.255 Mask:255.255.255.0
          inet6 addr: fe80::6:ff:fe00:15c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1492  Metric:1
          RX packets:1735305 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1398048 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:370662567 (353.4 Mb)  TX bytes:146673533 (139.8 Mb)

hsi0      Link encap:Ethernet  HWaddr 06:00:FD:01:00:5E
          inet addr:172.17.91.36  Bcast:172.17.255.255  Mask:255.255.0.0
          inet6 addr: fe80::400:fdff:fe01:5e/64 Scope:Link
          UP BROADCAST RUNNING NOARP MULTICAST  MTU:16384  Metric:1
          RX packets:1404685 errors:0 dropped:0 overruns:0 frame:0
          TX packets:162431 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

```
          RX bytes:346037613 (330.0 Mb)  TX bytes:21644741 (20.6 Mb)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1820 (1.7 Kb)  TX bytes:1820 (1.7 Kb)
```

[Ospf] Linux instance OSPF daemon configuration:

```
OSPF0036:~ # cat /etc/quagga/ospfd.conf
hostname ospf0036.ltic.pok.ibm.com
password zebra
enable password zebra
log stdout

! Static VIPA aka Dummy network
interface dummy0
ip ospf cost 1
ip ospf priority 0

! Service IP -
! Any Floating VIPA nic aka service IP device
interface dummy0:0
ip ospf cost 1
ip ospf priority 0

! Vswitch
interface eth0
ip ospf cost 2
ip ospf priority 0

! HiperSockets
interface hsi0
ip ospf cost 1
ip ospf priority 0

! Loopback
interface lo

!
interface sit0

! Some insight into the following lines:
```

```
! ID is the Dummy0 (DNS NAME of host)
! Dummy0 Lan 32 bit full host mask
! HiperSockets0 Lan 16 bit mask
! VSwitch Lan 24 bit mask

!
router ospf
ospf router-id 10.20.50.36
network 10.20.50.36/32 area 2.2.2.2
network 172.17.91.36/24 area 2.2.2.2
network 192.168.72.36/24 area 2.2.2.2
area 2.2.2.2 stub

!
line vty
exec-timeout 0 0
!
```

[Zebra] Linux instance zebra configuration:

```
LITOSPF2:~ # cat /etc/quagga/zebra.conf
!
hostname ospf0036.ltic.pok.ibm.com
password zebra
enable password zebra
log file /var/log/quagga/quagga.log

! Static VIPA
interface dummy0
ip address 10.20.50.36/32
ipv6 nd suppress-ra

!
interface dummy0:0
ipv6 nd suppress-ra

!
interface eth0
ip address 192.168.72.36/24
ipv6 nd suppress-ra

!
interface hsi0
ip address 172.17.91.36/24
ipv6 nd suppress-ra

!
interface lo
```

```
!
interface sit0
ipv6 nd suppress-ra

!
ip forwarding

!
line vty
exec-timeout 0 0
```

[Opsys] Linux instance operating system identification:

```
OSPF0036:~ # cat /etc/issue
Welcome to SUSE Linux Enterprise Server 11 (s390x) - Kernel \r (\l).

OSPF0036:~ # uname -a
Linux OSPF0036 2.6.27.19-5-default #1 SMP 2009-02-28 04:40:21 +0100 s390x s390x s390x GNU/Linux
```

[Procs] Processes running during the experimental observations:

```
OSPF0036:~ # ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1116   384 ?        Ss   Nov30   0:02 init [3]
root         2  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S<   Nov30   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        S<   Nov30   0:01 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Nov30   0:19 [events/0]
root         6  0.0  0.0      0     0 ?        S<   Nov30   0:00 [khelper]
root         7  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kintegrityd/0]
root         8  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kblockd/0]
root         9  0.0  0.0      0     0 ?        S<   Nov30   0:00 [cqueue]
root        10  0.0  0.0      0     0 ?        S<   Nov30   0:00 [cio_chp]
root        11  0.0  0.0      0     0 ?        S<   Nov30   0:00 [cio]
root        12  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kslowcrw]
root        13  0.0  0.0      0     0 ?        S<   Nov30   0:00 [appldata]
root        15  0.0  0.0      0     0 ?        S    Nov30   0:00 [pdflush]
root        16  0.0  0.0      0     0 ?        S    Nov30   0:03 [pdflush]
root        17  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kswapd0]
root        18  0.0  0.0      0     0 ?        S<   Nov30   0:00 [aio/0]
root        19  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kmcheck]
root       316  0.0  0.0      0     0 ?        S<   Nov30   0:07 [kjournald]
root       371  0.0  0.2   3196  1476 ?        S<s  Nov30   0:00 /sbin/udevd --d
root       666  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kstriped]
root      1098  0.0  0.2   3968  1504 ?        Ss   Nov30   0:00 /sbin/srcmstr
100       1122  0.0  0.1   2972   908 ?        Ss   Nov30   0:00 /bin/dbus-daemo
101       1239  0.0  0.6   7864  3096 ?        Ss   Nov30   0:00 /usr/sbin/hald
root      1260  0.0  0.6   8856  3360 ?        Ssl  Nov30   0:00 /usr/sbin/conso
root      1267  0.0  0.2   3828  1268 ?        S    Nov30   0:00 hald-runner
root      1815  0.0  0.1   2848   900 ?        Ss   Nov30   0:00 /sbin/syslog-ng
root      1818  0.0  0.1   2048   604 ?        Ss   Nov30   0:00 /sbin/klogd -c
root      1824  0.0  0.1   2536   744 ?        Ss   Nov30   0:00 /sbin/rpcbind
root      1856  0.0  0.1  11196   888 ?        S<sl Nov30   0:00 /sbin/auditd -s
root      1858  0.0  0.0      0     0 ?        S<   Nov30   0:00 [kauditd]
root      1868  0.0  0.1   2540   920 ?        Ss   Nov30   0:00 /usr/sbin/cron
```

```
root      1894  0.0  0.9  23852  4724 ?        Sl   Nov30   0:23 /usr/sbin/rsct/
root      1948  0.2  2.0  28392 10448 ?        Sl   Nov30   8:49 /usr/sbin/rsct/
root      2271  0.0  0.2 117628  1348 ?        Ssl  Nov30   0:00 /usr/sbin/nscd
daemon    2291  0.1  0.1   5172   732 ?        Ss   Nov30   7:22 /usr/sbin/slpd
root      2304  0.0  0.2   6500  1372 ?        Ss   Nov30   0:02 /usr/sbin/sshd
root      2321  0.0  0.1   2120   640 ttyS0    Ss+  Nov30   0:00 /sbin/mingetty
quagga    5285  0.0  0.3   6608  1704 ?        Ss   Dec01   0:00 /usr/sbin/zebra
quagga    9104  0.1  0.5   7712  2700 ?        Ss   Dec02   1:59 /usr/sbin/ospfd
root     10970  0.0  0.6   9900  3300 ?        Ss   10:55   0:00 sshd: root@pts/
root     10972  0.0  0.5   4780  3032 pts/0    Ss   10:55   0:00 -bash
root     11028  5.2  0.2   2852  1016 pts/0    R+   11:04   0:00 ps aux
```

[Routes] Example Linux Instance Routing Table -

```
OSPF0036:~ # route -n  | sort
0.0.0.0         192.168.72.5    0.0.0.0         UG    3      0        0 eth0
10.20.50.20     172.17.91.20    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.21     172.17.91.21    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.23     172.17.91.23    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.24     172.17.91.24    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.25     172.17.91.25    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.26     172.17.91.26    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.27     172.17.91.27    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.28     172.17.91.28    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.29     172.17.91.29    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.30     172.17.91.30    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.31     172.17.91.31    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.32     172.17.91.32    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.33     172.17.91.33    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.34     172.17.91.34    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.37     172.17.91.37    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.38     172.17.91.38    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.39     172.17.91.39    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.40     172.17.91.40    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.41     172.17.91.41    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.42     172.17.91.42    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.43     172.17.91.43    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.44     172.17.91.44    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.45     172.17.91.45    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.46     172.17.91.46    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.47     172.17.91.47    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.48     172.17.91.48    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.49     172.17.91.49    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.50     172.17.91.50    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.51     172.17.91.51    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.52     172.17.91.52    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.53     172.17.91.53    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.54     172.17.91.54    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.55     172.17.91.55    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.56     172.17.91.56    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.57     172.17.91.57    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.58     172.17.91.58    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.59     172.17.91.59    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.60     172.17.91.60    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.61     172.17.91.61    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.62     172.17.91.62    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.63     172.17.91.63    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.64     172.17.91.64    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.65     172.17.91.65    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.66     172.17.91.66    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.67     172.17.91.67    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.68     172.17.91.68    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.69     172.17.91.69    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.70     172.17.91.70    255.255.255.255 UGH   2      0        0 hsi0
```

```
10.20.50.71     172.17.91.71    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.72     172.17.91.72    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.73     172.17.91.73    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.74     172.17.91.74    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.75     172.17.91.75    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.76     172.17.91.76    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.77     172.17.91.77    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.78     172.17.91.78    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.79     172.17.91.79    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.80     172.17.91.80    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.81     172.17.91.81    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.82     172.17.91.82    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.83     172.17.91.83    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.84     172.17.91.84    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.85     172.17.91.85    255.255.255.255 UGH   2      0        0 hsi0
10.20.50.86     172.17.91.86    255.255.255.255 UGH   2      0        0 hsi0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
169.254.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 hsi0
172.17.91.0     0.0.0.0         255.255.255.0   U     0      0        0 hsi0
192.168.72.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.74.0    192.168.72.5    255.255.255.0   UG    3      0        0 eth0
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
Kernel IP routing table
```

[IOCPGuide] IOCP User's Guide (ICP) -
Chapter 1: “IOCP Introduction”, Section Title: “Internal Queued Direct Communication (HiperSockets) support”. IBM Publication SB10-7037-07. Available on-line as of the time of writing at:
https://www-304.ibm.com/servers/resourcelink/lib03010.nsf/0/B7315162CC7511ED852574EA004D791A/$file/SB10-7037-07.pdf

[Arch] - A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF -
Dow, Markos, Loveland. Available on-line under the “Related Publications” section at the time of this writing at:
http://www-03.ibm.com/systems/services/platformtest/servers/systemz_library.html

[Scaling] - A Note on Scaling
http://www-03.ibm.com/systems/services/platformtest/servers/systemz_library.htmlThe IBM System z10™ Enterprise Class (z10 EC™) current architectural limit for the maximum number of Linux instances connected to a single HiperSockets network is 4096 as achieved by optimally using each of the maximum 12288 subchannels [IOCPGuide]. The IBM VSWITCH technology is unbounded in the number of Linux instances that may connect pending system memory for consumption by z/VM. Thus it is theoretically possible to have 4096 hosts in this architecture, it is likely that other factors will become limiting when attempting to scale.

IBM Systems and Technology Group
Route 100
Somers, New York 10589
U.S.A.
Produced in the United States of America,
05/2010

ZSW03165-USEN-00