

A Reference Implementation Architecture for Deploying a Highly- Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

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

Table of Contents

Abstract.....	3
1. Introduction and motivation.....	3
1.2 Prior work.....	4
2. Technologies and techniques.....	4
2.1 VSWITCH.....	4
2.2 Direct OSA and Channel Bonded OSA.....	6
2.3 HiperSockets.....	6
2.4 High Availability IP addressing.....	7
2.4.1 Virtual IP Addressing (VIPA).....	7
2.4.2 Floating IP address (Service IP address) and floating VIPA.....	8
2.5 Open Shortest Path First: What it is, and why would you want it.....	10
2.5.1 Leveraging OSPF with Virtual IP Addressing (VIPA).....	12
3 Implementation Alternatives.....	13
3.1 Single subnet, automated NIC failover, and clustered service IP addresses.....	13
3.1.1 Is this solution good enough for your needs?.....	13
3.2 Multiple subnets, Dual-homed hosts, and static VIPA with OSPF.....	14
3.2.1 Is this solution good enough for your needs?.....	14
3.3 Multiple subnets, Dual-homed hosts, and floating VIPA with OSPF and TSA.....	15
3.3.2 Is this solution good enough for your needs?.....	15
4. Experimental evaluation.....	15
4.1 Network topology diagram and key network design points.....	16
4.2 VIPA.....	17
4.3 VIPA & OSPF.....	17
4.3 Key OSPF considerations.....	18
4.3.1 OSPF Costing.....	18
4.3.2 OSPF Stub Area.....	18
4.3.3 OSPF Designated Router Consideration: Router Priority.....	19
4.3.4 OSPF ABR to ABR interconnectivity consideration.....	19
4.4 Setup and baseline configuration points.....	19
4.4.1 Baseline OSA discussion.....	20
4.4.2 Baseline OSPF discussion.....	20
4.4.3 Linux for IBM System z - Zebra and OSPF configuration.....	20
4.4.4 Static VIPA, floating VIPA discussion.....	21
4.4.5 Baseline Key metrics discussion: The value that OSPF brings to the table.....	22
4.5 Failover-resiliency scenarios.....	28
4.5.1 First Failure: Loss of the HiperSockets connection.....	28
4.5.2 – Second Failure: Loss of the Ethernet device.....	29
4.5.3 – Third Failure: Loss of the HiperSockets again.....	30
4.5.4 – Fourth Failure: Hard failure of the server hosting the floating VIPA.....	31

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

4.5.5 Fifth Scenario: Server recovery.....	32
4.6 Summary of observations (CPU and memory intensiveness).....	32
5. Discussion and future work.....	33
6. Conclusion	33
Appendix	34
References.....	34
Reference configurations	36
Reference z/OS OSPF configuration – complete profile.....	36
Reference z/OS TCP/IP (partial) profile – only key configuration statements	38
Linux for IBM System z reference configuration for one Linux for IBM System z guest (Zebra.conf).....	39
Linux for IBM System z reference configuration for one Linux for IBM System z guest (Ospfd.conf)	40
Hints and Tips	42
Quagga OSPF MTU interoperability	42
Ifconfig considerations for a Zebra interface	42

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Abstract

This paper demonstrates a viable, highly available, networking infrastructure based on a sophisticated combination of dynamic routing (Open Shortest Path First) and highly resilient networking technologies, including Virtual Internet Protocol Addressing (VIPA), HiperSockets™ and IBM VSWITCH, to provide increased application server availability. Our system has been deployed and tested on a set of virtual Linux® servers running on an IBM System z10™, but the approach is applicable to practically any Linux system with some modification. We have observed empirically that this architecture provides fault tolerance while incurring only minimal operational overhead when implemented with due care. It should be noted that this pilot study was performed using few nodes, in order to ensure correctness of operation and proof of concept. A future test, and subsequent report, will examine this solution at scale.

1. Introduction and motivation

With the increasing number of cloud computing providers offering hosted infrastructure, a key measure of their pricing models has been charge per compute, or charge per guaranteed uptime. We contend that in order to maintain the high levels of availability needed to maintain those guarantees, sophisticated networking implementations are required.

In addition, as locally hosted data centers become more virtualized, systems are often provisioned with no direct form of access at all. The sole mechanism of access to many of these systems for their end users is with some form of networking technology.

Realizing these issues, it quickly becomes apparent that networking can become a weak point in application availability. But what do we mean when we say high availability?

In many ways, HA is concerned with a system's ability to provide continuous service despite individual component failures. Understanding all of an application's Single Points of Failure (SPOF) is the first step in deploying a HA solution. Therefore, building a highly-available network accessible solution begins by selecting:

- Highly reliable system components (such as IBM System z® servers)
- Reliable operating systems (for example, z/OS® or Linux for IBM System z)
- Dynamic routing protocols (such as Open Shortest Path First)
- Resilient application software

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Layered on top of your applications, it may also be prudent to use a class of automation software which enables mechanisms to deal with individual failures in the event that your software does not do so already. Two examples of automation software used in HA systems are the IBM Tivoli® System Automation for Multiplatforms product, and the Linux-HA.org package.

1.2 Prior work

IBM mainframes, along with z/OS, have a long published history when it comes to providing HA. While it is not this paper's intention to provide details on previously established HA best practices, this paper will leverage those practices when possible, citing the respective reference publication(s) where the additional information can be found.

2. Technologies and techniques

This paper documents the implementation of dynamic network routing and failover capabilities, utilizing technologies such as:

- HiperSockets
- Virtual IP Addressing (VIPA)
- Open Shortest Path First (OSPF) dynamic routing protocol
- Open System Adapter (OSA)
- Linux for IBM System z
- z/VM®, and z/VM Virtual Switch (VSWITCH)

We illustrate how these technologies can be combined with the Tivoli System Automation for Multiplatforms (TSA) solution to increase overall availability. We start by providing a brief overview of some key IBM System z technologies that contribute fundamentally to the high availability architecture we have implemented. If you are already familiar with these technologies, feel free to skip to Section 2.4 "High Availability IP Addressing".

2.1 VSWITCH

Our solution begins with VSWITCH. Running Linux under z/VM enables the use of z/VM's virtualized LAN capabilities: Guest LANs and VSWITCHs. A z/VM Guest LAN is a simulated LAN that is entirely contained within a given z/VM instance. As is the case with a HiperSockets virtual LAN, a z/VM Guest LAN in its default configuration has no external connectivity [VLAN1] [VLAN2].

A z/VM VSWITCH is essentially a special type of Guest LAN that can have a built-in connection to the external network. The VSWITCH can connect directly to an OSA-Express QDIO port. It acts like a real network switch, providing a transparent bridge to the outside network. When connected to a VSWITCH, Linux guests become part of the same subnet as the hosts on the external network to which the VSWITCH is physically connected. In other words, a VSWITCH allows you to flatten your network topology [Vswitch1].

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

VSWITCH is one mechanism that can insulate your software servers from the outage of a particular OSA network device. Redundant OSA ports can be defined to a VSWITCH in either active/backup or link aggregation mode, or a combination of both modes to provide an excellent means of insulating your software servers from the loss of a particular OSA adapter. In active/backup mode, there is no load balancing between the redundant OSA ports because this mode uses one port at a time. If one port fails, the VSWITCH will fail-over to the next port in its list. No special configuration of external network switches is required for VSWITCH active/backup mode.

In link aggregation mode, workload is balanced across all aggregated ports, providing a higher-bandwidth “fat pipe” for network traffic. Should one port fail, traffic simply flows over the surviving ports. This mode typically does require some additional configuration of the external switch to which the aggregated ports are connected [LinkAgg1]. A combination of both approaches can also be configured, such that should an entire aggregation group lose connectivity, it will fail-over to an individual backup port.

z/VM uses a special virtual machine type, called a “controller,” to manage and administer the VSWITCH. This controller machine owns the OSA devices and manages their availability to the VSWITCH. The controller machine does not participate in the IP data flow. Redundant VSWITCH controllers can be defined to support a single VSWITCH. If the active controller fails or loses its IP connectivity, clients will see only a brief pause in traffic while the backup takes over, and then traffic will resume where it left off. If the failing controller is autologged (z/VM terminology for started automatically), z/VM will immediately restart this controller and it will become the new backup.

HA aspects can be combined cumulatively by defining redundant VSWITCH controllers with redundant OSA ports assigned to a VSWITCH. If a pair of controller service machines are initialized and registered as controllers when a new VSWITCH is defined, redundant OSA ports defined for that VSWITCH will be automatically spread across the controllers by z/VM. In this case, failure of the active OSA port will not only shift to the backup OSA port, but if that OSA port was the last one associated with that controller machine, then the controller will change as well. In this way, the VSWITCH is simultaneously protected against different potential failures [TestRepVswitchContr] [VswitchRedundancy].

VSWITCHes can be configured to run in two modes. The first is Layer 2 mode (ETHERNET), in which all destinations are identified by MAC (Media Access Control) addresses, and all hosts maintain their respective ARP (Address Resolution Protocol) caches. The second mode is Layer 3 mode (IP), in which all destinations are identified by IP addresses, and ARP processing is offloaded to the OSA-Express adapter. A given VSWITCH can operate in only one mode. The use of link aggregation requires Layer 2 mode. Also note that there is no way to physically connect a HiperSockets virtual LAN directly into a VSWITCH LAN. Joining these two

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

networks requires an intermediate gateway, such as a multi-homed host residing on the Central Processing Complex (CPC) that is acting as a router.

2.2 Direct OSA and Channel Bonded OSA

z/OS and Linux LPARs and z/VM guests running on IBM System z servers can also connect directly to OSA-Express adapter ports. The Linux bonding driver [Bond1] [Bond2] can also be used to join multiple Layer 2 mode OSA-Express devices into a single logical device that can be configured in various modes, including active/backup, and link aggregation modes, with fail-over characteristics similar to what was described earlier for VSWITCH [Bond1].

From the perspective of the Linux system, the device representing a directly-connected OSA-Express port appears identical to that of a VSWITCH connection. It makes no difference to routing protocols or applications which type of connection is used. For this paper, the z/VM Linux guests used VSWITCH and HiperSockets connections, but not direct OSA-Express connections. Our z/OS LPARs used HiperSockets and direct OSA-Express connections (but not VSWITCH, because they were not running as guests under z/VM).

2.3 HiperSockets

HiperSockets is a technology deployed to add an additional networking conduit to our virtual machines without adding the burden typically associated with traditional networks such as cabling and equipment (switches, routers, and hubs), as well as the respective power, administration, and floor space requirements. In addition, HiperSockets can improve reliability and availability because there are no network hubs, routers, adapters, or wires to fail. Because HiperSockets communication is through the processor's system memory, there is no means for directly connecting a HiperSockets virtual LAN to an external physical network. Therefore, interconnectivity between a physical network and a HiperSockets LAN can be provided through a multi-homed host that is acting as a router.

Adding a HiperSockets connection is an inexpensive way to make each Linux guest multi-homed, which is fundamental tenet of serious HA deployments. By providing server connectivity to more than one broadcast domain, dynamic routing can be leveraged to ensure that packets can be redirected around a failure in the network.

HiperSockets technology provides very fast TCP/IP communications between z/OS and Linux servers running in different LPARs and z/VM guests within a single IBM System z server. HiperSockets is a Licensed Internal Code (LIC) function that emulates the Logical Link Control (LLC) layer of an OSA-Express QDIO interface. It uses internal Queued Input/Output (iQDIO) protocol, setting up I/O queues in the IBM System z processor's memory. Traffic then passes between virtual servers at memory speeds, totally eliminating the I/O subsystem overhead and external network delays. The virtual servers that are so connected form a "virtual LAN".

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Multiple independent HiperSockets virtual LANs are supported within a single IBM System z processor, as well. [HipersocketsImp].

Traditionally, HiperSockets supported only the Layer 3 (Network and IP Layer) transport mode. The IBM System z10 Enterprise Class (z10 EC™) server added support for Layer 2 (Link Layer). HiperSockets Layer 2 capability is supported by native Linux for IBM System z LPARs and by Linux as a guest under z/VM [SysConHandbook].

2.4 High Availability IP addressing

While VSWITCH and HiperSockets provide acceptable networking redundancy, redundancy in itself is not always sufficient to provide a satisfactory level of high availability. Dynamic routing protocols, such as Open Shortest Path First (OSPF), enable networks to take advantage of redundant connectivity to route traffic around failed links. While OSPF provides an additional level of availability to a solution, utilizing Virtual IP Addressing as part of the OSPF configuration will further increase a solution's availability.

In high availability configurations, it is important to always present end users with a consistent IP address for a given service, regardless of changes to the underlying infrastructure used to deliver that service. One means for doing this is through the use of Virtual IP addressing (VIPA). [VipaHA].

2.4.1 Virtual IP Addressing (VIPA)

A traditional IP address is associated with each end of a physical network link. Within an IP routed network, failure of any intermediate link, or failure of a physical network adapter will disrupt end user service unless there is an alternate path available through the routing network.

A Virtual IP Address (VIPA) removes the physical network adapter as a single point of failure by associating an IP address with the server's TCP/IP stack, subsequently making the VIPA accessible over all of the stack's physical interfaces and not limiting it to a specific physical network attachment. Therefore, because a VIPA does not have a physical network attachment associated with it, a VIPA is active as soon as the TCP/IP stack is active and it becomes accessible across each of the stack's respective physical interfaces as each interface becomes active.

To the routed network, a VIPA appears to be a host destination (a 32-bit network prefix for IPv4 protocol) on a multi-homed TCP/IP stack. When a packet with a VIPA destination reaches the TCP/IP stack that owns that particular VIPA, the IP layer recognizes the address as an address in the TCP/IP stack's Home list, and passes it up to the transport protocol layer in the stack for application processing. Note that VIPA should be on a different subnet than other IP addresses defined for that host. External routers must then be told that they can reach that VIPA through IP addresses associated with the physical network adapters on that host.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Defining a VIPA to z/OS involves some formal configuration to the TCP/IP stack definitions in the z/OS TCPIP.PROFILE data set [ComSrv]. For Linux for IBM System z, different approaches are used depending on whether underlying OSA devices are being used in Layer 2 or Layer 3 mode. If they are in Layer 3 mode, then multiple steps are involved. First, a dummy0 networking device is defined to the Linux TCP/IP stack and given its own IP address. Then, that IP address is added to each underlying OSA device's Ethernet definition using the "qethconf vipa add" command, or by directly editing a special VIPA subdirectory within the sysfs file system directly [DeviceDrivers]. For Layer 2 mode, all that is needed is to define an IP address to a dummy0 device. This approach associates the dummy 0's respective IP address with the Linux server's TCP/IP stack, effectively removing the IP address from any of the server's physical interfaces. In this paper, we use the Layer 2 approach.

The z/OS Communications Server supports two types of Virtual IP Addressing (VIPA); Static and Dynamic [CS-HA-Redbook].

A static VIPA is an IP address that is associated with a particular TCP/IP stack. Using either ARP takeover or a dynamic routing protocol (such as OSPF), static VIPAs can enable mainframe application communications to continue, unaffected by network interface failures. As long as a single network interface is operational on a host, communication with applications on the host will persist. A common practice is to use a server's Static VIPA in an installation's DNS strategy. This approach allows an installation the flexibility to non-disruptively add, remove, or replace network interfaces to an existing server, as well as move an existing server to another CPC within the routed data center without having to worry about the underlying networking infrastructure. Also, because a Static VIPA uniquely identifies a server and because every OSPF router must have a unique router-id, we used each server's static VIPA (z/OS or Linux) as the OSPF router-id for each respective z/OS and Linux for IBM System z server in our environment.

A Dynamic VIPA (DVIPA) can be defined on multiple stacks and dynamically moved between TCP/IP stacks in the sysplex. One stack is defined as the primary or owning stack, while the others are defined as backup stacks.

Linux supports the use of static VIPAs with straightforward configuration. In this paper we will demonstrate an approach for creating capability similar to a dynamic VIPA, a VIPA that can float between Linux systems, by combining static VIPA technology with the notion of a floating service IP address.

2.4.2 Floating IP address (Service IP address) and floating VIPA

Within the context of this paper, the terms "*floating IP address*", "*service IP address*", "*floating VIPA*", and "*floating service IP address*" will be used interchangeably. In high availability configurations, it is important to always present end users with a consistent IP address for a given service, regardless of changes to the underlying infrastructure used to deliver that service. One means for doing this is through the use of a floating IP address, sometimes known as a *service IP* address [ServiceIP].

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

The technique used to define such an address is known as IP aliasing. IP aliasing assigns an additional “alias” IP address to a network interface that the interface will respond to. Unlike a VIPA, the alias must be on the same subnet to which that interface is already connected, so that no additional routing definitions are necessary.

On Linux, alias IP addresses are assigned using the `ifconfig` command similar to typical IP address, but with a small twist. Consider the following example:

```
# ifconfig eth0          192.168.71.114 netmask 255.255.255.0
# ifconfig eth0:0       192.168.71.97  netmask 255.255.255.0
```

In this example, our `eth0` device’s physical IP address is `192.168.71.114` and its first alias is `192.168.71.97`. Note that both are on the same subnet, `192.168.71.0/24`. Additional aliases could be defined by simply incrementing the alias number (`eth0:1`, and so forth). A subsequent `ifconfig` display command will show both addresses being associated with `eth0`. Also, it is important to note that existing routing definitions for the `192.168.71.0/24` subnet must already be in place so that both the physical and the alias IP addresses will dynamically be recognized and subsequently be correctly routed for both of these IP addresses. The same approach is applicable regardless of whether the network interface device is operating in Layer 2 or Layer 3 mode.

A service IP address is commonly used in active/cold-standby or active/passive clusters, where multiple systems are defined to the cluster, but a target application (and its associated IP address) can be active on only one node at a time. Should that system fail, the application and its associated service IP address are dynamically moved to a surviving node in the cluster.

In this example, an application that delivers service to end users would be addressed using the alias, or service IP address, of `192.168.71.97`. In the event that the host fails, we can re-alias that same service IP address to another network adapter on another host in the same subnet, restart the application on that host, and our application would then resume providing service to users through the customary IP address. The additional step of issuing a gratuitous ARP command is often needed in order to direct network switches to flush their ARP cache of any residual information about the previous location of the service IP address. By using this pattern, the application’s service IP address floats across network adapters on different hosts, while each network adapter maintains its own, adapter-specific, physical IP address for maintenance or host monitoring purposes.

In order to minimize outage windows, an automation product is typically used to detect loss of connectivity to an application, move its service IP to another host, then restart the application on this new host. IBM Tivoli System Automation for Multiplatforms [TSA] and the open source Linux-HA package [LinuxHA-org] both support the notion of a floating service IP address.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

The combination of VIPAs with floating IP addresses on Linux provides very similar function to dynamic VIPA on z/OS. Associating an IP address with a Linux server's TCP/IP stack through a dummy device on its own subnet creates a static VIPA that removes a physical interface as a potential single point of failure, just as a static VIPA does for z/OS. As long as at least one network interface remains operational on the Linux host, communication with applications on that host will persist. Now, if an additional IP address is aliased to that dummy device (on the same subnet as dummy device's base IP address), and that aliased IP address is treated as the VIPA, then that VIPA can be floated across systems that also have dummy devices defined (in that same subnet) just like any other service IP address. In effect, this creates a "floating VIPA" that is similar in concept to the z/OS dynamic VIPA support. The VIPA protects against loss of an individual network adapter on a given host, and its ability to float (along with its associated application) protects against loss of the entire host.

Our solution also used Tivoli Systems automation (TSA) to provide another level of availability. We relied on TSA to both monitor and to manage our Linux for IBM System z server farm's floating VIPA. For example, when TSA detected that all network connectivity to the node currently hosting the VIPA had been lost, TSA would move the VIPA to the dummy0 device on a new server within our Linux server farm, providing nearly uninterrupted continuous availability to the end user.

The one catch to automating Network Interface Card (NIC) failover through the use of a VIPA is that it requires some means to automatically inform routers in the network how to reach the VIPA through a different physical network adapter after a NIC failure. This is commonly accomplished through the use of a dynamic network routing protocol, such as Open Shortest Path First (OSPF).

The sections that follow explain how each of these technologies were combined and how they harmoniously contributed to satisfying our high availability expectations.

2.5 Open Shortest Path First: What it is, and why would you want it

Open Shortest Path First (OSPF) is a widely used interior gateway dynamic routing protocol which uses link state information to build a mapping of the routing topology within an autonomous system (AS). An OSPF AS consists of a single core area known as the OSPF Backbone (area 0) and typically contains several other areas, such as stub areas, totally stubby areas, or not-so-stubby areas (NSSA).

Each of these areas must have either a direct or virtual connection to the OSPF Backbone. 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) 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

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

it belongs to. Please refer to RFC 2328 for the complete OSPF Version 2 for IPv4 definition. For additional information, you might find the following brief introduction found on Wikipedia useful: http://en.wikipedia.org/wiki/Open_Shortest_Path_First.

Every router in the OSPF Backbone Area (Area 0) maintains a complete map of the entire network topology. Any changes to the network topology (due to link-state changes) are propagated throughout the OSPF Backbone Area. Therefore, every time a router receives a link state update, it 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 topology, as well as the respective link stability within those networks, OSPF can be both processor and memory intensive due to the required re-convergence calculations. While the re-convergence time for OSPF to recalculate the shortest path to each destination is very small, this time can become increasingly large in very large, very unstable networking infrastructures. Though these performance problems may be minimized by utilizing stub areas, it is beyond the scope of this paper to explore all the performance and resource impacts, guidance on suggested solutions, or quantification of the performance impact, with any degree of confidence.

Two other types of OSPF areas are Stub and Totally Stubby areas. A Stub Area is a non-backbone area that is connected to the backbone area through an Area Border Router (ABR). The Stub Area does not receive advertisements about destinations that are in other Autonomous Systems. Such advertisements are called “external link state advertisements” because they refer to Autonomous Systems external to this Autonomous System. The Stub Area knows only about intra-area destinations within the Stub Area. It reaches external destinations through default routes sent to it by the ABR. With smaller link-state databases and smaller routing tables, Stub Areas consume less CPU storage and fewer CPU cycles. [CommServer-Redbook®]

Nodes in a Totally Stubby Area consume even less CPU storage and fewer CPU cycles for OSPF processing because they maintain knowledge only of the intra-area destinations and the default routes to reach inter-area and external destinations. [CommServer-Redbook]

The most common and recommended way to use OSPF in the z/OS environment is to place the z/OS node into an OSPF Stub, or into a Totally Stubby area. This paper advises using the established z/OS recommendations and extends those recommendations to the Linux for IBM System z servers, as well.

Properly architecting OSPF into a solution can increase the level of availability by removing some of the potential Single Points of Failures (SPOFs). Multi-homed servers combined with a VIPA removes a solution’s dependency on a physical network adapter by associating the VIPA to the server’s TCP/IP stack so that the address is accessible through all of the server’s active network interfaces. If one interface experience a link failure, either planned or unplanned, the

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

OSPF routing protocol will dynamically recognize that a route to that same host is still available over one of the server's other remaining active interfaces.

By design, our OSPF deployment addressed both our network load balancing and our route convergence processing overhead requirements. In a fully redundant, fully meshed networking configuration, OSPF may be able to utilize Equal Cost Multi-Pathing (ECMP), which will result in our network traffic (load) being evenly distributed across more networking equipment. When two or more equal cost routes exist to the same destination, the OSPF router can direct one connection through a first switch, while a second connection will be routed through a different network switch.

By designing a fully redundant network topology, ECMP reduces route convergence overhead. Specifically, if equal cost routes are present in a routing table, failure of the active route will cause immediate fail-over to the remaining equal cost route(s) without OSPF convergence computation or latency.

2.5.1 Leveraging OSPF with Virtual IP Addressing (VIPA)

While availability to a VIPA on z/OS can be supported in a statically routed flat network (a single broadcast domain) provided that the VIPA is also defined in the flat network's subnet, doing so requires utilizing DVIPA, which is not available in a Linux for IBM System z environment. Because Linux has no concept of a DVIPA, a much more creative approach is needed to ensure that the solution continues to present end users with a consistent IP address for a given service whenever a change to the underlying infrastructure occurs, or when the floating service IP is moved across the server farm.

So why did we call this section 'Leveraging OSPF with VIPA'? Earlier in this document, we described how a VIPA is represented as a 32-bit host address. Well, by configuring OSPF to include the VIPA as part of the server's OSPF advertisements, routes to the respective VIPA will be dynamically recalculated whenever OSPF detects that a route to the VIPA has either failed (physical failures), or if the VIPA has been moved from one server to another, either manually or dynamically by TSA.

The same fundamental principle applies to Linux with respect to providing HA to the "floating VIPA". By including the necessary definitions in all of the Linux OSPF configuration files (Ospfd.conf), routing updates will dynamically be advertised by the routing stack whenever the floating VIPA is moved by TSA.

While the initial setup for this HA deployment may sound very complicated and be viewed as being much higher in "startup costs", you may find that the long term return on investment (ROI) could be much greater. We endorsed this deployment to satisfy our own needs, and recognize that each installation will need to conduct their own ROI assessment.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

3 Implementation Alternatives

There are various possible approaches for increasing the availability characteristics of a networking infrastructure. However, higher levels of availability are usually associated with greater implementation complexity. Different environments will have different availability requirements, and many people will prefer not to incur more complexity than is required to meet their availability objectives. So it is useful to understand several alternatives and choose the best fit for a given environment. Three such alternatives are described here, with the last one being the focus of this paper.

- Single subnet, automated NIC failover, and clustered service IP addresses
- Multiple subnets, dual-homed hosts, and static VIPA with OSPF
- Multiple subnets, dual-homed hosts, and floating VIPA with OSPF and TSA

3.1 *Single subnet, automated NIC failover, and clustered service IP addresses*

A common, simple, and effective means of increasing the availability of network access to an application is to combine these three items: redundant network interface adapters, automation of resource-level network interface adapter failover, and a floating/service IP address.

For example, Linux guest machines running under z/VM could be connected to the external network through virtual network interface adapters provided by a z/VM VSWITCH. That VSWITCH in turn could make use of redundant OSA adapters, and leverage its built-in active/backup or link aggregation technology for automating OSA port failover, as described earlier. Such an approach provides protection against loss of a physical network adapter. A service IP address for an application could be aliased to a given Linux guest machine's virtual network adapter (provided by VSWITCH); and clustering software such as TSA could automate moving that service IP address and its associated application in the event of a host outage.

As an alternative, if Linux is being run natively in an IBM System z LPAR, OSA channel bonding could be used to automate OSA port failover, and the service IP address could be aliased to the bonded interface generated by the Linux bonding driver.

3.1.1 Is this solution good enough for your needs?

For many deployments, the use of redundant network interface devices with built-in failover capability, along with floating service IP addresses, will be sufficient to meet the availability objectives for server network access. This combination eliminates the server's network interface and the server itself as SPOFs and therefore, in a single broadcast domain (flat network) LAN environment, this will often be good enough.

However, these technologies depend on the server and its applications being accessed through a single subnet (single broadcast domain) and a single class of network adapter. If failures in the

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

external network halt access to that entire subnet, the server's applications will not be reachable. Likewise, if the underlying VSWITCH or channel bonding device driver support experiences problems, there is no alternative path for reaching the application. Alternatively, if a server's network interfaces are already connected to different subnets, channel bonding, redundant VSWITCH connections, and standard service IP addresses are not applicable. In order to diversify the subnets and networking technologies through which an application can be reached, additional technologies are required.

3.2 Multiple subnets, Dual-homed hosts, and static VIPA with OSPF

To diversify access paths to an application across multiple subnets and classes of network adapters, a different approach could be taken. A host could be defined with multiple network adapters of different types, creating a "dual-homed" host. For example, a z/VM virtual server machine could be defined with access to both a virtual network adapter from a VSWITCH, and to a HiperSockets adapter that in turn is connected to z/OS LPAR with a path out to the external network. Each of these connections must be over a different subnet. To provide a single, common IP address for the target application, a static VIPA could be defined that is reachable through either interface. The use of a VIPA in turn requires the use of a dynamic network routing protocol such as OSPF to automate network route failover from one underlying interface to the other.

In a native Linux LPAR, OSA channel bonding could provide one path into the VIPA, and a HiperSockets path could provide the other. OSPF is similarly required in this configuration for the same reason.

3.2.1 Is this solution good enough for your needs?

Though dual homing a Linux virtual server by itself provides alternate communication mechanisms, much of the benefit is not realized unless a dynamic routing protocol is use on top of these adapters. Various attempts at working around the need for dynamic routing are incomplete solutions that are bound to miss edge cases. A proper HA deployment requires a fully meshed, fully redundant networking infrastructure consisting of both physical and virtual networks, Dynamic VIPAs (or Service IPs), and dynamic routing acting in concert. This configuration provides protection for an application running on a given host against subnet failure, individual network adapter failure, or even failure of an entire class of network adapter (such as an entire VSWITCH).

Even with that level of network sophistication, such solutions are only able to protect against loss of network interfaces within a host, and do not provide protection against a complete host outage (and moving the associated VIPA somewhere else). In order to gain this missing piece of network automation, we need to look at mechanisms for automated floating VIPA movement.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

3.3 Multiple subnets, Dual-homed hosts, and floating VIPA with OSPF and TSA

For those situations where you need additional software control logic to move resources between hosts, clustering solutions such as Tivoli System Automation for Multiplatform (TSA) may be appropriate. TSA helps meet high levels of availability and prevent service disruptions for critical applications and middleware running on heterogeneous platforms and virtualization technologies. The product can initiate, execute, and coordinate the starting, stopping, restarting, and failing-over of individual application components and application resources such as virtual IP addresses.

As discussed in section 2.4.2, the use of TSA can be combined with static VIPA on Linux for IBM System z to create a “floating VIPA”, which is conceptually similar to the dynamic VIPA support offered by z/OS. Because dynamic VIPA is built-in to the z/OS infrastructure, dynamic VIPA offers some additional benefits not possible with this externally-generated solution, but the core benefit of a VIPA that can float between systems is still realized.

3.3.2 Is this solution good enough for your needs?

This approach offers very robust availability characteristics by merging the benefits of redundant network adapters on each subnet and floating VIPAs with a dynamic routing protocol. It also removes multiple single points of failure, including network interfaces on the same subnet, the subnet itself, an individual class of network adapter, and server that an application is running on.

However, this approach will not be suitable for everyone. It is vulnerable to the previously-described concerns about cumulative overhead and memory management concerns introduced by OSPF. By introducing OSPF into a traditional floating-service-IP architecture, it adds considerable complexity to what is otherwise typically a fairly straight forward implementation. Also, OSPF is an interior gateway protocol that is intended for use within a given site. By itself, it usually is not sufficient for tying together autonomous systems residing at geographically dispersed sites. Nevertheless, this does promise to be a highly available solution, and so it is the one we focused on testing and which we describe in this paper.

4. Experimental evaluation

This section of the paper begins by describing our network topology, followed by citing the key availability metrics captured in a baseline scenario. We then demonstrate the contrast in these metrics as we systematically cause failures in various components, such as networking links and an active Linux for IBM System z server. We then conclude with a recovery scenario, demonstrating the solution’s ability to provide continuous availability.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

4.1 Network topology diagram and key network design points

The following diagram provides a visual representation of our test environment. Some of the key aspects of this deployment, which will be described in more detail in each of the ensuing sections, are as follows:

- Six servers (three z/OS and three Linux) resided across two CPCs, an IBM System z9[®]-Enterprise Class (z9[®] EC) (2094) and an z10 EC (2097).
- Each server was configured with a unique static VIPA/dummy0: address
- HiperSockets were enabled on each CPC and were configured in each server's TCP/IP definitions.
- The z/OS servers shared two OSA features, with each OSA being connected into different networks.
- The three Linux guests were connected to a VSWITCH, utilizing two OSA features in an active/backup arrangement, with each OSA connected into different interconnected switches, but within the same subnet in each of those interconnected switches (this is depicted by the double line from the dual OSAs supporting the VSWITCH connecting into the network cloud in the diagram).
- Those same three Linux guest are also connected to that CPC's HiperSockets adapter.
 - So while the VSWITCH is not connected to the 192.168.74 subnet, the Linux guests running under its z/VM can still reach that subnet using a HiperSockets connection to a z/OS image that is connected to that subnet.
- Two interconnected, fully redundant OEM network switches were configured as the OSPF ABRs, supporting a stub area for exclusive use by our z/OS and Linux for IBM System z servers.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

While all the technologies mentioned in this paper contributed to our solution's overall HA, the combined use of these technologies (VIPA + TSA + OSPF) provided the greatest contribution in our HA deployment.

4.3 Key OSPF considerations

This section discusses the key OSPF features that we leveraged in our environment.

4.3.1 OSPF Costing

Our costing strategy was simple; the lowest costs were assigned to the OSPF interfaces representing any of the virtual technologies in our infrastructures (networks and hosts) so that it would always be less expensive to direct traffic over one, or two virtual interfaces, rather than over a single physical interface. For example, the OSPF cost associated with a VIPA interface was set to 1, while we assigned a cost of 4 for a HiperSockets interface and a cost of 8 to any connection relying on an OSA. For example, we used the following costing values: VIPA = 1, HiperSockets = 4, OSA = 8.

This ensured that if a direct connection to reach the destination (VIPA) over HiperSockets was available, that the sum of the HiperSockets cost (4) and the VIPA Interface cost (1) would always be less than the cost associated with sending the traffic across a physical network to the destination VIPA through OSA with a total cost of 9; $OSA(8) + VIPA(1)$.

4.3.2 OSPF Stub Area

The most common and recommended way to use dynamic routing in the z/OS environment is to define the stack as belonging to an OSPF stub area, or even better, as belonging to a totally stubby area. Stub areas minimize storage and CPU processing at the nodes that are part of the stub area because they maintain less knowledge about the topology of the Autonomous System (AS) than do other types of non-backbone routers.

Nodes in a totally stubby OSPF Area consume less CPU storage and fewer CPU cycles for OSPF processing than other types of OSPF areas (Backbone and basic stub area) because they need to maintain knowledge only about intra-area destinations and the default routes to reach inter-area and external destinations.

While a totally stubby OSPF Area deployment may be a better choice for larger environments, we found that for our environment, an OSPF stub area was acceptable for our 6-node deployment.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

4.3.3 OSPF Designated Router Consideration: Router Priority

OSPF routers must form adjacencies with one another across each multi-access network that they are attached to, in order to exchange and maintain link state information in the form of Link State Advertisements (LSA). A special router called a Designated Router (DR) on each shared multi-access medium performs most of the OSPF protocol activities for that network, such as synchronizing database information and informing members of the broadcast network of changes to the network. The DR must be adjacent to all other routers on the broadcast medium. Every network or sub-network on a broadcast network must have a DR and preferably a Backup Designated Router (BDR). [CommServer-Redbook, page 169]

In our topology we configured the two ABRs to serve as the DR and BDR. This was accomplished by setting a nonzero OSPF Router Priority value on each of the ABRs for each VLAN, while setting a Router Priority value of zero on each of the IBM System z servers for those same multi-access network connections.

It should be pointed out that there is one case that required special consideration with respect to our DR and BDR selection in our environment. 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, an IBM System z node must be elected as an OSPF DR on a HiperSockets LAN. [CommServer-Redbook, page 170] We selected the z/OS nodes as the DR and BDR to satisfy this OSPF requirement.

4.3.4 OSPF ABR to ABR interconnectivity consideration

OSPF advertises the state of links and not actual routing table entries (EIGRP is an example of a routing protocol that advertises routing table entries). Therefore, implicit in this design is the requirement to limit OSPF processing on the Linux for IBM System z and z/OS servers.

While it is beyond the scope of this paper to fully explore the ABR configurations in detail, it is very important to point out that our ABRs were intentionally interconnected with links configured in both the OSPF Backbone and OSPF stub areas to prevent the ABRs from ever advertising sub-optimal routes through the stub area during certain types of router/switch failures. *Figure 1: Network* illustrates this configuration point.

4.4 Setup and baseline configuration points

Two of our three z/OS servers resided on the same IBM System z9-2094 CPC, while the third z/OS server, along with all three of our Linux for IBM System z servers resided on the same IBM System z10-2097 CPC. A HiperSockets interface was configured on each CPC (z9 and z10™). Each of these HiperSockets were included as OSPF interfaces in all of the respective z/OS and Linux for IBM System z OSPF configurations to provide an additional level of network availability. All of our servers, z/OS and Linux for IBM System z, were also

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

configured with a Static VIPA, which is the same IP address that should be leveraged in an installation's Domain Name Server strategy.

4.4.1 Baseline OSA discussion

Each z/OS server was configured to use two distinct OSA features. Each OSA feature belonged to separate subnets, each of which was defined in different networking switches configured as OSPF ABRs. This was intended to provide a fully meshed, fully redundant networking infrastructure to safeguard against potential router/switch, and power failures.

The Linux for IBM System z servers utilized VSWITCH in Layer 2 (ETHERNET) mode with two OSAs in an active/standby (NOGROUP) configuration, each of which belonged to the same subnet.

4.4.2 Baseline OSPF discussion

Our controlled test environment consisted of three z/OS and three Linux for IBM System z servers, with each server configured to run the OSPF routing protocol in a common OSPF stub area. Two interconnected IP routers were deployed as the OSPF Area-Border Routers (ABRs).

4.4.3 Linux for IBM System z - Zebra and OSPF configuration

The Zebra.conf file contains the network interface and IP address definitions that are utilized by the Quagga (OSPF) stack. While including the physical interfaces and their respective IP addresses may seem intuitive, we would like to point out that the Dummy0:0 interface statement must also be included in the Zebra.conf file. We would also like to point out that both the Dummy0:0 interface and the Service IP address (for example, the "floating VIPA") must be included in the ospfd.conf conf file for each Linux for IBM System z Server that TSA can potentially move the floating VIPA to. This is done to ensure that the floating VIPA will dynamically be included in the OSPF LSA advertisements when TSA moves the floating VIPA to the respective server.

For example, the following Zebra.conf and Ospfd.conf excerpts illustrate these configuration relationships.

```
zebra.conf:
! Server - Static VIPA
interface dummy0
ip address 172.31.0.34/32
ipv6 nd suppress-ra

!
interface dummy0:0
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```
ipv6 nd suppress-ra
!

ospfd.conf:
!Server - Static VIPA
interface dummy0
ip ospf cost 1
ip ospf priority 0
!
! Service IP - Floating VIPA
interface dummy0:0
ip ospf cost 1
ip ospf priority 0
!
router ospf
network 172.31.0.34/32 area 2.2.2.2
network 172.31.200.1/24 area 2.2.2.2
area 2.2.2.2 stub
!
```

4.4.4 Static VIPA, floating VIPA discussion

As previously mentioned in section 2.4.1, a Static VIPA is associated with a server's TCP/IP stack. By including the Static VIPA in the respective server's OSPF configuration, availability to a multi-home server is increased because connectivity to the server, with the Static VIPA, is not limited to a single network interface; rather connectivity is extended to the server across all of the physically attached network interfaces.

On our Linux systems this static VIPA served several purposes. First, it provided a unique, highly available IP address, which was defined in our DNS strategy in order to provide assured access to the host.

Also, because a Static VIPA uniquely identifies a server and every OSPF router must have a unique router-id, we used each server's static VIPA to satisfy the OSPF router-id requirement on each of our Linux for IBM System z and z/OS servers.

Clearly the most interesting use of the Static VIPA within the scope of this paper is that the Static VIPA provided a base address for the dummy0 device, upon which we could then use TSA to create an alias representing our floating VIPA on any one of our systems in the cluster.

The floating VIPA was an address on the same subnet as the static VIPA, and benefited from being included in the same OSPF route advertisements as the static VIPA. It provided the

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

means to access the application for which we were attempting to provide high availability (in our case, a Web server). When the host of this Web server failed or became otherwise inaccessible, TSA could move the floating VIPA to another system in the cluster by re-aliasing it to the dummy0 device on that system. When that floating VIPA was reestablished, TSA could then restart the Web server on this new system, enabling users to access it over the same IP address that they had been using.

4.4.5 Baseline Key metrics discussion: The value that OSPF brings to the table

In a statically routed environment, typically one default route is manually defined. The server will lose vital connectivity if a failure occurs anywhere along this default route (that is: network interface, LAN wire/fiber, switch port). We avoided this situation by deploying OSPF.

By design, each OSPF ABRs is responsible for injecting a default route into the OSPF stub area that points to itself. Therefore, we begin our walk through analysis by examining the default routes that were injected by the OSPF ABRs into the stub areas.

From the routing tables captured on each of the z/OS hosts we see that there are four default routes. This is reasonable because these hosts have two separate paths, each in different networks to each of the two OSPF ABRs.

Default	192.168.71.4	UGO	000000	OSA920
Default	192.168.71.5	UGO	000000	OSA920
Default	192.168.74.4	UGO	000000	OSA930
Default	192.168.74.5	UGO	000000	OSA930

Similarly, the following Linux for IBM System z Servers also received a default route from each of the two ABRs. Note that these servers had connectivity only into one Ethernet network, therefore it is unreasonable to expect them to also receive four default routes, as was the case for the z/OS nodes. However, it is important to also point out that the two default routes that are defined, in themselves, have already provided us with a higher level of availability than would a statically routed deployment.

For example, in a statically routed environment, only one static route would be active at a time. Therefore, if we lose connectivity to the router that the active static route references, we lose the availability to respond to traffic outside the immediate broadcast domain. OSPF removes this availability exposure by dynamically maintaining the active default routes for the entire stub area.

```
N IA 0.0.0.0/0 [9] area: 2.2.2.2
                via 192.168.71.5, eth1 ← ABR #1
                via 192.168.71.4, eth1 ← ABR #2
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Note: IA indicates that this default route (0.0.0.0/0) is an OSPF Intra Area route, as opposed to an Inter Area route. OSPF, by design, will select Intra Area routes over Inter Area routes whenever available.

Because the floating VIPA represents our solution's key service, the rest of this discussion will focus on network connectivity to the floating VIPA. We start by pointing out that we explicitly assigned the TSA Floating Service IP address (172.31.200.1) to the dummy0:0 device on the LITOSPF1 Linux server. The routing capture below illustrates that this floating VIPA is directly associated with (aliased to) the respective server's dummy0 device.

```
N      172. 31. 200. 1/32      [1] area: 2. 2. 2. 2
                                   directly attached to dummy0
```

Notice how dummy0 is represented and not the dummy0:0.

The next entry of interest is in the z/OS routing tables. The following entry, captured on a z/OS host that does not reside on the same CPC as any of the Linux for IBM System z servers, shows us that the host at IP address (192.168.71.34), which is LITOSPF1, is the 'shortest path' (next hop) to reach the floating service IP address(172.31.200.1/32). This route indicates that the floating VIPA is accessible only across the physical network, which is reasonable because a lower cost HiperSockets path is not a viable route to the floating VIPA for this particular example. Rather, the shortest path in this instance is in fact across the physical network, through an OSA interface (OSA920)

```
172. 31. 200. 1/32      192. 168. 71. 34      UGH0      000000  OSA920
```

Next, we examine the routing table entries for the only z/OS host (Z2) that resides on the same CPC as the LITOSPF1 server. From this entry we see that OSPF has determined the shortest path to the floating VIPA is using a HiperSockets interface (Z2HIP40K), just as we would expect based on our OSPF interface costing strategy presented earlier (re: 4.3.1 OSPF Costing)

```
172. 31. 200. 1/32      172. 18. 91. 34      UGH0      000000  Z2HIP40K
```

The next few examples were taken from both of the other two Linux servers, which correctly illustrate that the floating VIPA (172.31.200.1) resides on the 172.18.91.34 host (LITOSPF1) and that the HiperSockets path (hsi1) is the shortest (least costly) path to that host for both of these servers. It is important to point out that a routing table provides us with the "next hop", or the next host that the packet will be directed to in order to reach its final destination and not necessarily indicate the final host. Of course, in this particular case, we know that the next hop and the final destination are one and the same.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```
litospf2: sho ip ospf route
      N    172.31.200.1/32      [5] area: 2.2.2.2
                                   via 172.18.91.34, hsi1
```

```
litospf3: sho ip ospf route
      N    172.31.200.1/32      [5] area: 2.2.2.2
                                   via 172.18.91.34, hsi1
```

The following routing table entry was taken from one of the OSPF ABRs, which also reflects that the floating VIPA is accessible via LITOSPF1. This specific ABR's only path to the LITOSPF1 server is through the physical Ethernet lan (Vlan 71).

```
0      172.31.200.1 [110/2] via 192.168.71.34, 00:00:56, Vlan71
```

Next, we will illustrate how utilizing a floating VIPA increases availability. We begin with the floating VIPA aliased to LITOSPF1's dummy0:0 device. Due to the fact that all three of our Linux servers were on the same CPC, we need to analyze only the routing table captured from one of our Linux for IBM System z guests.

The following routing table excerpt was extracted from the LITOSPF2 guest and is presented to illustrate a few key points.

- First, notice how dual default gateways are injected into the Linux for IBM System z guest's routing table by the two OSPF ABRs.
- Next, notice how LITOSPF2 has learned that the best route to the floating VIPA, as well as the best route to all of the other Linux for IBM System z guests is via the HiperSockets connection.
- Furthermore, we see how OSPF has properly advertised that the best route to the z/OS server (Z2) residing on the same CPC is via the HiperSockets network, even though a route through a directly attached physical network (192.168.71.0/24) is available.

```
litospf2.ltic.pok.ibm.com# sho ip ospf route
```

```
===== OSPF network routing table =====
N IA 0.0.0.0/0      [9] area: 2.2.2.2
                                   via 192.168.71.5, eth1 1-
                                   via 192.168.71.4, eth1 1-
N    172.18.0.0/16  [4] area: 2.2.2.2
                                   directly attached to hsi1 2-
N IA 172.20.110.0/24 [9] area: 2.2.2.2
                                   via 192.168.71.5, eth1 3-
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```

N    172.31.0.34/32    [5] area: 2.2.2.2    4-
                        via 172.18.91.34, hsi1
N    172.31.0.35/32    [1] area: 2.2.2.2    5-
                        directly attached to dummy0
N    172.31.0.36/32    [5] area: 2.2.2.2    6-
                        via 172.18.91.36, hsi1
N    172.31.0.39/32    [9] area: 2.2.2.2    7-
                        via 192.168.71.39, eth1
N    172.31.0.44/32    [5] area: 2.2.2.2    8-
                        via 172.18.91.44, hsi1
N    172.31.0.45/32    [9] area: 2.2.2.2    9-
                        via 192.168.71.45, eth1
N    172.31.200.1/32   [5] area: 2.2.2.2    10-
                        via 172.18.91.34, hsi1
N    192.168.71.0/24   [8] area: 2.2.2.2    11-
                        directly attached to eth1
N    192.168.74.0/24   [9] area: 2.2.2.2    12-
                        via 192.168.71.4, eth1
                        via 192.168.71.5, eth1
N IA 192.168.75.0/24   [9] area: 2.2.2.2    13-
                        via 192.168.71.4, eth1
===== OSPF router routing table =====
R    100.1.1.3         [8] area: 2.2.2.2, ABR 14-
                        via 192.168.71.5, eth
      100.1.1.4         [8] area: 2.2.2.2, ABR 15-
                        via 192.168.71.4, eth1
===== OSPF external routing table =====

```

Routing table notes:

1. Dual default gateways are injected by redundant ABRs
2. Network route, HiperSockets- directly attached
3. Network route-accessible through ABR #2
4. Host route to LITOSPF1 VIPA through HiperSockets
5. Host route to its own VIPA-directly attached
6. Host route to LITOSPF3 VIPA through HiperSockets
7. Host Route to z/OS Z1 VIPA on different CPC through eth1
8. Route to Host z/OS Z2 on same CPC through HiperSockets
9. Route to Host z/OS Z3 on different CPC through eth1
10. Route to Service IP on LITOSPF1 through HiperSockets
11. Network route – directly attached
12. Network route - accessible through both ABRs
13. Network route –accessible through ABR #1

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

- 14. ABR #2
- 15. ABR #1

The next routing table excerpt, extracted from the same LITOSPF2 Linux for IBM System z guest, demonstrates the connectivity resulting when one of the ABRs fails. This example correctly reflects that not only is one default gateway in effect, but all of the previous routes to the z/OS and Linux for IBM System z peers all remained intact.

```
LITOSPF2 routing tables
litospf2.ltic.pok.ibm.com# sho ip ospf rou
===== OSPF network routing table =====
N IA 0.0.0.0/0          [3] area: 2.2.2.2      1-
                        via 192.168.71.5, eth1
N   172.18.0.0/16      [1] area: 2.2.2.2      2-
                        Directly attached to hsi1
N IA 172.20.110.0/24   [3] area: 2.2.2.2      3-
                        via 192.168.71.5, eth1
N   172.31.0.34/32     [2] area: 2.2.2.2      4-
                        via 172.18.91.34, hsi1
N   172.31.0.35/32     [1] area: 2.2.2.2      5-
                        directly attached to dummy0
N   172.31.0.36/32     [2] area: 2.2.2.2      6-
                        via 172.18.91.36, hsi1
N   172.31.0.39/32     [3] area: 2.2.2.2      7-
                        via 192.168.71.39, eth1
N   172.31.0.44/32     [2] area: 2.2.2.2      8-
                        via 172.18.91.44, hsi1
N   172.31.0.45/32     [3] area: 2.2.2.2      9-
                        via 192.168.71.45, eth1
N   172.31.200.1/32    [2] area: 2.2.2.2     10-
                        via 172.18.91.34, hsi1
N   192.168.71.0/24    [2] area: 2.2.2.2     11-
                        directly attached to eth1
N   192.168.74.0/24    [3] area: 2.2.2.2     12-
                        via 172.18.91.44, hsi1 13-
                        via 192.168.71.5, eth1 14-
===== OSPF router routing table =====
R   100.1.1.3          [2] area: 2.2.2.2, ABR
                        via 192.168.71.5, eth1
===== OSPF external routing table =====
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Routing table notes:

1. One Default route injected by the surviving ABR #2
2. Network route- HiperSockets subnet
3. Network route still accessible through ABR #2
4. Route to Host LITOSPF1 VIPA through HiperSockets
5. Route to Host LITOSPF2 VIPA through HiperSockets
6. Host Route to LITOSPF3 VIPA through HiperSockets
7. Route to Host z/OS Z1 on different CPC through eth1
8. Route to Host z/OS Z2 on same CPC through HiperSockets
9. Route to Host z/OS Z3 on different CPC through eth1
10. Route to Service IP on LITOSPF1 through HiperSockets
11. Network route – directly attached
12. Equal cost Network routes
13. Accessible through z/OS Z2, same CPC HiperSockets
14. Accessible through ABR #2

The next series of routing table examples are presented to systematically illustrate the baseline connectivity prior to inducing any of the cascading failures. These baseline routing table excerpts show that the floating VIPA is accessible through the HiperSockets network for the three Linux for System z guests and two z/OS images, all of which reside on the System z10-EC, but the two z/OS images residing on the System z9-EC will need to traverse the physical network to reach the floating VIPA, as expected.

```
LITOSPF1:
           N    172. 31. 200. 1/32    [1] area: 2. 2. 2. 2
                                           directly attached to dummy0

LITOSPF2:
           N    172. 31. 200. 1/32    [2] area: 2. 2. 2. 2
                                           via 172. 18. 91. 34, hsi1

LITOSPF3:
           N    172. 31. 200. 1/32    [2] area: 2. 2. 2. 2
                                           via 172. 18. 91. 34, hsi1

Z2:
    172. 31. 200. 1/32    172. 18. 91. 34    UGHO    000000    Z2HIP40K

Z1:
    172. 31. 200. 1/32    192. 168. 71. 34    UGHO    000000    OSA920

Z3:
    172. 31. 200. 1/32    192. 168. 71. 34    UGHO    0000000000    OSA920

ABR:
    172. 31. 200. 1 [110/2] via 192. 168. 71. 34, 00:17:00, Vlan71
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

4.5 Failover-resiliency scenarios

In this section we walk through a series of cascading failures, each of which are designed to illustrate our solution's key resiliency and availability attributes. Here is a summary of the starting state of routes to our floating VIPA, which initially resided on LITOSPF1.

OSPF router	Destination	Gateway	Interface	Comment
LITOSPF1	172.31.200.1/32	directly attached	dummy0	directly attached
LITOSPF2	172.31.200.1/32	172.18.91.34	hsi1	via HiperSockets
LITOSPF3	172.31.200.1/32	172.18.91.34	hsi1	via HiperSockets
Z1	172.31.200.1/32	192.168.71.34	OSA920	via OSA
Z2	172.31.200.1/32	172.18.91.34	Z2HIP40K	via HiperSockets
Z3	172.31.200.1/32	192.168.71.34	OSA920	via OSA
ABR	172.31.200.1	via 192.168.71.34	vlan71	

4.5.1 First Failure: Loss of the HiperSockets connection

For this scenario we caused a failure in the HiperSockets connection on the same CPC where the three Linux Servers reside, and examine only the key portions of the routing tables on each node to illustrate how the floating VIPA is still available and accessible. The resulting routes are listed in the following table.

OSPF router	Destination	Gateway	Interface	Comment
LITOSPF1	172.31.200.1/32	directly attached	dummy0	no change
LITOSPF2	172.31.200.1/32	192.168.71.34	eth1	now via eth1 (OSA)
LITOSPF3	172.31.200.1/32	192.168.71.34	eth1	now via eth1 (OSA)
Z1	172.31.200.1/32	192.168.71.34	OSA920	no change
Z2	172.31.200.1/32	192.168.71.34	PVT71NW	now via OSA
Z3	172.31.200.1/32	192.168.71.34	OSA920	no change
ABR	172.31.200.1	via 192.168.71.34	vlan71	no change

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

On LITOSPF1, we see that the VIPA is still aliased to its dummy0 device, as expected. On LITOSPF2 and LITOSPF3 we now see that the floating VIPA is accessible through Ethernet (eth1), as expected. The ABR does not see any change, as expected.

As expected, both Z1 and Z3 don't see any change to their routing table, either before, or after the HiperSockets failure. In both cases, their respective routing tables reflect that the floating VIPA is accessible using their Ethernet interface (OSA)

Prior to the HiperSockets failure, the next hop from Z2 to the floating VIPA was to the LITOSPF1 host through 172.18.91.34, which represents the HiperSockets side of the connection on LITOSPF1. But now the next hop (which is still to LITOSPF1) for that route is to LITOSPF1's Ethernet connection (192.168.71.34).

The conclusion is that Z2's routing table was properly updated to reflect that the floating VIPA was still available, and that the next hop to it was still to the same host as it was before the HiperSockets failure, demonstrating that Z2 never lost availability to the floating VIPA.

4.5.2 – Second Failure: Loss of the Ethernet device

The next failure in our cascading sequence involved causing a failure in the last functional network interface on LITOSPF1, which is still where the floating VIPA resides. This scenario is intended to show that TSA correctly moves the VIPA when the server that is currently hosting it may still be up and running, but TSA recognizes that the server no longer has any networking connectivity.

During this scenario, the ABR indicated that the OSPF dead timer had lapsed for one of its OSPF neighbors, as expected.

```
OSPF-5-ADJCHG: Process 100, Nbr 172.31.0.34 on Vlan71 from FULL to DOWN, Neighbor Down: Dead timer expired
```

Observations:

As the table below shows, LITOSPF1 has no routing table because it no longer has any active network interfaces. We also see that LITOSPF2 is now advertising that the floating VIPA has been aliased to its dummy0 device by TSA.

When we look at LITOSPF3, we see that the floating VIPA is accessible using the HiperSockets, as expected, because the previous scenario did not disable the HiperSockets path across the entire CPC, rather it only failed on the LITOSPF1 guest.

The same is true for the Z2 z/OS image, in that the shortest path to the floating VIPA is once again across the HiperSockets network. Also, when we examine the routing tables on both the

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Z1 and Z3 z/OS images, we see that they both recognize that the floating VIPA is now accessible through the LITOSPF2 host (192.168.71.35), and they will still traverse the network to reach that VIPA, as expected.

The final entry in the table below shows that the ABR's routing table correctly reflects that the floating VIPA was moved to the LITOSPF2 guest as well.

OSPF router	Destination	Gateway	Interface	Comment
LITOSPF1	172.31.200.1/32	none	n/a	remained up, but no NW connectivity
LITOSPF2	172.31.200.1/32	directly attached	dummy0	
LITOSPF3	172.31.200.1/32	172.18.91.35	hsi1	back to using HiperSockets
Z1	172.31.200.1/32	192.168.71.35	OSA920	through network
Z2	172.31.200.1/32	172.18.91.35	Z2HIP40K	back to using HiperSockets
Z3	172.31.200.1/32	192.168.71.35	OSA920	through network
ABR	172.31.200.1	via 192.168.71.35	vlan71	as expected

4.5.3 – Third Failure: Loss of the HiperSockets again

It is not a good day for a HiperSockets device in our test environment. This scenario caused a failure in the HiperSockets adapter on the server currently hosting the floating VIPA, LITOSPF2. For this scenario, prior to inducing a failure in the HiperSockets adapter on the LITOSPF2 guest, we had to reinstate all connectivity to the LITOSPF1 guest in order to maintain the TSA quorum.

Once again, all of the expected route convergence was experienced from each server in our test environment, as the table below illustrates.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

OSPF router	Destination	Gateway	Interface	Comment
LITOSPF1	172.31.200.1/32	192.168.71.35	eth1	via network, as expected
LITOSPF2	172.31.200.1/32	directly attached	dummy0	no change
LITOSPF3	172.31.200.1/32	192.168.71.35	eth1	via network, as expected
Z1	172.31.200.1/32	192.168.71.35	OSA920	via network, as expected
Z2	172.31.200.1/32	192.168.71.35	PVT71N W	changed to go over the network, as expected
Z3	172.31.200.1/32	192.168.71.35	OSA920	via network
ABR	172.31.200.1	via 192.168.71.35	vlan71	no change, expected

4.5.4 – Fourth Failure: Hard failure of the server hosting the floating VIPA

For this scenario, the Linux Server hosting the floating VIPA, LITOSPF2, was caused to fail to demonstrate how TSA will move the floating VIPA and how all of the OSPF routers will converge on that movement.

TSA moved the floating VIPA to the LITOSPF1 guest and once again, all of the expected route convergence was experienced from each server in our test environment, as the table below illustrates.

OSPF router	Destination	Gateway	Interface	Comment
LITOSPF1	172.31.200.1/32	directly attached	dummy0	directly attached
LITOSPF2	n/a	n/a	n/a	Server is down
LITOSPF3	172.31.200.1/32	172.18.91.34	hsi1	via HiperSockets, as expected
Z1	172.31.200.1/32	192.168.71.34	OSA920	via network, as expected
Z2	172.31.200.1/32	172.18.91.34	Z2HIP40K	via HiperSockets, as expected
Z3	172.31.200.1/32	192.168.71.34	OSA920	via network, as expected
ABR	172.31.200.1	via 192.168.71.34	vlan71	as expected

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

4.5.5 Fifth Scenario: Server recovery

The recovery scenario does not present any surprises, in that the floating VIPA should not move. In this scenario the failed server, LITOSPF2, is recovered and, as expected, all of the routing tables have correctly converged on the floating VIPA residing on LITOSPF1, as the following table demonstrates.

OSPF router	Destination	Gateway	Interface	Comment
LITOSPF1	172.31.200.1/32	directly attached	dummy0	directly attached, as expected
LITOSPF2	172.31.200.1/32	172.18.91.34	hsi1	via HiperSockets, as expected
LITOSPF3	172.31.200.1/32	172.18.91.34	hsi1	via HiperSockets, as expected
Z1	172.31.200.1/32	192.168.71.34	OSA920	via network, as expected
Z2	172.31.200.1/32	172.18.91.34	Z2HIP40K	via HiperSockets, as expected
Z3	172.31.200.1/32	192.168.71.34	OSA920	via network, as expected
ABR	172.31.200.1	via 192.168.71.34	vlan71	expected

4.6 Summary of observations (CPU and memory intensiveness)

A commonly held concern when running any piece of software on a virtual execution environment is how much resource will be consumed individually on a per node basis. Another often overlooked concern is the cumulative effects of executing that code on multiple virtual nodes concurrently. There has been some speculation that using OSPF would incur costly overheads and performance penalties.

A recommended way to use dynamic routing in the Linux for IBM System z environment is to define an OSPF stub area, or even better, define an OSPF totally stubby area. Stub areas minimize storage and CPU processing at the nodes that are part of the stub area because they maintain less knowledge about the topology of the Autonomous System (AS) than do other types of non-backbone routers. They only maintain knowledge of intra-area destinations and summaries of inter-area destinations and default routes within the AS in order to reach external destinations. A totally stubby area receives less routing information than a stub area. A totally stubby only knows of intra-area destinations and default routes within the stub area to reach external destinations.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

It is our recommendation that you make the external routers be DR or BDR, and do not allow z/OS systems to be DR or BDR, in order to reduce the routing burden for z/OS systems.

In our empirical observation for a small set of systems deployed in this experiment, overhead was not measurable during normal operation, while overhead approached a few percent during active state exchanges. The amount of routing state changes can be minimized by constructing your network similar to what is shown in Figure 1 presented earlier in this document. However, we did not attempt to perform rigorous, repeatable performance measurements, nor did we explore a many-virtual-server configuration to assess the cumulative OSPF overhead associated with that (more realistic) environment.

5. Discussion and future work

As with any investigation we have not been able to pursue all of the interesting questions that have been asked. We will list a few of the more interesting items here that we hope to revisit when time permits, with the active encouragement of other interested parties who may want to investigate these questions. Specifically we are interested in examining the further nuances of equal cost multi-pathing, and examining the worst case performance of OSPF on strategic network topologies designed to minimize OSPF overhead.

6. Conclusion

We have demonstrated various levels of high availability suitable for cloud computing and virtual network infrastructure architectures. In addition, we feel the solutions we have described in this paper are generally applicable to other types of networking deployments. We have demonstrated that a high level of resilience can be obtained by combining floating VIPA and dynamic routing. We have further shown these techniques to be straightforward to implement, and have empirically observed low overhead in a very simple, sparse virtual server configuration.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Appendix

References

[VLAN1]

IBM Infocenter: Linux information > System z > Linux on System z performance > End-to-End Performance of a WebSphere® Environment Including Edge Components > System setup for the WebSphere environment performance tests > z/VM Guest LAN type setup
<http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaag/tuningWAS/lanset.htm>

[VLAN2]

Linux on IBM eServer™ zSeries® and S/390®: TCP/IP Broadcast on z/VM Guest LAN, Page 2 REDP-3596-00
<http://www.redbooks.ibm.com/redpapers/pdfs/redp3596.pdf>

[Vswitch1]

IBM Form Number REDP-3719-00, Linux on IBM eServer zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4 <http://www.redbooks.ibm.com/abstracts/REDP3719.html?Open>

[LinkAgg1]

z/VM Virtual Switch Link Aggregation
<http://www.vm.ibm.com/virtualnetwork/linkag.html>

[Bond1]

<http://www.linuxfoundation.org/en/Net:Bonding>

[Bond 2]

<http://linux-ip.net/html/ether-bonding.html>

[ComSrv]

z/OS V1R11.0 Communications Server IP Configuration Guide (SC31-8775-15),
<http://www-03.ibm.com/systems/z/os/zos/bkserv/r11pdf/#commserv>

[DeviceDrivers]

Device Drivers, Features, and Commands (kernel 2.6.31) - SC33-8411-03 , Page 131,
<http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/docu/lk31dd03.pdf>

[SysConHandbook]

IBM System z Connectivity Handbook, IBM Form Number SG24-5444-10
<http://www.redbooks.ibm.com/abstracts/sg245444.html>

[HipersocketsImp]

HiperSockets Implementation Guide, IBM Form Number SG24-6816-01,
<http://www.redbooks.ibm.com/abstracts/sg246816.html>

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

[VipaHA]

High Availability Architectures for Linux on IBM System z, Page 29, IBM Document ID: WP100752

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100752>

[VswitchRedundancy]

IBM Form Number REDP-3719-00, Page 21, Linux on IBM eServer zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4

<http://www.redbooks.ibm.com/abstracts/REDP3719.html?Open>

[ServiceIP]

High Availability Architectures for Linux on IBM System z, Page 10, IBM Document ID: WP100752

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100752>

[TestRepVswitchContr]

z/OS V1R8.0 System z Platform Test Report for z/OS and Linux Virtual Servers, June 2007, Page 172

<http://publibz.boulder.ibm.com/epubs/pdf/e0z1p150.pdf>

[TSA]

<http://www-01.ibm.com/software/tivoli/products/sys-auto-multi/>

[LinuxHA-org]

<http://linux-ha.org>

[CommServer-Redbook, page nnn]

Communications Server for z/OS V1R9 TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing

<http://www.redbooks.ibm.com/abstracts/sg247532.html>

[CS-HA-Redbook]

ibm.com/redbooks: CS for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance SG24-7534-00

<http://www.redbooks.ibm.com/abstracts/sg247534.html>

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Reference configurations

Reference z/OS OSPF configuration – complete profile

The following reference configuration was extracted from one of our three z/OS images. The OSPF configurations from the other two z/OS servers are identical to this one, except for the IP address assignments, which must be unique per interface, and the use of different interface names.

```
-----  
;  
;  
; Static VIPA being used as the RouterId  
;  
RouterID = 172.31.0.44  
;  
-----  
;  
Area  
  Area_Number      = 2.2.2.2  
  Authentication_Type = None  
  Stub_Area        = YES  
  Import_Summaries  = YES  
;  
Comparison=2;  
;  
GLOBAL_OPTIONS Ignore_Undefined_Interfaces=YES  
;  
-----  
;  
; Purpose: z/OS - Linux for System z Joint OSPF work  
;  
OSPF_Interface  
  Name              = PVT71NW  
  IP_Address        = 192.168.71.44  
  Subnet_Mask       = 255.255.255.0  
  MTU               = 1492  
  Router_Priority   = 0  
  Cost0             = 2  
  Attaches_To_Area = 2.2.2.2  
;  
; Purpose: z/OS - Linux for System z Joint OSPF work  
;  
;  
;
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```
OSPF_Interface
  Name                = PVT74NW
  IP_Address          = 192.168.74.44
  Subnet_Mask         = 255.255.255.0
  MTU                 = 1492
  Router_Priority     = 0
  Cost0               = 2
  Attaches_To_Area   = 2.2.2.2
;
;-----
;
;
;  STATIC VIPA
;
OSPF_Interface
  IP_Address          = 172.31.0.44
  Name                = Z2SVIPA
  Subnet_Mask         = 255.255.0.0
  MTU                 = 65535
  Router_Priority     = 0
  Cost0               = 1
  Advertise_VIPA_Routes = HOST_ONLY
  Attaches_To_Area   = 2.2.2.2;
;
;-----
;  Hipersocket with 40K packet, 32K MTU
;
OSPF_Interface
  IP_Address          = 172.18.91.44
  NAME                = Z2HIP40K
  Subnet_Mask         = 255.255.0.0
  MTU                 = 32768
  Cost                = 1
  Router_Priority     = 20
  Attaches_To_Area   = 2.2.2.2;
;
;-----
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Reference z/OS TCP/IP (partial) profile – only key configuration statements

```
-----  
; Static VIPA for Z2  
;  
DEVICE Z2SVDEV    VIRT  0  
LINK  Z2SVIPA    VIRT  0 Z2SVDEV  
;  
-----  
; HIPERSOCKET  
-----  
DEVICE IUTIQDE2  MPCIPA  
LINK  Z2HIP40K  IPAQIDIO IUTIQDE2  
-----  
;  
; Purpose: z/OS - Linux for System z Joint OSPF Work  
; Pchid:221  
; Devices:920-92F  
; NW:192.168.71/24  
; TRLE:PET71NET  
;  
INTERFace PVT71NW  
  DEFINE IPAQENET  
  PORTNAME THUMP920  
  IPADDR 192.168.71.44/24  
  MTU 1492  
  NONROUTER  
  VLANID 671  
  VMAC  
;  
-----  
; Purpose: z/OS - Linux for System z Joint OSPF Work  
; Pchid:5B0  
; Devices:930-93F  
; NW:192.168.74/24  
; TRLE:PET74NET  
;  
INTERFace PVT74NW  
  DEFINE IPAQENET  
  PORTNAME THUMP930  
  IPADDR 192.168.74.44/24  
  MTU 1492
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```

NONROUTER
VLANID 674
VMAC
;
;
-----
HOME
172. 31. 0. 44      Z2SVIPA ; Z2 Static VIPA link
172. 18. 91. 44    Z2HIP40K ; 40K Packet, 32K MTU Hipersocket
;
BEGINRoutes
;
-----
;
;          SUBNET   FIRST
;   DESTINATION   MASK   HOP           LINK   PACKET SIZE
;-----
ROUTE 192. 168. 71. 0/24          =   PVT71NW      MTU 1492 REPL
ROUTE DEFAULT      192. 168. 71. 5   PVT71NW      MTU 1492 REPL
;-----
ROUTE 192. 168. 74. 0/24          =   PVT74NW      MTU 1492 REPL
ROUTE DEFAULT      192. 168. 74. 5   PVT74NW      MTU 1492 REPL
;-----
;
;
ENDROUTES
;
START IUTIQDE2 ; 40K packet, 32K MTU Hipersocket
START PVT71NW
START PVT74NW

```

Linux for IBM System z reference configuration for one Linux for IBM System z guest (Zebra.conf)

```

!
hostname litospf1.#####.#####
password #####
enable password #####
log file /var/log/quagga/quagga.log
!
! Static VIPA
interface dummy0

```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```
ip address 172.31.0.34/32
ipv6 nd suppress-ra
!
interface dummy0:0
ipv6 nd suppress-ra
!
interface eth0
ipv6 nd suppress-ra
!
interface eth1
ip address 192.168.71.34/24
ipv6 nd suppress-ra
!
! Hipersocket - 40K packet, 32K MTU
interface hsi1
ip address 172.18.91.34/16
ipv6 nd suppress-ra
!
interface lo
!
interface sit0
ipv6 nd suppress-ra
!
ip forwarding
!
line vty
exec-timeout 0 0
!
```

Linux for IBM System z reference configuration for one Linux for IBM System z guest (Ospfd.conf)

```
hostname litospf1.#####.#####
password #####
enable password #####
log stdout
!
! Server - Static VIPA
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

```
interface dummy0
 ip ospf cost 1
 ip ospf priority 0
 !
 ! Service IP - Floating VIPA
 interface dummy0:0
 ip ospf cost 1
 ip ospf priority 0
 !
 ! interface eth0
 ! ip ospf cost 2
 ! ip ospf priority 0
 !
 interface eth1
 ip ospf cost 2
 ip ospf priority 0
 !
 interface hsi1
 ip ospf cost 1
 ip ospf priority 10
 !
 interface lo
 !
 interface sit0
 !
 router ospf
  ospf router-id 172.31.0.34
  network 172.18.0.0/16 area 2.2.2.2
  network 172.31.0.34/32 area 2.2.2.2
  network 172.31.200.1/24 area 2.2.2.2
  network 192.168.71.0/24 area 2.2.2.2
  area 2.2.2.2 stub
 !
 line vty
  exec-timeout 0 0
 !
```

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF

Hints and Tips

This section describes tips gleaned from performing the configuration and testing described in this paper.

Quagga OSPF MTU interoperability

In order to form an OSPF adjacency, the MTU must match on neighboring interfaces. Quagga does not provide a mechanism for setting the MTU within the OSPF stack. Rather, Quagga inherits the OSPF MTU from the MTU set at the NIC level. This implementation imposes an additional level of planning in that MTU changes made at the NIC level will effect every network connection on the Linux server. Therefore, it may be necessary to determine an optimal MTU across the entire stub area, and then set the MTU on the ABRs to match the MTU set on the Linux servers.

Ifconfig considerations for a Zebra interface

Configuring an interface down (for example, `ifconfig eth1 down`) from outside of Zebra has the same effect as “shutting” the interface down from within Zebra’s command shell. However, if `ifconfig` is used to shut down an interface, displaying Zebra’s running configuration from within the Zebra shell (for example, by issuing the “`show running config`” command) will not reflect that the interface was shutdown.

Perhaps more importantly, a running Zebra service does not detect when an interface, such as a HiperSockets interface, is dynamically attached or reattached while Zebra is running. The Zebra process must be stopped and then restarted in order for both the Zebra and Quagga processes to recognize and utilize the newly attached interface. The implications of this requirement include the need to factor this additional step into recovery procedures for failed network interfaces.

A Reference Implementation Architecture for Deploying a Highly-Available Networking Infrastructure for Cloud Computing and Virtual Environments using OSPF



Copyright IBM Corporation 2009
IBM Systems and Technology Group
Route 100
Somers, New York 10589
U.S.A.

Produced in the United States of America,
12/2009
All Rights Reserved

IBM, IBM eServer, IBM logo, HiperSockets, Redbooks, S/390, System z, System z9, System z10, Tivoli, WebSphere, z9, z10, z10 EC, z/OS, zSeries and z/VM are trademarks or registered trademarks of the International Business Machines Corporation.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

InfiniBand and InfiniBand Trade Association are registered trademarks of the InfiniBand Trade Association. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.