Friday, January 22. 2010

Less known Solaris features - IP Multipathing (Part 8): Classic IPMP

IPMP itself is a really old feature. It's in Solaris for several versions now. Just the implementation I've described before is a new one. But in Solaris 10 you don't have this new IPMP implementation. Solaris 10 still uses the old implementation. I will call the old implementation classic IPMP. The basic mechanism of new and classic IPMP is pretty much the same: Providing failure detection mechanisms and switch something to do a failover thus the data address stays available. But internally it's a completely different implementation. While the new mechanism is certainly the future of IPMP, I' m pretty sure you will old mechanism more often in the wild.

PrerequisitesThis example works with the same configuration, but you need a system with Solaris 10 or an Opensolaris System with a build earlier than 107. I just use Opensolaris on my lab machines, thus i used an virtualized Solaris 10 to explain the configuration of classic IPMP.

I will use the following addresses:

192.168.178.200 vhivemind-prod
192.168.178.201 vhivemind-e1000g0
192.168.178.202 vhivemind-e1000g1

I will demonstrate this on a recent release of Solaris 10:

```
bash-3.00# cat /etc/release
                       Solaris 10 5/09 s10x_u7wos_08 X86
          Copyright 2009 Sun Microsystems, Inc.  All Rights Reserved.
                       Use is subject to license terms.
                          Assembled 30 March 2009
```

Link based classic IPMPAn important difference to the new IPMP implementation is the point that you doesn't create a distinct IPMP interface because the concept of such a thing doesn't exist in classic IPMP. With link based classic IPMP you just put the interfaces in a group

```
bash-3.00# ifconfig e1000g0 plumb
bash-3.00# ifconfig e1000g1 plumb
bash-3.00# ifconfig e1000g0 vhivemind-prod netmask + broadcast + group production0 up
bash-3.00# ifconfig e1000g1 group production0 up
bash-3.00# ifconfig -aLet's have a short look onto the network configuration.
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=201000843 mtu 1500 index 15
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=201000843 mtu 1500 index 16
        inet 0.0.0.0 netmask ff000000
        groupname production0
        ether 8:0:27:6d:9:be
```

The data address is directly bound to one of the interfaces. It's important to know,that even when the ifconfig output suggest something different, outbound data flows to the network on both interfaces, not just the one which holds the data address.

Now unplug the cable connecting to e1000g0:

```
bash-3.00# ifconfig -a
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=219000802 mtu 0 index 15
        inet 0.0.0.0 netmask 0
```

```
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=201000843 mtu 1500 index 16
        inet 0.0.0.0 netmask ff000000
        groupname production0
        ether 8:0:27:6d:9:be
e1000g1:1: flags=201000843 mtu 1500 index 16
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
```

The data address was moved away from e1000g1 and a logical interface was created to hold it instead.

Probe based classic IPMPThe failure detection by IPMP probes is available in classic IPMP as well. Again all the configuration is done via ifconfig

```
bash-3.00# ifconfig e1000g0 plumb
bash-3.00# ifconfig e1000g1 plumb
bash-3.00# ifconfig e1000g0 vhivemind-e1000g0 deprecated -failover netmask + broadcast + group production0 up
bash-3.00# ifconfig e1000g0 addif vhivemind-prod netmask +  broadcast + up
Created new logical interface e1000g0:1
bash-3.00# ifconfig e1000g1 vhivemind-e1000g1 deprecated -failover netmask + broadcast + group production0 up
bash-3.00# ifconfig -a
```

Please note that you have to use the deprecated option to set the DEPRECATED flag on your own. New IPMP do this automagically. Forgetting this option leads to interesting, but not always obvious malfunctions. Let's check the network configuration.

```
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=209040843 mtu 1500 index 11
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g0:1: flags=201000843 mtu 1500 index 11
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
e1000g1: flags=209040843 mtu 1500 index 12
        inet 192.168.56.202 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:6d:9:be
```

Both interfaces have their test addresses. The data address is configured to an additional logical interface. As it's the only interface without the -failover statement, this interface is automatically managed by IPMP. Now remove the cable from the e1000g0 networking card.

```
bash-3.00# ifconfig -a
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=219040803 mtu 1500 index 11
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=209040843 mtu 1500 index 12
        inet 192.168.56.202 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:6d:9:be
e1000g1:1: flags=201000843 mtu 1500 index 12
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
```

The virtual interface with the data address has moved from e1000g0 to e1000g1

Making the configuration boot persistentMaking the configuration boot-persistent works works pretty much the same in both implementation. As we used ifconfig commands again, we can use the hostname.* files. We just have to translate the command lines accordingly:

Link-based IPMPAt first we configure the e1000g0 interface by creating the file /etc/hostname.e1000g0 containing a single line.

vhivemind-prod netmask + broadcast + group production0 up

Afterwards we do the same for e1000g1. We create a file named /etc/hostname.e1000g1 and put the following line (and just this line) in it:

group production0 up

Now reboot the system. After a few moments you can get a shell and check your configuration.

```
# ifconfig -a
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=201000843 mtu 1500 index 2
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=201000843 mtu 1500 index 3
        inet 0.0.0.0 netmask ff000000 broadcast 0.255.255.255
        groupname production0
        ether 8:0:27:6d:9:be
```

Everything configured as we've planned it.

Probe-based IPMPOkay, let's do the same for the probe-based IPMP. This is the /etc/hostname.e1000g0 file configuring the test address on the physical interface and the data address:

```
vhivemind-e1000g0 deprecated -failover netmask + broadcast + group production0 up \
addif vhivemind-prod netmask + broadcast + up
```

The file /etc/hostname.e1000g1 with the following line will configures the e1000g1 interface of our system at boot:

```
vhivemind-e1000g1 deprecated -failover netmask + broadcast + group production0  up
```

Okay, reboot your system and you should yield an ifconfig output like this one afterwards.

```
# ifconfig -a
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=209040843 mtu 1500 index 2
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g0:1: flags=201000843 mtu 1500 index 2
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
e1000g1: flags=209040843 mtu 1500 index 3
        inet 192.168.56.202 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:6d:9:be
```

Everything is fine.

Posted by Joerg Moellenkamp in English, Oracle, Solaris at 12:40

Hi, I have been checking your articles, however I am having a hard time just to find an ip fail over to another server instead of the same server to another network interface. Basically I just need this same functionality, but to do the failover to a network interface on another server. How can I go on doing this?
Anonymous on Apr 23 2013, 22:09

You need a availability protection framework. Perhaps as simple as VRRP, but depending on your requirements it's possible you have to use a real cluster framework as Oracle Solaris Cluster.
Anonymous on May 3 2013, 09:32