Getting Started With Crossbow

Pre-requisites

- Solaris Nevada build 81 or Solaris Express Developer Release
- Nemo-compliant networking card(bge, e1000g, xge, nxge, …)
- bfu and acr scripts
- Solaris Networking Virtualization bfu archives

Table of Contents

Virtualizing the Networking Devices

A single physical networking card is presented as multiple virtual cards, which are called vnics.

A vnic acts like any networking device. It has its own MAC address. An IP interface may be plumbed over a vnic, which can then be assigned an IPv4 or IPv6 address.

Example:

- List the physical links on the system:

```
# dladm show-link
bge0            type: non-vlan  mtu: 1500       device: bge0
ath0            type: non-vlan  mtu: 1500       device: ath0
```

- Create a virtual NIC over bge0:

```
# dladm create-vnic -d bge0 1
# dladm show-vnic
LINK        OVER             SPEED  MACADDRESS         MACADDRTYPE
vnic1       bge0             0 Mbps  2:8:20:22:51:dc    random
```

Note that a random MAC address was automatically assigned to the VNIC. The VNIC and its MAC address will persist if the host is rebooted.

- A new data link is created:

```
# dladm show-link
bge0            type: non-vlan  mtu: 1500       device: bge0
ath0            type: non-vlan  mtu: 1500       device: ath0
vnic1           type: non-vlan  mtu: 1500       device: vnic1
```

- Bring up an IP interface on vnic1:

```
# ifconfig vnic1 plumb
# ifconfig vnic1 dhcp start
# ifconfig vnic1
vnic1: flags=201004843<UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4,CoS>
    mtu 1500 index 4
    inet 129.146.109.26 netmask fffffe00 broadcast 129.146.109.255
    ether 2:8:20:22:51:dc
```

Now, the virtual NIC is visible to all classical network monitoring tools, such as netstat(1M):

```
    # netstat -ian
        Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis
Queue
    lo0   8232 127.0.0.0     127.0.0.1      1624   0     1624   0     0      0
    lo0   8232 127.0.0.0     127.0.0.1      0      N/A   1618   N/A   N/A    0
    ath0  1500 10.192.0.0    10.192.11.51   19036  0     3857   0     0      0
    ath0  1500 10.192.0.0    10.192.11.51   17468  N/A   3805   N/A   N/A    0
    vnic1 1500 129.146.108.0 129.146.109.26 371    0     12     0     0      0
    vnic1 1500 129.146.108.0 129.146.109.26 10     N/A   3      N/A   N/A     0
```

Bandwidth Management

The bandwidth may be limited for a whole VNIC by specifying the maxbw property (expressed in Mbps) when the vnic is created:

```
# dladm create-vnic -d bge0 -p maxbw=15 3
# dladm show-vnic
LINK        OVER             SPEED  MACADDRESS         MACADDRTYPE
vnic1       bge0            0 Mbps  2:8:20:22:51:dc    random
vnic3       bge0            0 Mbps  2:8:20:41:c2:71    random
# dladm show-linkprop -p maxbw vnic3
LINK         PROPERTY        VALUE          DEFAULT        POSSIBLE
vnic3        maxbw           15             —              —
```

A bandwidth limit can also be set on an existing data-link such as a physical NIC or an existing VNIC:

```
# dladm set-linkprop -p maxbw=300 bge0
# dladm show-linkprop -p maxbw bge0
LINK         PROPERTY        VALUE          DEFAULT        POSSIBLE
bge0         maxbw           300            —              —
```

A finer grain limit could be set on a per-transport or on a per-protocol basis, on top of any data-link. For that we use the new command flowadm(1m) to create a new flow of packets defined by the description of the packets matching the flow (transport, local_port, etc …). The limit on the bandwidth that can be used by that flow is specified using the maxbw flow property.

```
# flowadm add-flow -l vnic2 -a transport=tcp tcp_flow
# flowadm set-flowprop -p maxbw=100 tcp_flow
# flowadm show-flow
flow name      flow attributes                          policy attributes

tcp_flow       v4:tcp                                   (L)   100 Mbps
```

Incoming and outgoing TCP packets will be subject to a maximum of 100Mbps of throughput. Other traffic will use all remaining bandwidth available for vnic1.

We can observe the accumulated statistics about the packets used by the tcp_flow, by running kstat in the global zone:

```
    bash-3.00# kstat -n tcp_flow
    module: unix                            instance: 0

            name:   tcp_flow                        class:    flow
            crtime                          29.903360333
            ierrors                         0
            ipackets                        10177
            obytes                          0
            oerrors                         0
            opackets                        0
```

```
        rbytes                          550094
        snaptime                        17969.794047786
```

Per-flow bandwidth utilization may also be monitored in real time by invoking netstat -K.

In our example with tcp_flow, this is a snapshot of the real-time output screen, while running netperf between two zones, one attached to vnic1 and the other to vnic2:

```
Flow           Link          iKb/s     oKb/s     iPk/s     oPk/s

vnic1          bge0      106236.18    534.43   9068.56   1266.79
vnic2          bge0         534.89 106235.75   1267.78   9067.58
vnic3          bge0           0.46      0.00      0.98      0.00
vnic4          bge0           0.46      0.00      0.98      0.00
tcp_flow       vnic2        428.95      0.00   1016.77      0.00

Totals                    13400.12  13346.27  11355.07  10334.38
```

IP Instances and Zones

A zone with an exclusive IP stack has its own instance of the global tables and variables used in the TCP/IP stack. This allows a zone to be connected to a separate LAN or VLAN without sharing any networking state or policies with other zones. The zone with its exclusive IP instance has its own IP routing table, ARP table, IPsec policies and security associations, IP Filter rules, TCP/IP ndd tunables, etc.

To create a zone with an exclusive IP instance, set ip-type=exclusive in zonecfg(1M).

Since an exclusive zone has control over its ARP and IP internal structures, an IP address does not need to be set for the zone by the global zone administrator any more. Simply pick a physical interface to be assigned to the zone, or specify a VNIC which you previously created using dladm(1M).

The following is a sample of zonecfg input:

```
create -b
set zonepath=/export/home/Zones/z3
set autoboot=false
set ip-type=exclusive
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add inherit-pkg-dir
set dir=/etc/crypto
end
add net
set physical=vnic3
end
```

Copy it into a file (e.g. /var/tmp/zcfg.in).

Then run:

zonecfg -z z3 -f /var/tmp/zcfg.in

(Note: the following output is normal)

z3: No such zone configured Use 'create' to begin configuring a new zone.

zoneadm will now show the newly created zone 'z3' in a configured state:

```
# zoneadm list  -cv
  ID NAME             STATUS     PATH                           BRAND    IP

   0 global           running    /                              native  shared
   1 z1               running    /export/home/Zones/z1          native  excl

   2 z2               running    /export/home/Zones/z2          native  excl

   - z3               configured /export/home/Zones/z3          native  excl
```

Note the 'excl' under the 'IP' column indicating an exclusive IP instance for the zone z3.

The zone needs to be installed:

```
bash-3.00# zoneadm -z z3 install
```

Note: depending on the CPU speed, the installation may take around 10 minutes.

```
bash-3.00# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP

   0 global           running    /                              native  shared
   1 z1               running    /export/home/Zones/z1          native  excl

   2 z2               running    /export/home/Zones/z2          native  excl

   - z3               installed  /export/home/Zones/z3          native  excl
```

Boot the zone:

```
bash-3.00# zoneadm -z z3 boot
```

and connect to the zone's console by running zlogin -C z3, and initialize the naming services. You will have the usual interactive post installation dialog, set the hostname (zone's name), the time zone, the zone's root password, etc ..

The zone is now ready. You may bring up the network on the zone:

```
z3 console login: root
Password:
Feb 15 10:43:55 z3 login: ROOT LOGIN /dev/console
Sun Microsystems Inc.   SunOS 5.11      snv_55  October 2007
#
# zonename
z3
#
# ifconfig -a plumb
# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index
1
        inet 127.0.0.1 netmask ff000000
vnic3: flags=201000842<BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 2
        inet 0.0.0.0 netmask 0
        ether 2:8:20:41:c2:71
```

Note: dladm show-link cannot yet be run from a non global zone. However, ifconfig -a will plumb all non loopback interfaces that were assigned to the zone by the global zone administrator.

```
    # ifconfig vnic3 1.1.1.3/24 up
    # ifconfig -a
    lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index
1
            inet 127.0.0.1 netmask ff000000
    vnic3: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 2
            inet 1.1.1.3 netmask ffffff00 broadcast 1.1.1.255
            ether 2:8:20:41:c2:71

    # netstat -rn

    Routing Table: IPv4
      Destination           Gateway           Flags  Ref     Use     Interface
-------------------- -------------------- ----- ------ ---------- ---------


    1.1.1.0              1.1.1.3              U         1          1 vnic3

    224.0.0.0            127.0.0.1            U         1          0 lo0

    127.0.0.1            127.0.0.1            UH        1         36 lo0
```

The zone can now communicate with other hosts on the network.

```
    # ping 1.1.1.1
    1.1.1.1 is alive
```

Note: Just like a single zoned system, /etc/hostname.vnic3 or /etc/dhcp.vnic3 can be used to have the network automatically initialized with a static or dynamic IP address upon reboot of the zone.

Back to the global zone:

```
    bash-3.00# ifconfig -a
    lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
            inet 127.0.0.1 netmask ff000000
    ath0: flags=201004843<UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4,CoS> mtu 1500 index 3
            inet 192.168.1.132 netmask ffffff00 broadcast 192.168.1.255
            ether 0:b:6b:4d:b1:4
    bash-3.00#
    bash-3.00# netstat -rn

    Routing Table: IPv4
      Destination           Gateway           Flags  Ref     Use     Interface
-------------------- -------------------- ----- ------ ---------- ---------


    default              192.168.1.1          UG        1         45 ath0

    192.168.1.0          192.168.1.132        U         1          4 ath0

    224.0.0.0            192.168.1.132        U         1          0 ath0

    127.0.0.1            127.0.0.1            UH        2         68 lo0
```

The IP interface 'vnic3' is not exposed to the global zone, or other zones.

Further details about IP instances may be found in the Presentation to OpenSolaris User Group and the IP instances Architecture.

CPU Resources with Network Virtualization

NICs and VNICs may be bound to a subset of the processors available on a system. When such binding is established, most of the packet processing will be executed on the bound CPUs.

This feature is particularly useful for deeper separation of the CPU resources between zones and containers. It extends that separation to account for most of the CPU cycles spent anonymously in the networking subsystem on behalf of a zone.

The example below illustrates a setup that shows the difference in CPU utilization when this feature is in use.

The NIC will still interrupt a system defined CPU(s), which are currently not under the virtualization control. However, incoming packets are quickly dispatched to be processed by a CPU from the zone's processor set.

The instructions for creating a CPU resource pool with 4 CPUs out of a T-10000 multi-core system are described here

Configure a zone work1zone with an exclusive IP stack, and connect it to a vnic called vnic1, as described above. (In this example, we use vnic1 over bge1, which was assigned by the system to interrupt the CPU 8).

Assign the work1-pool pool to work1zone:

```
# zonecfg -z work1zone set pool=work1-pool
```

Reboot work1zone so that it binds to the new pool:

```
# zlogin work1zone init 6
```

Running iperf between the work1zone container and an external host, will actually show that processors 4-7 are mainly used. Most CPUs outside that set are occasionally used at a 5% or less of the time, for other internal load. This is the expected behavior because the application (and all its system calls) are actually hosted by the container.

```
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0  106   277  175    0    0    0    2    0     0    0   1   0  99
  1    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
  2    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
  3    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
  4    0   0    0  3765    0 7496    7   76  355    0  4556    2  19   0  79
  5    0   0    0  3590    0 7175    7   77  336    0  4310    2  19   0  80
  6    0   0    0  1504    0 3009    3   15  137    0  1912    1   8   0  92
  7    0   0    0  1108    8 2196    1    6   91    0  1394    1   6   0  94
  8    0   0 7247  7711 7706    9    0    0  860    0     0    0  56   0  44
…
 22    0   0 9114  6688    0 13659    0    0 1176    0     0    0  64   0  36
 23    0   0    0     3    0    5    0    0    0    0     1    0   0   0 100
```

CPU # 22 was being used at 64% of the time, all inside the system. That is the cost of processing incoming packets for work1zone. That cost is being charged to a CPU that is *not* member of the processor set that the container is assigned to.

To minimize the unfair utilization of CPU resources induced by work1zone inbound traffic, vnic1 needs to be also "bound" to the CPUs of work1-pset:

From the global zone, run:

```
# dladm set-linkprop -p cpus=4,5,6,7 vnic1
```

A second experiment with iperf shows:

```
  CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
    0    0   0  118   270  168    0    0    0    1    0     0    0   0   0 100
    1    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
    2    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
    3    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
    4    0   0 2370  4477    0 9045   30   95  623    0  2799    1  33   0  66
    5    0   0 2702  4630    0 9366   26   88  608    0  2331    1  32   0  67
    6    0   0 2995  4459    0 9017   27   83  737    0  1778    1  34   0  65
    7    0   0    0  3225    5 6491    3   51  167    0  4494    2  20   0  79
    8    0   0 7574  7587 7574   26    0    1 2767    0     5    0  51   0  49
…
   22    0   0 1243  1088    0 2175    0    0  344    0     4    0   2   0  98
   23    0   0    2     8    0   15    0    0    1    0     0    0   0   0 100
```

Note in particular: . CPUs 4-7 idle time decreased, and their sys time has increased since they work more to handle work1zone's packets. . CPU #22 is barely working at 2% instead of the previous 64%.