PETER BAER GALVIN

Pete's all things Sun: Crossbow

Peter Baer Galvin is the chief technologist for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is co-author of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://www.galvin.info and twitters as "PeterGalvin."

pbg@cptech.com

AS SOLARIS 10 GETS READY TO ENTER its fifth year, it's already looking rather aged. The distance between it and its offspring, OpenSolaris, is growing, and it's changing slower than it used to. In fact, there are a large set of features available in OpenSolaris that are not, and likely will not be, integrated back into Solaris 10. Unfortunately, the mainstream production operating system from Sun is Solaris, and the move to Solaris "next," which will be based on OpenSolaris, is still in the future.

The good news is that the changes in OpenSolaris are so extensive that it's difficult for Sun to reintegrate those changes into Solaris 10. The bad news is that to use these new features, we have to use the less-supported (at least by ISVs, if not by Sun) OpenSolaris. Those features are becoming so compelling that, for some environments, it might be worth the added OpenSolaris production challenges to gain access to those new features. One such feature is ZFS deduplication, which removes blocks that are already stored within ZFS when a write request is received, replacing them with pointers to the existing blocks, rather than storing yet another copy of that same block. That functionality is the major feature of Data Domain, which was recently sold to EMC for a very large sum of money. That is not the topic this month (but will certainly be included in a future column). Rather, another exciting, innovative, and important OpenSolaris feature is beckoning our attention: Crossbow.

Crossbow is a major new feature of OpenSolaris, hiding within its simple interface powerful virtualization and resource management abilities. Crossbow makes it possible to implement an entire network within a single Solaris instance and to put fine-grain controls on how much data is flowing from and to each network component. Crossbow was integrated into OpenSolaris this year, in build 106, with its first official release coming in OpenSolaris 2009.06. The paper describing Crossbow was published at the USENIX LISA '09 conference and won the conference's Best Paper award [1]. In this column I explain what Crossbow is and how to implement and manage its features.

Overview

Crossbow builds on the previous Solaris networking improvement projects, including Firetruck (which brought speed and multi-threading improvements to Solaris 10). Crossbow is a new virtualization layer within OpenSolaris, part of the core network feature set. Most of the features are off by default but are easily enabled and are efficient, powerful, and easy to use.

The features of Crossbow include:

- The ability to implement an entire virtual-wire network (vWire), virtually, within OpenSolaris, including firewalls, servers, routers, and switches
- Fine-grained network resource management Quality of Service (QoS), including per-protocol bandwidth limits, traffic priorities, CPU assignments, and VLAN tags
- vWires that can be reduced to a set of objects and rules which can be modified or replicated, allowing network modeling, debugging, and performance tuning within a virtualized network
- The ability to take advantage of native hardware features for security and performance
- Full zone/container and Xen domain awareness

Essentially, the feature set of Crossbow decouples an application from the physical network, allowing flexible network design based on application needs rather than underlying physical components. Such an application can then be moved or duplicated more easily, since the same virtual network can be deployed on varying physical components.

Crossbow provides all of these features without significant performance impact on network traffic (according to testing by Sun [1]). Solaris shops will very likely want to take advantage of these new features.

The remainder of this column explores these features and demonstrates how to use them, including the very impressive (but unsupported) vWire Builder. To get started you might want to read the "How To" guide published by Sun [2], and the Crossbow documentation [3].

Exploration

There is a bit of new terminology that comes with Crossbow, which is necessary to understand Crossbow's features and functions.

- VNIC: A virtual network interface controller. Exactly the same as a physical network NIC, but virtual. A VNIC can be plumbed, `ifconfig`ed, `snoop`ed, and `dladm`ed.
- Etherstub: A virtual switch (named a "stub" because it is similar to the programming concept of stubbing a subroutine as a placeholder before writing the full routine). It connects VNICs.
- Flow: A resource management component. Zero or more can be assigned to a link (physical or virtual) to implement QoS. Flows can control services (protocols and local and remote ports), transports, and local and remote IP addresses and subnets.

A physical port does not need to be plumbed or configured: a VNIC can provide all of that functionality. A VNIC within an Etherstub cannot send traffic directly outside that virtual switch, but the traffic can be routed outside the Etherstub.

Crossbow takes advantage of hardware features provided by the NICs. For example, most modern NICs have hardware classification capabilities that Crossbow uses to create "hardware lanes"—collections of hardware resources such as ring buffers and DMA channels that accelerate and manage performance.

"Traffic flows" can be created to manage the resources used by these other components. Traffic flows span the whole network stack from NIC through sockets, allocating and limiting resources. For instance, we could limit all UDP traffic traveling through a NIC to a certain maximum amount of bandwidth. Traffic flows can be used to limit traffic to containers as well. Traffic flows are managed via the `flowadm` command.

Because VNICs are essentially a full hardware NIC, virtualized, features such as `snoop` work with them. Before Crossbow, the only way for a container to manage a NIC and perform functions such as `snoop` was to dedicate a NIC port to the zone via the "exclusive-ip" container configuration feature. With Crossbow, VNICS can be given to containers and managed (and `snoop`ed by the container). One limit on VNICs is that they cannot be created on top of other VNICs. Like containers, VNICs are one layer deep. VNICs are managed by the `dladm` and `ifconfig` commands. Multiple VNICs configured on the same NIC cause the automatic creation of a virtual switch to connect all of the NIC's VNICs. Each NIC with VNICs gets its own independent virtual switch.

The Etherstub feature is similar to the auto-created virtual switches, except that it is independent of the NIC hardware. It can be used to create a virtual network that allows communication between VNICs within the kernel, without any hardware interaction. VNICs that are not part of the same virtual switch cannot talk to each other, while those that are part can communicate. Figure 1 shows the kinds of network configurations that can be created by Crossbow.

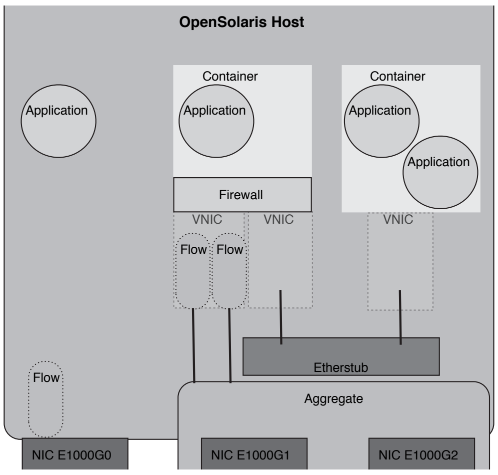


FIGURE 1: A SOLARIS SYSTEM WITH CONTAINERS AND CROSSBOW FEATURES

How do these commands work?

To create a new virtual NIC named "vnic1" on physical network device "e1000g0", execute:

```
$ pfexec dladm create-vnic -l e1000g0 vnic1
```

Because they are cheap, let's create another one called "vnic2":

```
$ pfexec dladm create-vnic -l e1000g0 vnic2
```

To check the status of all VNICs, execute:

```
$ pfexec dladm show-vnic
```

```
LINK    OVER       SPEED    MACADDRESS        MACADDRTYPE  VID
vnic1   e1000g0    1000     2:8:20:84:99:e7   random       0
vnic2   e1000g0    1000     2:8:20:82:e:fb    random       0
```

Notice that the MAC address is assigned at random, by default. Controls are available within the dladm command to allow manual MAC address management.

If the VNIC is going to have properties set, you can set them at creation time or afterward. For example, to create VNIC "vnic3" and limit the bandwidth of all traffic though that NIC to 40MB/sec:

```
$ pfexec dladm create-vnic -l e1000g0 -p maxbw=40 vnic3
```

To modify an existing VNIC, "vnic1," to have all of its activity execute only on CPU 1, and to give it a lower priority than any "medium" and "high" priority VNICs on the same NIC, you could say:

```
$ pfexec dladm set-linkprop -p cpus=1,priority=low vnic1
```

To get a detailed list of all the properties of VNIC vnic3, execute:

```
$ pfexec dladm show-linkprop vnic3
LINK    PROPERTY    PERM    VALUE      DEFAULT    POSSIBLE
vnic3   autopush    -w      --         --         --
vnic3   zone        rw      --         --           --
vnic3   state       r-      unknown    up         up,down
vnic3   mtu         r-      1500       1500         --
vnic3   maxbw       rw      40         --         --
vnic3   cpus        rw      --         --         --
vnic3   priority    rw      high       high       low,medium,high
vnic3   tagmode     rw      vlanonly   vlanonly   normal,vlanonly
```

To show the status of all VNICs on the system, execute:

```
$ pfexec dladm show-vnic
LINK    OVER       SPEED    MACADDRESS        MACADDRTYPE   VID
vnic1   e1000g0    1000     2:8:20:84:99:e7   random        0
vnic2   e1000g0    1000     2:8:20:82:e:fb    random        0
vnic3   e1000g0    40       2:8:20:a7:28:cc   random        0
```

Now let's create a virtual network switch and place two virtual NICs on that switch. This would be useful, for example, to allow two zones to talk to each other, but not on any physical network ports.

```
$ pfexec dladm create-etherstub inet1
$ pfexec dladm create-vnic -l inet1 vnzone1
$ pfexec dladm create-vnic -l inet1 vnzone2
```

To show the status of all links within the system, execute:

```
$ dladm show-link
LINK       CLASS       MTU     STATE      OVER
e1000g0    phys        1500    up         --
vnic1      vnic        1500    up         e1000g0
vnic2      vnic        1500    up         e1000g0
vnic3      vnic        1500    up         e1000g0
inet1      etherstub   9000    unknown    --
vnzone1    vnic        9000    up         inet1
vnzone2    vnic        9000    up         inet1
```

To continue that example, when building the zones, the two zones would be configured to use VNICs "vnzone1" and "vnzone2." Note that instant and periodic information about various aspects of network traffic is available via the usual "-i" option to the commands. For example:

```
$ dladm show-link -s -i 5
LINK      IPACKETS  RBYTES    IERRORS  OPACKETS  OBYTES     OERRORS
e1000g0   49732     54930394  0        7829      758620           0
e1000g0   2         134       0        1         182              0
```

More fine-grained traffic management is performed via the flowadm command. For example, to manage port 80 traffic on "vnic1," first a flow is created that describes that traffic. We give it the name "httpflow":

```
$ pfexec flowadm add-flow -l vnic1 -a transport=tcp,local_port=80 httpflow
```

Next, we shape that traffic. For example, here we limit port 80 traffic on "vnic1" to 100Mb/sec (full duplex):

```
$ pfexec flowadm set-flowprop -p maxbw=100M httpflow
```

We could also set a limit on how much traffic we send to other systems' port 80s by using the property "remote_port." To list all flows on the system and examine the state of the httpflow flow:

```
$ pfexec flowadm show-flow
FLOW      LINK      IPADDR       PROTO   PORT   DSFLD
httpflow  vnic1     --           tcp     80     --
$ pfexec flowadm show-flowprop httpflow
FLOW      PROPERTY      VALUE   DEFAULT        POSSIBLE
httpflow  maxbw         100     --             100M
httpflow  priority      --      --
```

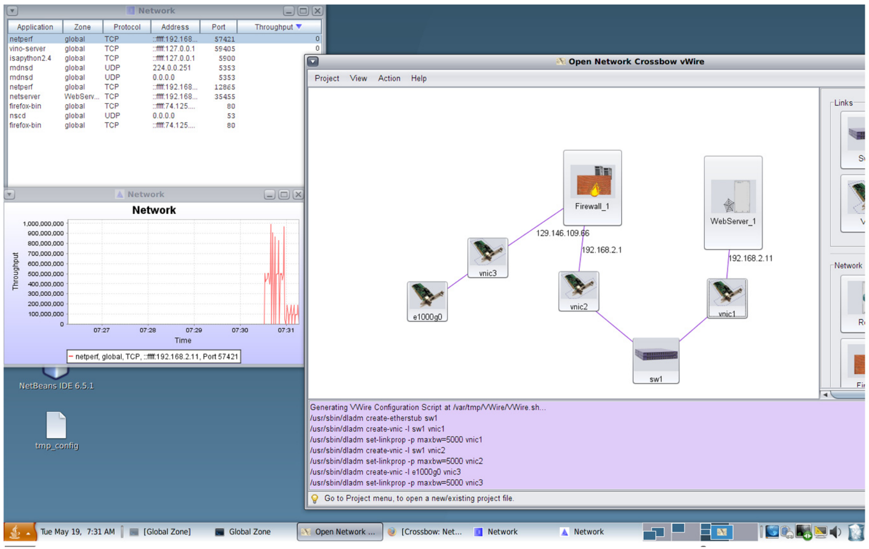
Flow statistics are available via the expected:

```
$ pfexec flowadm show-flow -s
```

Crossbow also uses the accounting subsystem to track network traffic. Here we enable that network accounting and examine its status:

```
$ pfexec acctadm -e basic -f /var/log/net.log net
$ pfexec acctadm net
   Net accounting: active
   Net accounting file: /var/log/net.log
   Tracked net resources: basic
   Untracked net resources: src_ip,dst_ip,src_port,dst_port,protocol,dsfield
$ pfexec dladm show-usage -f /var/log/net.log
```

A fairly astounding example of what can be done with Crossbow comes via the unsupported but very cool vWireBuilder tool [5]. Figure 2 shows a screen shot of the tool in use. In this example I've dragged a few network components onto the canvas, right-clicked on them to set properties such as IP addresses, and then compiled and executed the configuration. vWireBuilder then created this network configuration within my system by creating containers for each system function (firewall, Web server) and VNICs and Etherstubs for the network connections. It started a Web server within the Web server container. IP QoS was enabled by right-clicking on the various VNICs and setting bandwidth limits. I then added a network traffic load-generator, and vWireBuilder monitored the traffic via a drop-down menu item. Note that it did not configure the firewall within the firewall container; that would still need to be done by hand. The entire creation, from starting the tool to having the network configuration instantiated within my system, required only a few minutes. There is a demo video included with the tool that walks through some of its uses [5].



Application	Zone	Protocol	Address	Port	Throughput
netperf	global	TCP	::ffff:192.168...	57421	0
vino-server	global	TCP	::ffff:127.0.0.1	59405	0
isapython2.4	global	TCP	::ffff:127.0.0.1	5900	
mdnsd	global	UDP	224.0.0.251	5353	
mdnsd	global	UDP	0.0.0.0	5353	
netperf	global	TCP	::ffff:192.168...	12865	
netserver	WebServ...	TCP	::ffff:192.168...	35455	
firefox-bin	global	TCP	::ffff:74.125....	80	
nscd	global	UDP	0.0.0.0	53	
firefox-bin	global	TCP	::ffff:74.125....	80	

```
Generating VWire Configuration Script at /var/tmp/VWire/VWire.sh...
/usr/sbin/dladm create-etherstub sw1
/usr/sbin/dladm create-vnic -l sw1 vnic1
/usr/sbin/dladm set-linkprop -p maxbw=5000 vnic1
/usr/sbin/dladm create-vnic -l sw1 vnic2
/usr/sbin/dladm set-linkprop -p maxbw=5000 vnic2
/usr/sbin/dladm create-vnic -l e1000g0 vnic3
/usr/sbin/dladm set-linkprop -p maxbw=5000 vnic3
```

FIGURE 2: VWIREBUILDER CREATING A VWIRE

The Future

Crossbow seems to be a great framework for the next generation of Solaris networking features. Challenging and previously impossible network configurations, including fine-grained performance management, are made simple by its elegant feature set. For a just recently implemented facility, it works well and is feature-rich. Certainly more features will appear over time to refine and expand its functionality. In the future, the vWireBuilder GUI could capture a network configuration and allow its modification and then its instantiation with an OpenSolaris host. Such a feature would be a powerful addition to the system administrator's toolkit.

Conclusions

Project Crossbow extends the functionality and performance of the already impressive OpenSolaris network stack. Solaris 10 and OpenSolaris provide basic support for 10Mb through 10Gb HBAs, link aggregation at layer 2, IP multipathing (IPMP) at layer 3, VLAN tagging, routing, firewalling, and traffic snooping. Crossbow adds to this virtual NICs, virtual switches, and fine-grained IP QoS management of all networking resources. This new addition provides all of the components needed to implement, analyze, and performance-manage fully virtual networks. These features should allow Solaris administrators to design networks independent of the host hardware, and in the future the use of tools like vWireBuilder should allow for creation, modification, capture, and duplication of full virtual network infrastructures. The current state of Crossbow is expectedly in flux. There are demonstrations and blogs that refer to commands that no longer exist and to commands that may exist in the future. This results in a bit of confusion when trying out the features of Crossbow, but that exploration is rewarded with hints of a terrific new network feature set in OpenSolaris. See the Project Crossbow Wiki for more documents, discussion, details, and information on future features [6].

Tidbits

The USENIX LISA conference ran in November in Baltimore, and it had great content. I enjoyed teaching my Solaris tutorials and sitting in on some excellent presentations by other tutorial instructors and speakers. It was nice to see such a strong showing by Sun there, in the form of talks by Sunay Tripathi [7] about Crossbow and Bryan Cantrill [8] about the analytics in the Sun 7000 (fishworks) products. There were also Birds of a Feather sessions by Sun to present various activities that Sun is undertaking and solicit feedback. You can watch videos of these presentation, or listen to MP3s, via the LISA '09 Web site.

In OpenSolaris, as of build 129, ZFS has in-line deduplication. This is an important milestone for OpenSolaris, as companies that sell applications that perform in-line deduplication (Data Domain) are selling like $2B hotcakes. To try deduplication, first you must get your hands on the OpenSolaris 2009.06 distribution. After installing that, you need to update it to the latest developer build of OpenSolaris via these commands:

```
# pkg set-publisher -O http://pkg.opensolaris.org/dev opensolaris.org
. . .
# pkg image-update
. . .
```

Follow any extra instructions that image-update command requires. The net result should be a new OpenSolaris boot environment that incorporates the latest bits. A reboot into the new boot environment should bring up an OpenSolaris containing ZFS deduplication. Enabling deduplication is as trivial as setting a new property on an existing ZFS pool (to deduplicate future blocks written anywhere in that pool) or to an existing file system to just deduplicate inbound blocks being written there.

```
# zfs set dedup=on zpool
# zfs set dedup=on zpool/home
```

Rather astounding! There are more details, and discussions of the algorithms used, in Jeff Bonwick's blog: http://blogs.sun.com/bonwick/entry/zfs_dedup.

REFERENCES

[1] https://db.usenix.org/events/lisa09/tech/full_papers/tripathi.pdf.

[2] http://www.opensolaris.com/use/crossbow_network_containers.pdf.

[3] http://hub.opensolaris.org/bin/view/Project+crossbow/Docs.

[4] http://wikis.sun.com/display/OpenSolaris/vWireExample.

[5] http://hub.opensolaris.org/bin/view/Project+crossbow/demo.

[6] http://hub.opensolaris.org/bin/view/Project+crossbow/Features.

[7] http://blogs.sun.com/sunay/.

[8] http://blogs.sun.com/bmc/.