Bro and the ARC Series Adapters – A Strong Defender

Bro open source software is a great platform for building a Network Intrusion Detection System (NIDS). However, Bro throughput, like that of any IDS solution, is challenged by ever climbing network bandwidth. It starts with packet capture. At higher bandwidths it is far too easy to drop packets or consume too much CPU. Higher bandwidths require the scaling of Bro by using multiple CPU cores. That scaling depends upon specialized, and often costly hardware or software libraries maintained outside the Bro community.

Fortunately, there is a cost-effective solution to all of these challenges: a Bro plug-in that integrates your application with CSPi's Myricom ARC series of Network Adapters with Sniffer10G software. The plug-in is part of the official Bro distribution. By simply changing the library you link with, users of *libpcap*, *WinPcap,* and *PF_RING* can leverage CSPi's Myricom® Sniffer10G hardware and software. This delivers big benefits to packet-oriented applications.

- **Multiple applications see the same packets** - run multiple applications simultaneously against the same packets with zero copying in the background (for example, Bro with Snort). This works through libpcap and is controlled by environment variables or by PF_RING
- **Scale a single application** – distribute network flows among multiple copies of libpcap. Bro "frontend" scalability depends upon this. Many Snort users also leverage this.
- **Reduce CPU usage** - send all packets directly to your application in user-space, completely bypassing the OS kernel. This frees up CPU cycles and helps applications keep up with Myricom adapters which never drop a packet
- **Lossless packet capture** - Target uniquely large memory buffers, thereby eliminating packet drops due to rate matching challenges between hardware and software
- **Exact time stamps** - Our network adapters can deliver up to ± 3 ns hardware timestamps given high-quality timing signals attached to the coax or ± 30 ns with a quality PTP grandmaster. Myricom adapters can also extract and present Arista switch timestamps to your application

Bro Plugin

Recent Bro distributions contain a plug-in that attaches Bro directly to our Sniffer10G API. The benefits over using libpcap are:

- Even lower CPU utilization because one less data copy is needed
- You don't need to relink Bro against the CSPi Myricom version of libpcap
- BroControl will setup the worker scaling for you

Details
BroControl has built-in support for host-based load-balancing using either Sniffer10G or PF_RING (which often has Sniffer10G under it). Instead of creating separate entries for each

BRO worker, you specify the number of workers and the load-balancing algorithm, and BRO sets up everything, including the appropriate environment variables. In order to achieve the kernel bypass, high performance mode, you must use the “snfn” sniffer interface, and not the kernel interface (such as eth2). Using the kernel interface will result in poor performance and packet loss.

There are 3 new keywords in the node.cfg file: lb_procs, lb_method, and lb_interfaces.
You will need to configure Bro with this option:
./configure --with-pcap=/opt/snf/

Then in your node.cfg file you should make any worker using the Sniffer10G drivers look similar to this:

[worker1]
type=worker
host=1.2.3.4
interface=snf0
lb_method=myricom
lb_procs=10
pin_cpus=3,4,5,6,7,8,9,10,11,12

The flow-based load balancing will be automatically enabled. The pin_cpus will cause each BRO worker to run on a particular core, and stay on that core. That will help cache. As a general rule, you should process as much as possible on the NUMA node that the CSPi/Myricom card is connected to. If the card is on the PCIe bus for socket 0, then use cores on NUMA node 0. Processing packets on another socket will work, but will entail additional overhead to transfer the packet over the inter-socket QPI bus.

Are there any known issues with broctl capstats? The results might not be what you expect when you experience packet loss. If you are running multiple instances of BRO, then the way CSPi/Myricom statistics are reported are that number of received packets is by instance, so if you have 10 instances, you add up the 10 values for received packets to get the total number of packets on the link. However, each instance of BRO also reports the TOTAL number of lost packets. So if worker 1 drops 10 packets and worker 3 drops 25 packets, each worker will report 35 lost packets – that is worker 1 will report 35 lost packets, worker 2 (which didn’t lose any packets) will report 35 lost packets, and worker 3 will also report 35 lost packets.
So to compute the total number of packets BRO processed, you take the number of packets received for each worker, and add the number of lost packets for 1 worker only.
Normally, all packets for a Myricom port go to 1 ring. Any application which opens that ring gets all the packets. If you want to do load balancing across ‘n’ threads or application instances, based on packet flow, then you set the SNF_NUM_RINGS to a value between 2-32 and the packet flows will be distributed among those rings.

This allows you to engineer your performance by running multiple instances of BRO or Suricata on a server and distributed the flows from a port among those multiple instances.
When a Sniffer device is opened, the Myri10GE interface is disabled.

What does your node.cfg file look like?

Here is my node.cfg:

```
[manager]
type=manager
host=localhost
[proxy-1]
type=proxy
host=localhost
[worker-1]
type=worker
host=localhost
interface=snf0
lb_method=myricom
lb_procs=9
pin_cpus=3,4,5,6,7,8,9,10,11
env_vars=SNF_FLAGS=0x1
```

If you don't set the sharing option SNF_FLAGS=0x1, you can't have >1 processes open snf0.

From the Sniffer v3 User Guide:
Set the variable SNF_FLAGS = 0x01. This allows >1 application to open a sniffer device at the same time. Since each BRO worker is a separate process, it means multiple applications are trying to open a sniffer device at the same time. BRO is not multi-threaded.

Running BRO and another application at the same time: There are a couple of variables you want to set in order to run 2 applications:

1. You need to set SNF_FLAGS = 1
   This will enable sharing, so that multiple applications can open the same port.
2. Set the SNF_APP_ID differently for each application

BRO is special in that variables you set in the shell are not passed on through BROCTL. For example, to use the PCAP interface with BRO, without the plugin, I have the following node.cfg file:

```
[manager]
type=manager
host=localhost
[proxy-1]
type=proxy
host=localhost
```

```
[worker-1]
type=worker
host=localhost
interface=snf0
lb_method=myricom
lb_procs=9
pin_cpus=3,4,5,6,7,8,9,10,11
env_vars=SNF_FLAGS=0x1,SNF_APP_ID=2
```

If you set the SNF_DEBUG_MASK=3, you will get some debug information when the sniffer port is opened, either directly, or through libPcap. This is helpful in debugging because you will see if the APP_ID is set correctly, and if the sniffer library is being used. You just have to find the place where the BRO output is stored from each instance.

Because BRO is not run in the same window as you launch, it doesn't get the environment variables, so the SNF_APP_ID will not be set, and it will end up being -1. In order to share the port, the SNF_APP_ID has to be different from the default -1 value, and you can't mix one app with ID=-1 and another with ID=2. Either they all have -1, or they are all >0.

Additional information that may be helpful:
We supply sample utilities in /opt/snf/bin/tests : the snf_simple_recv -t -p 0 will tell you how many packets are coming in port 0 every second, it's a great sanity check to verify everything works fine. And snf_pktgen -p 1 will generate packets on port 1, if you want to do a loopback to verify. Both commands have a "-h" to get a list of all the options.

And things to think about:

How many cores on your machine? How many numa nodes? Are you running hyper-threading?

Your best performance is to run BRO on the same NUMA node as the PCIe slot or to at least maximize the number of cores on the same NUMA node as the PCIe slot.

On some systems, the cores are numbered sequentially per NUMA node, and on others, it alternates where the even cores are on one NUMA node and the odd cores are on the other NUMA node.

Ideally, you want to minimize the amount of traffic over the Intel QPI bus between CPU sockets.