Argus with Netmap : Monitoring traffic at 10Gbps line rate with commodity hardware

@FlowCon 2014

by Harika Tandra
Software Engineer at GLORIAD,
University of Tennessee, Knoxville

htandra@gloriad.org

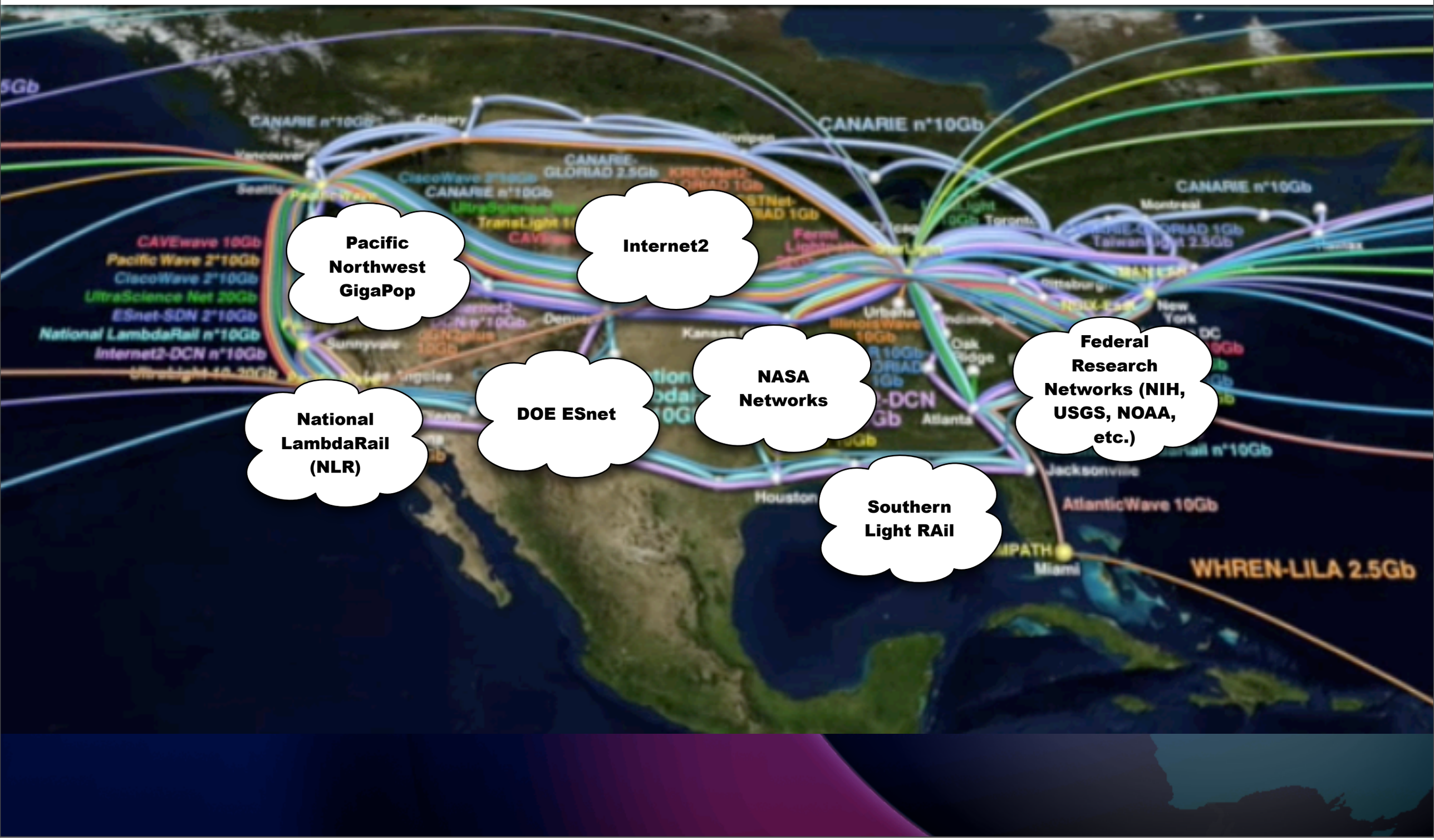
The Global Ring Network for Advanced Applications Development (GLORIAD)

The Global Ring Network for Advanced Applications Development (GLORIAD)

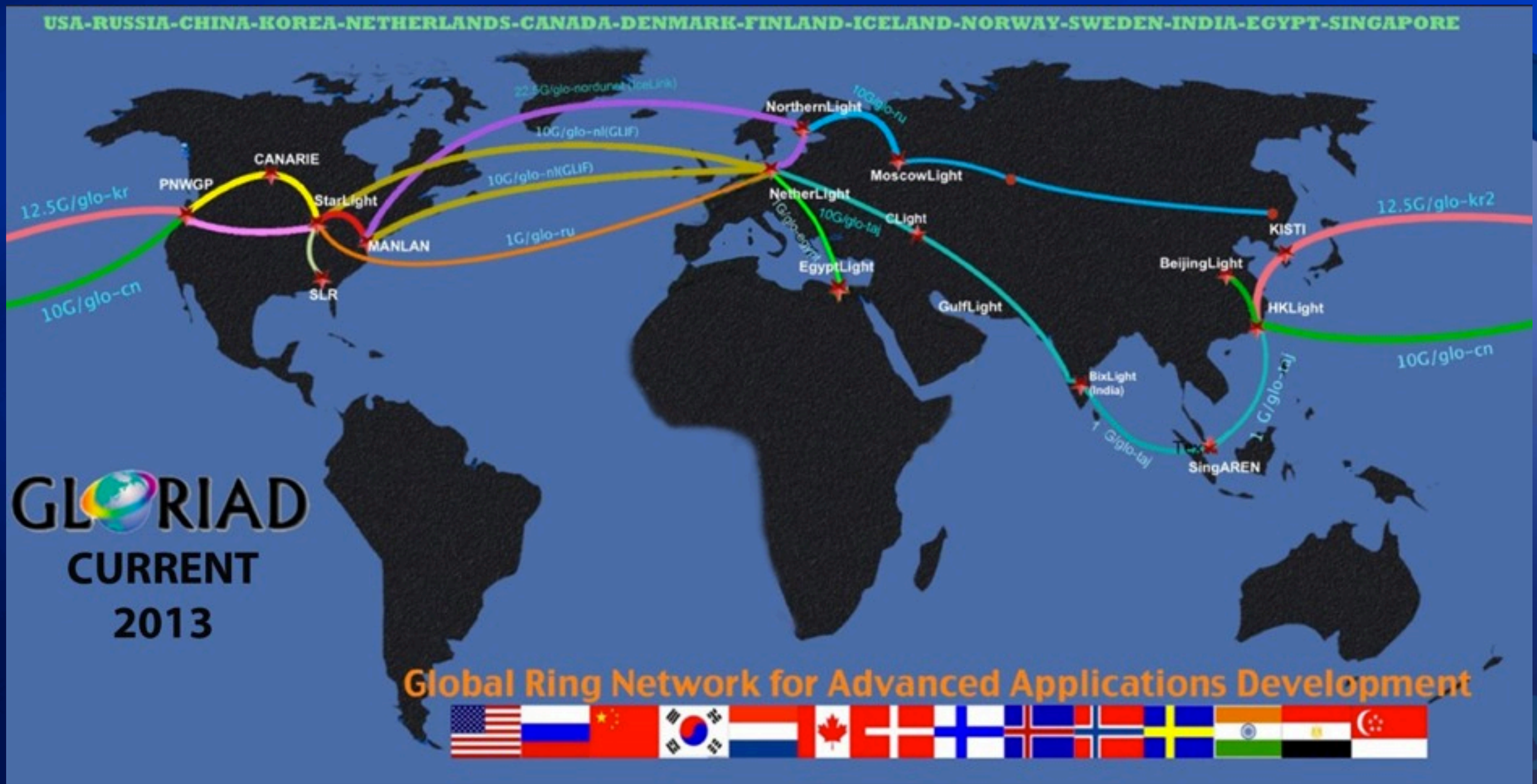
- GLORIAD is a "ring of rings" fiber-optic network around the northern hemisphere connecting US Research and Educations (R&E) networks to international R&E networks.
- NSF-funded project.

GLORIAD Operations

All these US R&E networks are connected to international peers via GLORIAD and similar international R&E networks.



Global Ring Network for Advanced Applications Development (GLORIAD)



Partners: SURFnet, NORDUnet, CSTnet (China), e-ARENA (Russia), KISTI (Korea), CANARIE (Canada), SingaREN, ENSTInet (Egypt), Tata Inst / Fund Rsrch/Bangalore Science Community, NLR/Internet2/NLR/NASA/FedNets, CERN/LHC

Global Ring Network for Advanced Applications Development (GLORIAD)

You have probably used GLORIAD unaware - if you ever visited any China Academy of Science site, Russian institution, Korean institution or other GLORIAD partner institution.

Partners: SURFnet, NORDUnet, CSTnet (China), e-ARENA (Russia), KISTI (Korea), CANARIE (Canada), SingaREN, ENSTInet (Egypt), Tata Inst / Fund Rsrch/Bangalore Science Community, NLR/Internet2/NLR/NASA/FedNets, CERN/LHC

Current GLORIAD-US Deployment of Argus

Seattle Force-10 Router

Chicago Force-10 Router

10G SPAN port

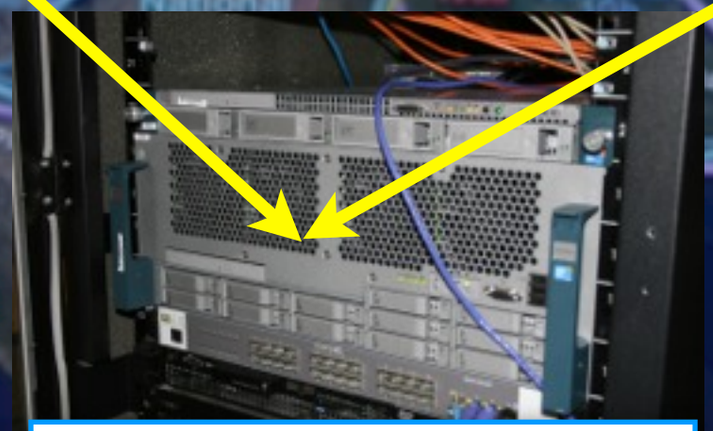
10G SPAN port

SEATTLE ARGUS NODE

DELL R410 servers -
1) Processors - 2 x Intel xeon X55670, 2.93GHz (Quad cores)
2) Memory - 8 GB (4 x 2GB) UDDIMMs
3) Hard drive - 500GB SAS
4) Intel 82599EB 10G NIC
5) OS - FreeBSD
6) Netmap
7) running argus daemon sending data to radium server in Knoxville

CHICAGO ARGUS NODE

DELL R410 servers -
1) Processors - 2 x Intel xeon X55670, 2.93GHz (Quad cores)
2) Memory - 8 GB (4 x 2GB) UDDIMMs
3) Hard drive - 500GB SAS
4) Intel 82599EB 10G NIC
5) OS - FreeBSD
6) Netmap
7) running argus daemon sending data to radium server in Knoxville



KNOXVILLE - Argus data collection and analysis.

Current GLORIAD-US Deployment of Argus

Seattle Force-10 Router

Chicago Force-10 Router

10G SPAN port

10G SPAN port

SEATTLE ARGUS NODE

DELL R410 servers -
1) Processors - 2 x Intel xeon X55670, 2.93GHz (Quad cores)
2) Memory - 8 GB (4 x 2GB) UDDIMMs
3) Hard drive - 500GB SAS
4) Intel 82599EB 10G NIC
5) OS - FreeBSD
6) Netmap
7) running argus daemon sending data to radium server in Knoxville

CHICAGO ARGUS NODE

DELL R410 servers -
1) Processors - 2 x Intel xeon X55670, 2.93GHz (Quad cores)
2) Memory - 8 GB (4 x 2GB) UDDIMMs
3) Hard drive - 500GB SAS
4) Intel 82599EB 10G NIC
5) OS - FreeBSD
6) Netmap
7) running argus daemon sending data to radium server in Knoxville

Current GLORIAD-US Deployment of Argus

Seattle Force-10 Router

Chicago Force-10 Router

10G SPAN port

10G SPAN port

SEATTLE ARGUS NODE

DELL R410 servers -
1) Processors - 2 x Intel xeon X55670, 2.93GHz (Quad cores)
2) Memory - 8 GB (4 x 2GB) UDDIMMs
3) Hard drive - 500GB SAS
4) Intel 82599EB 10G NIC
5) OS - FreeBSD
6) Netmap
7) running argus daemon sending data to radium server in Knoxville

CHICAGO ARGUS NODE

DELL R410 servers -
1) Processors - 2 x Intel xeon X55670, 2.93GHz (Quad cores)
2) Memory - 8 GB (4 x 2GB) UDDIMMs
3) Hard drive - 500GB SAS
4) Intel 82599EB 10G NIC
5) OS - FreeBSD
6) Netmap
7) running argus daemon sending data to radium server in Knoxville

Scaling our argus nodes for the rapidly growing traffic volume. Will reach 10Gbps and above soon.

Scaling to 10Gbps

- Monitoring 10Gbps links :
  - Commerical boxes (mostly give only Netflow data)
  - Specialized hardware like Endace DAG cards that scale to 10Gbps or higher rates
  - Software based solutions - packet capture accelerators like **Netmap**, PF_RING, DNA
- This presentation gives :
  - Introduction to Netmap API
  - Results of Argus using Netmap

Scaling to 10Gbps

- Monitoring 10Gbps links :
  - Commerical boxes (mostly give only Netflow data)
  - Specialized hardware like Endace DAG cards that scale to 10Gbps or higher rates
  - Software based solutions - packet capture accelerators like **Netmap**, PF_RING, DNA
- This presentation gives :
  - Introduction to Netmap API
  - Results of Argus using Netmap

Introduction to Netmap

Netmap

- New framework for high speed packet I/O
- Very efficient framework for line-rate raw packet I/O from user space
- Part of the FreeBSD head from version 9.1 onwards
- Also supports Linux (but I didn’t test it in Linux)

Netmap Cont..

- Implemented for several 1 and 10 Gbit/s network adapters - Intel, Realtek,nvidia

- Fig. - Sending ~14.8Mpps on our test server with Intel 10G card (the peak packet rate on 10 Gbit/s links)

Sending packets using Netmap Pkt-gen (packet size = 60B)

```
# date ; ./pkt-gen -i ix0 -f tx -T 1000
Mon Apr 29 09:19:31 EDT 2013
source_hwaddr [231] source hwaddr 90:e2:ba:2d:84:ac
extract_ip_range [118] extract IP range from 10.0.0.1
extract_ip_range [130] 10.0.0.1 starts at 10.0.0.1
extract_ip_range [118] extract IP range from 10.1.0.1
extract_ip_range [130] 10.1.0.1 starts at 10.0.0.1
extract_mac_range [136] extract MAC range from 90:e2:ba:2d:84:ac
extract_mac_range [151] 90:e2:ba:2d:84:ac starts at 90:e2:ba:2d:84:ac
extract_mac_range [136] extract MAC range from ff:ff:ff:ff:ff:ff
extract_mac_range [151] ff:ff:ff:ff:ff:ff starts at ff:ff:ff:ff:ff:ff
main [1035] Not using pcap on ix0
main [1055] map size is 207712 Kb in fd = 3
main [1064] number of queues 8
main [1078] mmapping 207712 Kbytes
Sending on ix0: 8 queues, 1 threads and 1 cpus.
10.0.0.1 -> 10.1.0.1 (90:e2:ba:2d:84:ac -> ff:ff:ff:ff:ff:ff)
main [1131] Wait 2 secs for phy reset
main [1133] Ready...
sender_body [617] start
sender_body [654] drop copy
main [1246] 14764960 pps
main [1246] 14882423 pps
main [1246] 14879564 pps
main [1246] 14879674 pps
main [1246] 14882116 pps
main [1246] 14880205 pps
main [1246] 14879513 pps
main [1246] 14880577 pps
main [1246] 14881788 pps
main [1246] 14880409 pps
main [1246] 14879631 pps
main [1246] 14882204 pps
main [1246] 14879753 pps
main [1246] 14880028 pps
main [1246] 14881814 pps
main [1246] 14880663 pps
main [1246] 14879296 pps
main [1246] 14880782 pps
main [1246] 14881804 pps
main [1246] 14880240 pps
main [1246] 14880247 pps
main [1246] 14880067 pps
^Csigint_h [169] Cancelling thread #0

main [1246] 756035 pps
Sent 0 packets, 60 bytes each, in 0.00 seconds.
Speed: nanpps. Bandwidth: nanbps.
#
```

In Netmap mode

- NIC is partially disconnected from host stack
- Program exchanges packets with the NIC through Netmap rings
- Netmap rings are in a preallocated shared memory region

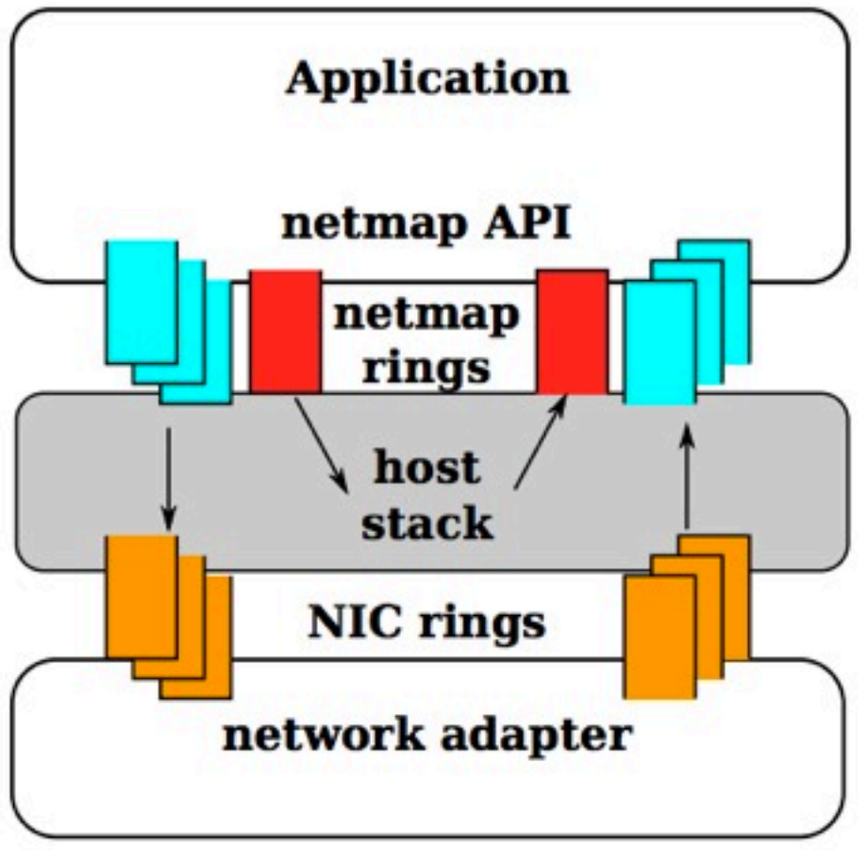


Figure 3: In netmap mode, the NIC rings are disconnected from the host network stack, and exchange packets through the netmap API. Two additional netmap rings let the application talk to the host stack.

Source- http://info.iet.unipi.it/~luigi/papers/20120503-netmap-atc12.pdf

Netmap rings

- Packet buffers
  - Fixed size (2KB), shared by userspace & kernel.
  - Buffers between *curr* and *curr* + *avail -1* are owned by userspace
- Netmap rings are like NIC rings
  - Owned by userspace except during system calls

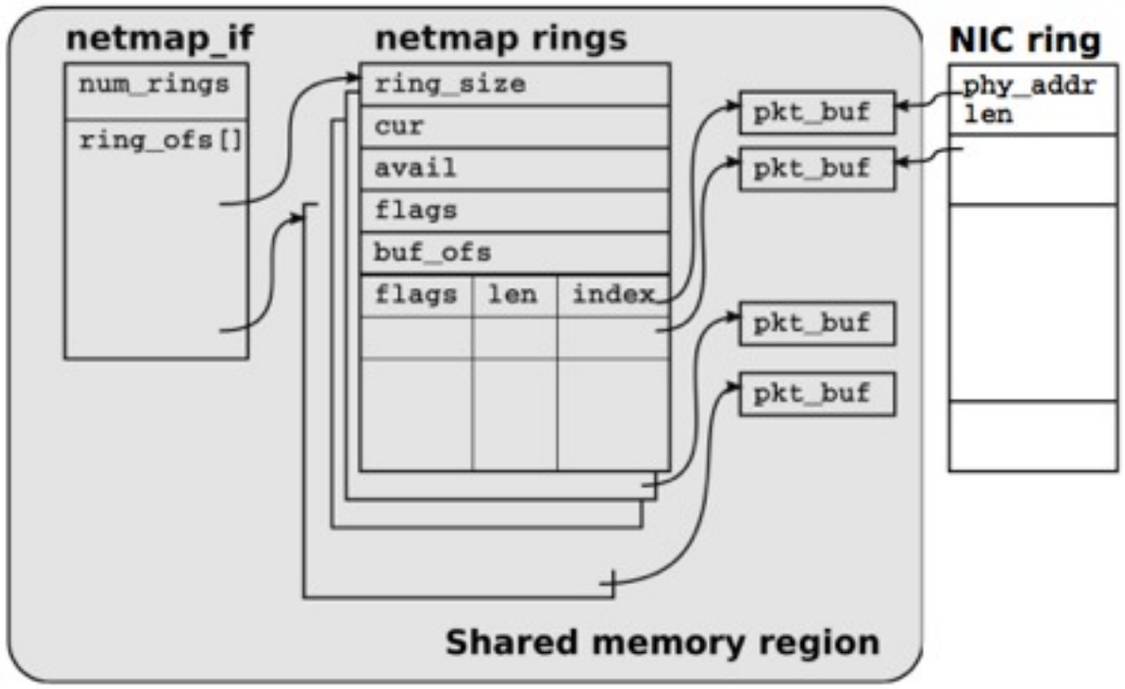


Figure 4: User view of the shared memory area exported by netmap.

Netmap API

- Netmap mode : open special device ***/dev/netmap*** and issue ***ioctl(.., NIOCREG, arg)***
- Receive side : ***ioctl(.., NIOCRXSYNC)***
  - Get info about number of packets available to read from OS.
  - Packets are immediately available through slots starting from ***cur***
- System call only validates the ***cur*** field and synchronizes content of slots between the Netmap and hardware rings

Skeleton code (Recieving packets in netmap mode)

```
for(j = 0 ; j < num_rings; j++) {
    //Looping through all rings
    rx_ring = NETMAP_RXRING(tnifp, j); //tnifp is pointer to netmap ring
    //received from NIOCREG system call
    if (rx_ring->avail == 0) //no pkts available
        continue;
    u_int cur = rx_ring->cur;
    while(rx_ring->avail != 0) { //Read all packets in this ring
        struct netmap_slot *slot = &rx_ring->slot[cur];
        rx_ring->slot[cur].len = src->ArgusSnapLen;
        char *p = NETMAP_BUF(rx_ring,slot->buf_idx);
        //Process the packer --- callback function
        src->ArgusInterface[0].ArgusCallBack((u_char *)src, hdr, p);
        cnt_pkt++;
        rx_ring->avail--;
        cur = NETMAP_RING_NEXT(rx_ring,cur); //Move to next slot in the ring
     }
}
```

Argus with Netmap

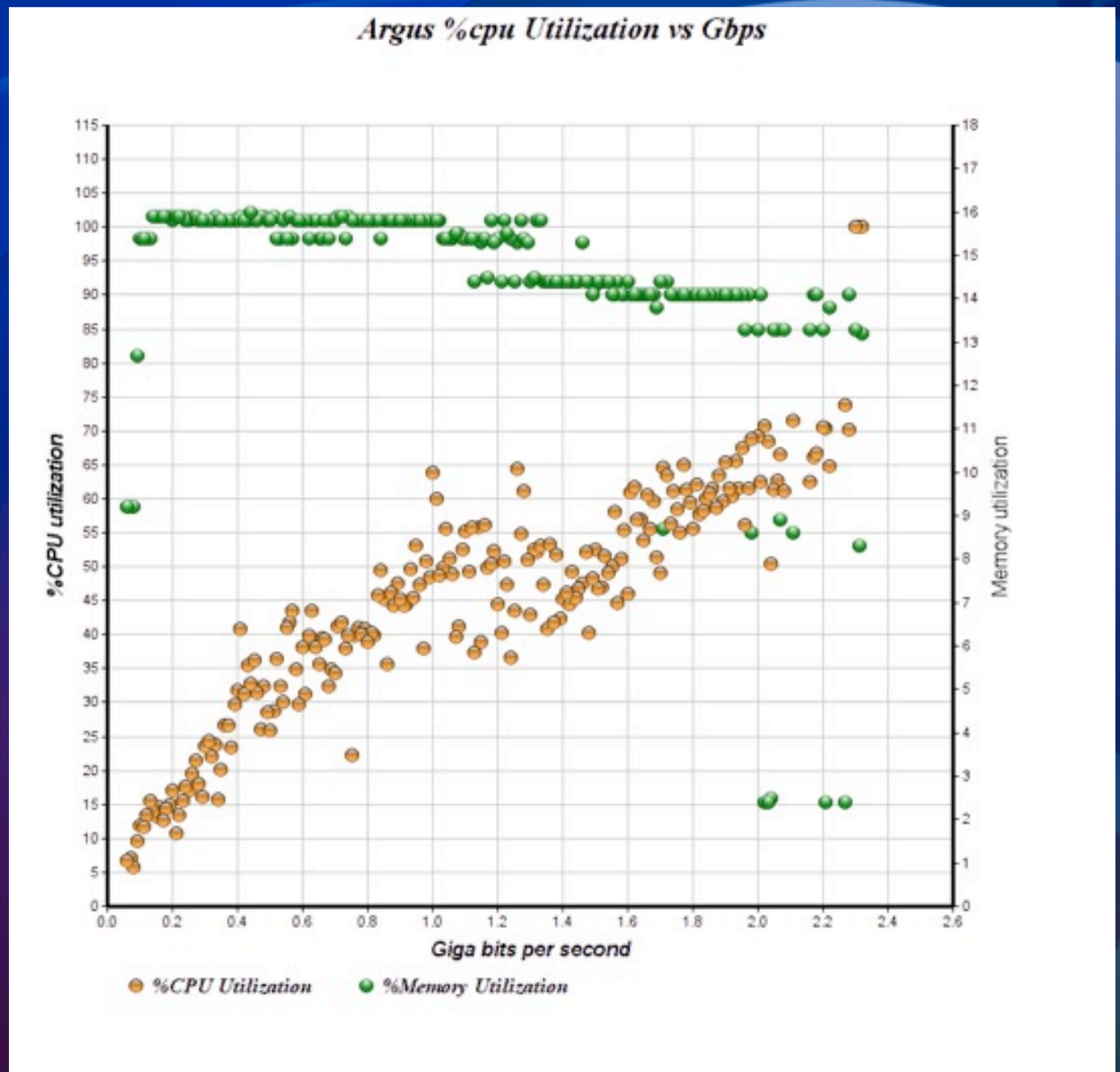
Porting Netmap on to argus

- Modified argus code to use Native Netmap API
- Server Specs: Dell R610 2 x Quad core Intel Xenon 5600 processor, 16GB Memory, Intel 10G 82599EB chipset
- FreeBSD 9.1

Performance with Netmap

Plot of %CPU and %Memory utilization Vs traffic in Gbps.

- Single threaded argus is able to capture upto ~2.5Gbps.
- This test is with traffic on Gloriad network. With avg. 2500 flow/ sec.



Next steps

- Multi-threaded argus to process packet headers from each queue separately
- Taking advantage of multiple cores. Netmap API allows for binding a ring to specific core or process

References

1. http://info.iet.unipi.it/~luigi/papers/20120503-netmap-atc12.pdf
2. http://luca.ntop.org/10g.pdf
3. http://info.iet.unipi.it/~luigi/netmap/

Thank you!