OpenSolaris Project
Crossbow: Network Virtualization & Resource Partitioning
http://www.opensolaris.org/os/project/crossbow

Sunay Tripathi	Kais Belgaied	Nicolas Droux
Sunay.Tripathi@Sun.Com	Kais.Belgaied@Sun.Com	Nicolas.Droux@Sun.Com

1 Introduction

OpenSolaris Project Crossbow provides the building blocks for network virtualization and resource control by creating virtual stacks around any service (HTTP, HTTPS, FTP, NFS, etc.), protocol (TCP, UDP, SCTP, etc.), or Virtual machines like Containers and xVM (Xen, and logical domains based hypervisor).

Each virtual stack can be assigned its own priority and bandwidth on a shared NIC without causing any performance degradation. The architecture dynamically manages priority and bandwidth resources, and can provide better defense against denial-of-service attacks directed at a particular service or virtual machine by isolating the impact just to that entity. The virtual stacks are separated by means of hardware classification engine such that traffic for one stack does not impact other virtual stacks.

○ *Crossbow Features*

The Crossbow features falls under 3 broad categories:

- **Virtualization Features** - Creation of Virtual NIC (VNIC) which is built on top of dedicated resources like Rx/Tx rings, DMA channels, kernel queues and threads , CPUs, and available bandwidth. This allows multiple Solaris Containers or Virtual machines to share the available bandwidth and host networking resources based on policies or resource partitioning without any negative performance impact due to virtualization. Virtual NICs, Etherstubs, and virtual switching are the key features that help us build Virtual Networks and higher level services like Virtual Routers, Virtual Firewalls, Virtual Load Balancers etc.

  Note that bandwidth limits, priorities and CPU resources can be associated with any data link (Physical NICs, link aggregations, etc.)

- **QoS, DiffServ, DDoS and Observability features** - Ability to separate out services and protocols and assigning them dedicated resources (such as Rx/Tx rings, DMA channels, kernel threads and queues, and CPUs) and dedicated bandwidth (both limits and guarantees) over a shared NIC without any performance penalties. Crossbow implements Bandwidth control, DiffServ, Packet steering etc by means of software and hardware classification and dynamic polling. These mechanisms allow to efficiently mitigate DDoS attacks: under attack, packets from the identified attackers (or attacking subnets) are channeled through a separate path using classification. Bandwidth for such packets can be curtailed as needed, and the offending packets are not allowed to enter the system.

  In addition, users have the ability to observe in real time or historically the traffic usage for various services and data links at no extra overheads.

- **Scaling the networking stack with threads, Cores and NICs** – Crossbow parallelizes the networking workload across multiple threads and cores using hardware or software fanout. It creates a unique path comprising of NIC hardware (Rx/Tx Rings, DMA channels, various registers), kernel threads and kernel queues, and has affinity to CPUs (cores and threads). There is no synchronization, shared locks, or shared counters between various lanes which allows the stack to linearly scale not only with cores and threads, but also as more NICs are added to the system. NIC hardware features such as multiple Receive and Transmit rings and hardware classification and fanout provide virtualization and scaling at no performance penalty.

  Furthermore, Crossbow moves the networking stack from an interrupt model to a dynamic polling model. Polling allows packets to be processed in the form of chains, which reduces per packet processing overhead, context switching, and thread pinning associated with interrupt mode. This results in increased throughput and reduced latency as illustrated later in this document.

Target Markets

- **OS/Network/Server Consolidation** - The application, network and server consolidation environments where both OS and network virtualization play a big role. This market is typically driven by the cost of owning and managing physical machines and physical networks. The sweet spot for these horizontally scaled environment are the 2-4 socket machines. From TCO perspective, these blades have only one physical NIC (1Gb or 10Gb) but are trying to run multiple virtual machines (Xen, Containers, LDoms) which have to share the NIC resources and the available bandwidth.
  The problem gets worse because for several decades we have been designing applications to go as fast as possible and congestion control is the job of the transport layer (if at all). So if one virtual machine is using UDP based traffic, then other virtual machines on the same system using TCP traffic will suffer badly. Even within same transport (TCP for instance), bulk throughput applications like ftp/http etc will have a very negative impact on interactive traffic and latency sensitive applications.
  The goal of the project Crossbow is to allow different virtual machines to share the common NIC in a fair manner without needing any intervention on the switch and allow system administrators to set preferential policies where necessary (e.g. the ISP selling limited B/W on a common pipe) without any performance impact. The traffic for each virtual machine (VNIC) is isolated from other traffic using the underlying NIC hardware itself. Coupled with VNICs, Vswitches, Vrouters, etc. people can replicate arbitrary complex physical networks in a box. Developers can debug distributed applications which send real packets over the '***Virtual Wire***' before customer deployment. Students and system and network administrators can learn networking technologies, do bottleneck analysis, etc without needing a real network.
- **Traditional QOS and application consolidation** - Existing host based QOS mechanisms are very complex to setup and typically come with a sizable performance penalty and increase in latency. The big part of the problem is the interrupt based delivery mechanism for inbound packets and the QOS being implemented by a separate layer (typically between NIC driver and IP). The network and transport layer of the host stack are unaware of the QOS layer. Packets are already delivered to the host memory by means of interrupts and the QOS layer needs to classify the packets to various queues before it can apply the policies. In case the packet can not be processed because the bandwidth usage for that class is exceeded, it sits in a queue while still

consuming system memory.

- **Network Machines** - Off the shelf CPUs are becoming fast enough to keep up with multiples of 10Gb NICs at hardware level. Sun's Niagara2 CPUs offer 2 10Gb NIC on the CPU itself and with large number of core and threads becomes an ideal platform for mid range to high end networking devices. The prime issue has been performance due to lack of scaling in different OSes. In spite of the performance issues faced by software stacks (in keeping up with small packets at line rates), Linux and its various distros has been immensely successful in low end networking device space. Crossbow is part of OpenSolaris. It improves not only single CPU performance but also the scaling across large number of cores, threads and NICs, thus making OpenSolaris an ideal platform for all kinds of networking devices. With special emphasis on forwarding, a single Niagara2 CPU with Crossbow can deal with forwarding 64 bytes packets across 10-12 1gigE NICs or a few 10 Gb NICs close to line rates.

  One of the major goals of Network Machines effort (using Crossbow as technology differentiator) is to offer Opensource software that runs on off the shelf hardware and provides basic functionality like routing, load balancing, firewalling along with the technology necessary to do management (both web and 'cli' based), fail-over (using VRRP etc) such that developers in the network space can focus on building bigger and better services instead of re inventing the wheel every time they want to offer a new functionality.

  Coupled with Network and Server Virtualization features in OpenSolaris, building multi tenancy and overlay networks (without needing specialized hardware) becomes a low cost exercise.

Availability and Licensing

Project Crossbow is an Opensolaris project and is available at
http://www.opensolaris.org/os/project/crossbow

At the above link, you can find source code, iso images, bfu archives, and live CDs. Various Design documents and presentations are available at
http://www.opensolaris.org/os/project/crossbow/Docs

As of July 2008, Crossbow had finished 3 rounds of beta testing and is already in use in various external deployments. It is targeting integration in regular OpenSolaris release by late 2008. Check the OpenSolaris website above for participation, accessing bits and latest schedule etc.

Licensing

Various aspects of Crossbow technology are patent protected under US and International patent laws. The source code is available under OpenSolaris 'CDDL' license and all users of OpenSolaris (of which Crossbow is a kernel component) are free to use the code or source as per the license. 'CDDL' license details can be found at http://www.sun.com/cddl. Contact the author if you desire to use the technology, source code or binary outside of 'CDDL' framework.

2 Technical problems in existing architectures

As mentioned earlier, the host based QOS systems work as an extra layer between the network and the stack and as such are pretty inefficient in providing the QOS services required of them. But that is not all. The existing interrupt driven packet delivery model precludes any kind of policy enforcement and fair sharing. When a NIC interrupt is raised, it is at a high priority and the CPU has to context switch whatever its processing to deal with the interrupt. Very often, the processing of a critical packet is interrupted to deal with the arrival of a non critical packet.

The anonymous packet processing in the kernel is another major problem in virtualizing the stack and enforcing any kind of bandwidth resource control (including fairness). 80% of the work is already done for an incoming packet when the stack determines that no one is actually interested in the packet and it needs to drop it. In other words, the cost of dropping unwanted packets is too high. Everything in the host flows through common queues and is processed by common threads which make enforcing policies based on traffic type very difficult. Receiving or transmitting each packet impacts processing on any other packet on that particular CPU.

In most of the virtualized environments, the pseudo NIC in the virtual machines has no way of knowing about the hardware capabilities of the real hardware (even simple things like hardware checksum). Additionally, there is no mechanism to share the NIC in a fair manner. The transition of typical packet from the dom0 to domU also causes severe performance problems.

Modern NICs vendors have been aware of the advent of virtualization. Most 10G NICs offer multiple transmit and receive queues, hardware level packets classification, multiple interrupts `and multiple MAC addresses, etc. The networking stack is lagging behind. It has not evolved to discover and utilize those capabilities.

3 Virtualization Components and Resources being Virtualized

The aim of Crossbow's architecture is to meet the requirements which range from just needing a set of dedicated resources (both hardware and software) all the way to full Virtualization (which includes dedicated resources, data structures, configuration and tuning). Before describing the architecture, we'll define in this paragraph what are the resources in question that can be virtualized by Crossbow and what are the broad categories for Virtualization:

○ *Resources that can be virtualized*

As far as network processing is concerned, we need to look from top to bottom to understand what are the useful resources as far as network processing is concerned:

- The NIC hardware itself - Consists of the hardware Receive and Transmit queues (know as Rx and Tx rings), DMA channels, hardware Classifier, MAC addresses, interrupts, and any other registers than can cause contention.
- The driver - The NIC driver typically maintains a set of buffer (commonly known as receive and transmit descriptors) and any other tunable.

- The MAC and Data Link Layer - Any data structures, queues, locks, and processing elements. The most common ones are the software receive and transmit queues (soft rings) and their processing threads, software classifier. All these together constitute a 'pseudo hardware layer' which mirrors a real NIC hardware and can be used where NIC has limited or no virtualization capabilities.
- IP Layer - The connection classifier (known as IP classifier), the serialization queues ('squeue'), the worker threads for the 'squeue', other IP data structures like 'ill_t' and any other tuning.
- The Routing tables - Today there is one common routing table per Solaris instance that can be virtualized if needed.
- Transport layer - Most of the transport layer code is processed either in context of application threads (on transmit), 'squeue' worker threads (both transmit and receive) and interrupt or soft ring threads which are all covered above (except application threads which are dealt with ensuring the application runs on dedicated processors). So the only thing we need to virtualize in transport layer is transport specific tuning and statistics.
- The available bandwidth - The issue is more of sharing the available bandwidth in case multiple application, containers, or virtual machines exist on the same physical machine.
- The processing elements - Apart from the queues themselves, its necessary to have dedicated CPUs per traffic type on which the various thread processing the traffic can run. This includes the CPUs where interrupts and various worker threads for a traffic type can be bound to.
- All networking configuration - The various configuration files which deals with assigning of IP addresses, netmasks, default routes, DNS servers, etc.

○ *What is Crossbow Virtualizing*

The answer depends on forms of Virtualization. If someone is just trying to do application consolidation i.e. having two very different application live on the same machine without interfering with each other, we can get away with less. The full Virtualization which entails separate virtual OS instances (likes of Xen and LDoms) will require more and in between Virtualization like Solaris Containers (or Zones) will require even more Virtualization.

The common part: In all cases, having dedicated queues, dedicated NIC hardware and dedicated CPUs to run the worker threads are the most important things that need to be addressed. Crossbow ensures that IP layer and below (including 'squeues', buffers, queues and threads in data link layer, driver and NIC hardware) can be virtualized. In addition, sharing and control over the available bandwidth and dedicated CPUs to run the interrupts and worker threads processing any particular traffic type (application or virtual machine related) is provided by Crossbow.

In some cases, only dedicated resources are needed while in other cases full Virtualization is required. We split the usage in 3 broad categories:

- **NIC** – The physical NIC is being virtualized so that a ***Virtual NIC (VNIC)*** can be created which has dedicated resources (as mentioned above), has security isolation derived from the hardware itself and does not face interference from traffic on other VNICs and NICs.
- **Solaris Containers** - This is the most complex of all scenarios. There is only one kernel with one virtual memory space and we need to virtualize the stack from top to bottom including

configuration for each Solaris Container. Apart from the "common case" mentioned above, we need to virtualize the transport layer, routing, configuration and have dedicated NIC resources, queues, bandwidth and CPU for each Solaris Container. The IP layer and above part has already been implemented as 'exclusive IP stack' for Solaris Containers or Zones.

- **Full Virtualization (Xen and LDoms) -** The "common part" applies here as well where we can create VNICs for each guest OS and dedicate bandwidth, CPU and NIC resources for the guest OS. In addition, both Xen and LDoms need additional pieces to map a NIC Rx/Tx rings directly on to the guest OS to avoid overheads of going through the host OS or service partition.

○ *Administrative Components*

Crossbow has following administrative components that a System Administrator needs to deal with

- Virtual NICs (VNICs) – As part of Crossbow, we virtualize the hardware and software resources, kernel threads and queues and bind them together under Virtual NIC. The VNIC is just like primary NIC (or legacy interface) which can be 'plumbed' using 'ifconfig'. For all practical purposes, there is no difference between NIC and VNICs. The VNICs are plumbed like legacy (Solaris 10 and before) NICs, and assigned IP addresses (static or DHCP) in the same manner. The MAC layer implements the VNICs where the inbound packet switching is done by hardware. The MAC pseudo hardware layer which implements a software classifier as well does the switching for packets between VNICs, broadcast and multicast packets.

  In Crossbow terminology, all NICs, VNICs, VLANs, Aggregations, etc are termed as '**Links**' (all L2 entities)

- All VNICs created over the same Etherstub, physical NIC or aggregation are connected by an L2 virtual switch implemented by the MAC layer. Virtual switching will distribute multicast and broadcast packets between VNICs and the wire. The virtual switch is also VLAN aware. Etherstubs allow users to create virtual switches that are independent from hardware.

- VLANS as VNICs – Crossbow allows VNICs to be assigned a VLAN tag. With Crossbow, VLAN data links are now transparently implemented as VNICs. This allows resource controls such as bandwidth limits to be assigned to VLAN data links.

- Exclusive IP stack (for Container) – Already implemented and allows a virtualized TCP/IP stack to Solaris Containers. Without rest of Crossbow, it requires a dedicated NIC or VLAN interface and doesn't support various properties and separation mentioned below.

The Link layer virtualization components like VNICs, etherstubs, etc are administered using '***dladm***' command. dladm allow the following data link properties to be specified or modified

- bandwidth limit or link speed
- processing priority (can be Real Time)
- cpu list on which both Rx/Tx processing will take place (can be dedicated CPUs)
- degree of parallelism (or fanout)

4 Flows (Service consolidation, QoS and DDos Defence)

Crossbow helps with the service consolidation as well. Two services with different needs can co exist on the system without interference from each other and can have dedicated resources from the NIC, kernel Queues and threads, CPUs, and Bandwidth across shared Links. In Crossbow terms, anything configured using 'flowadm' that is based on Layer 3 and/or Layer 4 classification rule is called a '***Flow***'. Since the services are always specified over a link using '***flowadm***' command, the system has unique L2-L4 information to do classification without any ambiguity.

Types of '*flows*'

- Service based (Service Consolidation) - This is the scenario where a particular application needs dedicated set of resources. Typically in case of critical apps, latency sensitive applications or applications which need to be limited, a 'flow' is created with a simple classification rule consisting of a protocol/transport and port (see /etc/services on how to identify standard services).
- Protocol based – Flows around just the transport information to give a preferential or policy based treatment to each protocol. Note that when a link is configured, it automatically separates out the protocols to avoid interference and give fairness between TCP and UDP traffic for instance.
- IP address and Subnet based – Flows can be configured on remote or local IP address or remote/ local IP subnet (both IPv4 and IPv6). This feature is used extensively in DDoS defence. When an IDS system (or system administrator) identifies an attacking IP address or subnet (or a particular service is under attack), a flow can be created around it with minimal or zero bandwidth where offending packets don't even come inside the system.
- Differentiated Services – Flows can also be created around DSCP labels to allow differential treatment to various services.

Note1: As per current implementation, to avoid ambiguity between different types of flows, the flows are mutually exclusive from each other on a per link basis i.e. Users can have multiple service based flows but then they cannot have a transport based or IP address based flow (as there is overlap between the two types).

Note2: As per current implementation, the flows are configured using software classifier only and do not get dedicated hardware resources. Both these limitations will be relaxed in future.

The following QoS properties can be configured on a per flow basis

- bandwidth limit
- processing priority (can be Real Time)

Flows and link activity can be seen in real time using 'netstat'. A new class was added to the Solaris Extended Accounting framework which allows users to see utilization history as well.

5 Performance and Scaling

○ *Single CPU Performance Improvements*

Crossbow architecture introduces '***Dynamic Polling***' where instead of per packet interrupt, the Link (NIC or Virtual NIC) is switched between interrupt and polling. Under load, the link is normally in poll mode and a kernel thread goes down to hardware to fetch all packets in the form of a packet chain.

Throughput – Since, under load (bursts etc) packets are delivered in chains via poll mode, the cost of going through the stack is amortized across the entire chain. Also, in polling mode, since no interrupts are coming, overheads due to context switching, mutex contention, thread pinning don't exist. The higher the packet per second rate, the more throughput improvements are seen. In addition, since the hardware already does the classification and we know packets are unicast packets meant for the links MAC address, the entire Data Link processing can be bypassed reducing the overheads further.

Effects of Dynamic Polling

Mpstat output for a link pre Crossbow

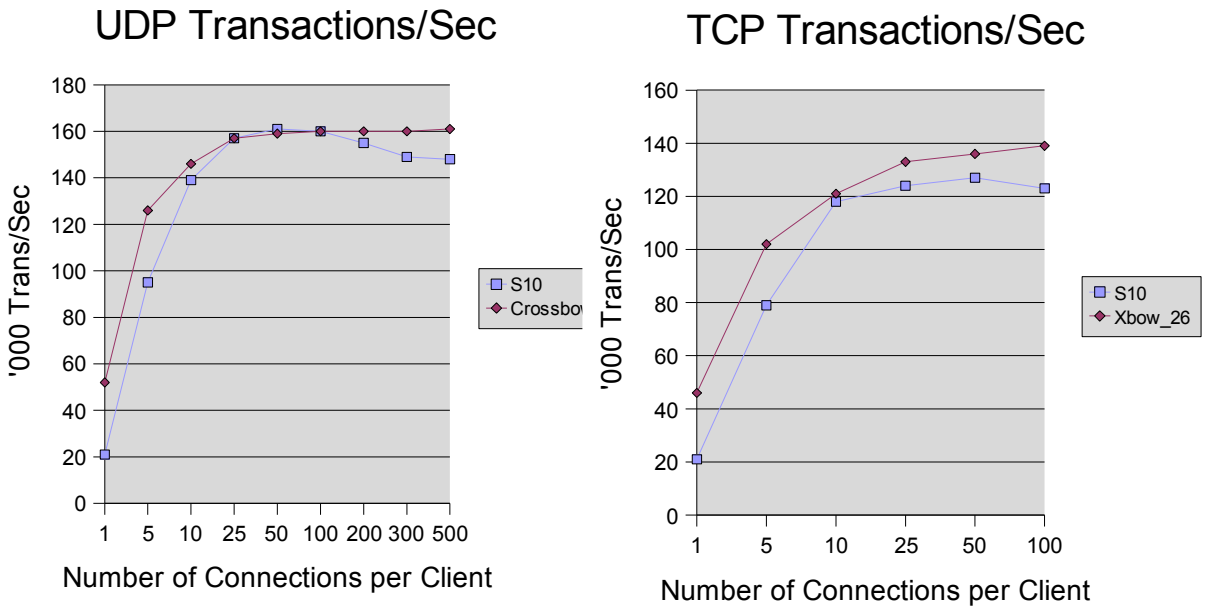
```
intr   ithr  csw   icsw  migr smtx  srw  syscl  usr sys wt idl
10818  8607  4558  1547  161  1797  289  19112  17  69  0  12
```

Mpstat output for a link post Crossbow

```
intr   ithr  csw   icsw  migr smtx  srw  syscl  usr sys wt idl
2823   1489  875   151   93   261   1    19825  15  57  0  27
```

Notice the drop in interrupts/sec (1st column); Context Switch (2nd column), etc

Latency – As mentioned above, because of '***Dynamic Polling***' and hardware classification, the per packet processing cost is reduced dramatically. This effect gets highlighted when latency is measure under load. The graph below shows the transactions per second (request/response) for 64 bytes packets. The system under test is a 2 socket dual core AMD processor based machine (Sun X4200) with single e1000g NIC. The X axis (showing the load) starts with a single thread per clients (3 clients in this case – each 4 socket dual core AMD processors with Intel e1000g NICs) and keeps increasing the number of threads per client. The Y axis measures the total transactions per second. The graph shows Solaris 10 behavior vs Crossbow/OpenSolaris behavior. Even when we get CPU limited on the system under test (the clients never get CPU or bandwidth limited), the transaction per second rate never drops with Crossbow while Solaris 10 (and other OSes) would see a downward slope with increased load.



- ***Scaling with threads and Cores***

Some of the NIC hardware (Sun Neptune PCI-E NIC which is also present on Niagara2 chip in both 1Gb and 10Gb link speeds) have the feature where multiple Rx and Tx rings can be assigned as target of a classification rule. Packets matching that rule (say MAC address for a link) are further distributed within the assigned Rx rings using a hash of source IP + port. This allows the incoming traffic to spread out to multiple Rx rings on a per NIC basis. In the High Level Architecture we see that all Crossbow data structures, threads and queues have a unique mapping with the Rx ring. This allows the packets chains to move through the system in a parallel manner and allows the networking stack to scale linearly with the number of threads and CPUs.

Even when the NIC does not support Virtualized Receive Side scaling features, the Crossbow pseudo layer implements the same functionality and allows even the legacy NICs to scale almost linearly.

6 High Level Architecture

The Crossbow architecture starts out by integrating network Virtualization and resource control as part of the stack architecture. The Solaris 10 network stack has already been designed for the next decade where the connection to CPU affinity is maintained and the upper stack has tight control over the NIC resources.

Crossbow builds on top of S10 stack by pushing the classification of packets based on services, protocols or virtual machines as far down as possible. If the NIC hardware itself has ability to divide onboard memory into segments/queues (know as Rx and Tx rings) which can preferably have their own

DMA channels and MSI-X interrupts, the stack programs the NIC classifier to classify incoming packets based on configured policies to different Rx rings. In case, the NIC doesn't have Rx/Tx rings and hardware classification, Crossbow Virtualized MAC layer also implements a '*pseudo hardware layer*' which consists of a more powerful software classifier, soft ring sets and soft rings which provide the same functionality as NIC hardware classifier and Tx rings but processing happens on associated CPU.

The Rx/Tx ring, the associated DMA channel, MSI-X interrupt, the serialization queue, the CPU, and processing threads are all unique for the service, protocol or virtual machine in question and can be assigned a unique MAC address and a Virtual NIC which becomes the administration entity that can be administered like a normal NIC. The NIC classifier drives the incoming packets to the correct Rx ring from where the 'SRS' (per link queue in MAC layer) owning the Rx ring (and VNIC) will pull the packets via polling mode based on fair sharing of resources or configured bandwidth. The interrupt mode is used only when the SRS has no packets to process and the Rx ring is empty. Each individual Rx ring is dynamically switched between interrupt and polling mode. Incoming packets that exceed the configured bandwidth limit remain in the NIC itself in their corresponding Rx ring and are pulled in the system only when they are ready to be processed. More details are given further in this section.

The creation of an administrative entity (VNIC) is optional and typically associated with a virtual machine like Solaris containers, Xen or LDoms. For application or protocol based resource control, a separate data path is created to provide the isolation and resource control but a VNIC is not needed.

Figure 1 shows (in blue) the various Crossbow components and their interaction with each other and other layers and administrative entities in Solaris.

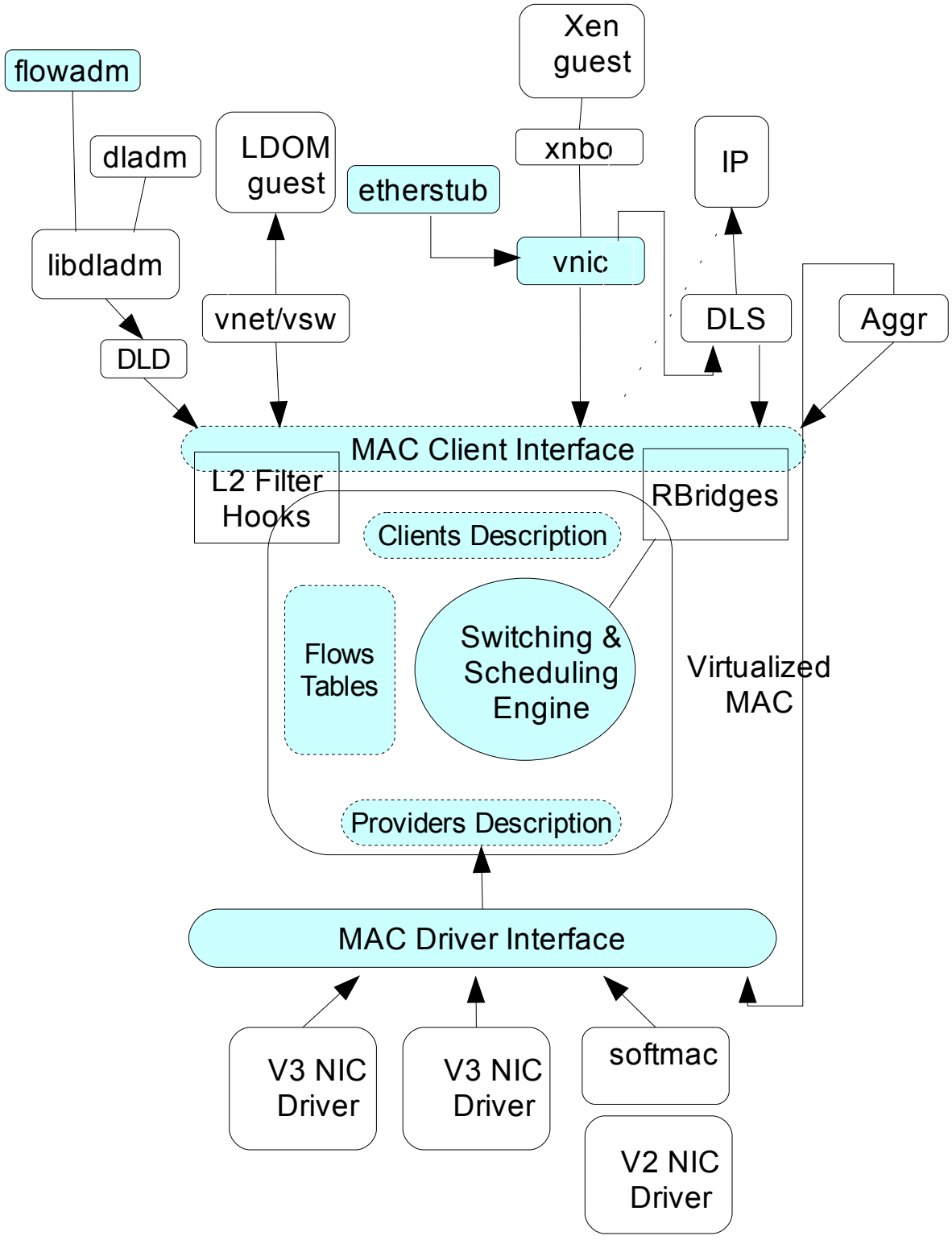


Figure 1

Key Data Structures and their relationship

Lets look at some key data structures and there relationship which is necessary to understand the high level architecture. These data structures are on per link basis:

- '*mac_client_impl_t*' - Each 'link' (a NIC, VNIC, aggr, VLAN, etc) is a client for MAC layer and is represented at the MAC layer by a unique data structure called '*mac_client_impl_t*'. This serves as a handle for control messages and various administrative operations.
- '*flow_entry_t*' - The classification rule (for example MAC address in case of NICs and VNICs; transport and port information for services; IP addresses or subnet for DDoS defence or flows based on subnets) and properties are contained in a '*flow_entry_t*' data structure. Both the hardware and software classifier are programmed with the same rule so the software classifier backs the hardware classifier and serves as the loopback path for VNIC to VNIC communication.
- '*mac_soft_ring_set_t (aka SRS)* and *rx_ring_t (aka Rx ring)*' - Each classification rule can point to one or more Rx rings (where packets are distributed to multiple Rx rings per classification rule using a hash on source IP + port for even greater scaling – this feature is supported by only few NICs like Sun's Neptune which is also on Niagara2 chip). The hardware Rx ring is represented in MAC layer by a '*rx_ring_t*' data structure. The *mac_soft_ring_set_t* and a NIC Rx ring (where present) have a 1 to 1 mapping and its the *SRS* which implements dynamic polling and B/W control by means of a queue and separate poll and worker threads. The same data structure is used on Tx side where it provides the outbound B/W control and flow control to upper layers.

  Even for older NICs like Intel '*e1000g*' and broadcom '*bge*' which don't have hardware based virtualization, we treat them as NICs with only one Rx/Tx ring and 1 classification rule and we do hardware based polling as long as there is only 1 link configured on the NIC. When a 2nd Link gets configured on these NICs, we disable dynamic polling and depend totally on software classification.
- '*mac_soft_ring_t*' (aka soft ring) – For Receive side, once the packet chain arrives in the *SRS* or '*mac_soft_ring_set_t*' either via interrupt path or poll path, and any B/W limit is enforced, packets are ready to move up the stack. Since the packets are for the entire link, we separate them out based on higher level protocol (IPv4 TCP, IPv4 UDP, Rest including all IPv6). We
- use a '*mac_soft_ring_t*' data structure (commonly known as soft ring) which has a queue and a worker thread for this purpose. The SRS and its associated soft ring act as one queue system making sure that packets get through when there is no backlog and even when there is backlog, they get queued at most once. This separation is necessary to avoid interference between different protocols and also for performance reasons. Specifically:
    - TCP does synchronization and mutual exclusion by means of its own serialization queue called 'squeue'. The 'squeue' like to do polling on receive side for performance and as such need a separate holding queue in the MAC layer which has only IPv4 TCP packets. Also, we don't need to do any data link layer processing so we can bypass the entire DLS layer.

- UDP packets can be sent up without any restrictions but for IPv4 UDP, we can bypass the DLS layer and small packet UDP has different performance requirements so we can't mix TCP and UDP together.
- We leave the rest of the IPv4 packets, broadcast and multicast packets and all IPv6 packets in a separate soft rings which goes through the data link layer because it needs additional processing (some options in case of IPv6).

Note that each soft ring worker thread is bound to a separate CPU where possible for performance reasons. In case, a cpu list is provided per link or a degree of fanout is specified, we create a set of 3 soft ring per cpu (as many as degree of fanout or cpus in the cpu list). The TCP packets are load balanced with the TCP soft rings based a hash for source IP + port.

- 'vnic_t' – This is a data structure specific to a VNIC and contains the VNIC specific information.

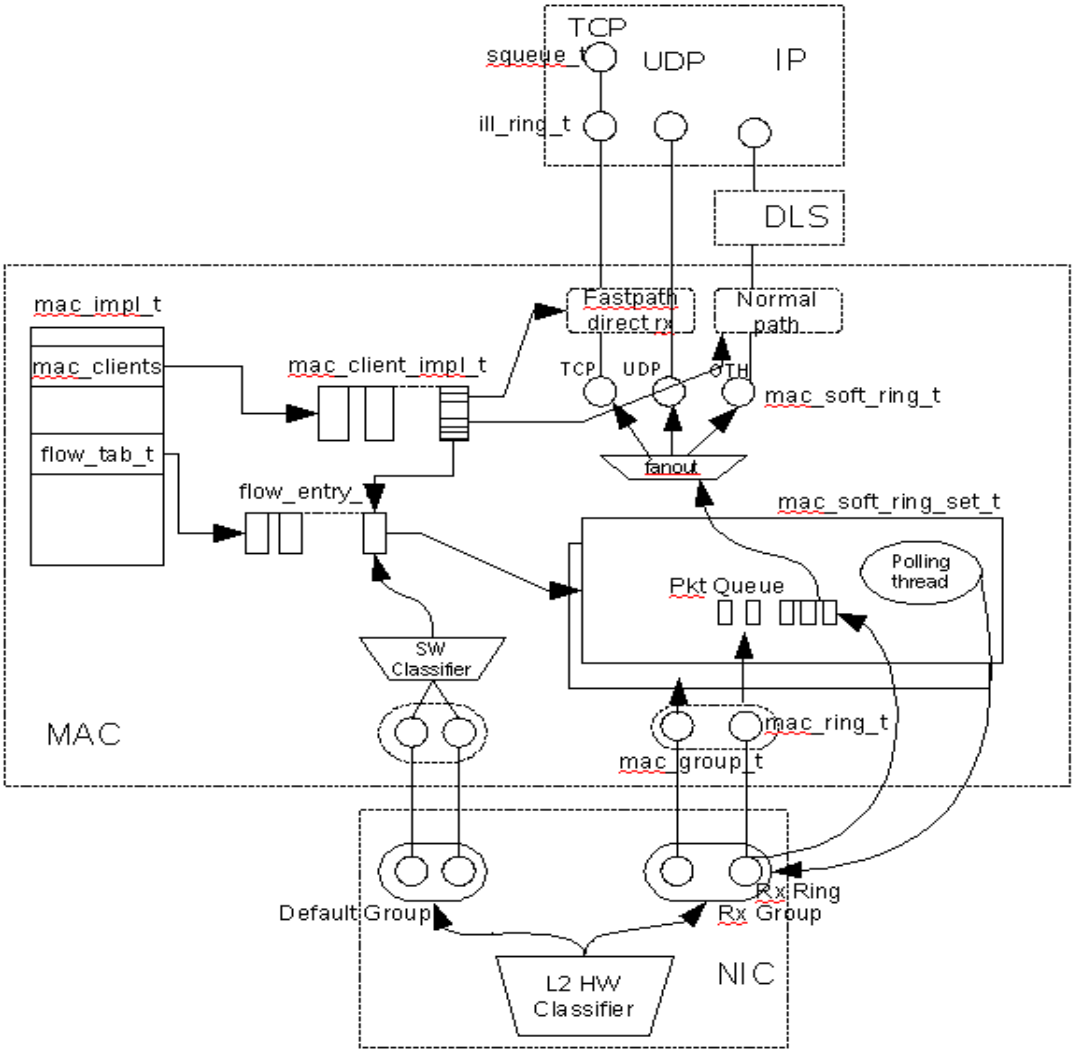


Figure 2

Figure 2 shows the relationship between various data structure and data path.

The entire layered architecture is built on function pointers know as 'upcall_func' and corresponding 'upcall_arg'. Every layer provides a pointer of its recv function as 'upcall_func' and a context as 'upcall_arg' to the layer below. This is how the packet path is constructed. Any layer can short circuit itself out by providing the 'upcall_func' and 'upcall_arg' of the layer above to layer below (and same for transmit side if needed). All context cookies for a layer work on reference based system when each layer pointed to it gets a reference and ensure that data structures don't get freed till all references are dropped.

Primary NIC

This is typically the NIC which is assigned the primary MAC address and is also linked to the device node attached to the physical hardware. The primary NIC represents the pre Crossbow interface (typically e1000g0, bge0, etc).

Virtual NIC

The Crossbow components can be separated into link level components and flows (which are implemented on top of Links). As of now, only link level components can benefit from hardware features including hardware classification. Configuring a new link via the 'dladm' command results in the creation of a '*mac_client_impl_t*', a '*flow_entry_t*, and *Rx and Tx SRSs* (with their associated soft rings). In case of a VNIC, a '*vnic_t*' is also created and all control operations on the VNIC are done through the VNIC device node. If the NIC is virtualization capable, Crossbow tries to assign one or more Rx rings (leaving the default Rx ring for unclassified traffic) as well and program the hardware classifier with the link MAC address to point to that Rx ring. The SRS associated with the Rx ring brings the packets into itself by dynamically switching between interrupt and polling mode and enforces any necessary B/W control. Link level promiscuous mode is also dealt with at this time. At this point, packets are separated into their respective soft rings and move up the stack.

We also program the software classifier with the corresponding '*flow_entry_t*' and SRS as the argument. Incoming packets via the default Rx ring (or in case no hardware classification) pass through the software classifier when packets for the link

are queued to the SRS associated with that link. The software classification rule always serves as a fall back path and allows the MAC layer to reassign the Rx ring for other purposes when necessary. The traffic for that link continues to flow via the software path.

Flows

The layer 3 and 4 level flows are created by means of '*flowadm*' command and are always created on top of links (NICs, VNICs, etc). The flows are also implemented by means of a '*flow_entry_t*' and SRS but flow processing is done afte+r the link level SRS has done its processing.

Virtualized MAC and MAC clients

Each link is a MAC client. Its '*mac_client_impl_t*' is the MAC handle used by both clients and drivers to identify a device. Each MAC client is associated with

- Receive callback function

- Set of unicast and multicast addresses, VLAN id
- Statistics
- Link state
- Promiscuous callbacks
- Rings, bandwidth limit, priority, set of CPUs, fanout

Keep in mind that entire set of data structures, queues and threads and packets flowing through them is on a per virtualized link basis. There are multiple links operating on their own CPUs and Rx rings in parallel and independent from each other. Figure 3 shows the incoming packet stream and how classification separates it into multiple virtualized packet streams on per link basis.

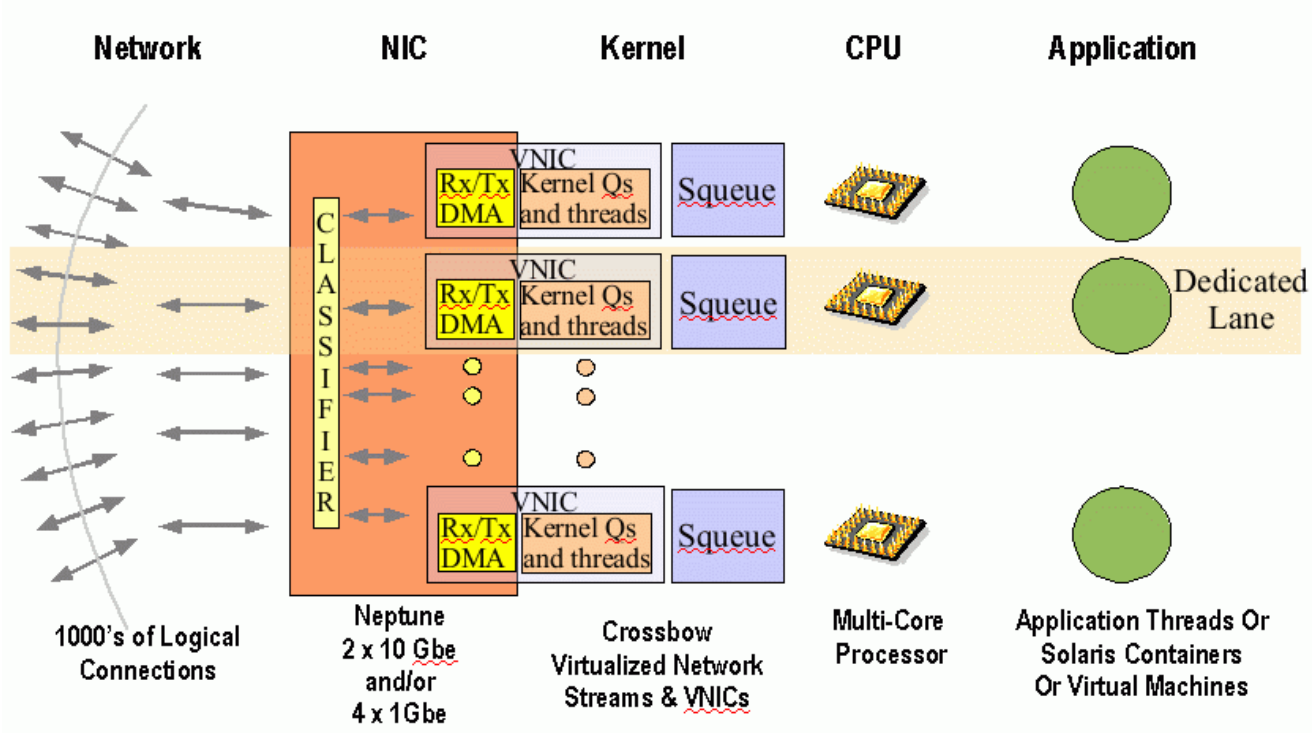


Figure 3

The pseudo Hardware Layer

In case where hardware doesn't have the necessary resources/capabilities or has exhausted them, the Crossbow MAC layer provides similar (and richer) functionality. In Crossbow terminology, this is

called The Pseudo hardware layer (part of MAC layer) that comprises of the hardware features and software feature together which offer unlimited classification and Rx/Tx ring capability. This allows the Crossbow functionality to be offered irrespective of Hardware capabilities. When available, hardware capabilities are used to offer better performance.

7 Hardware Resources and Classification

- Receive rings

  On the receive path, some drivers may also support a fixed or configurable number of receive rings. Incoming packets can be deposited into these rings in parallel. Packets can be picked off each ring independently. Receive rings have a finite capacity. When full, it is possible to specify the desired behavior, whether to drop packets destined to that ring, or to overflow to another ring.

- Receive Ring Groups

  One of the advanced virtualization features is the ability to bundle multiple Receive Rings in a single group. One or more MAC addresses may be assigned to a group. Incoming packets destined to the group's MAC address are delivered to any ring member, according to a programmable or predefined RTS policy (Receive Traffic Steering). Note that member rings are still dynamically polled individually and controlled by individual SRS. Rx ring groups can come with a predefined set of member rings, or they are programmable by adding and removing rings to/from them.

  Within a group, packets are delivered to individual Rx rings according to a Receive Traffic Steering policy. The choice of recipient ring can be made based on a Receive Load Balancing, a programmed High Level Classification, or a cascade of both.

- Receive Load Balancing

  Some NICs are able to balance the incoming flow of packets between Receive Rings based on a predefined or programmed policy. This is a feature particularly useful when a single CPU is not capable of handling the full line speed. Spreading the inbound traffic to multiple channels allows multiple processors to collaborate in receiving the packets. The ring selected for depositing an incoming packets is typically based on the value of a hash function computed over L3, L4 or other sections of the packet headers.

- Transmit rings

  Transmit rings (also referred to in some NICs specifications as Tx FIFOs or Tx Queues) allow sending multiple packet flows through the same device in parallel. A Transmit Ring is a software concept that refers to a set of transmit descriptors used for outgoing packet(s). The device driver notifies the hardware to begin reading the descriptors from system memory when the packets are ready to be sent. Hardware may DMA the content of descriptors to hardware descriptor queue and eventually DMA the packets data and start the transmission.

  Drivers supporting multiple transmit rings register the list of their opaque handles.

- Multiple Transmit Rings Groups

  Similarly to Rx rings, Transmit Rings can be bundled together. A Transmit Rings Group is a set of transmit rings with same capabilities (hardware checksum, LSO, etc.) and a Transmit Load

Balancer.

- High Level Classification

  Classification of incoming packets may be based on:

  - L2 criteria (MAC addresses, VLAN ID, etc)
  - L3 criteria (IP source and destination, IPv6 flow ID, protocol number ...)
  - L4 criteria (e.g. transport port number)
  - Combination of all.

  The outcome of the classification is depositing the packet that matches the criterion (also referred to as rule in some NIC specifications) into the programmed Receive Ring. Packets that do not match any classification rule are deposited into the default ring of the ring group.

8 Receive side Data Flow - Dynamic Polling

Each SRS is responsible for scheduling the receive side packet processing from its Rx ring. The packets arrive in the queue associated with the SRS by means of both interrupt and poll path. The SRS shares a common counter between itself and its soft rings which tracks the total packets queued in the system. This counter is used to decide when to switch the Rx ring between interrupt and polling mode.

As long as there is a backlog, the SRS keeps its Rx ring in poll mode. The poll thread is woken up to pick the packet chain from the Rx ring as soon as the backlog dips below a low water mark. If the backlog is below the low water mark, the poll thread continues to process the packet all the way to the upper layer (e.g. IP) without any context switch. The same applies for interrupt mode. If a backlog builds up, both poll or interrupt thread queue the packet in the respective soft ring where a soft ring worker thread takes up the packet processing.

The SRS poll thread and the SRS worker thread operate in a mutually exclusive manner. The SRS queue and SRS worker thread is used only if Bandwidth control is enforced. The SRS switches the Rx ring between interrupt and polling mode by means of special driver functions provided as part of MAC/ Driver API discussed ahead.

9 Transmit Side Data Flow

The transmit side data path also has a Tx SRS which is primarily pass through. The SRS is needed in case the Tx ring underneath runs out of transmit descriptors in which case it provides the flow control to upper layer and also a queue. Since the MAC layer transmit routines are available to MAC clients via direct function call, the MAC client API also provides mechanism for the client to specify whether packet needs to be dropped or queued when the Tx ring is flow controlled. The Tx SRS queue gets into play if a link has a bandwidth limit configured which is enforced by the Tx SRS.

If there are more than one Tx rings assigned to the link, then the Tx SRS is also accompanied by a soft ring (per Tx ring) where if a particular Tx ring gets flow controlled due to running out of descriptors, the packets get queued in the respective soft ring itself. In this scenario, the SRS chooses a Tx ring/soft ring by computing a hash on connection identifier so packets for same flow go out via same Tx ring and there are no reordering issues.

10 MAC/Driver Interfaces

In Solaris 10, a new framework called GLDv3 (Generic Lan Driver v3) was added to help IHVs write high performance device drivers to Solaris in a simple manner. The driver, as part of this framework, registers a few functions with the MAC layer that primarily deal with the hardware (transmit routine, receive routine, etc). The driver also gets in return the MAC layer routines (like '*mac_rx*') which needs to be called from the driver.

Crossbow extends the GLDv3 framework and categorizes the NIC hardware into

- Legacy NICs which have no virtualization capabilities
- NICs which have more than 1 Rx, Tx ring and are at least capable to MAC address based classification.
- NICs that support MAC_CAPAB_SHARES capability for guest domains direct access to some of the NIC's hardware resources.
- NICs capable of Layer2, Layer 3 and/or Layer 4 classification.

Additionally, drivers need to register the appropriate functions based on their capability level. For instance, drivers of the second category above have to register the Rx, Tx rings and any groups along with functions necessary to assign a MAC address to a ring/group and allow dynamic polling on a per ring basis.

11 Miscelaneous

Link Aggregations

Link aggregations were introduced by project Nemo as a way to scale up the bandwidth available to the network by aggregating multiple links connected to ports of the same switch. The inbound and outbound load is balanced between the ports members of he aggregation. This also assures higher availability through redundancy.

Link aggregations are constructed in a similar manner. The architecture was modified to use the new MAC client API between the aggr driver and each one of its members. The aggr driver was also ported to the new MAC driver interface.

Figure 4 below describes the components involved in the building and the functioning of an aggr. Note that the Outbound load balancer and the Rx handler for the data and LACP control messages are not significantly modified by Crossbow.

The aggr driver has a dual role. It still acts as a MAC driver for the MAC layer above it. This means that mac_register() is called for each instance created of the aggr pseudo device, and a mac_impl_t structure is allocated for it. The aggr driver interacts with MAC through the MAC driver interface (also referred to as the MAC provider interface) for advertising and negotiating capabilities, for transmitting and receiving data, and for reporting status notifications. No architectural changes were made for that aspect of aggr, beyond the porting to the modified provider interface. The main novelty for the aggr driver is the fact that it now is a consumer of the MAC client API. It opens an exclusive client to each port member of the aggregation, maintains a handle for that client, and uses the handle for all data and control operations with the member.

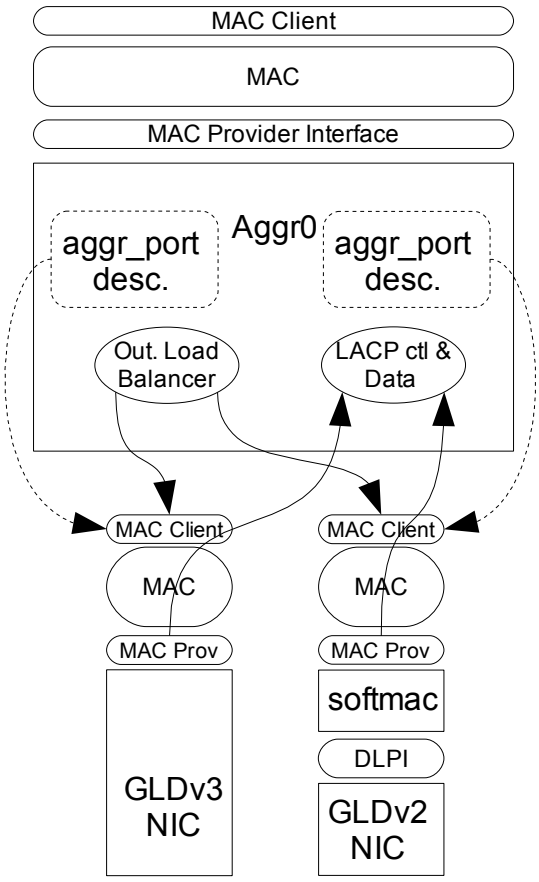


Figure 4

The aggr driver uses a project private MAC client function to query about its members support of multiple transmit and receive rings. Internally, the aggr driver keeps a pseudo-ring for each hardware ring from its members (mac_ring_t structure described above). In turn it advertises that pseudo ring to the MAC above it. The access routines for an aggr pseudo ring (polling and interrupt enable/disable for the receive rings, send routine for the transmit rings) simply retrieve the handle for the underlying member ring, and invoke the corresponding ring entry point.

12 Conclusion and Future Work

Crossbow's first putback is getting ready for integration into OpenSolaris. This includes all the features mentioned in section 1. As of writing this document, the beta refresh bits (and source code) were available on OpenSolaris and already in use by several customers and developers. Some ISVs have already made changes to their application to make use of various Crossbow features.

In future (after first putback), Crossbow will be pushing the envelop in terms of leveraging NIC classification engines and resource partitioning. Layer 3 and Layer 4 classification will be added along with other stateless off load features. Apart from bandwidth limits, Crossbow will be adding bandwidth guarantees as well. Some of the other key features slated to be worked on are

- Enhanced Real Time Networking support
- L3 and L4 hardware classification support
- Full data link layer bypass for all traffic (currently works for IPv4 TCP, UDP and SCTP only)
- Dynamic use of hardware resources
- Extensive API and a networking equipment like command line interface 'CLI'

Apart from the technology pieces, a major effort will be made to make Crossbow and OpenSolaris an ideal platform for network development. The technical pieces to make Crossbow work as a high performing router or firewall are already there and can be used as of writing this paper. An in kernel L3/L4 load balancer (on top of Crossbow bits) is already under works and early access bits are expected soon on OpenSolaris. Other projects that complement Crossbow and make OpenSolaris a compelling offering in the area of Virtualization and as a networking developers platform are Layer 2 filter, Rbridges, and Clearview. The latter allows links to have vanity names and also provides a MAC plugin for any legacy driver to work under Crossbow framework without any changes (includes the old monolithic DLPI drivers as well).

13 Additional Links

Below are some additional technical and non technical links to find out more details on Crossbow technology or to figure out how to use it.

OpenSolaris Crossbow page	http://www.opensolaris.org/os/project/crossbow
Discussion Group	http://www.opensolaris.org/os/project/crossbow/discussions
Source and Binary download	http://www.opensolaris.org/os/project/crossbow/snapshots
Documents etc	http://www.opensolaris.org/os/project/crossbow/Docs
Virtual MAC client APIs	http://www.opensolaris.org/os/project/crossbow/Docs/crossbow-virt.pdf

Hardware Resources and MAC/Driver API	http://www.opensolaris.org/os/project/crossbow/Docs/virtual_resources.pdf
Getting started with Crossbow	http://www.sun.com/bigadmin/features/articles/crossbow_net_virt.jsp
Crossbow Demo/Workshop	http://www.opensolaris.org/os/project/crossbow/Docs/Crossbow_Workshop.pdf
Use cases	http://www.opensolaris.org/os/project/crossbow/Docs/usecase.html http://blogs.sun.com/droux/entry/private_virtual_networks_for_solaris http://www.opensolaris.org/os/project/crossbow/handson