

# Crossbow: A Vertically Integrated QoS Stack

Sunay Tripathi  
sunay.tripathi@sun.com

Nicolas Droux  
nicolas.droux@sun.com

Thirumalai Srinivasan  
thirumalai.srinivasan@sun.com

Kais Belgaied  
kais.belgaied@sun.com

Venu Iyer  
venu.iyer@sun.com

Solaris Kernel Networking  
Sun Microsystems, Inc.  
17 Network Circle, Menlo Park, CA 94025, USA

## ABSTRACT

This paper describes a new architecture which addresses Quality of Service (QoS) by creating unique flows for applications, services, or subnets. A flow is a dedicated and independent path from the NIC hardware to the socket layer in which the QoS layer is integrated into the protocol stack instead of being implemented as a separate layer. Each flow has dedicated hardware and software resources allowing applications to meet their specified quality of service within the host.

The architecture efficiently copes with Distributed Denial of Service (DDoS) attacks by creating zero or limited bandwidth flows for the attacking traffic. The unwanted packets can be dropped by the NIC hardware itself at no cost.

A collection of flows on more than one host can be assigned the same Differentiated Services Code Point (DSCP) label which forms a path dedicated to a service across the enterprise network and enables end-to-end QoS within the data center.

## Categories and Subject Descriptors

D.4.4 [Operating Systems]: Network communication; C.2.4 [Computer-Communication Networks]: Network operating systems

## General Terms

Design, Performance, Security, Experimentation

## Keywords

Networking, Performance, Classification, Crossbow, QoS, Flows, DDoS

## 1. INTRODUCTION

Recent technological advances have resulted in the convergence of voice, video, multimedia, e-Commerce, and traditional data traffic on the Internet. However each type of traffic has different characteristics and requirements in terms of delay, jitter and bandwidth. In addition, Internet Service Providers (ISPs) have an obligation to support a level of service that customers paid for according to their service

class. Thus there is a need to support QoS in a scalable and manageable way, with low performance overhead.

Many QoS models have been proposed. For example, Integrated Services (Intserv) [7] offers end-to-end guarantees about service levels. Other models, such as Differentiated Services (Diffserv) [6], specify service differentiation behaviors locally on a per-hop basis. Offering true end-to-end QoS requires support from all entities including the source host, routers and the destination. In this paper we propose a QoS model for the end hosts that is superior in performance and can be used in conjunction with Diffserv.

*Crossbow* is the code name of the new OpenSolaris networking stack which vertically integrates QoS functionality from the NIC hardware all the way up to the transport and socket layers. It uses NIC hardware features aggressively for performance, security isolation, and for meeting the QoS requirements of applications.

This paper first takes a look at the problems with existing QoS solutions. It then describes an architecture that addresses some of these problems and shows how the architecture can be used to mitigate DDoS attacks. It then proceeds to show how the architecture can also be used to build an end-to-end QoS solution that spans the enterprise data center. Finally, it explores other work happening in this area and describes future direction.

## 2. ISSUES IN EXISTING ARCHITECTURES

Performance overheads, complexity, scalability, deployment and manageability issues, are some of the issues facing various QoS solutions.

Intserv suffers from complexity and scalability issues [11] attributable to the complicated signaling mechanisms and the need to maintain flow related state in intermediate routers. Intserv was consequently never widely deployed. On the other hand, Diffserv, although simpler, does not specify end-to-end guarantees by itself. In [12] the author points out that though deployment of QoS mechanisms in the Internet remains sparse, diffserv represents a good start to address the real-world QoS needs.

On the host side, QoS has been traditionally implemented as a separate layer between the Data link and the Network (IP) layers. The QoS layer does the Diffserv processing that is needed. However this model creates a significant performance and scalability bottleneck on high bandwidth 10 Gigabit Ethernet networks. In addition recent CPU architectures [19] are moving towards a massively multi-threaded

multi-core model rather than higher clock speeds. The cost of bringing the packet into the host and inspecting the packet headers has become increasingly prohibitive on such architectures and makes it almost impossible to honor the Service Level Agreement (SLA). Thus, there is a need to integrate the QoS functionality vertically with the network stack in order to amortize the various per packet costs and reduce the QoS overheads.

Intrusion Detection Systems constantly contend with DDoS attempts that exhaust CPU or network bandwidth resources [22]. Traditional QoS models do not solve this problem because they are structured high up in the stack and don't have a way to turn off the incoming attacking stream at the lowest level and to relieve system resources.

The essence of what QoS should be is lucidly brought out in [2] where the author points out the following QoS requirements: it must be bottom up, it needs to be supported at the lowest layer, below IP, and it needs to be extremely efficient and simple.

### 3. CROSSBOW ARCHITECTURE

Project Crossbow in OpenSolaris implements a new networking stack that has QoS functionality integrated in the stack itself instead of an add on layer. The approach uses NIC hardware classification and partitioning capability to allow the use of some of the most commonly used QoS features without any performance overheads. The project also attempts to make the concepts and configuration easy for users to understand and deploy.

The Crossbow Virtualization functionality [28] also takes advantage of the NIC hardware capabilities. A Crossbow virtualization lane consists of dedicated hardware and software resources for a particular type of traffic and defines a vertical path from the NIC hardware to the socket layer. NIC receive and transmit rings, interrupts etc. are examples of hardware resources, while kernel queues, threads, CPU bindings of kernel threads etc. are examples of software resources. One or more virtualization lanes may be assigned to a virtual machine.

Crossbow flows discussed in Section 3.1 are analogous to virtualization lanes. Flows may however be used even without any virtualization. Early packet classification and use of hardware rings achieve traffic separation and partitioning in both cases. Dynamic polling [28] on individual NIC receive rings, smooth transition between interrupts and polling, and support for multi-core CPUs contribute to performance in both cases. In the virtualization case, bandwidth limits are used to set the link speed of a virtual NIC.

The major components of the architecture discussed in this paper are already available for download as part of OpenSolaris kernel. Some of the features like hardware based Flows and enhancement to DDoS defense are works in progress.

#### 3.1 Flows

Bernet et al.[5] outline an informal management model for the Diffserv architecture including meters, markers, queuing disciplines and shapers. Crossbow, on the other hand allows the creation of flows which implement a mostly queueless scheduling engine for packet processing. On the receive side, packets are allowed to come in the system when they are scheduled to be processed. Similarly, on the transmit side, the applications generating traffic are flow controlled as needed thus largely eliminating the need for queuing.

A flow essentially creates a data path from the hardware to the transport layer. The path is created by configuring classification rules in the NIC, which result in steering packets at the hardware level to an assigned receive ring. The stack can identify and schedule the packets for a flow even before it brings them into the system memory or look at any headers. Dynamic Polling on a per receive ring basis ensures that packets for a flow are allowed in the system based on their assigned service levels. To summarize, the following components make up the flow:

**Classifying parameters (attributes)** – These can be attributes from Layer 3 or Layer 4 headers and can include host and subnet IP addresses (local and remote with variable length netmasks), transport protocol (TCP, UDP, SCTP, ICMPv6, etc), ports (local and remote), DSCP bits, etc.

**Properties** – A property of a flow determines how packets for that flow will be treated. Properties can be bandwidth limits and guarantees, processing thread priorities, and processing CPUs.

**System resources** – Consist of the following hardware and software resources:

**NIC resources** – Hardware receive and transmit ring (groups) and classification rule. Most modern NICs [15] [25] [20] support multiple Receive and Transmit rings and hardware classification features.

**MAC resources** – The key MAC resource is the construct called *Softring Set* (SRS). The SRS is a FIFO based with an attached poll and worker thread. It also implements the packet scheduling based on backlog, and specified bandwidth limits or guarantees.

On the receive side, a one to one mapping exists between a SRS and a NIC hardware receive ring. Thus, the SRS can switch the hardware ring between interrupt and poll mode without impacting any other flow or traffic. In addition, a SRS can also have a collection of *softnings* (hence the name softning set) which emulates the hardware ring group. Softnings are also queues with a worker thread running on unique CPUs (where possible) that are assigned to the softning. The purpose of softnings is to offer software based fanout to spread the incoming packet processing across multiple CPUs.

On the transmit side, the SRS can have a direct relation with a hardware transmit ring. The SRS schedules the packet transmission, manages the driver transmit buffers and flow controls the application when transmit ring is running out of buffers.

**IP and transport layer resources** – TCP and SCTP have vertical perimeter (squeues) [27], which includes queue poll and worker threads. Typically, a unique queue is assigned to the SRS (in the absence of rings) or to each soft ring within the SRS when software based fanout is enabled for the flow.

### 3.2 Receive-Side Processing

Packets for a flow are classified by the NIC hardware into the Receive ring for the flow. They enter the MAC layer either through the interrupt path or as a result of being polled by the MAC SRS's poll thread.

Bandwidth control is implemented by a simple average rate meter with the average computed every fixed period (currently 10 milli second) and enforced by the received side SRS. When the bandwidth limit is reached for the specified period, the SRS switches the receive ring into poll mode and packets are left in the hardware receive ring to be picked up by the poll thread according to the bandwidth constraints. The interrupt mode is enabled when there is no backlog (i.e. queued packets) and the arrival rate is within specified bandwidth limits for that particular period.

In the case of TCP, if the queue cannot keep up with the processing, it sets the underlying SRS or soft ring in poll mode. In turn, the SRS switches the receive ring to poll mode. In poll mode, the SRS attempts to continue polling periodically based on a low and high water mark to keep traffic flowing. However, if there is no backlog or new packet arrival, the SRS can set the receive ring back in interrupt mode.

It is worth noting that the entire system of SRS (any soft rings) and queue provides a contention free path without the need for any fine grained locks. Furthermore, the packets are normally not queued in the system at all because as soon as any backlog starts to build up, the hardware receive ring is switched to the poll mode and acts as the only queue in the path.

Figure 1 illustrates the TCP receive path for a flow. When the flow is added (e.g. flow for TCP packets), Crossbow programs the NIC's classifier to steer all TCP packets to a ring. Packets get either immediately delivered to SRS through interrupt mode or are picked later by means of an SRS poll thread.

The Crossbow MAC layer has a full featured software classification engine. It is used when the NIC is not capable of classifying based on L3/L4 headers, or is out of hardware receive rings. Software classification is performed very early to assign a packet to a flow and steer the packet to the SRS associated with the flow. The system can work in hybrid mode where hardware based flows can be combined with an unlimited number of software based flows. Traffic arriving through the default receive ring is software classified and delivered to the software based flows.

### 3.3 Transmit-Side Processing

On the transmit side, the application thread sends data directly to the flow's SRS. The packets are sent directly to the NIC's transmit routine provided that transmit buffers are available and the bandwidth limit for that period is not being exceeded. As a host, Crossbow uses application flow control on outbound data traffic rather than dropping packets. When a NIC runs out of descriptors, or if the bandwidth is being exceeded, the SRS exerts back pressure and the client is blocked from sending further data down. When the NIC has transmit descriptors available, it sends a notification to MAC to remove the blocked condition on the SRS. In turn, the IP layer is notified by the SRS to enable the client to resume sending data.

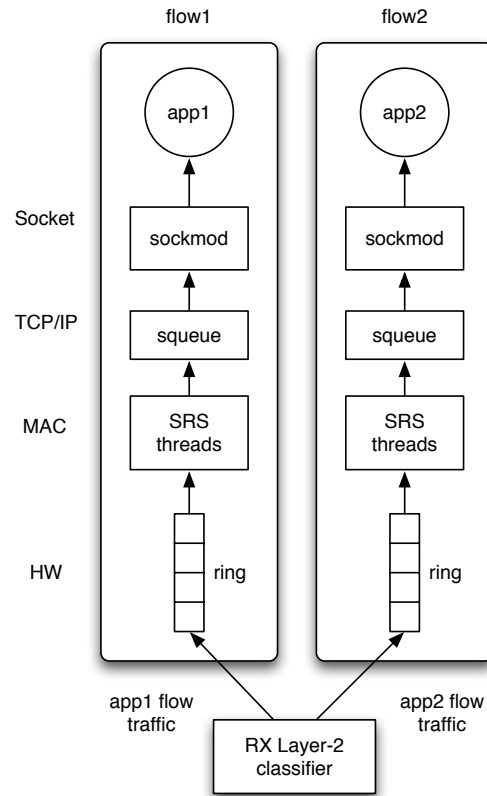


Figure 1: Crossbow hardware flows

### 3.4 Types of flows

As mentioned in Section 3.1, layer 3 or 4 attributes may be used to specify a flow over an interface or data link. The following types of flows are fairly common and are supported efficiently.

**Service-based flow** – Services are typically defined as a combination of a particular transport and well-known port. For example an HTTP service that uses TCP protocol over port 80 can be assigned its own resources by creating a TCP flow for the above protocol and local port combination.

**IP address-based flow** – Traffic that uses a particular local or remote IP address may be given its own resources by creating a flow that specifies the desired local or remote IP address(es).

**IP subnet-based flow** – Specifying a combination of IP address and subnet mask creates an IP subnet-based flow that can be given its own resources.

**DSCP label-based flows** – This is done by specifying the DSCP bits and the DSCP mask that is to be applied on incoming packets.

While it is possible to design a very generic classifier to handle any arbitrary combination of attributes, the challenge is to achieve it with minimal performance impact and also use NIC hardware classification support.

### 3.5 TCP and UDP flows

Given the extensive use of TCP and UDP protocols, we discuss some more details about TCP and UDP service based flows in this section. TCP is very sensitive to packet drops. It interprets packet loss as a measure of link congestion and self-paces its throughput accordingly. It is well known [16] that the bandwidth delay product of a TCP connection in steady state corresponds to the capacity of the channel between sender and receiver. Efficient bandwidth control is achieved by controlling that delay and minimizing packet drop.

In the case of a hardware based TCP flow, the dynamic polling mechanism of NIC receive rings implements bandwidth control on the inbound side. As mentioned in Section 3.2 the SRS polling thread picks up only as many packets from the NIC receive ring as allowed by the configured bandwidth limit. Bandwidth control for outbound packets is implemented similarly by introducing a suitable delay corresponding to the bandwidth limit. As mentioned in Section 3.3 the application is flow controlled, and thereby prevented from sending more data without packet drops. Inbound packets are not dropped as long as the NIC receive ring has sufficient receive descriptors and buffers to hold the entire TCP window's worth of data.

The bandwidth control mechanisms for UDP are similar to TCP on both inbound and outbound directions. However the UDP protocol does not have a built-in flow control mechanism. If the arrival rate of incoming packets for a UDP flow is greater than the bandwidth limit, the NIC receive ring will eventually fill up, and incoming packets will be dropped by the NIC. As it is the case for TCP traffic, the outbound bandwidth usage is regulated by flow controlling the application.

In the case of software based flows, a software queue is used in place of the NIC receive ring, and the bandwidth usage of inbound packets is regulated using a simple tail-drop mechanism. Hardware based flows inherit the drop model implemented by the NIC hardware.

While a single TCP connection can use at most 1 ring to avoid costly reordering, multiple connections of a high bandwidth flow can be spread among multiple hardware rings. Outbound traffic is spread across multiple transmit rings by the Crossbow stack, and inbound traffic is spread between multiple receive rings by the NIC hardware. When a bandwidth controlled flow is spread across multiple rings, the bandwidth counter for that flow is shared among the rings. This sharing introduces fairness issues in polling mode, which is an area for future work.

### 3.6 Flow Performance

Most of the QoS solutions function well on transmit side. However, bringing the packet into the system and looking at headers to identify either the packet destination or the QoS class constitute a major cost. Thus, QoS solutions are less efficient when processing received network traffic.

Figure 2, shows the advantage of Crossbow flows. The Software based flow performs some of the same operations as the Solaris 10 IPQoS [26]. However, by being integrated in the stack, Crossbow flows perform better than a traditional layered IPQoS implementation. The hardware based flows perform best with almost no overheads. The classification occurs in the hardware and an independent path exists through the stack. In addition, sizable advantages can be

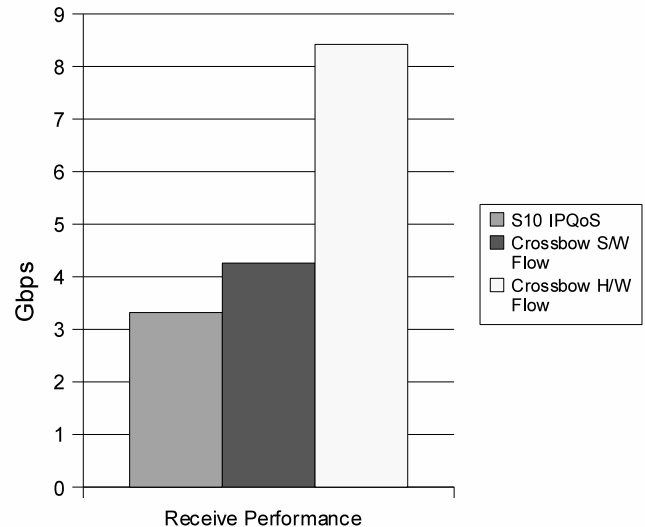


Figure 2: Receive Performance with a simple TCP based rule

Configured BW	Fedora 2.6 TC	Crossbow HW Flow
1Gbps	0.4Mbps	0.95Gbps
2Gbps	0.7Mbps	1.87Gbps
3Gbps	1.13Mbps	2.86Gbps
4Gbps	1.26Mbps	3.81Gbps
5Gbps	1.24Mbps	4.40Gbps

Table 1: Measured TCP bandwidth with Fedora 2.6 TC and Crossbow hardware flow vs configured bandwidth

obtained by using Dynamic Polling to schedule packet processing on the receive side.

In Table 1, we compare Crossbow hardware based flows with Fedora 2.6 TC[1] under bandwidth limits to see how well the integrated hardware and software approach worked. As it is evident from the graph, Crossbow hardware based flows allowed TCP to get very close to the bandwidth limit while Fedora 2.6 TC does not allow the throughput beyond a few megabits even when the configured limits were in gigabit range on a 10 Gb/s network. The issue was that TC does not schedule receive side processing and any packets exceeding the buffer limit are dropped hurting TCP performance. Crossbow Hardware based flow on the other hand employs the NIC receive ring and helps to adjust TCP RTT instead of dropping the packets. It would be worthwhile to note that on transmit experiments, both Crossbow and TC were able to get close to the configured limits. The commands necessary to configure both Crossbow and TC are shown in Section 5.

For all experiments, the test setup consisted of four identical machines. One machine acted as a system under test (SUT) and the other three machines acted as clients. Each machine was dual socket, quad core Intel machine with each core operating at 2.8 GHz. All four machines had an Intel 10 Gigabit Ethernet NIC and were connected to a dedicated Foundry 10 Gigabit Ethernet switch. Each client was sending 10 TCP streams to the SUT. The wire MTU was 1500 bytes, and the application write buffer size was 8 Kbytes.









