Crossbow

A Toolkit for Integrated Services over Cell Switched IPv6†

Dan Decasper[1], Marcel Waldvogel[1], Zubin Dittia[2], Hari Adiseshu[2]
Guru Parulkar[2], Bernhard Plattner[1]
[dan|mwa|plattner]@tik.ee.ethz.ch
[hari|zubin|guru]@arl.wustl.edu
[1]Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
Phone: +41-1-632 7019 Fax: +41-1-632 1035
[2]Applied Research Laboratory, Washington University, St. Louis, USA
Phone: +1-314-935 4586 Fax: +1-314-935 7302

Abstract

The project Crossbow *provides a framework to investigate services and mechanisms including resource management and packet scheduling for multimedia/multicast applications. In particular the Internet Protocol version 6 (IPv6, IP next generation, IPng) [7] protocol suite on top of ATM is considered to demonstrate possible synergy between ATM and IPv6. The presented architecture includes IPv6 and RSVP [17], running on BSD Unix using the 1.2 Gbps APIC (ATM Port Interconnect Controller) chip [6], as well as support for Ethernet networks.*

1 Introduction

The Crossbow system includes several components where each of them have a rather high level of complexity on there own and in conjunction with other system components. The goal of this paper must be therefore to give a rough overview of how these components work and interact and can't be a detailed description of any of them.

First of all, Crossbow introduces new components to a host's system architecture. It includes modifications and extensions of the network subsystem in a NetBSD Unix Kernel and provides a new daemon running in user-space. It is connected to the network through either an ATM or an Ethernet interface. Section 2 describes this system.

Second, a new IP on ATM architecture is proposed using Crossbow systems connected to an ATM switch and introducing a distributed packet scheduling algorithm. Section 3 describes the architecture.

Finally, section 4 shows a possible application of the Crossbow system set up for further research and demonstration purposes.

---

† This work is supported in part by Kommission für Technologie und Innovation (KTI), Ascom Tech, Ltd., Applied Research Lab (ARL) Washington University St. Louis, ETH Zürich, and Intel

2 Crossbow on a host

The main parts of Crossbow are the Toolkit Engine in the BSD Unix kernel and the QoS Control Program in user-space (Figure 1). QoS management is moved out of the kernel, wherever possible, into a user-level daemon process. Only processing that either needs to be applied to each IP packet or functions that need access to kernel data structures remain in the kernel as part of the Toolkit Engine. The QoS Control Program mainly handles resource reservation requests from applications or from the network and negotiates them with the Toolkit Engine. The Toolkit Engine implements an IPv6 stack, detects long live traffic flows, and applies toolkitized packet processing algorithms like packet schedulers to network data.

2.1 QoS Control Program

The QoS Control Program (QCP) is the central authority responsible for supervising all QoS requirements. In order to minimize kernel complexity, increase system maintainability and provide for convenient debugging of user level processes, all processing not directly involved with packet level manipulations (sending, receiving, forwarding) is handled by the QCP. Unlike other approaches [17], the kernel is not involved with policy and admission control, just with the per-packet traffic policing.

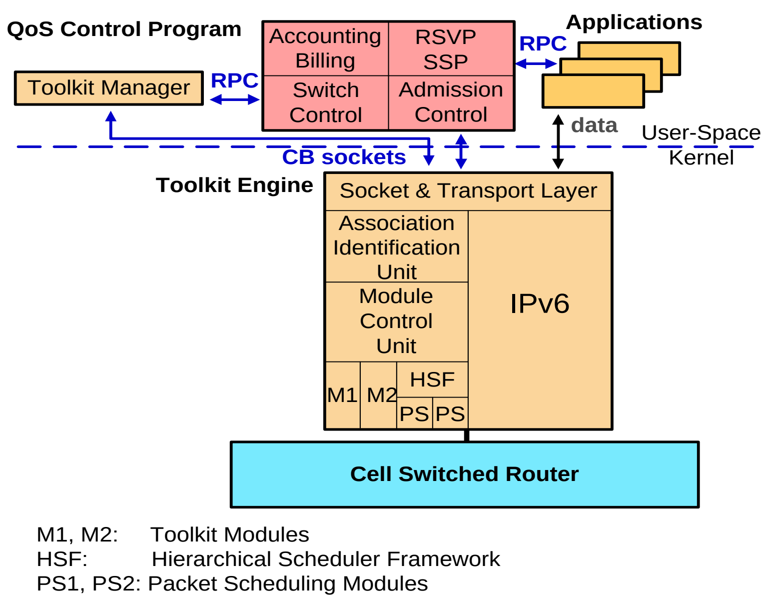

Figure 1: Crossbow's Component Overview

Applications wishing to have QoS assurance can communicate with the QCP, which offers an API providing a high level of abstraction, through remote procedure calls (RPCs). The QCP converts each application request into calls to the appropriate resource reservation protocols and switch control mechanisms, shielding all the implementation details from the application. Among other things, the API provides an easy, object-oriented way to open QoS-assured connections and to set and query QoS parameters associated with connections.

To provide a more flexible way of communication between the QCP and the kernel, a new socket type, similar the routing sockets introduced with 4.3 BSD Reno, is used. Sockets of this type offer a modular interface to the kernel for the QCP daemon. The Toolkit Engine in the kernel takes advantage of this event oriented approach. The kernel can therefore notify the application of important events in a much more flexible way than what is currently possible with signals and their rather crude semantics.

Since the same QCP is also used in Crossbow-based routers (see section 3, Crossbow as an IP on ATM router), it implements Ipsilon's General Switch Management Protocol (GSMP) [9], to set up and tear down cut-through flows. Support for accounting and billing modules is planned to be implemented in the future. For QoS signalling and resource reservation in the network, the QoS Control Program has built in support for RSVP [17] and SSP (State Setup Protocol, [2]).

The State Setup Protocol (SSP) was designed as a light-weight protocol combining the advantages of RSVP and IFMP [9], so that the individual flows could be mapped to specific ATM VCs, thus allowing the dynamic creation of QoS-guaranteed “cut-through” routes for individual flows. To make the protocol light-weight, it was stripped down to the very minimum, both in protocol and code complexity, while still trying to provide the basics necessary for any QoS the application would require. In contrast to RSVP, no PATH messages are sent, there is only one flow type, and one reservation style, greatly simplifying the protocol handling and making flow merging possible in all cases.

Since RSVP and SSP are wanted not only to co-exist, but also to co-operate, each QCP also acts as a gateway between the two protocols, by providing a “pseudo” link or interface, which effectively converts requests between the two protocols. To avoid having to unmarshal and decode the packets of one protocol and re-encoding it for the other, both resource reservation protocols were equipped with hooks which provide a controlled, simple way of intercepting the information in the clear before it is being sent out to the pseudo link.

2.2 Toolkit Engine

The IPv6 Toolkit Engine is made up of static units and dynamically loadable modules. The user can implement modules, load them into the kernel, and select different modules for different tasks. To allow that, the Toolkit Engine consists of a modified ‘off-the-shelf’ IPv6-stack (coming from INRIA, France [8]), the Association Identification Unit (AIU), the Module Control Unit (MCU) and the Hierarchical Scheduler Framework (HSF). The IPv6 stack is responsible for getting data from sockets, putting it into IPv6 packets and sending them to the network as well as receiving or forwarding packets (standard IP protocol stack processing). The AIU is used for flow detection and it returns suitable module instances providing extended functionality including IPv6 option processing, security related algorithms, or packet scheduling.

A flow is a sequence of data packets where all packets have five header fields (source address, destination address, source port number, destination port number, and protocol) and the input interface in common. Crossbow allows the user to bind Toolkit Modules to data flows by defining packet filters and selecting modules to work on these flows. Filters can be specified through the QoS Control Program. Filters are normally a six-tuple consisting of the same fields as a flow but in contrast to a flow, one or multiple filter fields can by wildcarded. The address fields are combined with a mask to determine the significant bits, and the other filter fields can be optionally wildcarded. By masking and wildcarding filter fields, the user can map multiple data flows to a single filter. For any of these filters and for any module type supported, the user may choose an instance of a module which runs on all packets matching the filter.

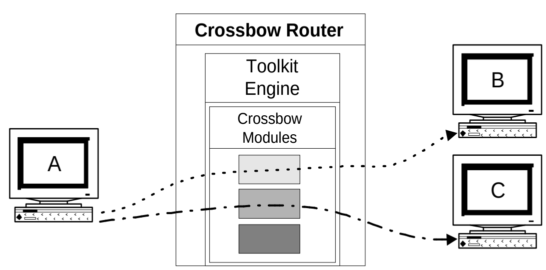

Figure 2: Host communication through a Crossbow router

Assume a host A sending data to the hosts B and C through a Crossbow router (Figure 2). The user may configure the Crossbow router so that packets coming from source A going to destination B are processed by different algorithms than packets going to destination C. The user does so by setting two filters and binding each of them to a different module instance. Every module instance is of a certain type. Five different module types are supported in the initial version: filter modules, authentication modules (for RFC 1826 authentication), encryption modules (for RFC 1827 encryption), IPv6 option modules (for RFC 1883 IPv6 option processing), and packet scheduling modules (cf. section 2.3). Filter modules can be implemented by the user to customize the filtering by looking at arbitrary packet fields beside the six-tuple mentioned above.

The Association Identification Unit stores user defined filters and holds Toolkit Module selections. As soon as an IPv6 packet enters the stack, it gets passed to the AIU, which associates it with a flow by tagging it with an identifier for the flow it belongs to. If the packet in question belongs to a yet unknown flow, a new flow entry is automatically created and mapped to filters stored in the AIU. Thereby, the user's module selections are assigned to the new flow. A flow always matches the most specific of the stored filters and inherits the module selection and settings from the filter. There is a wide range of filter scope. On one side, there is the default filter which contains wildcarded fields only and therefore matches every flow. On the other side, fully specified filters match exactly one flow. Between these two, adjustments of the scope of a module selection can be done to whatever scale is desired.

Routing lookups performed for newly created flows are implemented using a new, very high speed method described in [14]. The search algorithm for the best matching prefix uses binary search on hash tables organized by prefix lengths. The scheme scales very well as address and routing table sizes increase: independent of the table size, it requires a worst case time of $\log_2$(address bits) hash lookup. Therefore only 7 hash lookups are needed for IPv6 and several optimizations reduce that number even further.

Flow entries in the AIU expire after some time and are deleted if they are no longer used (soft-state). The IPv6 packet, traversing the IPv6 stack from bottom to top or vice versa, possibly comes to a place where toolkitized algorithms may by applied to it. The packet's flow tag is checked and the packet is passed to the desired algorithm's instance. This allows for the modularization of packet processing algorithms with very little performance penalties involved.

The Module Control Unit (MCU) manages various Toolkit Modules loaded into the kernel by `modload` shell commands. A default set of these modules is loaded at system boot time. Modules use a callback function to register themselves with the MCU. The callback function is used by the MCU for all communication with modules, including instance generation (modules may create different instances of themselves on request), instance configuration and instance deallocation. If a user wants to configure a module for a given filter, he forwards the commands to the AIU via the QoS Control Program telling the MCU to send a message to the module corresponding to the filter.

The Hierarchical Scheduler Framework (HSF) provides an environment for packet scheduling Toolkit Modules (cf. section 2.3).

2.3 Packet Scheduling Modules

Packet scheduling modules are one of the five module types currently supported in Crossbow. They give a good example of toolkitized code and the usage of toolkit mechanisms for code implementation. The Toolkit Engine provides a generic framework for hierarchical schedulers as described in [3]. Various packet scheduling algorithms like WF^2Q [4], Priority queuing or Deficit Round Robin (DRR) [13], will be implemented and integrated into the Hierarchical Scheduler Framework as Toolkit Modules.

Generalized Processor Sharing (GPS) [11], suggesting a bit-by-bit round-robin algorithm, achieves perfect equal sharing of the bandwidth between flows, even with different packet lengths. Unfortunately, GPS is not very well suited for real world implementation because of the small (bit) granularity. Using the APIC hardware (see section 2.4, APIC host adaptor) environment, a pacing-based scheduling variant is proposed. It is named "cell-by-cell" round-robin (CCRR) and will be implemented as a Toolkit Module. The idea is to map all best-effort flows to a fixed set of VCs that are reserved for best-effort traffic. Each best-effort VC is assigned to a different best-effort channel in the APIC. The scheme assumes that the pacing hardware of the network interface supports the notion of multiple best-effort channels, which are transmit channels on which data is sent out on every "available" cell slot, where availability of a slot means that no guaranteed flow or transit cell is using that slot. Every available cell slot of one of these channels is served by the APIC hardware in a round-robin way. This is a much better approximation to GPS than WFQ or WF^2Q.

2.4 APIC host adaptor

Although the Crossbow system architecture may run on Ethernet networks, one of the project's main focuses is to investigate IPv6 on ATM related algorithms and Integrated Service architectures requiring quality of service guaranties from the underlying network hardware. The device planned to use on ATM networks is the APIC (ATM Port Interconnect Controller) [6, 14], a 1.2 Gbps host-network adaptor chip which paves the way to the realization of extremely high performance I/O subsystems explored in conjunction with the Crossbow project. Very high bandwidth combined with built-in low latency mechanisms make the APIC particularly well suited for multimedia and latency-critical distributed computing applications. Features include full duplex AAL-5 segmentation and reassembly at 1.2 Gbps as well as cell pacing mechanisms based on proprietary d-heap technology which enables entirely independent pacing over all active connections, which is needed for CCRR described above.

3 Crossbow as an IP on ATM router

Crossbow will be used in a Cell Switched Router (CSR) environment implementing GSMP in every node's QoS Control Program. Multiple variants of CSR designs are known [1]. ATM routers may come with one or multiple IP processing elements (IPPE) connected to one or multiple ports of the ATM switch. IPPEs may map flows to virtual circuits (VCs) or routes to virtual circuits.

In the Crossbow project, the selected architecture is closely aligned with the one described in [12] which deals with multiple IPPEs and flow-to-VC mapping. The architecture introduces a very synergistic combination of IP and ATM which goes beyond the conventional approach of implementing IP over ATM in a strictly layered fashion. By allowing the IP processing layer to directly control and manipulate an underlying ATM switch core, IP can directly benefit from the hardware processing efficiencies of ATM switching technology.

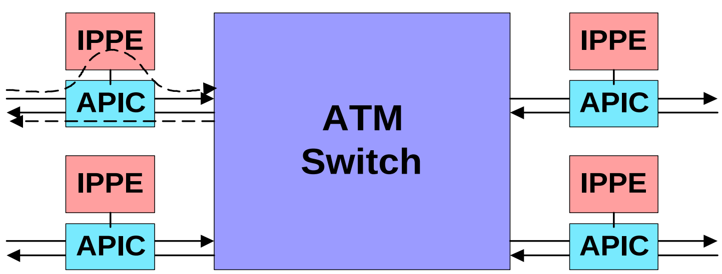

Figure 3: Crossbow router architecture

Each router is designed using an ATM switch and APICs connecting one IPPE to every switch port (Figure 3). IPPEs are general-purpose processors running the Crossbow network subsystem described in section 2. The IPPEs are connected to the switch through the APIC host-network adaptor. The APIC adapter concurrently supports two ATM links and can also switch ATM cells directly from one link to another one without going through the IPPE. Each APIC is configured with a list of VCs whose cells need to be forwarded to the attached IPPE. Otherwise, it is passed along the APIC chain. The Crossbow router permits the IPPE to dynamically configure ATM virtual circuits to optimize the handling of user information flows. When an IPPE receives an RSVP or SSP reservation message, it sends control cells to the embedded ATM switch, instructing it to configure a new virtual circuit on the selected outgoing link. The IPPE instructs the APIC to forward subsequent cells from this flow directly along the virtual circuit, without passing through the IPPE. This approach allows the IPPE to maintain full control over individual packet flows and modify its decisions as necessary, while amortizing the cost of the more complex decision-making processes over many IP packets. Therefore the usage of the APIC in conjunction with this architecture provides a scalable solution to packet processing at high link speeds.

The Crossbow router architecture also enables the IP layer to configure multicast virtual circuits by allowing IP multicast to be implemented directly at the hardware level by exploiting the cell-replication capability of the ATM switch. In particular, when the first of a stream of IP multicast packets is received at an IPPE, a multicast route lookup is performed. This yields an ATM VCI that was previously configured using ICMPv6 [5] and multicast routing protocols and locally bound to the given multicast address. The IPPE then modifies the virtual circuit table in the ATM switch's input port processor to forward cells to the proper multicast VCI. After that, the IPPE hands off control to the APIC as previously described, allowing the remainder of the burst to be processed at the cell level. This makes it possible to achieve high-performance IP multicast forwarding and allows a very large number of multicast applications to operate on the network at the same time.

IPPEs only process flows (packets) as they enter the CSR, outbound flows pass directly through the associated APICS. The routing decision is made at the IPPE connected to the input port. The same IPPE also handles packet scheduling for the outgoing link. Thus, multiple IPPEs simultaneously schedule packets for different output links. For this reason, all IPPEs need to cooperate using a "distributed" packet-scheduling algorithm. Implementing such an algorithm easily is provided by dynamically changing maximum rates allocated to each IPPE for each output link. IPPEs do traditional packet scheduling for all flows that they are responsible for, assuming that the output link rate of a single port is equal to the current maximum rate assigned to them. Clearly, the sum of these maximum rates assigned to all the IPPEs is always equal to the link rate (for each of the outputs). A dedicated protocol is used to assign these maximum rates to different IPPEs based on current traffic conditions at each of the output links.

Crossbow distinguishes three classes of traffic flows: reserved, elastic and best effort. Once established, reserved flows are cut-through routed and no dedicated packet scheduling in the IPPE is needed. Packets go straight through the APICs on both the input and the output port of the switch without any software processing. Elastic flows are flows providing a reserved "floor" (minimum) bandwidth and may acquire a larger amount of bandwidth then the reserved floor bandwidth if available. In support of elastic flows, packet schedulers will be implemented in software as Crossbow modules (cf. section 2.3). Best effort flows are scheduled using CCRR relying on the APIC's hardware pacing to implement equal sharing of all the remaining bandwidth between the flows (leftover after summing up bandwidth for reserved and elastic flows).

4 Applications

One of the planned multimedia applications in use for Crossbow is the Multi-Media Explorer (MMX) [16] (Figure 4).

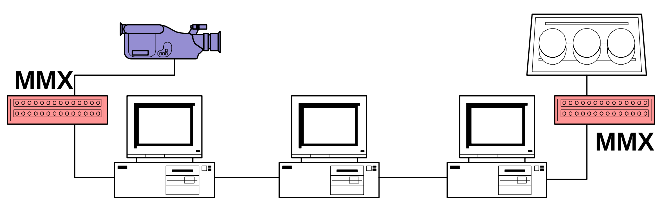

Figure 4: MMX and Crossbow

The MMX is a desktop multimedia interface that provides users with advanced video and audio communications capabilities. It generates JPEG compressed video at about 20 to 30 Mb/s from a video camera, in ATM cells. A regular PC running Crossbow will run an application receiving this stream and converting it to IPv6 packets. This PC/MMX combination will be an experimental source with a similar PC/MMX combination acting as a receiver. The router is a third PC in-between the source and destination, of course running Crossbow as well. With this framework, demonstration of full motion compressed video applications, including video-on-demand, and teleconferencing will be possible.

Besides the MMX and other similar multi-media applications, integration of the standard network applications like telnet and ftp with service guarantees is scheduled as well. An http-daemon and World Wide Web client will also be adapted.

5 Conclusion

This paper describes a way to provide an experimental platform for researchers and developers to investigate network related techniques and mechanisms in the field of IPv6 and ATM. It offers a fully featured Integrated Service environment including Quality of Service functionality for end user applications. By introducing a high degree of modularity and providing essential new network technologies like flow management and fast packet to flow mapping algorithms expected to play a very significant role in network kernels in the near future, scientist can focus on their specific subject of investigation without the need of knowing the implementation details of those mechanism. By writing network code in form of modules and including them at run time into the Crossbow system, much shorter turn around times and therefor increased productivity are expected.

The same network subsystem will be found on user’s desktops as well as on large backbone routers since most of the required functionality is equivalent for both types of applications. The CSR architecture in use offers a very scalable and high-performance solution expected to show that IP and ATM technologies can be mutually supportive.

Acknowledgments

The authors would like to thank Fred Kuhns, John DeHart, and Burkhard Stiller for suggesting modifications which helped to make this paper more understandable.

References

[1] Adiseshu, H., and Parulkar, G., “Towards a Scalable Integrated Service IP Router Design”, to be published

[2] Adiseshu, H., and Parulkar, G., “Scalable Integrated Services with IP/ATM Integration”, to be published

[3] Bennett, J.C.R. and Zhang, H., “Hierarchical Packet Fair Queueing Algorithms”, In *Proceedings of SIGCOMM’96*, August, 1996.

[4] Bennett, J.C.R., and Zhang, H., “WF2Q: Worst-case Fair Weighted Fair Queueing”, *INFOCOM’96*, March, 1996

[5] Conta, A., Deering, S., “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification”, RFC 1885, December 1995

[6] Dittia, Z., Parulkar, G., Cox, J., “The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques”, 1996

[7] Deering, S., Hinden, R., “Internet Protocol, Version 6 (IPv6), Specification”, RFC 1883, December 1995

[8] INRIA ftp site for IPv6 source code. ftp://ftp.inria.fr/network/ipv6

[9] Newman, P, et al., “Ipsilon's General Switch Management Protocol Specification”, RFC 1987, August 1996

[10] Newman, P, et al., “Ipsilon Flow Management Protocol Specification for IPv4”, RFC 1953, May 1996

[11] Parekh, A., “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks”, PhD dissertation, MIT, February 1992

[12] Parulkar, G., Schmidt, D., Turner, S., “a^{I}t^{P}m: a Strategy for Integrating IP with ATM”, In *Proceedings of SIGCOMM’95*, August 1995

[13] Varghese, G., Shreedhar, M., “Efficient Fair Queuing using Deficit Round Robin”, In *Proceedings of SIGCOMM’95*, August 1995

[14] Waldvogel, M., et.al., “Scalable High Speed IP Routing Lookups”. To appear in *Proceedings of SIGCOMM’97*, September 1997

[15] Web Pages on APIC. http://www.arl.wustl.edu/arl/refpapers/zubin/apicbrief.html

[16] Web Pages on MMX. http://www.icon-stl.net/~ststech/

[17] Zhang, Lixia, et al., “RSVP: A New Resource ReSerVation Protocol”, In *IEEE Network Magazine, Vol. 7, No. 5. pp. 8-18*, September 1993