QoS Routing using Active Networks in IXP 2400 Network Processor

***N.Saravana Selvam* [†] *and Dr.S.Radhakrishnan* [††],**

[†]Professor/CSE, Sree Sowdambika College of Engineering, Aruppukottai,
[††] *Senior Professor & Head/CSE, Arulmigu Kalasalingam College of Engineering, Krishnankoil .*

Summary
As technology is moving towards high computing environments with billions of users, devices, and services, the Quality of Service (QoS) becomes a necessity and an essential element for end-to-end real time applications. Since the Internet is becoming a converged network that can support emerging voice, video, and data applications in a unified manner. One of the key issues in such a converged network is how to identify feasible paths that can satisfy the quality-of-service (QoS) requirements of different real-time applications. This problem is commonly known as QoS routing (QoSR). In general, the main objective for a QoSR algorithm is to find feasible paths that can satisfy the given QoS requirements. While searching for a feasible path, it is necessary to be highly responsive to routing requests. The concept of active network has been recently adopted in order to provide a framework in which executable code within data packets executes upon intermediate network nodes and to facilitate delay services in the network. This paper describes the combination of the active network concepts with QoS routing provided by the Intel IXP2400. The information embodied in the transmitted packets is used by the microengines of the IXP to fulfill the QoS demands issued by various real time applications. Thorough examination is made for the performance and reliability of the Active QoS routing scheme for different traffic measures and for the corresponding QoS offered in terms of the end-to-end delay, jitter and packet loss.

Key words:
QoS, QoSRouting, Network Processor, IXP2400, Active Networks

1. Introduction

Routing mechanism is a key to the success of large-scale, distributed communication and heterogeneous networks. The increase in real-time applications such as Voice over IP, audio and video streaming in the public Internet has warranted QoS based routing. These applications have stringent performance requirements in terms of delay, delay jitter and loss rate. In real-time quality-of-service, delay variation is generally more critical than delay as long as the delay is not too high. Clearly, voice-based applications cannot tolerate more than a certain level of delay. The condition of varying delays may be expected to a greater degree in a shared medium environment with datagram, than in a network implemented over a switched substrate. Routing a real-time flow therefore reduces to an exercise in allocating the required network resources while minimizing fragmentation of bandwidth.

When network dimensions increase, traditional routing algorithms and resource allocation methods [1] do not scale well particularly in the presence of frequent traffic flow changes and the flow of prioritized packets. In topologically complicated networks, the lack of adaptability of routing and resource allocation algorithm could become disastrous for the offered Quality of Service (QoS). Routing and specifically resource allocation algorithms must have the ability to adapt in any network changes and cope in anytime network state changes (capacity of nodes and links, traffic within paths, load changes etc.).

QoS routing attempts to improve network utilization by diverting traffic to paths that would have not been discovered by traditional, non QoS sensitive routing. We believe that QoS routing in NP based router will be more useful and more effective in environments where traffic and network capacity are mismatched and alternate paths with lower load exist.

Present Routers are mainly based on Application Specific Integrated Circuits (ASICs) that are custom made and are not flexible enough to support diversified services. General Purpose Processors (GPP) offer flexibility in supporting new features by simply upgrading the software, but have difficulties in supporting higher bandwidth. Network Processors have emerged to provide both the performance of ASICs and the programmability of GPPs.

Network Processors allow multitasking and multithreaded programming. Multithreading provides a means for hiding latencies of various operations of an NP by allowing a Processor Engine (PE) to proceed with processing of alternate packets when processing of the current packet stalls for some reason such as a memory access. Multithreading can increase the utilization of PEs and can improve the overall performance of an NP. Here we are using all the eight threads present in the IXP2400, and the

Manuscript received March 5, 2010
Manuscript revised March 20, 2010

performance are measured with and without executing the threads in strict order.

2. Active Networks

Active and programmable networks increase the flexibility of the network infrastructure by giving users and service providers the opportunity to customize network elements to meet their own specific needs. *Active networking* will require more intelligent services from intermediate network nodes rather than just simple routing. In this paradigm packets could carry code within or refer to executables to build dynamically a service path between nodes on demand. Router needs to keep state regarding the service and expose it to the code. These codes travel inside network packets and are executed in intermediate nodes resulting in the modification of their state and behavior [2][3]. This new approach to dynamic routing based on active networking technologies, allows route allocation to respond to current factors in the network and subsequently ensures that network resources are used efficiently whilst also meeting QoS demands of the traffic.

3. QoS Routing

QoS routing is the process of selecting the path to be used by the packets of a flow based on its QoS requirements, e.g., bandwidth or delay. The exact definition of a flow is not important as long as it involves the same ingress and egress points from the network. The motivation for using a path selection that is sensitive to these requirements is the hope that it will help improve both the service received by users and the overall network efficiency. The improvement to the service received by users is in the form of an increased likelihood of finding a path that meets their QoS requirements.

A major problem with multimedia services is that their traffic demands are highly variable and modern high speed communication networks are rapidly becoming more dynamic and complex (changing the global state representation).This along with the increasing range of QoS requirements highlights the need for QoS routing to be able to dynamically adapt to changing demands.

3.1. Path Selection

The arriving traffic is identified as belonging to one of a number of categories such as video conferencing, ftp, e-mail, streaming audio and video and voice. In order to differentiate or categorize these different types of traffic flows, we can label or tag the packets so that when these packets arrive at a node, the label will be read by the microengine threads and the packet/stream type identified. Also for every traffic type each metric is weighed since it will hold differing levels of importance to each traffic type. After paths are pre-computed, the path selection phase picks a path for routing a particular request. Bandwidth however is the exception, since in all cases if the level of residual bandwidth if not sufficient to meet QoS requirements then that link is not feasible.

3.2. QoS Determination & Resource Reservation

To determine whether the QoS requirements of a flow can be accommodated on a link, a router must be able to determine the QoS available on the link. If there are critical flows that must be accorded higher priority than other types of flows, a mechanism to route such prioritized flows must be implemented in the network router. For a given network load, a high priority flow should be more likely to get a certain QoS from the network than a lower priority flow requesting the same QoS.

4. Background on Network processors

NPs are microprocessors designed specifically to build packet switches. NP provides the speed of an ASIC and at the same time is programmable. An NP consists of a number of on chip processors that can provide high throughput for network packet processing and application level tasks. The work presented in this paper is based on the fully programmable Intel ® IXP 2400 processor. Each processing element (PE) has eight hardware thread contexts that enable thread context switches that have zero or minimal overhead. It can examine and forward packets independently without using the host processor, bus, or memory. The non-preemptive nature of the threads simplifies synchronization within a microengine.

NPs use multiple execution engines each of which is a multithreaded processor core to hide DRAM latency to increase their overall computing power. NPs may also contain hardware support for hashing, CRC calculation, etc., not found in typical microprocessors. Figure 1 shows a schematic of an NP. Additional storage is also present in the form of SRAM and DRAM to store program data. In general, processing engines are intended to carry out data-plane functions. Control plane functions could be implemented in a co-processor, or a host processor.

An NPs operation can be explained in terms of a representative application:
(1) A thread on one of the processing engines finds that a new packet has arrived in the receive buffer of one of the input ports. (2) It reads the packet’s header into its registers. (3) Based on the header fields, it looks up a forwarding table to determine which output queue the packet needs to go. Forwarding tables are organized

carefully for fast lookups , and are typically stored in the high-speed SRAM. (4) The thread moves the rest of the packet from the input interface to packet buffer. It also writes a modified packet header in the buffer. (5) A descriptor to the packet is placed in the target output queue, which is another data structure stored in SRAM. (6) One or more threads monitor the output ports and examine the output queues. When a packet is scheduled to be sent out, a thread transfers it from the packet buffer to the ports' transmit buffer.

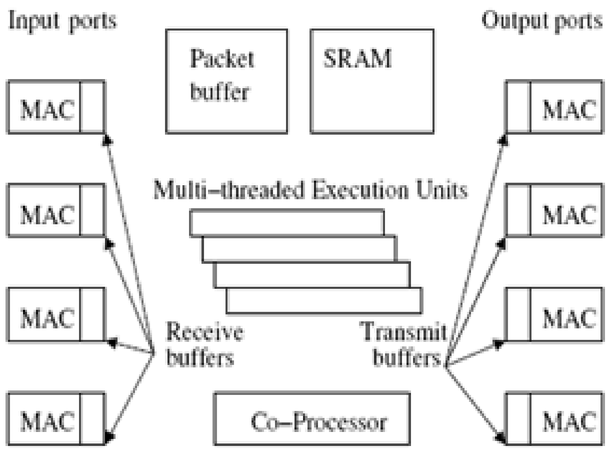

Fig. 1: Schematic diagram of a typical NP

5. Packet Filters

Simplifying, packet filtering is the process of examining a network packet's headers and deciding its fate. Examining a packet involves using bit-map-based comparisons of headers at well-known bit positions for information needed to make the decisions. Packet operations may also involve changes to the packets for routing or other network operations. After a packet has been looked at or modified, the information gathered leads to additional actions. Examples of actions are: dropping the packet, forwarding to the next stage for further processing, routing to an exit point or dealing with an anomaly.

6. Implementation design

The programming environment provided with the IXP 2400, called the Intel IXA Software Development Kit (IXA SDK) , supports a programming framework called an Active Computing Element (ACE) that encapsulates the tasks that perform independent packet processing functions. Associated with an ACE may be a piece of code that runs on a microengine, which is called a MicroACE. Code that runs on the microengines in a MicroACE is frequently called a microblock. More than one microblock may be assigned to a microengine.

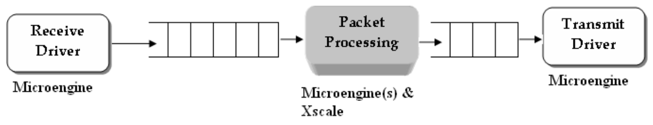

Fig. 2: Packet processing in NP

As shown in figure 2 the Ingress Micro ACE or the Receive driver handles the tasks associated with packet arrival. The Egress Micro ACE or the Transmit driver handles the tasks associated with packet transmission. The Forwarder MicroACE performs level 3 (dataplane level) forwarding of packets.

The Forwarder microblock does lookups for next hops in a forwarding table that is maintained in SRAM. With this microblock there are many possible exception conditions, such as the absence of routes for a particular destination, or the packet is an ARP or ICMP packet. In all such cases the Forwarder microblock routes the packets to its core component. The function of the *Stack* MicroACE is to take an incoming packet and present it to the protocol stack. The Stack MicroACE is used for packets that are destined for the Strong ARM.

The *Packet Filter* MicroACE encapsulates the packet filtering functionality, including the microblock that has the packet filter algorithms implemented in microblock code and filter table management in the C language, implemented as the core component. The API is used by the core user interface implementation to manipulate the filter table.

On the IXP2400 processor, the first decision in our application design is the division of work between microengines and the Intel XScale core. Microengines handle all, or most of the per packet processing. The Intel XScale core typically handles infrequently arriving packet types that require more complicated processing.

The simulator is configured to send various types of packet streams with two input ports and five output ports, each with a data rate of 1000 Mbps with receive/transmit buffer sizes of 65536 and 16384 respectively are chosen for this application.

Multimedia streams are artificially simulated and packets are injected into the Network Processor from the network through the Media Switch Fabric Interface. Traffic generated in this work consists of RTP/UDP packets, UDP packets and TCP packets. All these packets are uniformly distributed in the traffic. Then the packet data and its meta data are kept in DRAM and SRAM respectively. The packets are then forwarded to the MicroEngines for processing.

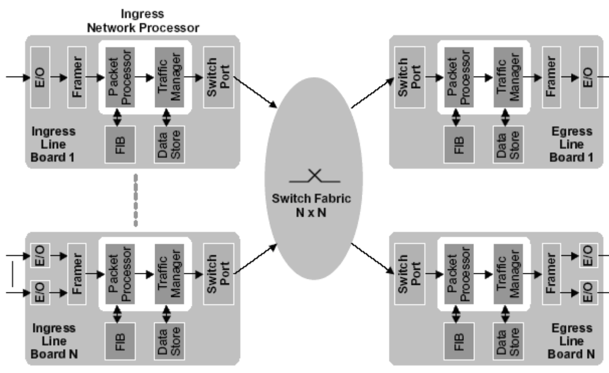

Fig. 3: Architecture of NP

Each incoming packet needs to be assigned to one of the available processors. But it is usually desirable to deliver packets of each flow in order. The main reason for doing so is to prevent *fast-retransmit* feature of TCP from resending the packets which are delivered to the destination out-of-order. When the source of a TCP flow receives acknowledgments with out-of-order sequence number, it assumes that the packets are lost in the network. But this might have simply happened because of out-of-order acknowledgement by the receiver. These unneeded retransmits can lead to significant bandwidth loss unless the router guarantees in-order routing of each flow's packets.

To do so, the router needs to identify the active flows and track them by keeping some sort of flow state. Normally this packet flow needs to be shared among different threads/NPs requiring synchronization between processing elements. Flow identification is a classification problem which can be handled by assigning a hash key to each flow using unique flow identifiers like source and destination IPs, port numbers and transport ID. Then all packets of each flow will be assigned to a fixed network processor for the lifetime of the flow. Here high priority packets are always routed to the output port 0.subsequent output ports are used for other priority flows.

Main subsequent advantage of this approach is that no longer the flow state information needs to be shared between different competing NPs/threads so synchronization will not be necessary. After processing the input stream, processed packets are driven into the network through the *switch fabric*, and scheduled in egress for output.

Two programming models exist for running the same series of packet processing components on multiple threads. When multiple microengines are used to collectively execute the same set of microblocks, the threads in these microengines can execute in an ordered or unordered fashion. Intel IXA framework supports both models of execution. Unordered thread execution means each thread retrieves a packet and process it as quickly as possible. The processing is independent of the other threads in the pool. Figure 4a & 4b shows that the Ordered thread utilization under-utilizes the processing power of the IXP micro engines by increasing the delay and jitter level when maximum load is given, so running our packet processing code on multiple threads does much to address the utilization problem. Here unordered thread utilization give us lot more computing power allowing packets to process much faster.(Figure 4a). Separate port for QoS packets with unordered execution of thread provides a good performance result.

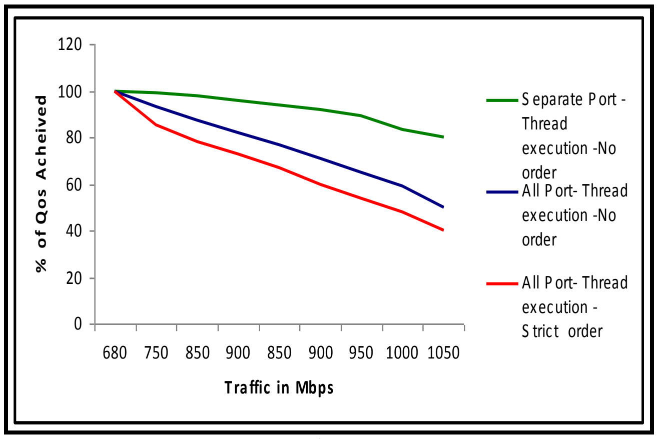

Fig. 4 a: Order of thread execution

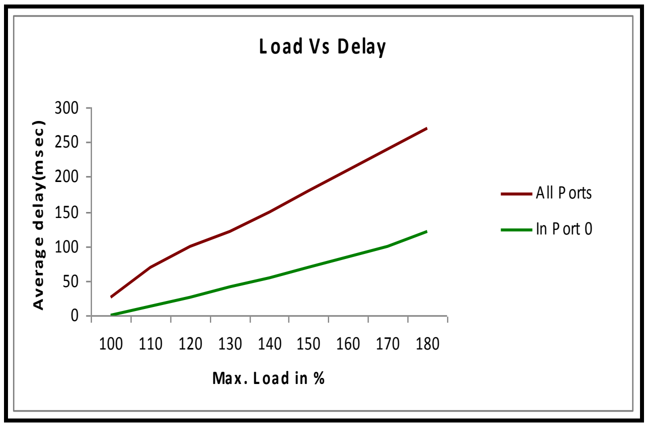

Fig. 4 b Load VS delay

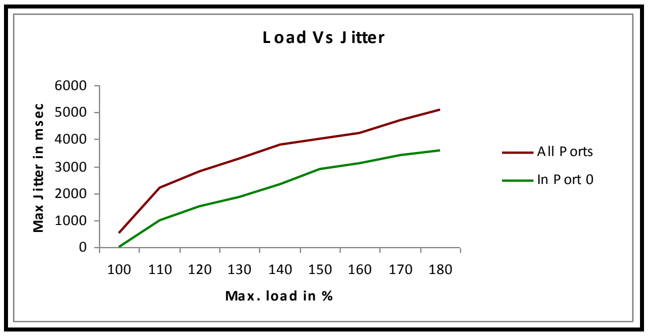

Figure 4 c

7. Summary and Conclusion

This paper shows how the end-to-end quality of service can be improved with active networks. Based on existing QoS capabilities, the active networking approach proposed here provides the necessary tools for dynamic translation between different QoS schemes and, therewith, enables efficient linkage of QoS parameters to build end-to-end services

References

[1] A. Chavez, A.Moukas, P.Maes, Challenger: A Multi-agent System for Distributed Resource Allocation Proceedings of the First International Conference on Autonomous Agents (1997), pp. 323–331. 228.
[2] Psounis K. Active Networks; Applications, Security, Safety and Architectures, IEEE Communications Surveys, Vol. 2, No. 1, First Quarter 1999
[3] Calvert, K.L.; Bhattacharjee, S.; Zegura, E.; Sterbenz, J. Directions in Active Networks. IEEE Communications Magazine, October 1998
[4] Werner Bux, et al. Technologies and building blocks for fast packet forwarding. *IEEE Communications Magazine*, pages 70–77, January 2001.
[5] Mark Kohler. NP complete. *Embedded Systems Programming*, page 45, November 2000.
[6] E.Johnson and A.Kunze, "*IXP Programming – The Microengine coding guide for the Intel IXP1200 Network Processor family",* Intel Press.
[7] Andreas Kind. *The Role of Network Processors in Active Networks*. IBM Zurich Research Lab, 2003.
[8] Intel, "lntel IXP2800 network processor,"
[9] http://www.intel.com/design/network/products/npfarmly/ixp2800.htm.

N. Saravanaselvam is working as a Professor in Department of Computer Science and Engineering at Sree Sowdambika College of Engineering, Aruppukottai, Tamilnadu, India. He has completed his B.E. Electronics and Communication Engineering and M.E. Computer Science and Engineering in Arulmigu Kalasalingam College of Engineering , Krishnankoil Under Madurai Kamaraj University, Madurai. Now he is a research scholar of Anna University, Chennai. He has guided more than 25 B.E./M.E. Projects. His field of interest is Network Engineering.

Dr.S.Radhakrishnan is presently working as Senior Professor and Head, Department of Computer Science and Engineering at Arulmigu Kalasalingam.College of Engineering, Krishnankoil, TamilNadu. He has completed his M.Tech. and Ph.D., in Biomedical Engineering from Institute of Technology, Banaras Hindu University. He has guided more than 50 M.E./M.Tech. Projects and 10 M.Phil. Thesis. Currently ten candidates are working for Ph.D. under his guidance. His fields of interests are Network Engineering. Computer Applications in medicine and evolutionary computing. He is also serving as Project Director (Network technologies) in TIFAC CORE in Network Engineering at Arulmigu Kalasalingam College of Engineering. He has more than 20 publications to his credit.