©2010 International Journal of Computer Applications (0975 - 8887)
Volume 1 – No. 16

Resource Management in Ambient Network Using Network Processor

Mrs. K.Sashi Rekha
M.E.,Lecturer(Computer Science and Engineering),
Dr.N.G.P. Institute of Technology,
Coimbatore.

Mrs. Nagalakshmi Venugopal
M.E (P.hd).,Lecturer (Computer Science and Engineering),
Dr.N.G.P. Institute of Technology,
Coimbatore.

Mr. D.Selvam
M.E. Lecturer (Information Technology),
Dr.N.G.P. Institute of Technology,
Coimbatore.

ABSTRACT

The Ambient Network project aims at designing a future networking environment where today's networks (cellular, wireless, fixed) are seamlessly integrated offering a richer and smarter networking experience to applications and users. An efficient resource management method to deal with different characteristics of the heterogeneous technologies is the need of the hour. IXP 2800 network processor is the high end device designed for 10 gigabit data rates with typical usage in high speed packet forwarding systems and ambient networks. This project aims at using network processors for solving resource management issues in ambient networks. The problem of fair allocation among contending traffic flows on a link has been extensively reasearched. Moreover, conventional resource scheduling algorithms depend strongly upon the assumption of prior knowledge of network parameters and cannot handle variations or lack of information about these parameters. In this paper a novel scheduler called the Composite Bandwidth and CPU Scheduler (CBCS). Which jointly allocates the fair share of the link bandwidth as well as processing resource to all competing flows. CBCS also uses a simple and adaptive online prediction scheme for reliably estimating the processing time of the packet.

Keywords:

Ambient Network, Network Processor, Scheduling, Distributed Applications, Packet-Switched networks

1. INTRODUCTION

Recent Technology has become indispensable to human life. Today number of network offers different kind of services to unlimited number of end users, but they are not getting satisfactory services from service provider. For instance existing mobile and wireless link layer technologies like Wireless Local Area Network, Global System for Mobile Communication, in third generation network, etc, lack a common control plane in order to enable end-users to benefit fully from the offered access connectivity. In addition access to these networks is often restricted due to security and business consideration. Although static, pre-established roaming agreements can extend the scope of these subscriptions to some other networks, there is no technology to automatically and transparently select the best and cost effective link for the end-user. Major factor that affects the services offered by these networks is congestion.
The solutions for these problems are provided in next generation communication networks with coexistence of multiple technologies and user devices of integrated fashion. One such technology is *Ambient Network*. Ambient Network aims to provide solutions encountered in current mobile and wireless networks. As ambient network compose and decompose, topology and traffic patterns changes rapidly and makes it difficult to rely on long-term network planning and dimensioning .To overcome these difficulties, mechanisms are needed to dynamically adapt changes in traffic demand and to utilize the available resources fairly. In this project a new networking concept known as Ambient Control Space and its functionalities are introduced and discussed about the design of congestion control .In this project we also identify and analyze the challenges of ambient network pose to resource management.
The main objective and goal of this project is to make better use of available resources by adapting the routing function to the current traffic situations. So that we can

- ❖ Maximize the throughput.
- ❖ Increase Fairness in resource allocation.
- ❖ To increase the packet transmission rate with minimum delay.

2. AMBIENT NETWORK

The Ambient Network project aims at creating scalable and affordable network solutions for mobile and wireless systems beyond 3G.Ambient Network (AN) contains a set of one or more nodes and devices, which share a common control plane called the Ambient Control Space (ACS). It aims to enable the cooperation of heterogeneous networks belonging to different operator or technology domains to overcome the difficulties encountered in current generation of network. Norbert Niebert, Andreas Schieder (April 2004) has analyzed the formation of Ambient Network based on three-design principle,

- ❖ Ambient Networks build upon Open Connectivity and Open Networking functions.
- ❖ Ambient Networks are based on Self-Composition and Self-Management.
- ❖ Ambient Network functions can be added to Existing Networks.

Architecture and Components of Ambient Networks

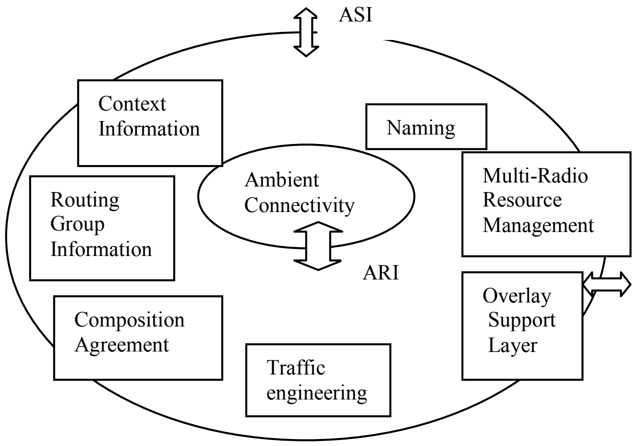


Fig. 1: Components of Ambient Network

2.1 Ambient Control Space

The ACS (Ambient Control Space) is the internal of an Ambient Network. It has the functions that can be accessed and it is in full control of the resources of the network. Ambient control space can be subdivided into the actual control functions and the control space framework functions, which are not explicitly shown but assumed to implement the loop surrounding the connectivity plane. Today's Internet Networking Technology according to Norbert Niebert, Andreas Schieder (April 2004) lack this common plane .The control space framework comprises all functions necessary to allow the control functions to plug into control space, execute their control tasks and coordinate with other functions present in the control space as said by Chen .Z, M.Mohamed ali (2004). These ACS functions can be used as a plug and play feature in existing networks.
There are three interfaces present to communicate with an ACS. These are:

- ❖ ANI: Ambient Network Interface. If a network wants to join in, it has to do so through this interface.

- ❖ ASI: Ambient Service Interface. If a function needs to be accessed inside the ACS, this Interface is used.

- ❖ ARI: Ambient Resource Interface. If a resource inside a network needs to be accessed, this interface is used.

2.2 Functional Area (FA)

FA is a concept to group functions into topic – related sets for easier reference and discussion. In this project, network processor is used inside gateways i.e. inside ANI and performs functions such as Traffic management, Queue management, Packet Processing, Packet classification etc. Network Processor used here mainly concerns with congestion control Functional Area. Congestion control Functional Area consists of connectivity plane (ACY) with some functionality as depicted in Figure 2 .In this project these functions are assumed to be performed by network processor. Collections of these functionalities are called CC-FA-CY (congestion control Functional area connectivity) and CC-FA-CS (Congestion control Functional Area Control space) respectively. CC-FA-CY includes mechanisms and techniques to interact with legacy solution such as TCP.CC-FA-CS includes all the functions that are necessary to interact with all other functional areas.

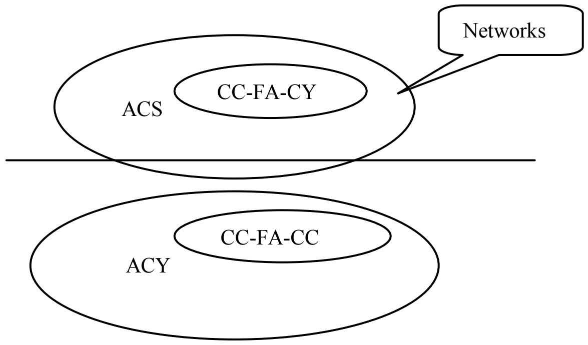


Fig. 2: Congestion control Functional Area.

2.3 Network Composition

Network Composition as demonstrated by Jorge Andres Colas (2005) is a new architectural concept introduced in ambient Networks to enable control – planes inter working and sharing of control functions among networks. Different networks may cooperate with each other dynamically for various purposes. The main intention as analyzed by Jorge Andres Colas (2005) is to provide inter working without manual intervention and prior signing of agreements between different network operators. To ensure a smooth and seamless cooperation agreement has to be made among all networks involved. A new composed network may be created when individual AN make an agreement to compose.
Generic Ambient Network signaling protocol suggested by Jorge Andres Colas (2005) is used to exchange signaling information of functional areas inside ACS.A composed network consists of all logical and physical resources and services each of its members contribute according to the composition agreement.

2.4 Characteristics of Ambient Networks

The characteristics of Ambient Networks are:

- ❖ Heterogeneity: Ambient Networks are based on a federation of multiple networks of different operators and technologies.
- ❖ Mobility: In dynamically composed network architectures, mobility of user group clusters would support effective local communication.
- ❖ Composability: An Ambient Network can be dynamically composed of several other networks. Cooperating Ambient Networks could potentially belong to separate administrative or economic entities. Hence, Ambient Networks provide network services in a cooperative as well as competitive way.

The Ambient Network Interface (ANI) facilitates cooperation across different Ambient Networks.

- Explicit Control Space: Provisioning (at least a subset of) the Ambient control.
- Space Functions: When Ambient Networks and their control functions are composed, care must be taken that each individual function controls the same resources as before: by composing two Ambient Networks, resources shall not become a common asset but rather an asset that can be traded.

3. Network Processor

Network Processor (NP) are network devices specifically designed to store, process and forward large volumes of data packets at wire speed with strong programmability. Processing of packet at wire speed has resulted in the creation of Integrated Circuits that are optimized to deal with this form of packet data, such as ASIC-based switches and routers. General-purpose processor offer programming flexibility, but they lack packet-processing performance. Network Processor has specific features or architecture that is provided to enhance and optimize packet processing within these networks.

3.1 Intel IXP2800 Network Processor

The IXP2800 is the high-end device of a family of network processors developed by Intel Corporation. It is designed for 10 Gigabit/sec data rates, with typical usage in packet forwarding systems. According to Matthew Adiletta, Mark Rosenbluth, Debra Bernstein (Aug 2002), It can be configured with large amounts of dynamic and static storage for buffering hundreds of thousands of packets for up to a million Internet Transmission Control Protocol (TCP) connections.

3.2 The XScale™ Processor

The XScale processor is compliant with the ARM Version 5TE (Advanced Risc Machines), and runs at 700MHz.Normally, it is used as a system control plane processor, handling exception packets and doing management tasks. It contains independent 32KB instruction and data caches, and a full capability memory management unit. The XScale has uniform access to all system resources, so it can efficiently communicate with the microengine though data structures in shared memory.

3.3 The IXP2XXX Microengine

Several goals guided the specification of the ME:

- High frequency to allow for sufficient instructions per packet. The ME has a six-stage pipeline and runs at 1.4 GHz.
- Large register set. Having many registers minimizes the need to shuffle program variables back and forth between registers and memory.
- Multiple threads. Given the disparity in processor cycle times vs. external memory times, a single thread of execution often blocks waiting for external memory operations to complete. Having multiple threads available allows for threads to interleave operation—there is often at least one thread ready to run while others are blocked. This makes more productive use of the other ME resources, which would otherwise be idle. There are eight hardware threads available in the ME. Each of the eight threads will always be in one of four states.
- Inactive—some applications may not require all eight threads. Unused threads can be kept in an inactive state by setting the appropriate value in a configuration register.
- Executing—the executing thread is the one in control of the ME. Its PC is used to fetch the instructions that are executed. A thread will stay in this state until it executes an instruction that causes it to go to sleep state (there is no hardware interrupt or pre-emption; thread swapping is completely under software control). At most, one thread can be in executing state at any time.
- Ready—In this state, a thread is ready to execute but is not because a different thread is executing. When the executing thread goes to sleep state, the MEs thread arbiter selects the next thread to go to the executing state from among all the threads in the ready state. The arbitration is round robin.
- Sleep—In this state, the thread is waiting for some external event(s) to occur (typically, but not limited to, an IO access). In this state the thread does not arbitrate to enter the executing state. At most, one thread can be in executing state at a time; any number of threads can be in any of the other states.

3.4 Registers

Each ME contains four types of 32-bit data path registers:

- 256 general-purpose registers
- 512 transfer registers
- 128 next neighbor registers
- 640 32-bit words of local memory

GPRs are used for general programming purposes. They are read and written exclusively under program control. GPRs, when used as a source in an instruction, supply operands to the execution data path. When used as a destination in an instruction, they are written with the result of the execution data path.

Transfer registers are used for transferring data to and from the ME and locations external to the ME (for example, DRAMs, SRAMs, etc).

Next Neighbor (NN) registers are used as an efficient method to pass data from one ME to the next, for example, when implementing a data-processing pipeline.

Local Memory (LM) is addressable storage located in the ME. LM is read and written exclusively under program control.The distinction between LM and the registers described above is that the LM address is computed by the program at run-time, whereas the register addresses are determined at compile time and bound in the instruction.

3.5 The DRAM Cluster

The DRAM cluster provides three independent DRAM controllers, each of which controls external Rambus DRAMs (RDRAMs). The reason for three channels is to provide sufficient data buffering bandwidth for 10Gb network applications. DRAMs are a good choice for a data buffer because they offer excellent burst bandwidth and are much denser and cheaper per bit relative to SRAM. Each DRAM controller, running at 133MHz provides 17Gb/s of bandwidth, shared between reads and writes. The three DRAM controllers provide hardware interleaving of the DRAM address space (often referred to as *striping*). This is done to spread accesses evenly to prevent "hot spots" in the memory.

3.6 The SRAM cluster

The SRAM cluster consists of four independent SRAM controllers, each of which controls external Quad-Data-Rate (QDR) SRAMs. The reason for four channels is to provide sufficient control information bandwidth for 10 GB network applications. SRAMs are a good choice for control information, which tends to have many small data structures such as queue descriptors and linked lists. Each SRAM controller, running at 200MHz, provides 800MB/s of read bandwidth and 800MB/s of write bandwidth. In addition to the normal read and write access, the IXP2800 SRAM controllers provide three additional hardware functions.

- *Atomic read-modify-write operations: increment, decrement, add, subtract, bit-set, bit-clear, and swap.*

The atomic operations are useful for implementing software semaphores. They can also be used for multiple processes that modify a shared variable without using conventional mutex to obtain ownership. This is more efficient, since it eliminates the mutex operation altogether in this case.

- *Linked-list queue operations.*

This hardware accelerates enqueue and dequeue to linked-list operations by eliminating the read-to-write or read-to-read latency. For example, to do an enqueue, software must read the current list tail and then use it as an address to write the new link to memory. The SRAM controller keeps the tail address in on-chip registers and does the enqueue write locally; this saves the time that would have been spent by the microengine to get the tail value and then simply use it as the address for the write.

- *Ring operations.*

A ring is also sometimes called a *circular buffer*. It consists of a block of SRAM addresses, which are referenced through a head and tail pointer. Data is inserted at the tail of the ring and removed from the head .The SRAM controller keeps the head and tail pointers in on chip registers and increments them as they are used. The advantage is that multiple processors can add data to and remove data from the rings without having to use a mutex to obtain ownership.

3.7 The Media-Switch-Fabric Interface

The Media and Switch Fabric (MSF) Interface is used to connect an IXP to a physical layer device (PHY) and/or a switch fabric. The MSF consists of separate receive and transmit interfaces. Each of the receive and transmit interfaces can be separately configured. The receive and transmit ports are unidirectional and independent of each other. Each IXP2800 port has 16 data signals, a clock, a control signal, and parity signals. There is also a flow control port consisting of a clock, data, parity, and ready status bits, and it is used to communicate between two IXP2800 chips, or an IXP2800 and a switch fabric interface. The IXP2800 supports 10Gb/s inbound traffic and 15Gb/s outbound or 15Gb/s inbound and 10Gb/s outbound.

Incoming packets are received into the Receive Buffer (RBUF). Outgoing packets are held in the Transmit Buffer (TBUF). The RBUF and TBUF are both RAMs and store data in sub-blocks (referred to as *elements*), and are accessed by either the microengines or XScale™.The RBUF and TBUF each contain 8KB of data. The element size is programmable as 64 bytes, 128 bytes, or 256 bytes per element. The microengine can read data from the RBUF to the microengine inbound registers using the MSF [read] instruction. The microengine can promote data from RBUF to DRAM directly using the DRAM [rbuf_rd] instruction. The microengine can promote data into the TBUF along with status via writes from the outbound transfer registers using the MSF [write] instruction. The microengine can control movement of data from DRAM directly to the TBUF using the DRAM [tbuf_wr] instruction.

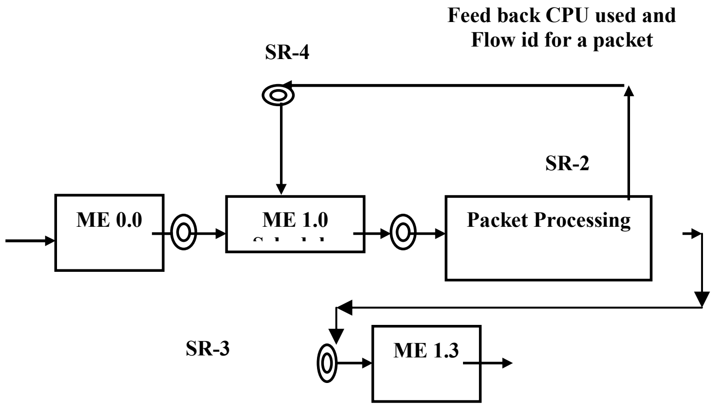
3.8 Resource Allocation

Fair allocation of shared network resources among multiple users is an intuitively desirable property. The Link bandwidth is not the only resource that is shared by the traffic flows as they traverse the network. A routers processor is often also a critical resource to which all competing flows should have fair access. For each incoming packet the router has to perform several activities like computing checksum, performing a forwarding table lookup, processing variable length option, etc. As the processing requirement of different packet vary widely, the issue of fairness in the allocation of the processing resources gains significance.

The overall fairness cannot be achieved by fair sharing of the link bandwidth alone or merely through fair allocation of processing resource alone. Therefore, for better QoS and overall fairness in resource allocations for the contending flows, it is vital that the processor and bandwidth scheduling schemes should be integrated. A Novel scheduler called the Composite Bandwidth and CPU Scheduler (CBCS) Algorithm as discussed by Fariza Sabrina, Salil S. Kanhere and Sanjay K.Jha is used to allocate resource fairly.

CBCS can schedule multiple resources adaptively, fairly, and efficiently among all the competing flows. Scheduler employs a simple and adaptive online prediction scheme called modified Single Exponential Smoothing (SES) for determining the packet execution times. Packets from each flow are first processed by the processor and then transmitted onto the output link. The joint allocation of the processing and bandwidth resource is accomplished by the composite scheduler, which selects a packet from the input buffers and passes it onto the CPU for processing. No scheduling action takes place after the processing; the packets processed by the CPU are stored in the buffer between the processor and the link and are transmitted in a first come first serve order.

4. ARCHITECTURE



4.1 System Design

The design consists of modules for packet reception, processing and transmission. This design is implemented using Intel Network Processor IXP2800.Purpose of using Intel Network Processor is that, when compare to Motorola and IBM Intel Network Processor has flexibility in programming so that it can adapt to changing technologies. In addition Intel Network Processor provides high performance. The pipeline consist of following modules

- ❖ RX Block
- ❖ Packet Classification Block
- ❖ Queue Management Block
- ❖ Decision Making Block
- ❖ TX Block

The Proposed system comprises of Network Processor IDE that provide a real time environment and are designed to use inside gateway. Here gateway is used as network interface that is used to exchange information about various networks. The checkpoint node functionalities are embedded within this network processor. Different types of real time and non real time data are given as input to different ports. The incoming packets are classified based in TCP port as real time and non real time data and are placed in separate queues. These queues are served based on weighted round robin scheduling algorithm. The selection of routes and packets are transmitted in such a way that network congestion is reduced. The efficiency of packet transmission is also increased considerably.

4.2 Allocation of Microengine

The checkpoint node functionalities are assigned to each microengine as below; For example: Microengine 0:0 is assigned to receive block. Microengine 1:0 is assigned for CBCS scheduler, Microengine 0:1,0:2,1:1,1:2 is assigned for packet processing and forwarding block, Microengine 1:3 is assigned for transmitter block.

4.3 System Function

The sequence of traffic monitoring, packet processing and optimum route selection in order to avoid packet loss and hence to decrease the overall latency is given below.

- ❖ System is configured. Script files are added in startup menu.
- ❖ Data stream from external packet generator or packgen are assigned to input ports.
- ❖ Start Simulation
- ❖ Receive packet from MSF.Signal indicates occurrence of packet at MSF interface.
- ❖ Only after signaling, buffers and thread are allocated to incoming packets. If payload is large additional buffers are allocated from buffer pool.
- ❖ Address of header and metadata are stored in queue descriptor. Only metadata information is handed over to classification block. This metadata information is written in SRAM memory using command sram_write.
- ❖ Packets are validated. Packets should follow RFC1812 rule. Some of the rules are packet header should be five bytes of length, Time to live field should not be zero, source and destination address should not be class D or class E address, Packets with zero address should be dropped.
- ❖ Packets are serviced using CBCS algorithm.
- ❖ Packets are then handed over to transmit block using dispatch loop and are then transmitted Processing of next packet is done. If no packet is present in the interface its stop processing.

5. System Modules

5.1Packet Rx to CBCS Scheduler Message Structure

On receiving a new packet, each thread in this block checks the Start of Packet (SOP) and End of Packet (EOP) bits of the packet, identifies the port of the packet, allocates a DRAM buffer for the packet on start of a new packet. Moves the data from the receive buffer to DRAM buffer, signals the next stage of the pipeline on EOP, and cleans up the state for the next round. This stage requires the following four operations. They are 1) SRAM read to allocate a new buffer; 2) DRAM write to move the packet data into the DRAM buffer; 3) SRAM write to update the packet descriptor information; and 4) a scratch ring write to signal the next pipeline stage when the entire packet has been reassembled with the packet data.
The packet Rx microengine sends enqueue messages to the CBCS scheduler microengine via SR-1 contain three long words of data.

5.2 CBCS Scheduler to Packet Processor Message Structure

The CBCS scheduler to the packet processor messages (via SR-2) contains 2 long words of data. The packet processor microengines cache (i.e., store) the first long word data in local memory and uses the same data to generate transmit message to the packet transmitter microengine. Also the sopBufferOffset value is used to access the packet metadata from the SRAM memory using the dispatch loop functions. The second long word value is used later as a part of Packet Processor to Scheduler feedback message.

5.2.1CBCS Implementation Details

Microengine local memory is used for keeping CBCS scheduler variable such as Quantum (or credit increment), packet counts for the flows or queues, credit counter per flow, estimated CPU requirements (per packet per flow) etc. The local memory is used, as it's the fastest to access. However, SRAM can be used for allocating the variables when number of flows is extremely high. The CBCS scheduler is implemented using 4 threads e.g., initialization thread, enqueue thread, dequeue thread, and CPU prediction thread. After initialization is completed, the initialization thread sends signals to the enqueue, dequeue, and CPU prediction threads to begin their tasks as they wait on the initialization thread's completion signal.

5.2.2 Initialization Thread

Initialization thread sets the SRAM channel CSR to indicate that packet based enqueue and dequeue would be done, i.e., enqueue and dequeue of a full packet is done every time. The thread also initializes SRAM queue descriptors (and queue array) and the scheduler variables (e.g., it initializes the value of quantum, credit counter for the flows, estimated CPU requirements per flow etc). After initializing the scheduler variables, the thread terminates itself so that the microengine thread arbiter excludes this thread from its list.

5.2.3 Enqueue Thread

The enqueue thread waits for the signal from the initialization thread before starting its infinite loop. In each turn, the thread calls an SRAM API (e.g. scratch get ring) to read an enqueue message from SR-1 and specifies a signal number (as a parameter to the API call). The thread then swaps out to allow other threads to run as the SRAM read operation would take some time. After receiving the control back, the thread checks the presence of the signal (i.e., checks whether the enqueue message read operation is completed or not. Once the enqueue message is read, it checks the validity of the enqueue message, as there may not be any message in the ring. If the thread receives an invalid message, it does context swap and then goes for the next turn. As shown earlier in table 1, the third LW of packet metadata contains the packet size field. So, if the enqueue message is a valid message, the thread reads the third LW of the packet metadata from the SRAM using another API (e.g. sram read) and extracts the packet size for calculating the total resource requirement (i.e. both the CPU and bandwidth) for the packet. The CPU requirement data is taken from the global variable (per flow), which is constantly updated by the CPU prediction thread. The calculated total resource requirement is used by the dequeue thread for scheduling purposes, and therefore it needs to be stored. The enqueue thread calls an SRAM API (e.g., sram write) to write back the resource requirement data to the SRAM and specifies a signal number. While the write operation is in progress, the thread calls another API to enqueue the packet info in the SRAM queue corresponding to the flow-id. It may be mentioned that the enqueue is done using the packet Next pointer (calculated using the sopBufHandle member of the enqueue message). The thread increments the packet counts for the queue and waits for the SRAM write operation to be completed. The thread then does a context swap and goes for the next round.

RR Calculations

The total resource requirement (RR) for the incoming packets is calculated in nano seconds (ns) using the following equation.

RR= CPU Cost of the packet (ns) + Transmission cost of the packet (ns)

= CPU cost (ns) per CPU Cycle * Estimated CPU Cycles Requirement + Transmission cost per byte (ns) * Packet size in Bytes

Each microengine has clock frequency of 600 MHZ i.e., 600 millions cycles per sec. Therefore, CPU cost (ns) per CPU Cycle =5/3 ns. For a 100 Mbits network interface, the transmission cost per byte would be = 80 ns.

5.2.4 Dequeue Thread

Dequeue thread waits for signal from initialization thread before starting its infinite loop. In each CBCS round, the algorithm serves all the active or backlogged flows (i.e., the flows having one or more packets in the queue). So for each flow i, the algorithm checks whether the Queue Count i.e., QC [i] (stored in global variables) is positive or not. If QC[i] is positive, it adds quantum to the value of the Credit Counter of the flow i (i.e. CC[i]), otherwise it resets the CC[i] to 0 and tries to serve the next active flow. While serving flow I within each CBCS round, the algorithm checks whether both the CC[i] and the QC[i] are positive or not. If either of them is 0 or negative, the algorithm does a context swap (so that other threads get a chance to run) and then tries to serve the next active flow. Otherwise, the algorithm calls an SRAM API (e.g., sram dequeue) to dequeue a packet info from the SRAM queue corresponding to flow i and it waits for the dequeue completion signal. After dequeue, it decrements the queue count for flow i and then it checks the validity of the dequeued buffer handle (i.e., the packetNext ptr as enqueued in the enqueue operation). If the buffer handle is invalid, it does a context swap and then tries to serve the next packet from the same flow i. For a valid dequeue of a packet, the code calls another SRAM API to read the resource requirement (RR, which is the CPU requirement plus bandwidth requirement in nano seconds) from the 7th LW of the packet metadata in SRAM (as it was stored there during enqueue operation) and waits for the read operation to complete. On completion of the SRAM read, the system signals the thread and the code then decrements the CC [i] by the value of RR. The thread then generates a scheduler-to-processor message and enqueues the message to the scratchpad ring 2 (SR-2). However, before enqueuing the message in SR-2, it checks the fullness of the ring using IXP library API and waits if the ring is full. After sending the message to the processor, the thread swaps out and tries to serve the next packet from the same flow i.

5.2.5CPU Prediction Thread

This thread waits for the signal from the initialization thread before it starts its infinite loop. In each turn, the thread calls an SRAM API to read the processor-to-scheduler message from scratchpad ring 3 (SR-3) and specifies a signal number to wait on and then swaps out so that other threads can work while it is waiting for the read to complete. After reading the message, the thread validates the message and if it's a valid message, then it updates the estimated CPU requirement of the specified flow using SES estimation technique. The estimated CPU requirements (per packet) per flow are kept in global variables.

5.3 Packet Processor to CBCS Scheduler Feedback Message Structure

After processing of a packet is completed, the processor microengine sends a feedback message to the scheduler (via SR-3) that contains two long words of data. The packet processor to CPU scheduler message structure also uses the same data structure.

5.4 Packet Processor to Packet TX Message Structure

After processing of a packet is completed, the processor microengine sends a packet transmission message to the Packet TX micro engine (via SR-4) that contains just one long word of data.

6. CONCLUSION

CBCS is a low complexity scheduler, which has better fairness and performance characteristics as compared to an implementation consisting of separate schedulers of similar complexity. With the rapid growth in link bandwidth, the duration of time that is available to a router for making a scheduling decision is diminishing rapidly. Hence it is imperative that a scheduling algorithm can be easily implementable in real hardware systems. So we developed a real world implementation of the CBCS scheduler using a network processor such as the Intel IXP2800.This algorithm can be readily adapted for the joint allocation of a combination of different heterogeneous resources such as bandwidth and battery power in mobile ad hoc, memory and processor cycles in router.

7. REFERENCES

[1] N. Niebert, R. Hancock, H. Flinck, H. Karl, C. Prehofer, “Ambient Networks Research for Communication Networks Beyond 3G”, IST Mobile Summit Lyon, 2004.

[2] F. Sabrina and S. Jha, "A novel Architecture for resource management in Active Networks using a directory service", ICT 2003, Tahiti, French Polynesia, February 23 -1 March, 2003, pp: 45-52.

[3] IXP2800 Framework developer manual, as provided with the Intel IXA SDK 3.5.

[4] IXA Portability Framework Reference Manual, as provided with the Intel IXA SDK 3.5.

[5] IXP 2800 Hardware Reference Manual, as provided with the Intel IXA SDK 3.5.

[6] P. Pappu and T. Wolf, “Scheduling Processing Resources in Programmable Routers,” Proc. IEEE INFOCOM ’02, June 2002.

[7] A. Demers, S. Keshav, and S. Shenker, “Design and Analysis of a Fair Queuing Algorithm,” Proc. ACM SIGCOMM, pp. 1-12, Sept.
1989.

[8] Website,”the Wireless World Research Forum”,http://www.wireless-world-research.org.

[9] WWI Ambient Networks, http://www.ambient-networks.org.

[10] “Intel IXP2400 Network Processor Overview,” white paper,
http://www.intel.com/design/network/products/npfamily/ixp2400.htm, 2007.