# Max-Min Resource Allocation in a Network Processor†

Sang-Yoon Yi, Minsu Shin, Junsung Kim, Sachin Lal Shrestha and Song Chong
Network Systems Lab.
Dept. of EECS
KAIST
Daejeon 305-701, Rep. of Korea
Email:(upily, msshin, quasar, sachinls)@netsys.kaist.ac.kr, song@ee.kaist.ac.kr

***Abstract*— Routers process packets and forward them to appropriate output ports. There are two resources that packets contest to acquire within a router; processing resource and bandwidth resource. Processing resource includes parsing the contents of a packet/header and do classification, lookup, check sum etc. Bandwidth resource indicates output bandwidth of a router. These two contested resources make up two-dimensional resource allocation problem, which our MAX-MIN flow control algorithm addresses. We propose an intelligent explicit rate (ER) allocation algorithm based on the control-theoretic ER allocation algorithm. In the router model with two distinct resource constraints, at a given time, either one or both resources can be scarce. Depending on the scenario, our MAX-MIN flow control algorithm intelligently allocates resources using different adaptive operations for each steady state. The algorithm maintains per-flow state making it *simple* and *scalable*. At steady state, input flow rates and queue lengths asymptotically converges to a unique and fair equilibrium point. The fairness and intelligent adaptation is verified through simulation in Intel IXP1200 Software Development Environment.**

## I. INTRODUCTION

Active network is modifying the role of a router into a computing element and not just a packet forwarding machine. Traditional ASIC routers acting as forwarding machine is outdated due to mobile codes embedded in packets of active network. Active network might be far from practical realization, but fast changing multimedia trend enforce ASIC routers out of network world owing to longer Time to Market. Therefore, routers have evolved from traditional ASIC-based router to Network Processors(NP) based router. Later one is programmable alternative of the first one, which is more flexible and easy to implement making the Time to Market less. Packets are processed through programming pipeline leading towards error free packet routing to destination. Substantial increment in packet-life spent in processing pipelines give rise of processing bottleneck. Hence without considering processing resource constraint along with output Bandwidth, fair resource allocation in a router can't be fair enough, therefore, distinctively identifying two bottleneck resources in a router as: *Processing Resource* and *Bandwidth Resource*. Processing Resource includes all system resources of the network element or network node such as CPU computation capacity, memory capacity, and internal bus bandwidth. Bandwidth resource is output link capacity.

Link bandwidth is scaling to gigabits and more due to the advent of optical fiber [1], increasing the probability of router congestion due to processing and/or bandwidth resource bottleneck. So a flow control problem is devised for a programmable router modelled on *"Two Dimensional Paradigm"* considering both processing resource and bandwidth resource together.

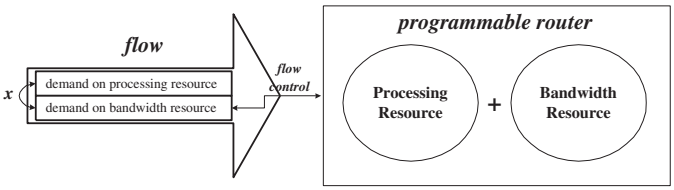



Fig. 1. Two-dimensional paradigm which considers both processing resource and bandwidth resource in flow control.

In the model presented in Figure 1, a flow has two demands namely, processing resource and bandwidth resource. For a packet, the demand on processing resource is the processing time ($cycles$) it takes to process a packet and then enqueue to the output port. Similarly, the demand on bandwidth resource indicates the requirement of adequate bandwidth to serve a packet in its entirety, i.e., length ($bytes$) of the packet determines the bandwidth requirement. For a flow, the demand on processing resource and bandwidth resource are defined as the rates, "$cycles/sec$" and "$bytes/sec$" respectively; the router allocates these resources to each flow for processing and forwarding. These two resources are shared amongst number of flows on competitive basis.

Processing resource requirement by nature is variant and thus can't be determined before the actual processing completes. As [5] suggested, the processing requirement of a packet depends on class or type of service sought by the packet. On the contrary, output bandwidth requirement of a packet is readily visible since packet length is available commodity. Each flow can have different value of $x$, a relation expressing demands for processing and output bandwidth, owing to the complexity of processing requirements of header processing and packet lengths. In active networks

†This work was supported in part by university IT research center program of the government of Korea.

[2] the intermediate nodes, called *active nodes*, are capable of performing customized computations on every packet. Since packets from multiple sources contend for both processing as well as bandwidth resources, the router is specially susceptible to ill-behaved packet sources that can generate packets at high rates and seize an unfair share of the bandwidth resource. Furthermore, packets from ill-behaved sources can completely dominate other packets by consuming much of the processing resource, thereby denying them both the processing resource and the bandwidth resource.

In the active networks [2], people have tried with various approaches to solve the two-dimensional resource allocation problem. In [3], two DRRs[8](Deficit Round Robin) are used to control resource allocation with a control feedback of a quantum which fills up the deviated fairness in the previous control period. This algorithm guarantees very strict fairness but with several weaknesses like per-flow buffers, per-flow states, and implementation complexity and possible only in both resources bottleneck.

The proposed algorithm computes Explicit Rate (ER) based on processing and/or bandwidth resource availability and the demand of each flow for these resources. ER values so generated are thus based on supply and demand at the time. Fairness is maintained by allocating additional processing and/or bandwidth resources to needy flows from those having adequate flow. With our algorithm we can achieve the following performance and complexity objectives.

- Algorithm has intelligence to identify which resource is bottleneck and accordingly compute fair MAX-MIN ER values for all the flows in MAX-MIN sense only by observing the lengths of two system buffers.
- Existence of an asymptotically stable (oscillation-free) fair-equilibrium point at which full utilization of the bottleneck resource is achieved.
- Low and scalable degree of implementation complexity with only per-flow state maintenance.

In the sections to follow, we depict the system model employing proposed MAX-MIN flow control algorithm and ER computational life cycle in section 2. We verify three steady state solutions of flow control and their existence guarantee in section 3. We discuss simulation environment with Intel IXP1200 Network Processor[7] and IXP1200 Development Tools followed by the simulation results in section 4. Conclusions and further works are in section 5.

## II. System Design

For the reader's convenience, the variables used throughout the analysis are summarized in Table I.

TABLE I

VARIABLES USED IN THE ANALYSIS AND DESIGN

| | |
|---|---|
| $a_i(k)$ | The source transmission rate of flow $i$ |
| $r_i(k)$ | The explicit rate of flow $i$ |
| $q_A(k)$ | The queue length processing buffer |
| $q_B(k)$ | The queue length transmission buffer |
| $q_T$ | The target queue length |
| $C_A$ | Processing resource capacity |
| $C_B$ | Transmission bandwidth resource capacity |
| $x_i$ | Per flow processing density |
| $T$ | Control Period [k, k+1] $j$ |
| $N$ | Number of flow |
| $p_i$ | Peak rate of flow |
| $Q$ | Set of bottleneck flow at a router |
| $\|Q\|$ | Number of flows in set $Q$ |
| $A, B$ | Controller Gain |

### A. System Model

Fig.2 represents the discrete-time system model of a programmable router employing the proposed algorithm. The node receives(Rx), processes(Ex), and transmits(Tx) packets in input flows. We have defined a new parameter, control period $T[cycles]$, signifying period at which system parameters are extracted and ER values are calculated. ER values so calculated is used for next control period and during this period ER values are not updated but calculated. During control period algorithm uses T-normalized* rate. So the router has two system capacities $C_A[cycles/T]$ and $C_B[bytes/T]$, corresponding to the two resource demands of a flow.

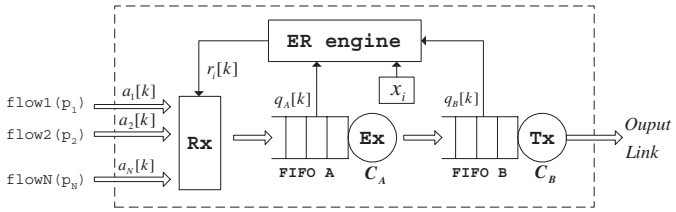



Fig. 2. System model of a router employing the proposed flow control algorithm.

$p_i$ is the peak rate of flow $i$. $q_A[k]$ and $q_B[k]$ are the queue lengths of two system buffers for processing and transmission respectively at control time $k$. Control time refers to a sample discrete time at which system snap shot is obtained. $x_i$ is the only per-flow state maintained by the router which denotes *processing density* for flow i.

$$x_i = avg\left(\frac{\text{demand on processing resource}}{\text{demand on bandwidth resource}}\right)[cycles/byte], \forall i \in N \tag{1}$$

$avg(\cdot)$ is a EWMA(Exponentially Weighted Moving Average) filter. $r_i[k]$ is the explicit rate(ER) computed by ER engine and allocated to each flow for the control period [k, k+1]. And $a_i[k]$ is the actual rate at which the router accepts data from flow $i$ during the control period $[k, k+1]$. $a_i[k]$ is the minimum of the peak rate of the flow and system provided ER rate.

$$a_i[k] = \min\{\, r_i[k], \quad p_i \,\}, \quad \forall i \in N \tag{2}$$

where $N$ denotes the set of all the input flows which pass through the router.

*Actually all control in a router is performed in discrete-time. Therefore the rate used in the control had better be "per-T" rather than "per-sec" from the controller's viewpoint.

### B. Algorithm

We say that "*flow $i$ is bottleneck at the router*" when the flow has $a_i[k] = r_i[k]$. $Q$ denotes the set of these flows. Then for the flows in $N - Q$, the input rate equals to the peak rate. This implies that the peak data rate of the flow is less than the explicit rate computed by the router and the flow cannot supply the router with enough data at the ER permitted by the router, so the router accepts data from the flow at its peak rate. The surplus amount of rate $r_i[k]$ - $a_i[k]$ is fairly allocated to other needy flows.

The ER is computed in two steps. The intermediate ER computation is based on PI control[†] of the two system FIFO queue model. This rate is BW MAX-MIN[‡] and determines the input rates of the bottleneck flows.

$$r[k] = \left[ r[k-1] - \frac{A}{|Q|}( q[k-1] - q[k-2] ) - \frac{B}{|Q|}( q[k-1] - q_T ) \right]^+ \quad (3)$$

then the final ER that is BW & CPU MAX-MIN[§] is computed as below.

$$r_i[k] = \left(\frac{q_T - q_B[k-1]}{q_T}\right) \frac{\frac{1}{x_i}}{\sum_{i\in Q}\frac{1}{x_i}} \; |Q|\, r[k] \; + \; \left(\frac{q_B[k-1]}{q_T}\right) r[k], \quad \forall i \in N \quad (4)$$

where $[\,\cdot\,]^+ = max[\,\cdot\,, 0]$.

$|Q|$ denotes the number of flows in set $Q$. $q[k]$ is the total queue length at control time $k$ and expressed as $q[k] = q_A[k] + q_B[k]$. The explicit rates $r[k]$ and $r_i[k]$ are computed in the control period $[k-1, k]$ with $[bytes/T]$ as unit. Then each flow is throttled so that transmit rate of data is at the rate $a_i[k] = \min\{r_i[k],\ p_i\}$ into the router during the control period $[k, k+1]$.

With the control algorithm (3), $q[k]$ approaches $q_T$ in steady state and we can optimize the performance of convergence by tuning the controller gain $A$ and $B$. Another notable feature of the control algorithm (3) is the normalization of the controller gain by the number of bottleneck flows. This normalization is indeed beneficial in such a way that it makes the performance of convergence to be virtually independent of the number of bottleneck flows at the router [4].

The final ER value computation algorithm (4) performs two important functions. First, it determines which resource is bottleneck by observing $q_B[k]$. Second, it allocates the bottleneck resource fairly to each flow in MAX-MIN sense. When bandwidth resource is the bottleneck, the controller allocates BW MAX-MIN ER($= r[k]$) to each flow. When processing resource is the bottleneck, the controller allocates CPU MAX-MIN ER$\left(= \frac{\frac{1}{x_i}}{\sum_{i\in Q}\frac{1}{x_i}}\; |Q|\, r[k]\right)$ to each flow that is inversely proportional to $x_i$. If both resources are bottleneck, then the controller allocates reasonable rate between the two ER values above to each flow based on the degree of bottleneck intensity determined by observing $q_B[k]$.

We can see that (4) consists of two entities. Each entity is the product of the bottleneck intensity and the fair ER for each resource. The bottleneck intensity is determined by the ratio of $\frac{q_T - q_B[k]}{q_T}$ and $\frac{q_B[k]}{q_T}$ for the two resources respectively.

From (4), we can express all the input flows in set Q as a single flow which has the total input rate as below.

$$\sum_{i\in Q} a_i[k] \;=\; |Q|\, r[k] \quad (5)$$

(5) is a notable feature of the proposed algorithm. The sum of the input rates of the bottleneck flows is determined by $|Q|$ and $q[k]$ which in turn determines $r[k]$. According to [4], the total queue length $q[k]$ converges to $q_T$ in the steady state by (3) and (5). We will discuss more about this in the next section.

## III. Analysis

Steady-state solution as well as the asymptotic stability are two prominent properties of our algorithm for MAX-MIN flow control in a programmable router.

### A. Steady State Solutions and Fairness

Fig.3 is the discrete-time queueing model of the network router model in Fig.2. Two different resources $C_A$ and $C_B$ are possible bottleneck sections and related with each other by $x_i$. Following are the assumptions employed for the analysis of the model.

- Input flows are *persistent* until the system reaches steady state. By persistent, we mean that the flow always has enough data to transmit into the router at its peak rate.
- There are no arrivals and departures of flows until the system reaches steady state. The number of flows remains constant through out the analysis.
- The processing density $x_i$ for each flow and the bandwidth capacity $C_B$ at the output link are constant until the system reaches steady state.
- The Ex. and Tx. buffer size are assumed infinite.
- Whenever Rate is mentioned in analysis and notations to follow it means T-normalized rate. So units of rate and capacity are [bytes/T] or [cycles/T] respectively.

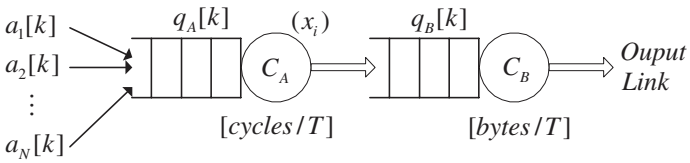



Fig. 3. Queueing model of the programmable router in Fig.2

As in fig.3 the input flows gets into buffer A with the average processing density and are processed by the CPU. If average processing density of the mixed flows in buffer A was a known quantity, then the $C_A$ divided by the average processing density would have been comparable with $C_B$ in "$bytes/T$".

The input rate at steady state for the flows in $N-Q$ converge to their peak rates $p_i$ respectively. Remaining system capacity, after assigning to flows in $N - Q$ denoted by $\hat{C}_A$ and $\hat{C}_B$ is

[†]Proportional Integral control
[‡]in agreement with the MAX-MIN fairness for bandwidth resource
[§]in agreement with the MAX-MIN fairness for processing resource

allocated to flow in $Q$. Therefore the system dynamics and the steady states are dependent on the flows in $Q$ and the two remaining system capacities after being allocated to the flows in $N-Q$. Hence we analyze the two-dimensional flow control problem by considering only the bottleneck flows. Then the remaining system capacities can be expressed as below.

$$\begin{array}{rcll} \hat{C}_A &=& C_A - \sum_{i\in N-Q} x_i\, p_i & [cycles/T] \\ \hat{C}_B &=& C_B - \sum_{i\in N-Q} p_i & [bytes/T] \end{array} \tag{6}$$

There exist three types of steady state in the two-dimensional flow control problem according to $\hat{C}_A$, $\hat{C}_B$, and $x_i$s of the flows in Q. In each steady state $q_A[k]$ and $q_B[k]$ reach different convergence values denoted by $q_A^*$ and $q_B^*$ respectively. $r^*$ and $a_i^*$ are the convergence values of $r[k]$ and $a_i[k]$ respectively. We look into these three steady states and their conditions in the following three sections respectively.

Before analyzing steady state behavior, we need to define following.

$$\bar{x}_n = \frac{\sum_{i\in Q} x_i}{|Q|}, \qquad \bar{x}_h = \left( \frac{\sum_{i\in Q} \frac{1}{x_i}}{|Q|} \right)^{-1} \tag{7}$$

where $\bar{x}_n$ and $\bar{x}_h$ are the numerical average and the harmonic average of the $x_i$ values of the bottleneck flows respectively. And $\bar{x}_h$ is always less than or equal to $\bar{x}_n$ since all $x_i$ are positive.

*1) Bandwidth Resource : $\hat{C}_A \geq \bar{x}_n \hat{C}_B$:* When the bandwidth resource is bottleneck, it is expected that $q_A[k]$ becomes zero. Then $q_B[k]$ approaches $q_T$ as $q[k]$ goes to $q_T$ as mentioned above. This implies that the sum of the input rates of the flows in Q equals the remaining transmission capacity $\hat{C}_B$ at steady state. The total processing demand of the bottleneck flows is given by

$$\sum_{i\in Q} x_i\, a_i^* = \bar{x}_n |Q|\, r^* \;=\; \bar{x}_n\, \hat{C}_B \;\leq\; \hat{C}_A \tag{8}$$

As in (8), the condition $\hat{C}_A \geq \bar{x}_n \hat{C}_B$ guarantees that the total processing demand of the flows in Q is less than or equal to the remaining processing capacity $\hat{C}_A$ because $\bar{x}_h \leq \bar{x}_n$. Therefore the first steady state scenario exists with following condition $\hat{C}_A \geq \bar{x}_n\, \hat{C}_B$ and the convergence values are given by

$$\begin{array}{rcl} q_A^* &=& 0, \quad q_B^* \;=\; q_T, \\ a_i^* &=& \left\{ \begin{array}{ll} \dfrac{\hat{C}_B}{|Q|}\,, & \forall i \in Q \\ p_i\,, & \forall i \in N-Q \end{array} \right. \end{array} \tag{9}$$

*2) Processing Resource : $\hat{C}_A \leq \bar{x}_h\, \hat{C}_B$:* As in the earlier case, its obvious that when processing resource is the only bottleneck point, $q_B[k]$ becomes zero. Then $q_A[k]$ approaches $q_T$ because $q[k]$ goes to $q_T$ as described in the previous section. This implies that the total processing demand of the flows in Q equals the remaining processing capacity $\hat{C}_A$ in the steady state. So we can write the following relations using (4).

$$a_i^* \;=\; \frac{\frac{1}{x_i}}{\sum_{i\in Q} \frac{1}{x_i}}\; |Q|\, r^* \tag{10}$$

$$\sum_{i\in Q} x_i\, a_i^* \;=\; \bar{x}_h |Q|\, r^* \;=\; \hat{C}_A \tag{11}$$

Consequently the total input rate of the bottleneck flows in the steady state is given by

$$\sum_{i\in Q} a_i^* \;=\; |Q|\, r^* \;=\; \frac{\hat{C}_A}{\bar{x}_h} \;\leq\; \hat{C}_B \tag{12}$$

As in (12), the condition $\hat{C}_A \leq \bar{x}_h \hat{C}_B$ guarantees the total input rate of the bottleneck flows is always less than or equal to the remaining transmission capacity. Therefore the second steady state does exists, as claimed, with following condition $\hat{C}_A \leq \bar{x}_h\, \hat{C}_B$ and the convergence values are given by

$$\begin{array}{rcl} q_A^* &=& q_T, \quad q_B^* \;=\; 0, \quad , \\ a_i^* &=& \left\{ \begin{array}{ll} \dfrac{\hat{C}_A}{|Q| x_i}\,, & \forall i \in Q \\ p_i\,, & \forall i \in N-Q \end{array} \right. \end{array} \tag{13}$$

While using conventional resource allocation algorithm considering only bandwidth resource, the input rates are uniformly distributed amongst all flows but the sum of the input rates is less then what we get from the proposed algorithm (5). Reason behind this can be explained with a fact that the flows of high processing density dominates other flows in the share of processing resource, thereby denying them both processing resource and bandwidth resource. These ill-behaved flows cause all flows to have uniform but less input rates.

*3) Both Resources : $\bar{x}_h \hat{C}_B \;<\; \hat{C}_A \;<\; \bar{x}_n \hat{C}_B$ :* Regardless of the controller the two-dimensional problem has this subtle region which originates from the different values of the processing density $x_i$. If all $x_i$s are the same, $\bar{x}_h$ becomes equal to $\bar{x}_n$ and this subtle region disappears. In this region, both system resources bottleneck together and both system buffers have finite queue length greater than zero. Hence, intuitively, in steady state, the desired result must converge in aforementioned region, and the proposed control algorithm (4) exactly achieves that goal.

Let's assume that $q_A[k] \to q_A^*$ and $q_B[k] \to q_B^*$ at steady state. But $q_A^* + q_B^* = q_T$ by (3). Since both resources bottleneck, we have the following two relations from (4) at steady state. Letting $\alpha = q_B^* \,/\, q_T$,

$$\sum_{i\in Q} a_i^*\, x_i = (1-\alpha)|Q| r^* \bar{x}_h \;+\; \alpha |Q| r^* \bar{x}_n \;=\; \hat{C}_A \tag{14}$$

$$\sum_{i\in Q} a_i^* = |Q| r^* \;=\; \hat{C}_B \tag{15}$$

From (14) and (15), we obtain

$$\hat{C}_A \;=\; (1-\alpha) \left( \bar{x}_h \hat{C}_B \right) \;+\; \alpha \left( \bar{x}_n \hat{C}_B \right) \tag{16}$$

We know that $0 < \alpha < 1$ and $\bar{x}_h \leq \bar{x}_n$. Therefore the condition for the third steady state is satisfied.

$$\bar{x}_h \hat{C}_B \ < \ \hat{C}_A \ < \ \bar{x}_n \hat{C}_B \tag{17}$$

This implies that the assumption is correct for the condition $\bar{x}_h \hat{C}_B < \hat{C}_A < \bar{x}_n \hat{C}_B$ and $q_A[k] \rightarrow q_A^*$ and $q_B[k] \rightarrow q_B^*$ at steady state. From (16), we obtain,

$$\alpha \ = \ \frac{\hat{C}_A - \bar{x}_h \hat{C}_B}{\bar{x}_n \hat{C}_B - \bar{x}_h \hat{C}_B} \tag{18}$$

The third steady state exists as we claimed and the convergence values are given by

$$q_A^* \ = \ (1-\alpha)\, q_T, \quad q_B^* \ = \ \alpha\, q_T \tag{19}$$

$$a_i^* \ = \ \begin{cases} (1-\alpha)\bar{x}_h \hat{C}_B \ + \ \alpha \dfrac{\hat{C}_B}{|Q|}\,, & \forall i \in Q \\ p_i\,, & \forall i \in N - Q \end{cases} \tag{20}$$

From (6), (16) and (20), we can verify that $\sum_{i \in N} x_i^* a_i^* = C_A$ and $\sum_{i \in N} a_i^* = C_B$. This implies that the processing resource and the bandwidth resource are fully utilized by the proposed algorithm when both resources bottleneck.

Lets consider an example case of 4 flows with different $x_i s$, $(x_1, x_2, x_3, x_4)$ = (1, 2, 4, 8). Then $\bar{x}_h = 2.13, \hat{x}_n = 3.75$. In figure 4, x-axis represents ratio of processing resource and bandwidth resource and also shows stable value of each flow shown in different bottleneck conditions. When a system is computationally bottleneck ($\hat{C}_A < \bar{x}_h \hat{C}_B$), flows have reverse proportionality to $x_i s$. If a system is bandwidth bottleneck ($\hat{C}_A > \bar{x}_n \hat{C}_B$), flows have identical rates due to bandwidth MAX-MIN fairness. If two resources are both bottleneck ($\bar{x}_h \hat{C}_B \ < \ \hat{C}_A \ < \ \bar{x}_n \hat{C}_B$), flows have rates as a linear combination of two different rates to utilize both resources fully.

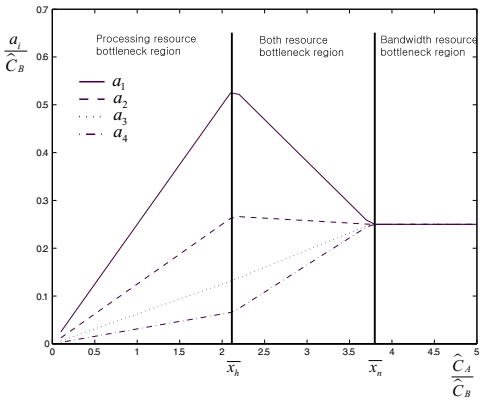



Fig. 4. Flow rates for 3 different bottleneck region

Necessary and sufficient condition and optimum gain values for Asymptotic Stability, using system of closed-loop equation of system dynamics, can be obtained for each of the three regions of the previous section. We found the optimum control gain values through computation.

$$A = 0.32, \quad B = 0.05$$

But due to the space limitation, we omit the discussion of Asymptotic Stability.

## IV. Simulation Results

The architecture of IXP1200, multi-coprocessors and multi-contexts, is a general architecture in today's network processors. IXP1200 consists of one StrongARM core(166Mhz) and six co-processors, referred to as microengine(166Mhz), where traffic management modules of control-plane are embedded into StrongARM and packet processing modules of data-plane into microengines. Scalability concerns in data-plane packet processing could be satisfied with the four zero context switching overhead hardware contexts in each of the six microengines.

There are several studies that evaluated the IXP1200. [10] concludes that the SDRAM storing packets is the bottleneck in IP forwarding. [9] concludes that the SRAM and microengines are the double bottlenecks in prototyping a DiffServ edge router with IXP1200. They have shown that the processing resource is easy to be bottlenecked in today's complex network services which motivated this work.

For simulation, an IP forwarding engine employing the proposed algorithm was implemented in the software development environment, referred to as IXP1200 Developers Workbench. Fig.5 represents the system model of the IPv4 (RFC 1812 compliant) forwarding engine implemented for the simulation. FIFO A is the buffer for processing and FIFO B is the buffer for transmission. Four microengines(ME0 - ME3) are used to receive packets from four MAC ports and microengine(ME4) carries packet processing of FIFO A, leaving Microengine(ME5) to transmit packets from FIFO B to the output link. The microengine codes for Rx MEs, FIFO queue and Tx ME are programmed based on the codes in [11]. The proposed control algorithm in (3) and (4) is performed by Strong ARM core in hardware system. In our simulations, we used step size T = 240000 cycles and the core clock cycle of microengine was 166MHz.

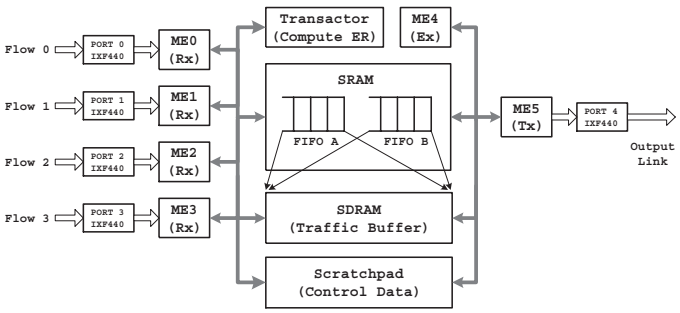



Fig. 5. System model of IP forwarding engine implemented for the simulation.

### A. Software Simulation Results

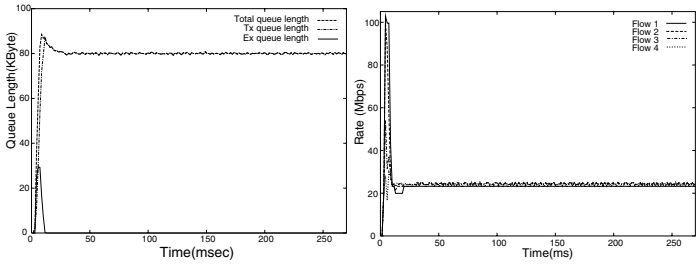
In this simulation, the length of the packets in all flows is fixed per flow for ease of applying per-flow processing density $x_i$. In our simulation, the packet processing is only header processing.

Fig.6 represents the queue dynamics and the input rates in the region of §III-A.1. ExQ and TxQ indicate the FIFO A and FIFO B respectively. All input flows have the peak rate of 100Mbps but the processing densities are different as described below, which are determined by packet length. $\hat{C}_A$ fixed at 240000 cycles/T and $\hat{C}_B$ values mentioned in the table below apply to the region of §III-A.1, thereby giving rise to Bandwidth resource bottleneck case. The simulation scenario is summarized in Table II under Bandwidth section. Theoretical Fair Rate and Actual Rate are denoted by FR and AR respectively.

In fig. 6, $q_B[k]$ reaches $q_T$ and $q_A[k]$ reaches zero. Since bandwidth resource is bottleneck, all flows have same input rates regardless of their $x_i s$ as in the second graph in fig. 6.

TABLE II
SCENARIO DETAIL

| Flow No. | Bandwidth | | Processing | |
|---|---|---|---|---|
| | Weight | FR/AR | Weight | FR/AR |
| 0 | 2.5 | 25/23.2 | 4.7 | 70.8/65.7 |
| 1 | 5 | 25/23.8 | 9.4 | 35.4/32.9 |
| 2 | 10 | 25/24.4 | 18.8 | 17.7/16.5 |
| 3 | 20 | 25/24.7 | 37.5 | 8.9/8.3 |
| | $\hat{C}_B = 100Mbps$ | | $\hat{C}_B = 300Mbps$ | |
| | $TxQ : 80015, ExQ : 0$ | | $TxQ : 150, ExQ : 79217$ | |



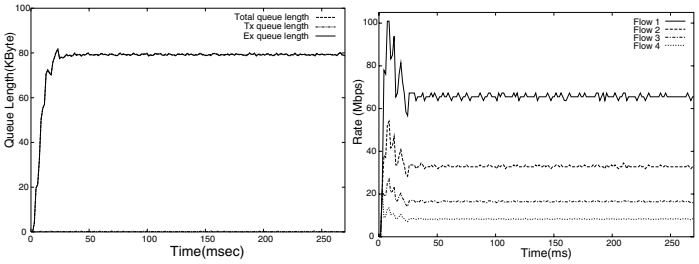
(a) Queue Length (b) Flow Rate

Fig. 6. Bandwidth Resource Bottleneck Case

Fig.7 represents the queue dynamics and the input rates in the region of §III-A.2. Unlike the scenarios in the previous section, the processing densities increased but the ratio among them remains unchanged. The simulation scenario is summarized in Table II under Processing title.

In Fig.7, $q_A[k]$ reaches $q_T$ and $q_B[k]$ reaches zero. The processing resource is bottleneck, so the input rate of each flow is inversely proportional to its $x_i$ as inducive from Fig.7.

Fig.8 represents the queue dynamics and the input rates in the region of §III-A.3. The processing densities, same as first case, were adjusted as below. The simulation scenario is summarized in Table III.



(a) Queue Length (b) Flow Rate

Fig. 7. Processing Resource Bottleneck Case.

TABLE III
SCENARIO DETAIL

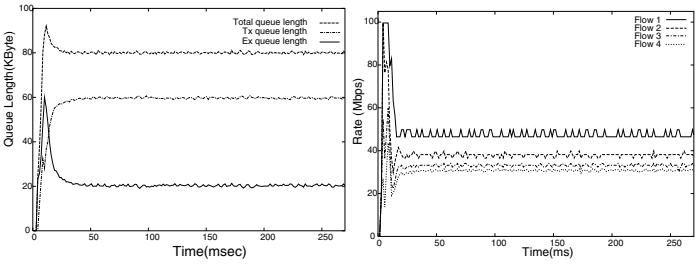
| Flow No. | $\alpha = 0.746$* | | $\alpha = 0.393$** | |
|---|---|---|---|---|
| | Weight | FR/AR | Weight | FR/AR |
| 0 | 2.5 | 50.3/47.4 | 2.5 | 78.4/75.3 |
| 1 | 5 | 40.1/37.9 | 5 | 49.1/46.1 |
| 2 | 10 | 35.1/33.2 | 10 | 34.4/31.5 |
| 3 | 20 | 32.5/30.8 | 20 | 27.1/24.3 |
| | $\hat{C}_B = 158Mbps$ | | $\hat{C}_B = 189Mbps$ | |
| | $TxQ : 59699, ExQ : 20316$ | | $TxQ : 31441, ExQ : 47703$ | |

*: The degree of Bandwidth bottleneck is more compared to Processing resource bottleneck where $\alpha = \frac{q_B^*}{q_T} = 0.746$ with theoretical value for $\alpha$ being 0.760.

**: The Degree of Processing resouce bottleneck is more compared to Bandwidth bottleneck where $\alpha = \frac{q_B^*}{q_T} = 0.393$ with theoretical value for $\alpha$ being 0.419.

Above values are obtained using equations (18) and $\alpha = \frac{q_B^*}{q_T}$.

In Fig.8, we can identify that $q_A[k]$ and $q_B[k]$ reach finite values less than $q_T$ as in (19). Since both the resources are bottleneck, they are fully utilized together. We can identify that the sum of input rates of all flows equals $C_B$ in Fig.8. According to (20), $a_i^*$ takes a value between $\frac{C_A}{|N|x_i}$ and $\frac{C_B}{|N|}$ . However, it is closer to the latter because $q_B$ is larger than $q_A$.

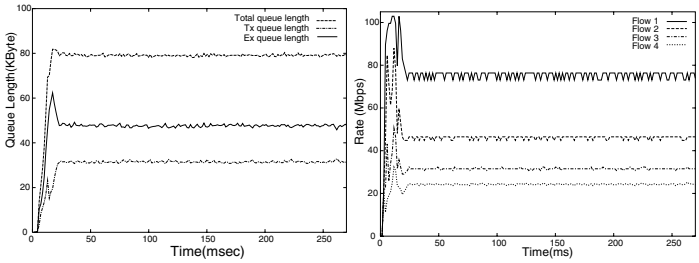


(a) Queue Length (b) Flow Rate

Fig. 8. Both resources bottleneck with more Bandwidth resource Bottleneck Case.

Fig. 9 represents a different scenario. In this scenario, $q_B$ becomes larger than $q_A$ and $a_i$ approaches closer to $\frac{C_A}{|N|x_i}$ than $\frac{C_B}{|N|}$ as in Fig. 9. Therefore the input rate of each flow is more affected by its processing density while the sum of the flows

still equals to $C_B$.




(a) Queue Length (b) Flow Rate

Fig. 9. Both resources bottleneck with more Processing resource Bottleneck Case.

Practically, full utilization of link capacity cannot be achieved due to inter-packet gap requirement that introduces a time delay between successive data packets mandated by the network standard for protocol reasons. Due to the reason mentioned above, there existed a difference between theoretical fair rate and actual rate.

## V. Conclusions

In this paper we have shown that the MAX-MIN flow control in a programmable router is a two-dimensional problem and we should approach the problem with *two-dimensional paradigm* which considers both *Processing Resource* and *Bandwidth Resource* together. We defined the MAX-MIN fairness in the two-dimensional paradigm and suggested a MAX-MIN explicit rate(ER) allocation algorithm for the two-dimensional flow control. In the extension of the flow control problem to the two-dimensional domain, we have found the existence of three steady state solutions with certain conditions. In each steady state, the convergence values of system parameters and MAX-MIN ERs have been determined. Further study into the proposed algorithm shows asymptotic stability with the given stability condition for each steady state. The results from the simulations using the Intel IXP1200 Evaluation Platform proved that the proposed algorithm achieves the goal of fairness and performance in each steady state.

Many further work remain in the extension of this study. A direct extension of this work is to apply the proposed algorithm to the source flow control in network level. When a source receives the ERs computed by the proposed algorithm from the network nodes in its routes, the only thing that the source should do is to select the minimum among the ERs and send data at the selected ER.

## References


[1] Dominic Herity, *Network Processor Programming, Embedded Systems Programming,* (7) (2001)

[2] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, *A Survey of Active Network Research, IEEE Communications* Volumn 35, Number 1, (1) (1997) 80-86.

[3] V. Ramachandran, R. Pandey, and S-H. Gary Chan, Fair Resource Allocation in Active Networks, *IEEE/CCN Proc. Ninth International Conference* (2000).

[4] Song Chong, Sangho Lee, and Sungho Kang, A Simple, Scalable, and Stable Explicit Rate Allocation Algorithm for MAX-MIN Flow Control with Minimum Rate Guarantee, *IEEE/ACM Trans. Networking* **9-3** (6) (2001).

[5] Tilman Wolf and Mark A. Franklin. Locality-aware predictive scheduling for network processors. *In Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 152.159, Tucson, AZ, November 2001.

[6] John G. Proakis, Dimitris G. Manolakis, Digital Signal Processing, Prentice-Hall International, Inc., 3rd edition.

[7] Intel, IXP1200 Data Sheet, Intel document number 278298-004, (5) (1995)

[8] M. Shreedhar, and G. Varghese, Efficient Fair Queueing Using Deficit Round-Robin, *IEEE/ACM Transactions on Networking* **4-3**(6)(1996) 375-385.

[9] Ying-Dar Lin, Yi-Neng Lin, DiffServ over Netowkr Processors: Implementations and Evaluation, *IEEE/Proceedings of the 10th Symposium on High Performance Interconnects Hot Interconnects* **4** (2002).

[10] Tammo Spalink, Scott Karlin, and Larry Peterson, Building a Robust Software-Based Router Using Network Processors, *Proceedings of the 18th ACM Symposium on Operating Systems Principles(SOSP)* (2000).

[11] Erik J. Johnson, Aaron R. Kunze, IXP1200 Programming, Intel Press, (2) (2002)