社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＱＡＩ－15（1）
2005／5／25

マルチプロセッサ向けルータアルゴリズムに関する検討

清水　敬司†　尾花　和昭†　高原　厚††　小口喜美夫†††

† NTT 未来ねっと研究所　〒239-0847 神奈川県横須賀市光の丘 1-1
†† NTT ビズリンク株式会社　〒101-0003 千代田区一ツ橋 2-5-5
††† 成蹊大学　〒180-8633 武蔵野市吉祥寺北町 3-3-1

あらまし　近年のハイエンドルータでは、機能追加の柔軟性と高速性を両立すべく、ソフトウェアにより機能変更が可能なマルチプロセッサシステムが利用されている。ところが、実装されているルータアルゴリズムのほとんどは、元来単一プロセッサ向けに設計されたものである。そこで本研究では、パケット処理プロセッサの概略モデルと、その上に実装するルータアルゴリズムについて検討を行う。特に TCP/UDP フローを意識したパケット処理を行うフローベースアルゴリズムに着目し、並列処理による性能向上を阻害する要素を明らかにする。公平な帯域共有を行いながら、高い帯域を利用しようとするフローにペナルティーを与えることができるフローベースの AQM プロトタイプ実装とその評価の結果を示し、マルチプロセッサ向きのアルゴリズムをいかに設計できるかについて議論する。
キーワード　パケットプロセッサ、並列処理、パラレル MXQ, アクティブキューマネージメント

Router Algorithms for Multiprocessor-based Internet Routers

Takashi SHIMIZU†, Kazuaki OBANA†, Atsushi TAKAHARA††, and Kimio OGUCHI†††

† NTT Network Innovattion Laboratories　Hikarinooka 1-1, Yokosuka, 239-0847 Japan
†† NTT BizLink Corporation　Hitotsubashi 2-5-5, Chiyoda-Ku, 101-0003 Japan
††† Seikei University　Kichijouji Kitamachi 3-3-1, Musashino, 180-8633 Japan

Abstract　This paper studies router algorithms in the context of executing them on multiprocessor systems, which are becoming increasingly common in modern high-speed Internet routers. By modelling the unique environment that is provided by such multiprocessor systems, we identify a couple of design issues for the successful parallel execution of router algorithms, especially by exploring flow-level parallelism. The successful execution depends on the level of concurrency in algorithms, and concurrency is restricted by access contention for shared data resources. We show how flow-level parallelization can reduce access contention, and how we can design multiprocessor-oriented router algorithms. As an example, we describe the MXQ algorithm, an flow-based AQM algorithm, that is efficiently realized on a multiprocessor system. The real implementation on a real high-speed multiprocessor-based router, and the results of the experiments in a laboratory testbed show that the algorithm can scale well up to tens of thousands of flows at the line rate of 10 Gb/s.
Key words　Packet Processor, Parallel MXQ, Active Queue Management

1. Introduction

To meet the conflicting requirements for high performance and flexibility, most of modern IP routers are designed to use, what is called, packet processors or network processors. The packet processor is a special-purpose VLSI chip in which a number of processing elements or purpose-built CPUs are embedded by utilizing advanced SoC (System-on-Chip) technology [3], [5]. Consider the fact that we have only 8ns for processing an 40-byte packet at 40 Gb/s link rate, and typical IP packet processing requires about 500 instructions. It is highly possible that such multiprocessor systems continue to be used for the next decade [2].

IP router algorithms are originally designed for a single processor system. Obviously, it is not straightforward to implement them onto such a multiprocessor system. Because they contain considerable amount of task

and data dependencies and conditional branches. However, if we view from a different perspective, focusing on the dependency among arriving packets, we can explore parallelism based on the granularity of "a flow", which is the stream of packets that share its source and destination.

The common wisdom in the Internet says that per-flow state at routers must be avoided because maintaining such a state in an environment as dynamic as the Internet would limit the speed of the routers. In practice, however, various types of flow-based algorithms are implemented onto modern routers, and are used in every segment of the Internet. Examples include flow-based policing and shaping, filtering, and so forth. The router architecture has been evolving for those decades utilizing the advancing VLSI technology. Therefore, the router algorithms must be reviewed in order to understand the state of the art.

This paper studies router algorithms in the context of executing them on multiprocessor systems, especially by exploring the flow-level parallelism. By modelling the unique environment that is provided by such multiprocessor systems, we identify a couple of design issues for the successful parallel execution of router algorithms.

The successful execution depends on the level of concurrency in algorithms, and concurrency is restricted by access contention for shared data resources, which need some form of arbitration. In this paper, we design the parallel version of an AQM algorithm, called MXQ [6], which is designed for not only providing fair bandwidth distribution, but also regulating high-bandwidth flows, and show how flow-level parallelization can reduce data access contention, and how we can design the multiprocessor-oriented router algorithms.

We have developed a prototype system by using a real packet processor, and evaluate this performance in a large scale laboratory testbed. The result shows that this parallel algorithm is feasible up to tens of thousands flows at the speed of 10 Gb/s.

2. Execution Environment

This section describes the simplified execution environment for router algorithms to explore the flow-level parallelism, and clarifies the issues for their successful parallel execution.

2.1 Hardware Architecture

Most of the packet processors in modern IP routers are designed to be software programmable. This is because routers must evolve to support emerging new standards that reflect diversified and ever-changing user requirements. If we assume that the standard IP forwarding algorithm is implemented as a software code, it requires about 500 instructions. It is difficult for a single processor system to satisfy the processing time requirements, even though it could use high clock frequency around 1GHz. Therefore, most of packet processors are designed as multiple processor systems to increase throughput by exploring parallelism [1], [2].

Figure 1 illustrates our packet processor model that consists of M processors, on-chip shared memory, and external memory banks. We assume that the sufficient bandwidth is available for all data communications among them. Most packet processors are designed to use pipeline techniques. Those pipeline techniques can be realized at various levels of tasks, such as the packet level, the instruction level, and so forth. For simplicity, our model assumes the whole tasks related to a single packet are assigned to the same processor.

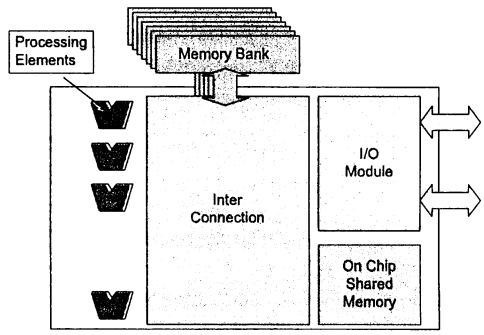

図 1 Packet Processor Model

2.2 Definition of flow

There are numerous definition of "a flow" in the literature. In this model, the flow is defined as what is called, the five tuple: Source IP address, Destination IP address, Source port number, Destination port number, and protocol number. In the case where the port numbers are not available (e.g. ICMP), the port numbers can be regarded as zero. We assume the flow is unidirectional throughout this paper, and do not consider to associate two flows with opposite directions, which actually related.

In addition, we mainly consider the flows identified "on the fly" [4] from packet header fields. We assume that the decision is completely local at the router, and do not consider the issues such as the synchronization of per-flow states among routers along the path.

2.3 Flow Table

Flow-based algorithms use a flow table to maintain flow-specific attributes. We may assume an entry in the flow table consists of a key and a set of attributes.

The key is a unique identifier in the flow table to access the set of attributes for a certain flow. The set of attributes may include not only the packet header fields, but also quality of service parameters, statistics counters, and algorithm-specific data.

In order to maintain the data consistency in a simple

manner, the flow-specific attributes must be locked at the time when they are read until when they are written back. However, each flow entry is independent by nature, thus, the attributes for different flows can be read and be written in parallel.

The size of the flow table must be sufficiently large to support from 10,000 up to 500,000 flows. Therefore, the flow table is stored on the external memory bank that is shown in Figure 1.

2.4 Shared data structures

In addition to the flow specific attributes, flow-based algorithms requires shared data structures such as the number of flows, queue length, fair share bandwidth, aggregated bandwidth, sorted lists of flows and so forth. Such shared data structures are accessed by a number of processing elements as well as the outgoing interface at every packet arrival and departures. Those are stored on the on-chip shared memory in Figure 1.

2.5 Structure of flow-based algorithms

The structure of flow-based algorithms consists of enqueueing part and dequeueing part. Enqueueing part includes all the steps until a received packet is placed in a packet buffer. Dequeueing part includes all the steps for transmitting packets to the switching fabric. Both parts includes the steps that are specific to flow-based algorithms.

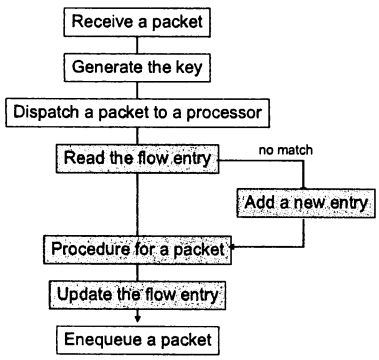

図 2 Enqueueing part

Figure 2 shows the flowchart of enqueueing part. After receiving a packet, we first generate a key for accessing the corresponding entry in the flow table. This key is first used for selecting the processor to dispatch this packet in this model. At the selected processor, we access the flow table entry with the key. If no matching key found, we create a new entry. Note that we do not write it into the flow table at this time. Then we perform the main procedure of the algorithm as well as standard IP procedure, such as forwarding table look up, and write the updated flow entry back into the flow table. Finally, we place a packet to a packet buffer.

The shaded boxes are the steps that we can parallelize by scheduling tasks efficiently. Obviously, the other steps must be designed so as not to limit the overall performance.

Figure 3 shows the flowchart of dequeueing part. After transmitting the previous packet, we select the packet to transmit next. We may generate a key for accessing the corresponding entry in the flow table if necessary, update the flow attributes, and transmit the packet to the switching fabric. The shaded boxes again are the steps that we can parallelize by scheduling tasks efficiently.

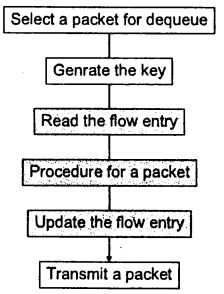

図 3 Dequeueing part

3. Concurrency of Algorithms

The successful execution depends on the level of concurrency in algorithms, and concurrency is restricted by access contention for shared data resources. The shared data resources in our model are flow table and shared data structures. Those shared resources may be accessed both in enqueueing part and dequeueing part. In order to maintain data consistency, once an entry or a shared data structure is read by a certain processor, the access must be postponed until the relevant procedure is completed, and the resulting value is written back.

Therefore, the level of concurrency can be evaluated by how much we can reduce access contention by scheduling tasks. The next step is to revise an algorithm to minimize the duration of processing over shared data structures, or to emulate them with approximation techniques, thus creating an multiprocessor-oriented algorithm.

3.1 Access to Flow Table

The access to the flow table during the enqueueing part can be arranged in a skewed fashion as shown in Figure 4. The schedule of flow table access is skewed aligned with the access time T_{ft}. We can guarantee that different processors never access the same flow entries by dispatching incoming packets to processors so that packets belonging to a certain flow always assigned to the same processor.

The skewed scheduling may work for the dequeueing part, only if the selection of the packet transmission be made in per-flow basis with regular intervals. However,

the most scheduling algorithms does not provide regular schedule to achieve efficient link sharing. In such a case, we would have to resolve access contention before deriving transmission schedule. This means that access contention of flow table may not be removed by this technique.

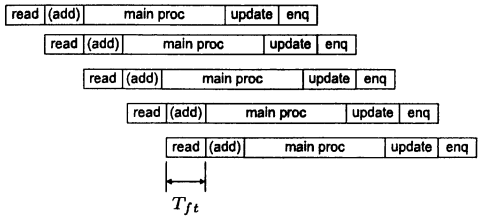

図 4 Skewed Execution Schedule

3.2 Access to Shared Data

The skewed scheduling can not work well for the accesses to the shared data structures. This is because they must be updated packet by packet. The overall performance of algorithms is determined by the duration required for the task over a shared data and the frequency of access.

For example, while queue length must be updated at every packet arrival in enqueueing part, and at every packet departure in dequeueing part, the link utilization must be updated only at every packet departure. This means there is the potential of performance improvement at most twice by using link utilization instead of queue length.

One of the shared variable frequently used in flow-based algorithms is the number of active flows. It is the number of flow entries whose packets has been recently received. It is realized by using aging techniques that utilize the flow attribute "the arrival time of the last packet". However, in order to maintain it accurately, they require considerable amount of flow table accesses, which might limit the overall performance.

4. Case Study

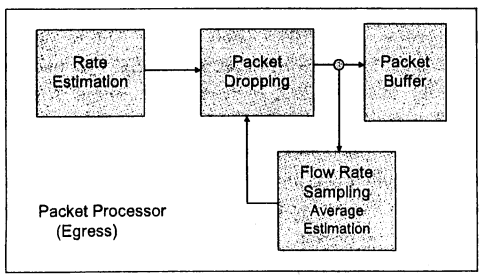

図 5 Block Diagram of Parallel MXQ

The MXQ algorithm [6] has been proposed as the flow-based algorithm that achieves efficient control of high-bandwidth flows in addition to fair bandwidth distribution.

In this section, we describe the parallel version of the MXQ algorithm. Our focus is on how the algorithm is designed that is friendly to multiprocessor systems.

4.1 Parallel MXQ algorithm

Figure 5 illustrates the block diagram of the MXQ algorithm. The following subsections describe each block in this diagram. We show a couple of modifications to allow us to reduce access to shared data structures.

4.1.1 Rate Estimation

When a packet of flow i arrives at time t, the flow rate $r_i(t)$ is calculated as a moving average of instant rate over a time period. For the computational efficiency, we use the linear approximation as follows.

$$r_i(t) = \left(1 - \frac{\Delta t}{T}\right) \times r_i(t - \Delta t) + \frac{Packetsize}{T}, \quad (1)$$

where T is the time factor, which is set to 1 second, and Δt is the interval from the previous packet. This step is executed in enqueueing part, and only use the flow attributes. Access contention is removed by the skewed scheduling.

4.1.2 Average Rate Estimation

MXQ needs to estimate the average rate F, which is expressed as

$$F = \frac{\sum_{i=0}^{N-1} r_i(t)}{N}, \quad (2)$$

where N is the number of active flows.

We sample the flow rate with the probability inproportional to the packet rate $\gamma_i(t)$ as follows.

$$\frac{1}{\gamma_i(t)} = \frac{\ell_i}{r_i(t)}, \quad (3)$$

where ℓ_i is the packet size of the arriving packet of flow i. Under this sampling technique, the expectation of the sampled flow rates can be regarded as the average rate.

We derive the average of samples by using a moving average method. The average rate F is updated by the following equation whenever a new sample is taken.

$$F \leftarrow (1 - \omega) \times F + \omega \times r, \quad (4)$$

where r is the sampled flow rate, and ω is a moving average weight, which is set to $\frac{1}{128}$. Practically, the examination of sampling is done in a granularity of 64K byte time in the system. In a 10 Gb/s link, we average over the interval of approximately 6 ms.

This step is executed in enqueueing part, and use one shared variable F. One multiplication and addition is required to update it. Note that this technique removes the use of "the number of active flows".

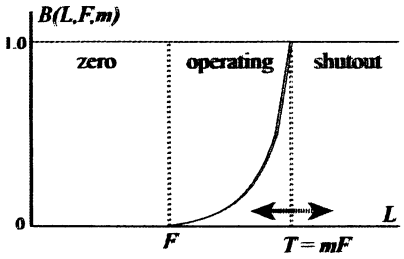

図 6 Simplified view of the function B, when $m > 1$

4.1.3 Drop Probability

Let L denote the flow rate calculated for an arriving packet. We define the moving shutout threshold T as $T = mF$. Here, F is the average rate, and m is the congestion indication that quantifies the level of congestion. We will discuss how to derive m in the next section. The drop probability B is described as in equation (5) for $m > 1$ (i.e. $F < T$), and equation (6) for $m \leqq 1$ (i.e. $T \leqq F$).

$$B(L, F, m|m > 1) = \begin{cases} 1 & T \leqq L \\ \left(\frac{L-F}{T-F}\right)^k & F \leqq L < T \\ 0 & L < F \end{cases} \quad (5)$$

$$B(L, F, m|m \leqq 1) = \begin{cases} 1 & T \leqq L \\ 0 & L < T \end{cases} \quad (6)$$

A simplified view of the drop probability B when $m > 1$ is shown in Fig. 6. When $m > 1$, the curve is divided into three regions: zero region ($0 \leqq L < F$), operating region ($F \leqq L < T$), and shutout region ($T \leqq L$). When a packet arrives, B is calculated using its rate estimate L. If the rate estimate L lies in the shutout region, the packet is dropped with probability 1.0. If it lies in the zero and operating region, the smaller the L is, the smaller the B becomes.

The boundary of the shutout region is determined by the moving shutout threshold T. If congestion occurs the value of m decreases, so the threshold T decreases, and the shutout region is extended towards the zero region. When congestion is lasting for a long time, which may be regarded as the existence of a unresponsive flow, the shutout threshold T moves further, and incoming packets of high bandwidth flows are aggressively discarded with probability 1.0. Note that in the case where $m \leqq 1$, the operating region disappears.

The slope in the operating region absorbs the temporal fluctuation of steady state flows. According to our evaluation, the ($k = 2$) gives the optimum performance for TCP flows.

4.1.4 Congestion Indication

The congestion indication m quantifies the level of congestion to adapt the algorithm to the changing link utilization. The queue length is used in the original MXQ algorithm. However, it is in appropriate as described earlier.

We used the output port utilization instead. The port utilization is sampled every 64 K byte time. If the current utilization exceeds the pre-determined target, we divide m by k, which is set to 1.03. On the other hand, it drops below that, we multiple m by k. Note that the output port utilization is only updated only at packet departures.

4.2 Platform of Implementation

Our implementation was built onto a high-end IP router which supports interfaces up to 10 Gb/s [1]. The architecture of this router was approximately matches our model. It should be noted that the same packet processors and memory banks are available both ingress and egress forwarding path. On a line card, two custom ASICs, each with sixteen 300 MHz processors, and 1.5GB DDR memory for packet processing function were embedded. They offered 25M packets per second processing for line rate performance under varying flow rates and the packet sizes ranging from 40 Byte. We made the most use of on-chip microcode programmability to customize packet processing functions to incorporate the parallel MXQ algorithm.

Since the MXQ was an AQM algorithm, we incorporated our code into the egress packet processor code. The code amounted to slightly over 300 lines. Consider that standard IP processing is mainly performed on the ingress packet processor, and is roughly 500 instructions. It is obvious that our implementation satisfied the processing time requirements.

4.3 Performance Evaluation

We evaluated the performance of the implementation in an large scale laboratory testbed. Using a number of real PCs, we created realistically randomized traffic, so that we could get a valuable insight to predict how the implementation would behave in a real environment.

4.3.1 Testbed Network

Two IP routers were connected with OC-192c (10Gb/s) link, and each router connected 14 PCs through dedicated GbE (1Gb/s) links. One from right hand side and one from left hand side were selected to constitute a pair, and traffic were sent and received between the pair all through the experiment. All of the PCs were not exactly the same: they were different in the number of CPUs, CPU's clock cycles, amount of memory and so forth. But the pair were selected so that they have the same performance. All of the PCs had one or two GbE Network Interfaces. The operating

system running on PCs were Linux.

4.3.2 Video Conferencing Traffic

One of the anticipated threats in the Internet is congestion of the traffic generated by video conferencing or video phone applications. The more the high-speed broadband services are used, the more the volume of such traffic will increase to cause congestion in access networks, peering points and trans-ocean links. Because such applications commonly use UDP for transporting video and audio data, the capability to regulating high-bandwidth flows will become necessary.

The video conferencing traffic consists of small constant-rate audio flows and high peak-rate, but very bursty video flows. The problem will occur when a burst of video flows causes temporal congestion at a router. It causes packet loss from any flows, including audio flows. It is well-known that packet loss for audio flows cause significant degradation in quality. Therefore, they must be protected from congestion. If a router can arbitrate flows based on their bandwidth, and regulate high-bandwidth flows, we will obtain the following benefits.

- Audio flow are protected from congestion. This means that users are able to continue their sessions, even though the quality of their video might be degraded temporary.
- Packet loss in video flows can be regarded as an indication to promote users to decrease the rate of video flows. They can reduce the size or frame rate.

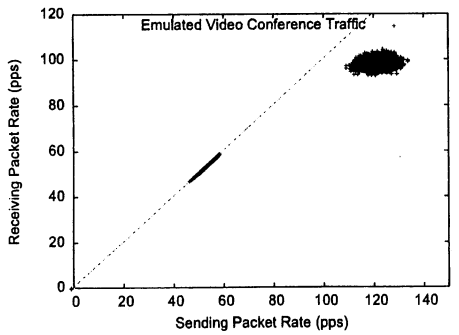

図 7 Emulated Video Conference Traffic

Figure 7 shows how our algorithm can arbitrate emulated video conference traffic. Each pair of PC generated two types of UDP flows.

(1) Audio: Average = 25kb/s, packet size = 64k byte,

(2) Video: Average = 1Mb/s, packet size = 1024k byte.

Using 14 set of PCs, we generated 17,870 flows, which is equivalent to 8,935 sessions. The total traffic is slightly higher than the target utilization that is set to 90Each point in this plot represents the sending packet rate (Horizontal Axis) and the receiving packet rate (Vertical Axis) of a flow. In total 17,870 points are plotted in this figure.

Obviously, the audio flows, the group of points around (55,55), are located on the no-loss line: They were completely protected from congestion. On the other hand, the video flows, the group of points in the right top region, are located below the no-loss line: They suffered from packet loss. In conclusion, it was confirmed that our implementation had the capability to support about 9,000 video conferencing sessions.

5. Conclusions

This paper studied flow-based router algorithms in the context of executing them on multiprocessor systems. We modelled the unique environment in modern multiprocessor-based system, and showed that flow-level parallelization could greatly reduce data access contention especially in enqueueing part of algorithms. In addition, we showed the careful design was needed for deriving multiprocessor friendly algorithms.

As an example, we described how the MXQ algorithm, an flow-based AQM algorithm, was efficiently realized on the multiprocessor system. The real implementation on a real high-speed multiprocessor-based router, and the results of the experiments in a laboratory testbed showed that the algorithm can scale well up to tens of thousands of flows at the line rate of 10 Gb/s. It should be noted that the number of flows created in our experiment was limited by the capability of PCs, and not by that of routers. The routers has the capability of over 500,000 flows.

文　献

[1] http://www.caspiannetworks.com/.

[2] http://www.cisco.com/en/us/products/ps5763/.

[3] D. E. Comer. *Network Systems Design using Network Processors Agere Version.* Pearson Prentice Hall, 2004.

[4] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts. On the scalability of fair queueing. In *Proc. of HotNets III*, November 2004.

[5] P. C. Lekkas. *Network Processors Architectures, Protocols, and Platforms.* McGraw-Hill, 2003.

[6] T. Shimizu, M. Nabeshima, I. Yamasaki, and T. Kurimoto. Mxq (maximal queuing): A network mechanism for controlling misbehaving flows in best effort networks. *IEICE Trans. on Information and Systems*, E84-D(5):596–603, May 2001.