Proceedings of the 20th IASTED International Conference
Parallel and Distributed Computing and Systems (PDCS 2008)
November 16 - 18, 2008 Orlando, Florida, USA

LOAD SCHEDULING FOR FLOW-BASED PACKET PROCESSING ON MULTI-CORE NETWORK PROCESSORS

Fei He[1,2], Yaxuan Qi[1,2], Yibo Xue[2,3] and Jun Li[2,3]
1 Department of Automation, Tsinghua University, Beijing, China
2 Research Institute of Information Technology, Tsinghua University, Beijing, China
3 Tsinghua National Lab for Information Science and Technology, Beijing, China
hefei06@mails.tsinghua.edu.cn

ABSTRACT
Load scheduling is critical to the performance of parallel processing network devices. With the rapid development of multi-core technology, efficient load scheduling scheme optimized for multi-core network processors becomes increasingly important and motivates intensive research today. In this paper, we study the relationship between two canonical scheduling schemes, packet-level scheduler and flow-level scheduler, and find out that scheduling at flow-slice level can further exploit parallelism while preserving per-flow packet-order. An adaptive load scheduling scheme at flow-slice level is proposed and evaluated. The experiment results show that this scheme can achieve better balance of workload than that of flow-level scheme, while keeping high cache utilization rate in typical system configurations.

KEY WORDS
Load Scheduling, Network Processor, Flow-based Packet Processing, Multi-core Processor

1. Introduction

With the rapid increase of network bandwidth, packet processing systems are facing more and more challenges. The biggest challenge is lack of computing resource. One solution to this challenge is parallel processing using multi-core network processor (NP) which employs multiple processing engines (PE) to handle network traffic in parallel in order to achieve both high performance and good scalability. Load scheduler is the key and fundamental module in parallel packet processing systems.

Another trend of packet processing system is that most applications are built within a common framework, which is called flow-based packet processing. Under different definitions of flow, applications including stateful inspection firewall, intrusion detection/prevention system, IPSec VPN, and flow-aware router can be regarded as flow-based applications. Basic operations of flow-based packet processing include flow classification, flow state lookup and updating. Since packets belonging to the same flow share the per-flow state maintained by the application, concurrently accessing and updating these shared flow-states by multiple processing engines can incur significant overhead due to mutual exclusion. Thus, load scheduler aware of the characteristics of flow-based packet processing is very essential to such multi-core packet processing systems. Such systems can achieve good performance only if the load scheduler distributes the traffic evenly among multiple processing engines.

There are two types of load scheduling scheme in the context of packet processing systems:

- **Packet-level Scheduler**, which dispatches each packet to processing engine independently.
- **Flow-level Scheduler**, which dispatches packets belonging to the same flow to a specific processing engine.

The advantage of packet-level scheduling scheme is that it can achieve a finer-grain parallelism. But it requires additional techniques to preserve packet order, which has great impact on system performance. When consecutive packets in a single flow are dispatched to different processing engines, packet-level scheduling scheme will incur significant overhead, which will greatly decrease system performance. One reason is that access to flow-state must be synchronized by mutual exclusion techniques such as locking. Another reason is that packet-level scheduling could lower the cache hit rate since each processing engine has its local cache [1].

Comparing to packet-level scheduling scheme, flow-level scheduling scheme can avoid the overhead incurred by synchronization and achieve high cache hit rate. However, it has coarse-grain parallelism in comparison to packet-level scheduling scheme, and cannot fully use all PEs' resource. In addition, it is hard for a flow-level scheduling scheme to achieve ideal work-load balance among multiple PEs by predicting the flow behavior, because flow characteristics, such as the size and the arriving pattern, are statistically various.

In this paper, we propose an adaptive scheduling scheme that distributes load at a sub-flow level (flow-slice). This scheme can achieve good balance of workload and high system throughput. Main contributions of this paper are:

- **Scheduling Granularity Comparison**: Backbone traces are used to compare the difference of scheduling at different granularity. Experiment results show that

scheduling at flow-slice level can exploit finer parallelism than flow-level.

- **Adaptive Scheduling Schemes**: We propose an adaptive load scheduling scheme which can exploit finer parallelism than flow-level scheduling scheme while still preserving per-flow packet-order. In shared cache multi-core systems, cache hit rate using our flow-slice level scheduling scheme is almost the same as that using flow-level scheduling scheme. Even in distributed cache systems, the decrease of cache hit rate is small and can be compensated with a little increase of cache size.
- **Cycle-driven Performance Simulation**: We evaluate different load scheduling schemes with a cycle-driven simulator using real-life backbone traces. The experiment results show that our scheme can reduce the packet loss rate to almost zero using relatively small buffer size, while keeping high efficiency of cache-utilization.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 explains the load scheduling problem and describes the system model. Section 4 describes the proposed adaptive scheduling scheme at flow-slice level. Section 5 provides the simulation results and performance evaluation. Finally, we come to our conclusions and discussion on the future work in section 6.

2. Related Works

Hash-based load scheduling scheme is a canonical flow-level scheduling scheme which incurs low overhead. Z. Cao et al. [2] evaluate the performance of different hash functions used in Internet traffic splitting, and find out that hashing using a 16-bit CRC over flow identifier gives good load balancing performance. D.G. Thaler et al. [3] propose another hash mapping scheme, Highest Random Weight (HRW), in the context of multi-server Web server systems. The main advantage of HRW over other hash schemes is that it can achieve fault tolerance with minimum disruption, which means that a minimum number of requests are remapped during server failures.

Since these two hash-based schemes only provide load balancing over the hash key space, they are vulnerable to traffic locality in Internet traffic [4]. Under the assumption of Zipf-like flow popularity distribution, W. Shi et al. [5] prove that hash-based schemes using the flow space as input space are not able to achieve load balancing. Accordingly, L. Kencl et al. [4] proposed an adaptive scheduling scheme for parallel packet forwarding system based on HRW. The scheme is an adaptive extension to the HRW scheme in order to cope with biased traffic patterns. The adaptor evaluates processor utilization periodically and compares it to a pair of thresholds to determine whether the system is unbalanced. If necessary, the adaptation is invoked to adjust the weights of every processing engine used in HRW. Another adaptive scheduling scheme [5] classifies flows into two categories: the aggressive flows and the normal flows, and applies different scheduling policies to the two classes of flows. This scheme exploits flow-level traffic characteristic to detect aggressiveness using a small number of packets in a flow. The problem of this scheme is that it still schedules packets at flow-level. It shifts only aggressive flows when the system is not balance to a certain extent, in order to minimize the adaptation disruption to the cache of the processing engines. But when some aggressive flows require processing capacity that exceeds what one PE can provide, shifting flows from one PE to another will not solve the problem of imbalance,.

These adaptive schemes [4][5] may cause packet reordering when the adaptation is invoked. When flows are shifted from an overloaded processing engine to a low-loaded one as the result of the adaptation, the original packet order may not be preserved. The occurrence of packet reordering, which can severely affect end-to-end TCP performance, should be avoided in packet forwarding systems. S. Kandula et al. [6] focuses on the traffic splitting problem in multipath routing. It exploits a simple observation that if the time between two successive packets is larger than the maximum delay difference between the parallel paths, the second packets can be routed to any paths without causing packet reordering.

The type of application running on processing engines is considered to have great impact on choosing or designing proper load scheduling schemes. In this paper, we take application characteristics, i.e. flow-based packet processing, into consideration when designing load scheduling scheme. Based on the similar observation of S. Kandula et al., we propose an adaptive scheduling scheme at flow-slice level that outperforms existing flow-level scheduling schemes while avoiding packet reordering caused by packet-level schedulers.

3. Problem Statement

3.1. System Model

There are N PEs to process packets dispatched from the traffic scheduler in typical multi-core network processor system. A packet destined to PE_i is appended to the input queue of PE_i, where $1 \leq i \leq N$. All these input queues share a fixed size of memory, which means that the length of an input queue is between zero and the buffer size B, and the limitation of a queue's length depends on the length of other queues.

We use P_i to denote the processing capacity of every PE, μ_i to denote the utilization rate of PE_i. By λ_i we denote the packet arrival rate at PE_i, which is the actual workload dispatched to PE_i. The total processing capacity is $P = \sum_{i=1}^{N} P_i$. The aggregate arrival rate is $\lambda = \sum_{i=1}^{N} \lambda_i$. In this paper, we only consider the case that each PE is homogeneous, i.e. $P_i = P/N$, for $1 \le i \le N$. Also, we use h_i to denote the cache hit rate of every PE.

The application considered in this model is called flow-based packet processing. Most of packet processing applications can be regarded as flow-based packet processing under different definition of flow. Flow-based packet processing applications access two types of data structures: *packet data structures* and *flow state structures*. Backbone traffic studies [7] show that packet data structures (including packet header, payload, etc.) exhibit little temporal locality. On the other hand, flow state structures (e.g. a hash table used for flow classification, etc.) exhibit considerable temporal locality [8]. The temporal locality of flow state in flow-based processing application is the most important characteristic that must be taken into consideration when designing the scheduling scheme, since the efficient of cache utilization for flow state structures has a great impact on the throughput of the application.

Therefore, two performance metrics are mainly concerned in our model: system's utilization rate and cache hit rate. These two metrics determine the throughput of the packet processing system.

3.2. Scheduling Objectives

The goals of load scheduling scheme for network systems are similar [2]. Firstly, the latency introduced in splitting the traffic must be strictly limited. With respect to flow-based packet process systems, packet-level traffic scheduling schemes may incur significant overhead induced by synchronized access to flow-state information. Therefore, flow-level traffic scheduling schemes are more suitable for flow-based packet processing systems comparing to packet-level scheduling schemes.

Secondly, because balance of workload is crucial for the system to achieve its full processing potential, the proposed scheduling scheme should minimize the imbalance of traffic. The responsiveness to load imbalance determines the system's utilization rate.

In addition to performance guarantee, a load scheduler for packet processing among multi-core NPs should possess

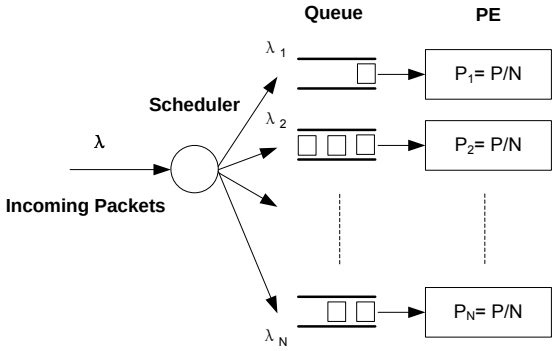

Figure 1. System model

the following properties:

- **Per-flow Packet Order Preservation**: The original packet order should be preserved, when packets belonging to one flow are dispatched to several PEs
- **High Cache Hit Rate**: The cache hit rate on a PE is mainly determined by the temporal locality in traffic dispatched to it. The cache hit rate can greatly affect the throughput of the system.

3.3. Source of Imbalance

The disadvantage of flow-level scheme is that under most circumstances it may not distribute traffic evenly. The most commonly used flow-level scheme is hash-based traffic scheduling scheme. The hash-based load scheduler maps the incoming flows onto each PE. Flow identifier v_i consists of a set of unvarying fields in packet header of a particular flow. The hash-based scheme uses a function H that maps v_i into the set of PE number. That is

$$H(v_i) \rightarrow \{1, 2, 3, ..., N\} \quad (1)$$

A typical example of a flow identifier would be the traditional 5-tuple, which is the combination of source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol type (PT). In this paper, we use the 5-tuple to define a flow, and we use flow bundle to refer to flows that have the same hash value.

There are two sources of load imbalance in flow-level schemes:

- **Coarse-granularity**: Flow-level scheduler restricts the available parallelism to the number of active flows. Thus, the size and randomness of input set to hash function is limited, which makes it difficult for a hash function to generate random outputs.
- **Heterogeneous Unit of Scheduling**: A hash-based flow-level scheduler dispatches flows as the unit of workload. Flow size affects the processing time. Because of the long-tailed distribution of flow size [10, 11], hash-based flow-level scheduler may not distribute workload evenly, even if hash function generates

Table 1. Experiment Dataset

Trace	Packet/second	Bandwidth (Mb/s)	Active Connection (conn/s)	New Connections (conn/s)
Abilene-III	225k	2150	617k	9960
CENIC-I	72k	498	115k	1076

random outputs.

4. Scheduling Scheme at Flow-Slice Level

The flow-level scheduling scheme increases temporal locality of flow state accessed by each PE, which consequently increases the cache hit rate. Its main disadvantage is that it may not distribute workload evenly. Scheduling schemes should be designed to achieve better balance of workload than the flow-level scheduling scheme while keeping high cache rate high. The scheme proposed in this paper can achieve this.

4.1 Scheduling at Flow-slice Level

Consider the scenario in Figure 2, where a series of packets arriving at the scheduler. Given two consecutive packets in a flow, P_1 and P_2, if P_2 reaches the scheduler after P_1 leaves the PE, the scheduler can distribute P_2 and the packets after P_2 to any PE without any possibility of reordering. In our model, we define *flow-slice* as consecutive packets in a flow the inter-arrival time between which is not larger than *maximum processing latency* (MPL) of PE.

Use T_i to denote the arrival time of packet *i* in a flow. Formula (2) is sufficient to determine the end of a flow slice.

$$T_{i+1} - T_i > \frac{B}{(1-\mu_i)P_i} = \frac{BN}{(1-\mu_i)P} \quad (2)$$

Recall that *B* is the buffer size in packets, μ_i is the utilization rate of PE_i, and P_i is processing capacity of PE_i.

4.2. Observation

Backbone traffic traces are used to observe the property of flow-slice in this section. The dataset used in this paper consists of backbone packet traces from NLANR PMA [9], collected at the OC192c Packet-over-SONET link from Internet2's Indianapolis Abilene router node (the Abilene-III trace) and the 10 Gb CENIC HPR backbone link (the CENIC-I trace) respectively. Table 1 summarizes our dataset.

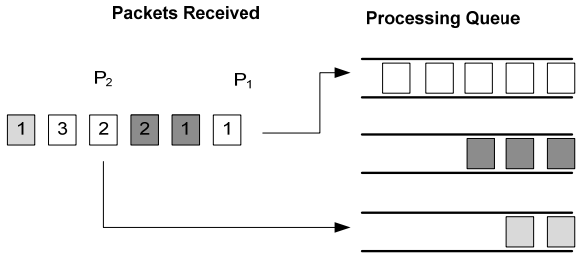

Figure 2. Scheduling at flow-slice level

S. Kandula et al. [6] point out that the main origin of flow-slice is the burstiness of TCP at RTT and sub-RTT scales, which is caused by ACK compression, slow-start, and other factors [10][11]. But in our application, the maximum processing latency is typically several microseconds, much smaller than a RTT. And we also find out that flow-slices exist not only in TCP traffic, but also in non-TCP traffic. Figure 3 shows that the proportion of flow-slice number to flow number in TCP traffic differs little from that in non-TCP traffic, when MPL varies from 1ms to 10ms.

In backbone network node, the number of active connection is the same order of magnitude as the number of packet per second (Table 1). We think another origin of flow-slice is the mixing of different flows.

The average flow length in IPLS-trace is 27.56, i.e. the proportion of number of packets to number of flows is 27.56. Figure 3 shows that the proportion of the number of flow-slices to the number of flows is larger than 10, which means there are more than 10 slices per flow in average, so scheduling at flow-slice level greatly finer the granularity of load splitting in comparison to scheduling at flow level.

Figure 4 and 5 shows the popularity of both flows and flow-slices in different size. It is showed that scheduling at flow-slice level also makes the size distribution of the splitting unit less skewed than scheduling at flow level.

4.3. An Adaptive Scheduling Scheme at Flow-slice Level

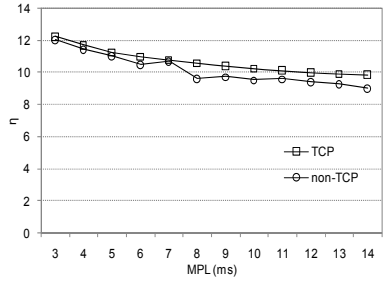

Figure 3. The proportion of number of flow-slices to number of flows (η) versus MPL

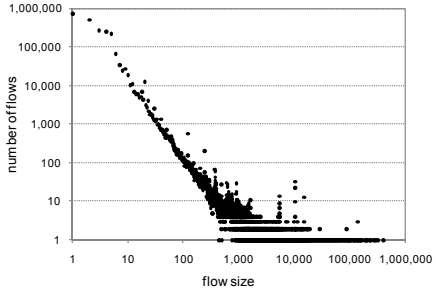

Figure 4. Flow size distribution, flow size versus number of flows

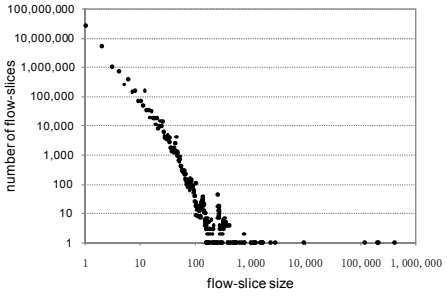

Figure 5. Flow-slice size distribution, flow-slice size versus num of flow-slices

```
pkt = Receive_packet();
flowID = Hash(pkt);
if (pkt.timestamp < flow_table[flowID].last_arrival + MPL)
{
      Dispatch(pkt, flow_table[flowID]. PE_ID);
}
else { // new flow-slice
      old_PE_ID = flow_table[flowID].PE_ID;
      if (queue[old_PE_ID] < THRESHOLD)
      {
            Dispatch(pkt, old_PE_ID);
      }
      else {
            new_PE_ID = Adaptation(pkt, old_PE_ID);
            flow_table[flowID].PE_ID = new_PE_ID;
            Dispatch(pkt, new_PE_ID);
      }
}
flow_table[flowID].last_arrival = pkt.timestamp;
```

Figure 6. Pseudo codes of the proposed scheduling scheme

We propose an adaptive scheduling scheme that distributes traffic at flow-slice level. The scheduling scheme proposed can be described using pseudo code in Figure 6. The scheduler uses a hash table to map flow-slices to PEs. Each table entry contains last arrival time and last PE ID.

In our model, there are N parallel PEs, and the aggregate packet arrival rate is λ. On the assumption of ideally balanced workload, the arrival rate at each PE is λ/N. The maximum processing latency (MPL) is $(B \cdot N)/\lambda$. For IPLS-trace, λ = 225kpps. A typical network process system, N is around 8 to 32 (we choose N = 16), and B ranges from 50 to 200 packets. Thus, MPL ranges from 3.55ms to 14.2ms.

5. Performance Evaluation

5.1. Simulation Parameters

We conduct cycle-driven simulations of adaptive load scheduling scheme based on a generalized network processing system, which has 16 parallel processing engines (PEs) with cache implemented. Flow table lookup and update (implemented using a hash table) is executed as the main process of a flow-based packet processing application in the simulator. The simulator is executed with two types of inputs: packet traces (described in Section 3.3), and a flow state table constructed using the packet traces.

Two types of cache model are implemented in the simulator: shared cache model and distributed cache model. All the PEs share one cache in shared cache model, while each PE has a separate cache in distributed cache model. LRU algorithm is used as the replacement algorithm in the simulator, and the line width of the cache is one word.

Given a trace and the number of PEs (N=16), the processing rate of each PE (μ_i) is estimated using $\mu_i = \lambda/N$. The average packet arrival rate (λ) is measured for each trace.

Parameters that have major impacts on system performance include: the processing buffer size B, the size of flow-slice table F, and the triggering threshold H. Performance metrics we are mainly concerned about are the packet loss rate, PE utilization rate, and cache hit rate.

5.2. Packet Loss Rate

In our simulations, packet loss happens only in the following situation. The workload is not properly balanced among the PEs, some PEs are idling while other PEs are overloaded. When the number of packets in a PE's queue increases to the limit of its buffer size, newly arriving packets are dropped.

Figure 7 shows packet loss rate of hash-based flow-level scheme and the flow-slice scheme proposed in this paper. The hash-based flow-level scheme has high packet loss rate, and increasing buffer size does help after buffer size is larger than 200 packets. When the buffer size is small, the flow-slice scheme has some packet losses. After the buffer size increases to over 200 packets, there is no packet loss any more.

5.3. Processing Engine Utilization Rate

Figure 8 shows the average PE utilization rate. The adaptive scheme scheduling at flow-slice level makes the system utilized close to its full potential. This means the workload is properly balanced among the PEs.

5.4. Cache Hit Rate

The cache hits in flow-based packet processing system are mainly resulted by the temporal reuse of flow state structures. Since the temporal reuse of these data structures occurs primarily when multiple packets belonging to the same flow are processed. Hence, in shared cache model, cache misses happen when two packets belonging to the same flow are separated by a large number of packets of other flows (the cache entry is replaced). In distributed cache model, cache misses can also be caused by shifting of flow-slice between PEs.

Figure 9 shows that there is not much difference between the cache hit rates of both schedulers when using shared cache model. Since the cache is shared by all PEs, the shifting of a flow-slice from one PE to another does not have much impact on the locality properties of the traffic.

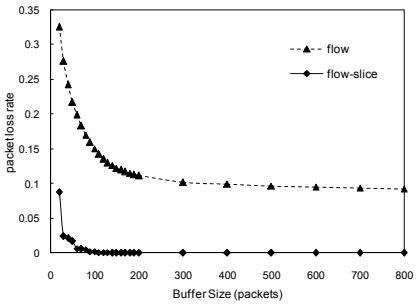

Figure 7. Packet loss rate versus buffer size.

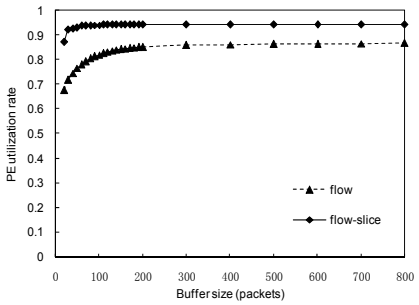

Figure 8. PE utilization rate versus buffer size.

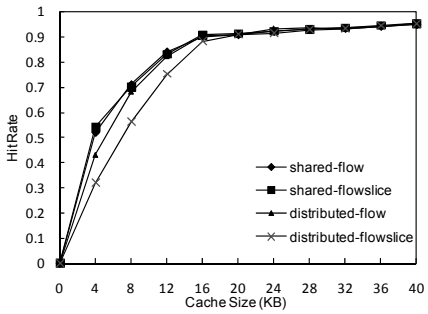

Figure 9. Cache hit rate of the schedulers

Using distributed cache model, the cache hit rate of flow-slice level scheduler is lower than flow-level scheduler when the cache size is small. However, when the cache size is relatively larger, the cache hit rates of both schedulers become almost the same. Despite the fact that flow-slice shifting affects the temporal locality of traffic reaching each PE, this result shows that a little increase in cache size can compensate for the reduction in cache hit rate caused by that.

6. Conclusion and Future Work

We analyze the difference between the different scheduling schemes at various granularities and show that scheduling at flow-slice level can exploit finer parallelism than flow-level and it can also make the size distribution of splitting unit less. We propose an adaptive load scheduling scheme that distributes traffic at flow-slice level based on our analysis. The proposed scheduling scheme exploits further parallelism than flow-level scheduling scheme while preserving per-flow packet-order.

In our experiment and performance evaluation, we use a cycle-based simulator with real-life backbone traces. The experiment results show that the proposed scheduling scheme can achieve much better balance of workload than flow-level scheme, and the packet loss rate is reduced to almost zero using a small buffer. In shared cache NP system, our scheme does not reduce the efficiency of cache utilization, and even in distributed cache NP system, a little increase in cache size can compensate for the reduction in cache hit rate caused by the shifting of flow-slices.

Our primary plans of future works involve implementing these scheduling schemes on different multi-core platforms, such as Cavium Octeon Processors [12] and Intel IXP Network Processors [13], to further evaluate the performance of the proposed scheduling scheme.

Acknowledgements

This work was granted by National High-Tech R&D (863) Plan of China (No. 2007AA01Z468). The authors also would like to acknowledge the colleagues in the Network Security Lab for their suggestions.

References

[1] T.L. Riché, J. Mudigonda, and H.M. Vin, Experimental Evaluation of Load Balancers in Packet Processing Systems, *Proc. 1st Workshop on Building Block Engine Architectures for Computers and Networks (BEACON-1)*, 2004

[2] Z. Cao, Z. Wang, and E. Zegura, Performance of hashing-based schemes for Internet load balancing, *Proc. IEEE INFOCOM*, 2000.

[3] D.G. Thaler and C. V. Ravishankar, Using name-based mappings to increase hit rates, *IEEE/ACM Trans. Networking, 6*(1), 1998.

[4] L. Kencl and J.L. Boudec, Adaptive load shareing for network processors, *Proc. IEEE INFOCOM*, 2002.

[5] W. Shi, M.H. MacGregor, and P. Gburzynski, Load balancing for parallel forwarding, *IEEE/ACM Trans. Networking, 13*(4), 2005.

[6] S. Kandula, D. Katabi, S. Sinha, and A. Berger, Dynamic load balancing without packet reordering, *ACM SGICOMM Computer Communication Review, 37*(2), 2007.

[7] K. Thompson, G. Miller, and R. Wilder. Wide-area traffic patterns and characterizations. *IEEE Network*, 1997.

[8] J. Mudigonda, H.M. Vin, R. Yavatkar. A Case for Data Caching in Network Processors. *http://www.cs.utexas.edul/~vin/pub/pdf/mudigonda04case.pdf*

[9] NLANR PMA: Special Traces Archive, *http://pma.nlanr.net/Special/*.

[10] H. Jiang and C. Dovrolis, The origin of TCP traffic burstiness in short time scales, Technical report, Georgia Tech., 2004.

[11] Z.L. Zhang, V. Ribeiro, S. Moon, and C. Diot, Small-time scaling behaviors of Internet backbone traffic, *Proc. IEEE INFOCOM*, 2003.

[12] Cavium Networks, *http://www.caviumnetworks.com/*

[13] Intel Network Processors, *http://www.intel.com/design/network/products/npfamily/index.htm*