International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 1 | ISSN : 2456-3307

Distributed Packet Buffers for High Bandwidth Switches and Routers

Maragoni Mahendar[1], Pinnapureddy Manasa[2]

[1]Assistant. Professor, Department of CSE, Avanti's Scientific of Technological and Research Academy. Hyderabad, Telangana, India

[2]M.Tech Student, Department of CSE, Avanti's Scientific of Technological and Research Academy, Hyderabad, Telangana, India

ABSTRACT

High-speed routers rely on well-designed packet buffers that support multiple queues, provide large capacity and short response times. Some researchers suggested combined SRAM/DRAM hierarchical buffer architectures to meet these challenges. However, these architectures suffer from either large SRAM requirement or high time-complexity in the memory management. In this paper, we present scalable, efficient, and novel distributed packet buffer architecture. Two fundamental issues need to be addressed to make this architecture feasible: 1) how to minimize the overhead of an individual packet buffer; and 2) how to design scalable packet buffers using independent buffer subsystems. We address these issues by first designing an efficient compact buffer that reduces the SRAM size requirement by (k - 1)/k. Then, we introduce a feasible way of coordinating multiple subsystems with a load-balancing algorithm that maximizes the overall system performance. Both theoretical analysis and experimental results demonstrate that our load-balancing algorithm and the distributed packet buffer architecture can easily scale to meet the buffering needs of high bandwidth links and satisfy the requirements of scale and support for multiple queues.

Keywords : Distributed Packet Buffers, SRAM, DRAM, RTT, Cisco CRS, TCP, HSD

I. INTRODUCTION

The phenomenal growth of the Internet has been fueled by the rapid increase in the communication link bandwidth. Internet routers play a crucial role in sustaining this growth by being able to switch packets extremely fast to keep up with the growing bandwidth (line rate).This demands sophisticated packet switching and buffering techniques. Packet buffers need to be designed to support large capacity, multiple queues, and provide short response times.

The router buffer sizing is still an open issue. The traditional rule of thumb for Internet routers states that the routers should be capable of buffering RTT_R data, where RTT is a round-trip time for flows passing through the router, and R is the line rate. In the author claimed that the size of buffers in backbone routers can be made very small at the expense of a small loss in throughput. Focusing on the performance of individual TCP flows, the author claimed in that the output/input capacity ratio at a network link largely determines the required buffer size. If the output/input capacity ratio is lower than one, the loss rate follows a power-law reduction with the buffer size and significant buffering is needed. Given everlasting controversy, nowadays, routers manufacturers still seem to favor the use of large buffers. For instance, the Cisco CRS-1 modular service card with a 40 Gbps line rate incorporates a 2 GB packet buffer memory per line card .In order to support fine-grained IP quality of service (QoS)

requirements, nowadays, a packet buffer usually maintains thousands of queues. For example, the Juniper E-series routers maintain as many as 64,000 queues. Given the increasing popularity of Open Flow, a packet buffer that supports millions of queues is always desired.

Furthermore, a packet buffer should be capable of sustaining continuous data streams for both ingress and regress. With the ever-increasing line rate, current available memory technologies, namely SRAM or DRAM alone cannot simultaneously satisfy these three requirements. This prompted researchers to suggest hybrid SRAM/DRAM (HSD) architecture with a single DRAM, interleaved DRAMs or parallel DRAMs sandwiched between SRAMs .In this paper, we briefly review previous work on packet Buffer architectures and present scalable and efficient hierarchical packet buffer architecture. This is our first attempt to combine the merits of two previously published packet buffer architectures consequently; the SRAM occupancy has been significantly reduced. By fully exploring the advantage of parallel DRAMs, we first propose a memory management algorithm (MMA) called Random Round Robin (RRR). Thereafter, we devise a "traffic-aware "approach which aims to provide different services for different types of data streams. This approach further reduces the system overhead. Both mathematical analysis and simulation demonstrate that the proposed architecture together with its algorithm reduce the overall SRAM requirement significantly while providing guaranteed performance in terms of low time complexity, upper bounded drop rate, and uniform allocation of resources. In one simulation, the proposed architecture reduces the size of SRAM by more than 95 percent and the maximal delay is only us-level, when the traffic intensity is 76 percent.

Existing System

The router buffer sizing is still an open issue. The traditional rule of thumb for Internet routers states that the routers should be capable of buffering RTT*R data, where RTT is a round-trip time for flows passing through the router, and R is the line rate. Many researchers claimed that the size of buffers in backbone routers can be made very small at the expense of a small loss in throughput.

Focusing on the performance of individual TCP flows, researchers claimed that the output/input capacity ratio at a network link largely determines the required buffer size. If the output/input capacity ratio is lower than one, the loss rate follows a power-law reduction with the buffer size and significant buffering is needed.

Proposed System

We devise a "traffic-aware" approach which aims to provide different services for different types of data streams. This approach further reduces the system overhead. Both mathematical analysis and simulation demonstrate that the proposed architecture together with its algorithm reduce the overall SRAM requirement significantly while providing guaranteed performance in terms of low time complexity, upper bounded drop rate, and uniform allocation of resources.

MODULES:

1. **Source:** It loads data and sends data to its router (source router).
2. **Source Router:** Source router uses leaky bucket mechanism to maintain the buffer in available bandwidth.
3. **Main Router:** Main router sends the forward packets from source to destination and backward packets from destination to source. It receives empty packets from destination to calculate the bandwidth of destination and ack packets to send the next packet to destination.
4. **Destination Router:** It sends empty, ack packets to centralized router.
5. **Destination:** Destination receives the data from destination router.

II. ARCHITECTURE

First introduced the basic hybrid SRAM/DRAM architecture [8] with one DRAM sand- witched between two smaller SRAM memories, where the two SRAMs hold heads and tails of all the queues and the DRAM maintains the middle part of the queues. Shuffling packets between the SRAM and t h e D R AM is under the control of a memory management algorithm. The principal idea behind their MMA is to temporarily hold amount of data for each queue in both the ingress and egress SRAM, so as to change the scattered DRAM accesses into a continuous one. Since the batch loads are strictly limited within each queue, the size requirement of SRAM for the HSD architecture is where Q is the number of FIFO queues. Whenever a FIFO queue accumulates b amount of data, it is transferred to the DRAM through a single write. A SRAM queue size of 2gb guarantees against queue overflow. Flow further suggested that the size of the tail SRAM can be further reduced by introducing a pipeline design. They also introduced a so-called the Earliest Critical Queue First (ECQF)-MMA for the egress, which reduces the size of head SRAM to Q By introducing an extra delay the ECQF- MMA now predicts the most critical queue (the one that goes empty or bears the biggest deficit first) and fetches the corresponding b- size chunk [10] of data from the DRAM in advance. This architecture, also known as Nemo, has been adopted by Cisco. Distributed Packet Buffer Architecture In our view, all packet buffering techniques so far have adopted a traffic-agnostic approach while designing the packet buffering algorithms. We m u s t clarify that even though existing approaches do use Q queues, each queue is treated the same by the buffer management algorithms. No effort is made to exploit the inherent characteristics of the corresponding traffic patterns like the arrival rate, burst sizes, transit time requirements through the router, etc. However, a traffic -aware approach to the problem, we believe, will yield new possibilities for conquering the scalability problem.

Per-Destination and Per-Packet Load Balancing

You can set load-balancing to work per-destination or per-packet. Per-destination load balancing means the router distributes the packets based on the destination address. Given two paths to the same network, all packets for destination1 on that network go over the first path; all packets for destination2 on that network go over the second path, and so on.

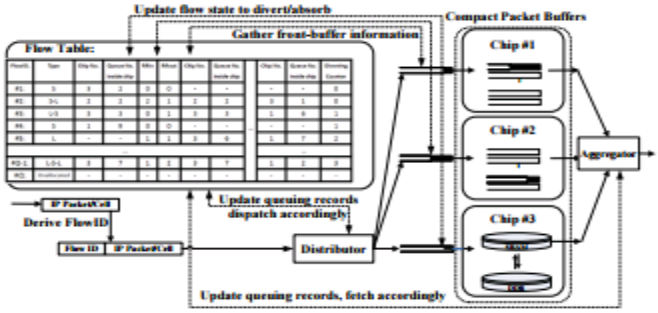

Figure 1. Distributed Packet Buffer Architecture

This preserves packet order, with potential unequal usage of the links. If one host receives the majority of the traffic all packets use one link, which leaves bandwidth on other links unused. A larger number of destination addresses leads to more equally used links. To achieve more equally used links use IOS software to build a route-cache entry for every destination address, instead of every destination network, as is the case when only a single path exists. Therefore traffic for different hosts on the same destination network can use different paths. The downside of this approach is that for core backbone routers carrying traffic for thousands of destination hosts, Per-packet load-balancing means that the router sends one packet for destination1 over the first path, the second packet for (the same) destination1 over the second path, and so on. Per-packet load balancing guarantees equal load across all links. However, there is potential that the packets may arrive out of order at the destination because differential delay may exist within the network. In Cisco IOS software, except the release 11.1CC, per packet load balancing does disable the forwarding acceleration by a route cache, because the route

cache information includes the outgoing interface. For per-packet load balancing, the forwarding process determines the outgoing interface for each packet by looking up the route table and picking the least used interface. This ensures equal utilization of the links, but is a processor intensive task and impacts the overall forwarding performance. This form of per-packet load balancing is not well suited for higher speed interfaces.

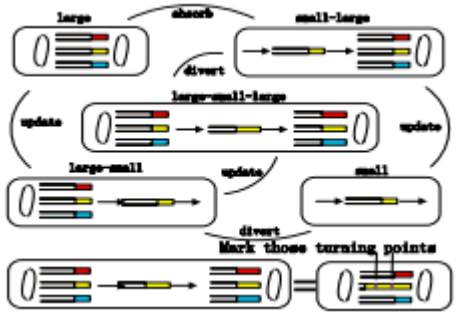

Figure 2. Flow Allocation among Multiple Subsystems

Figure 3 shows the state machine we have defined. Although there could be thousands of combinations, we only reserve six critical states. They are "unallocated", "large", "small" and three intermediate states "large-small", "small-large" and "large-small-large". Any flow can switch its state between "small" and "large" smoothly with certain constrains.

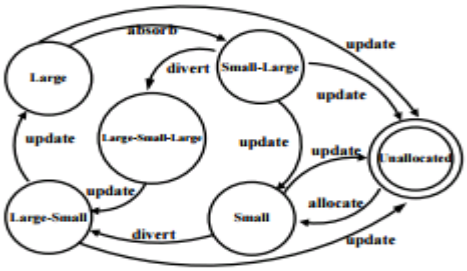

Figure 3. Flow States

Meanwhile, it is strictly controlled that any flow can only possess no more than three serving states at any time, i.e. at most 2 turning points. This helps the system minimize the overhead of state maintenance. Based on the state machine above, we devise a load-balancing algorithm. The pseudo code of our algorithm is shown in Fig. 4. The algorithm is naturally separated into three tasks that are implemented at the distributor, compact packet buffer subsystem and the aggregator respectively. The tasks communicate with each other through the centralized flow table.

Distributor:
For each cell, derive its flow ID, get access to the flow table accordingly.
If the state of this flow is "unallocated", **then**
1) Find a subsystem currently the lightest loaded by comparing the length of waiting lists in each front-buffer. In case all are equal, select randomly.
2) Allocate this new flow to an empty queue of the selected subsystem.
3) Record this information and update the flow state to "small".
4) Dispatch this cell accordingly.
Else if the state of this flow is "small" or "small-large", **then**
Dispatch the cell to the corresponding subsystem accordingly.
Else then
Dispatch the cell to the corresponding subsystem in per-flow round-robin*.

Each Compact Packet Buffer Subsystem:
Fetch one cell from the front-buffer, save it to the corresponding queue.
If the number of backlogged cells is no more than *THRESHOLD* **&**
current cell belongs to a flow which is originally severed as large **&**
it has been diverted for less than *MaxDivertTimes*, **then**
Update its flow state to small-large, bit-mark this cell as turning point.
Else if the number of backlogged cells is more than 2* *THRESHOLD* **&**
current cell belongs to a flow which is originally severed as small or small-large, **then**
Update its flow state to small-large, bit-mark this cell as turning point, increase the *MaxDivertTimes* by one.
Else then do nothing.

Aggregator:
Given a flow ID to fetch,
1) Delay this request for *DelayFactor* timeslots
2) Check the flow table.
3) Fetch one cell in per-flow round-robin* or from one subsystem only.
4) **If** it is a turning point, **then** update the flow state accordingly.

* per-flow round-robin: if a packet is the *i*-th cell in a flow, then it should be dispatched into DRAM *j*, where $j = i \bmod k$.

Figure 4. Load-balancing algorithm

III. SIMULATIONS

Our experimental results are presented in this section. Unless otherwise specified, the default for all the experiments is as specified in Table 1.

Table 1. Default Parameters

DelayFactor	100
THRESHOLD	10
The maximal depth of front-buffers	100
Number of subsystems	4

We define a timeslot as the minimal working span where each subsystem is capable of processing exactly one cell. Since there are four subsystems, for each timeslot, at most four cells are generated depending on the traffic intensity. Analyses on real-life traces indicated that the top 10% of flows account for over 90% of the packets and the bytes transmitted [23]. To be modest, in the following simulations, top 20% of flows accounts for 80% overall cells. Meanwhile, in order to observe the dynamic behavior of the entire system, the simulations are always separated into three phases.

Assume the simulation lasts for X timeslots. For the first 0.2*X timeslots, there is only input without output where cells are backlogged. In this way, we can create an initial backlog and simulate the situation when the congestion happens. After 0.2*X+1 timeslots, a full-speed output begins while input maintains. With backlogged cells in the first phase, we can monitor the system performance in detail, especially how the load-balancing algorithm behaves. After 0.5*X timeslots, the input stops while only the output maintains fetching any backlogged cells. In this way, we can simulate the situation when the system is lightly loaded. Moreover, we choose the parallel system (i.e. PHSD in [19]) as the basic reference standard of our distributed system. Because it is the best parallel architecture we known so far which represents the previous "flow-agnostic" approaches.

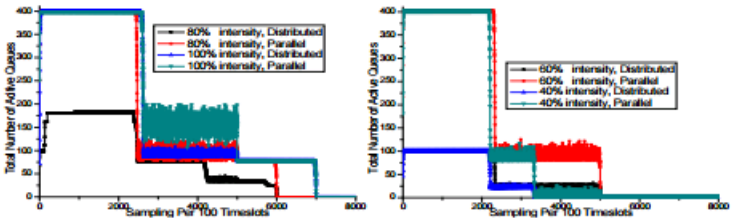

Figure 5. The overall active queues for both architectures

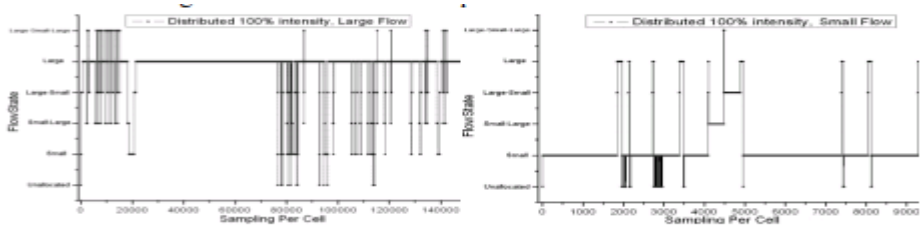

Figure 6. Flow-aware services based on probability method

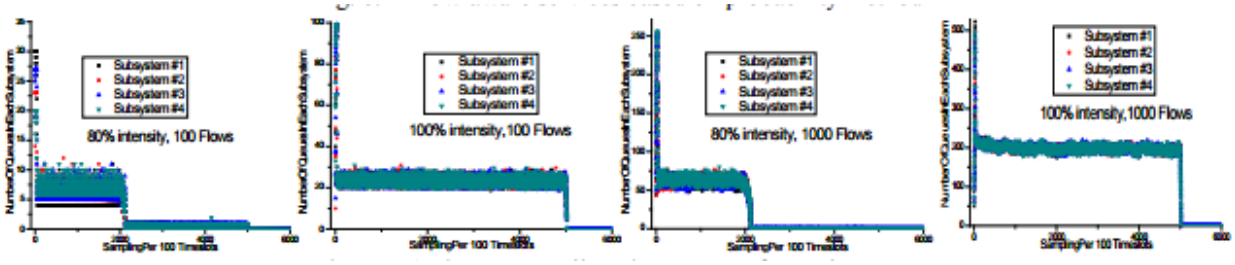

Figure 7. Active queues allocations among four subsystems

Since our algorithm does not refer the flow assignment status in balancing, we are curious about the actual distribution of active queues among subsystems. Fig. 7 shows the distribution of active queues among multiple subsystems. By increasing flow number from 100 to 1000, we observe that the unbalanced distribution is greatly improved which matches pervious mathematical analyses. We also monitor the length of front-buffers. As shown in Fig. 8, the average FIFO lengths for both algorithms are much less than 100 and the distributed system always outperform the parallel one. We clearly observe that the average FIFO length of distributed system stays at a constant around 16, which matches the mathematic analyses as well. Besides smaller average value of FIFO length, the distributed system also performs more smoothly.

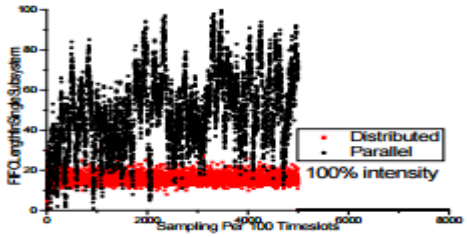

Figure 8. The number of backlogged cells in a front-buffer

IV. CONCLUSION

Unlike the previous approaches, our design dispenses with both the head and tail caches, keeping only tiny distributed front-buffers inside each subsystem. Each flow is only mapped to approximately one queue in a compact buffer. Thus, it maintains much less physical queues compared to other approaches and reduces the size of SRAM significantly. The significance of this research lies in the bold new direction towards distributed scalable buffer design that we plan to pursue. In particular, our approach yields a scalable, independent subsystem based packet buffer architecture that can easily be tailored to meet the specific line rate and traffic requirements for different switches, and match the flow-level requirements including bandwidth and delay guarantees within a switch. Our approach will also yield a power efficient buffer design, where parts of the buffer may be switched on and off based on the real-time traffic arrival rates and buffering requirements.

V. REFERENCES

[1]. S. Iyer, R. Kompella and N. McKeown, "Designing packet buffers for router line cards", in IEEE Transactions on Networking, vol.16, Jun. 2008, Issue 3.

[2]. Samsung SRAM Chips, http://www.samsung.com/global/business/semiconductor/products/sram/Products_HighSpeedSRAM.html

[3]. Samsung DRAM Chips, http://www.samsung.com/global/business/semiconductor/prod ucts/dram/Products_DRAM.html

[4]. J.Corbal, R.Espasa, and M.Valero, "Command vector memory systems: High performance at low cost," In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, pp.68-77, October 1998.

[5]. K.G. Coffman and A.M.Odlyzko, "Is there a "Moore's Law" for data traffic?" Handbook of Massive Data Sets, eds., Kluwer, 2002,pp.47-93.

Author Details

Maragoni Mahendar has Received B.TECH Degree in Computer Science Engineering (C.S.E) from Avanthi's Scientific Technological & Research Academy, Gunthapally, Rangareddy in 2012, under Jawaharlal Nehru Technological University Hyderabad and Masters Technology in Computer Science Engineering (C.S.E) from Nova College of Engineering & Technology, Jafferguda, Ranga Reddy 2014, under Jawaharlal Nehru Technological University Hyderabad. He is dedicated to teaching field since the last 4 years. His field of interest includes Cloud Computing, Data science & Big Data. He published Two International Journals and Participated in 4 International conferences. At present working as Asst.Professor, Department of Computer science and Engineering in Avanthi's Scientific Technological & Research Academy, Ranga Reddy, Telangana, India.

Email Id : m.mahender527@gmail.com

PINNAPUREDDY MANASA (14PT1D5809) M.Tech Student in Dept. of CSE from Avanti's Scientific of Technological and Research Academy. Telangana, INDIA.