Smartbuf: An Agile Memory Management for Shared-Memory Switches in Datacenters

Authors : Hamed Rezaei, Hamidreza Almasi, and Balajee Vamanan

2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)

Presented by Aishwarya Shrestha

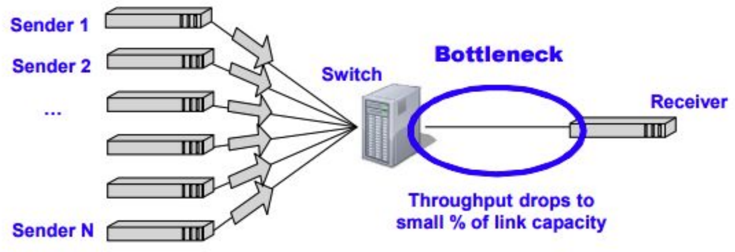
Agenda

Background

1. **Dependency** on internet-based services has been **increasing**. These services operate within data centers and play a crucial role in determining the overall experience.
2. Such **distributed architecture** provides strong protection against failures.
3. This setup has two important consequences:
   - ❏ The total response time depends on the time taken to complete the slowest data flows (99th percentile)
   - ❏ The network experiences **sudden spikes** in traffic referred to as "**incast**." (aka partition-aggregate traffic)

Background - Incast

1. Incast, also known as partition-aggregate traffic
2. Occurs in high-performance computer networks when a large number of clients simultaneously send a large amount of data to a single server.
3. This can cause the server's network interface to become congested, leading to reduced network performance and increased latency.
4. Cause bottlenecks and long completion times, especially for short data transfers that finish in less than one round-trip time (RTT).



Simple setup to observe incast

Background - Problems

1. Congestion control mechanisms (to avoid incast)
    a. efficient deadline-aware transport (EDT) and
    b. deadline-aware control (DAC)

2. The problem of achieving **quick response times** during incast traffic is well recognized.

3. The switches currently used in data centers, which have shared-memory switches with shallow buffers, make the problem even worse.

4. Reason : Incast traffic causes a build-up of data in the buffers and results in overflow. This leads to dropped packets and increased waiting time.

Background - Problems and approches

5. Recent congestion control algorithms address the incast problem
   But effectiveness of these approaches is limited by:
   a) round trip time (RTT)
   b) algorithms’ convergence times (ranges from a few to several tens of RTT)
6. This is not a major concern for longer flows. However, incast flows are usually brief (e.g. less than 1 KB) and last only a few RTT.

7. In this case, packet scheduling techniques would not help to solve incast problem

8. To avoid costly TCP timeouts, the sufficient buffer space can be allocated, it can help absorb incast events.

   the amount of buffer space required per port is proportional to the product of the bandwidth-delay.

Background

The amount of buffer space required per port is proportional to the product of the bandwidth-delay.

1. According to Moore's law, the network bandwidth scales faster than the memory. As a design choice, is **not feasible** to design data centers with large buffers to absorb incast events.

2. Fortunately, not all ports are likely to experience synchronized flows (i.e., incasts) at the same time and their buffer demands change over time

3. This diversity in the port presents **opportunity** to dynamically allocate memory to ports

4. Existing proposals for **dynamic memory allocation**
   a. Identify congested ports and
   b. allocate a portion or all of the memory equally among these ports.

   Does not accurately estimate the degree of congestion experienced by each individual port.

Design

1. The authors have provided strategy for managing **network buffer occupancy**.

2. The goal of Smartbuf is to prevent network congestion and maintain high network performance
3. Idea:
    a) To **learn the maximum** amount of buffer occupancy that each burst of incoming data can cause and then,
    b) **Limit** the buffer occupancy allocated for each network port that experiences similar bursts to that maximum level
    c) Can maintain consistent level of performance and prevents from dropped packets.

Design

4. Authors have acknowledged that, high the fan-in (incast degree) is a good predictor of the buffer demand

Jargons

1. Queue length : the number of items waiting in line to be processed
2. The gradient of queue length : rate of change in the length of a queue over time
3. Fan-in : the number of inputs also known as incast degree

5. High fan-in events –(causes) ➔ increase in **queue length** (short flow) ➔(indicates) buffer demand
i.e. fan-in (AKA incast degree) is a strong indicator of the buffer demand

This is because many of the flows during these bursts tend to be short.

6. This key observation i.e. one-on-one association between fan-in and buffer demand enabled the authors to estimate buffer demands of ports and allocate space accordingly.

Design

7. The gradient of queue length is directly proportional to the aggregate incoming rate, (i.e., **average sending rate * fan-in**)

8. Key insight: Measuring fan-in is difficult so instead of using fan-in, authors use the gradient of queue length (local information) to calculate the level of incoming data and potential congestion

9. To further reduce the sensitivity of the approach to event-based switch measurements, the authors smoothed (averaged) the gradient of queue lengths as the signature for the bursts.

This can lead to more accurate buffer management and improved overall performance of the system.

10. To accurately capture the presence of packets sent back-to-back within a burst and improve accuracy, they have applied Exponentially Weighted Moving Average (EWMA) over a limited time period.

Design

11. The switch begins observing the burst signatures and records the largest buffer occupancy observed at a port for every identifier (the smoothed gradient of queue length).

12. It continuously learns these key-value pairs and updates the values whenever a higher upper bound is encountered.

13. This information, represented as a set of key-value pairs, is used to identify future burst signatures and allocate buffers on a per-port basis.

14. Due to the dynamic workloads, the entries eventually age out after some time (i.e., soft state), and new signatures are learned.

15. Therefore, dynamically allocating buffer space (like so) to ports based on their demand is crucial for obtaining high performance in modern datacenter networks.

Implementation - Smartbuf algorithm for buffer allocation

1. Smartbuf is designed to capture the behavior of network bursts and use this information to accurately predict the buffer requirements of future bursts.

2. The goal is to have a better understanding of burst patterns in order to allocate buffer resources more effectively and improve network performance.

3. Two phases of algorithm: Learning phase and Real phase.

Implementation - Smartbuf algorithm for buffer allocation

4. Learning phase : the switch keeps track of the **gradient of the queue length** for each port, along with the **maximum queue occupancy** for each gradient, for every packet that is removed from the queue.

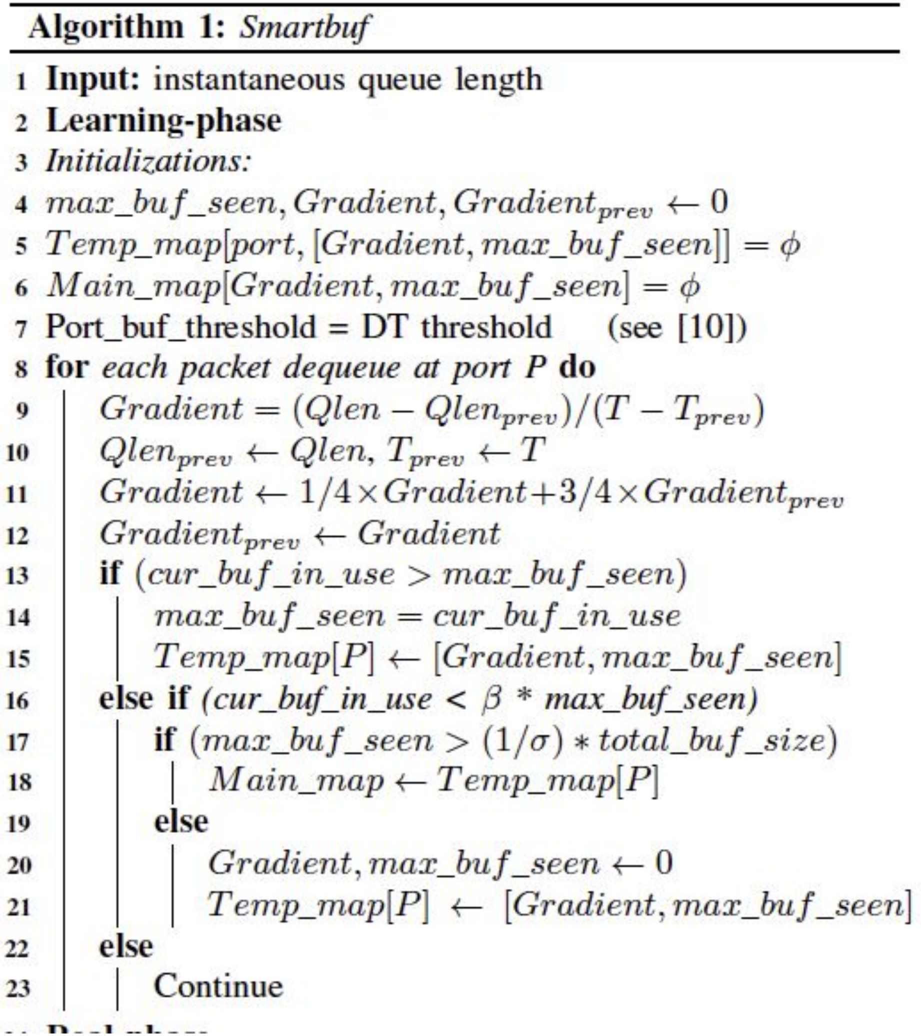
Temp_map (per port information) : port ID and a pair of gradient of queue length and maximum buffer occupancy [Line 5]

Main_map (global across all ports) : records the burst-related information only [Line 6]

Smartbuf kicks in only if a burst is detected.

Else port's buffer threshold is calculated [Line 7]

Main_map is global across all ports



Algorithm 1: *Smartbuf*

```
1  Input: instantaneous queue length
2  Learning-phase
3  Initializations:
4  $max\_buf\_seen, Gradient, Gradient_{prev} \leftarrow 0$
5  $Temp\_map[port, [Gradient, max\_buf\_seen]] = \phi$
6  $Main\_map[Gradient, max\_buf\_seen] = \phi$
7  Port_buf_threshold = DT threshold    (see [10])
8  for each packet dequeue at port P do
9      $Gradient = (Qlen - Qlen_{prev})/(T - T_{prev})$
10     $Qlen_{prev} \leftarrow Qlen$, $T_{prev} \leftarrow T$
11     $Gradient \leftarrow 1/4 \times Gradient + 3/4 \times Gradient_{prev}$
12     $Gradient_{prev} \leftarrow Gradient$
13     if ($cur\_buf\_in\_use > max\_buf\_seen$)
14         $max\_buf\_seen = cur\_buf\_in\_use$
15         $Temp\_map[P] \leftarrow [Gradient, max\_buf\_seen]$
16     else if ($cur\_buf\_in\_use < \beta * max\_buf\_seen$)
17         if ($max\_buf\_seen > (1/\sigma) * total\_buf\_size$)
18             $Main\_map \leftarrow Temp\_map[P]$
19         else
20             $Gradient, max\_buf\_seen \leftarrow 0$
21             $Temp\_map[P] \leftarrow [Gradient, max\_buf\_seen]$
22     else
23         Continue
```

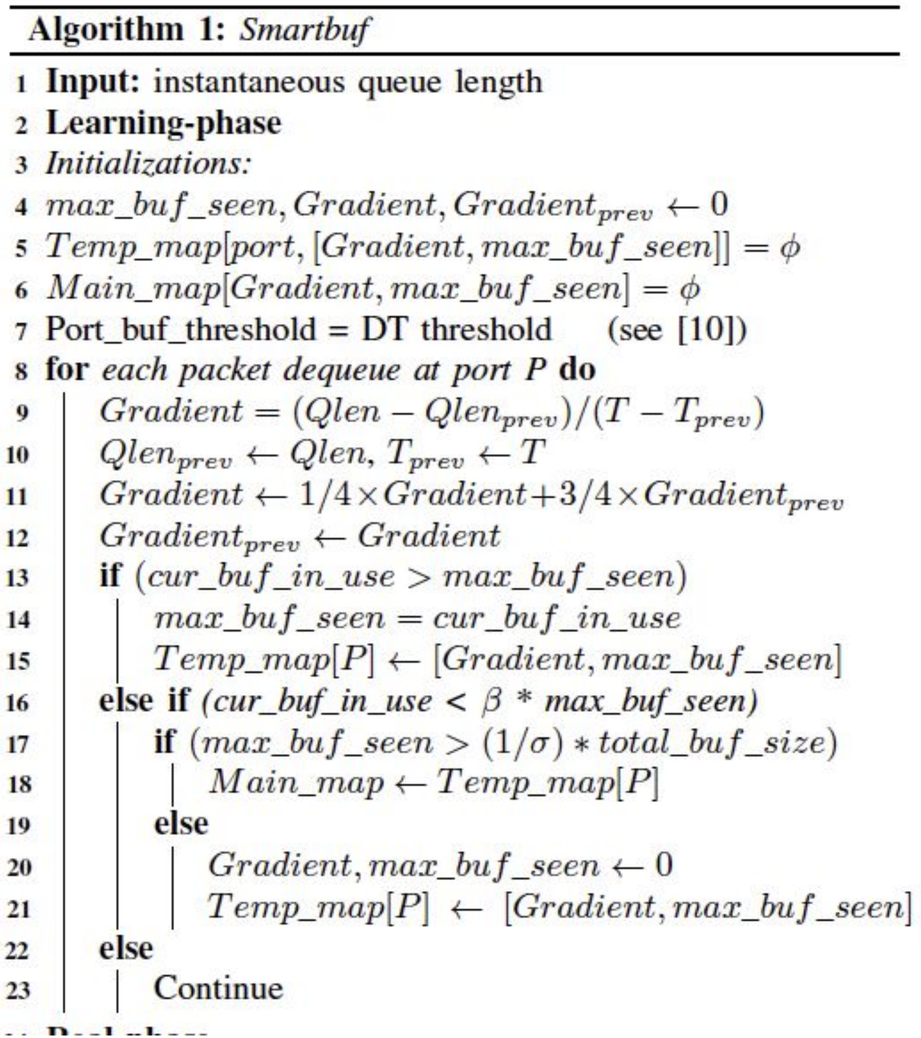
Implementation - Smartbuf algorithm for buffer allocation

5. After this initial setup it calculates the gradient of queue length

6. Smartbuf calculates the moving average of gradients while giving greater importance to previous samples, by assigning them a higher weight (e.g. 0.75). This step is crucial to the accuracy of the algorithm, as it significantly impacts the accuracy of the calculations performed by Smartbuf. [**Line 9 - 12**]

7. In **lines 13–15**, algorithm updates Temp_map entries if a larger buffer occupancy is observed. this step is important in recording the correct highest observed buffer occupancy

8. **Lines 16-23** focus on inserting Temp_map values into the Main_map (i.e., moving burst related information to the global hash map)



Algorithm 1: *Smartbuf*

```
1  Input: instantaneous queue length
2  Learning-phase
3  Initializations:
4  $max\_buf\_seen, Gradient, Gradient_{prev} \leftarrow 0$
5  $Temp\_map[port, [Gradient, max\_buf\_seen]] = \phi$
6  $Main\_map[Gradient, max\_buf\_seen] = \phi$
7  Port_buf_threshold = DT threshold     (see [10])
8  for each packet dequeue at port P do
9      $Gradient = (Qlen - Qlen_{prev})/(T - T_{prev})$
10     $Qlen_{prev} \leftarrow Qlen, T_{prev} \leftarrow T$
11     $Gradient \leftarrow 1/4 \times Gradient + 3/4 \times Gradient_{prev}$
12     $Gradient_{prev} \leftarrow Gradient$
13     if $(cur\_buf\_in\_use > max\_buf\_seen)$
14         $max\_buf\_seen = cur\_buf\_in\_use$
15         $Temp\_map[P] \leftarrow [Gradient, max\_buf\_seen]$
16     else if $(cur\_buf\_in\_use < \beta * max\_buf\_seen)$
17         if $(max\_buf\_seen > (1/\sigma) * total\_buf\_size)$
18             $Main\_map \leftarrow Temp\_map[P]$
19         else
20             $Gradient, max\_buf\_seen \leftarrow 0$
21             $Temp\_map[P] \leftarrow [Gradient, max\_buf\_seen]$
22     else
23         Continue
```

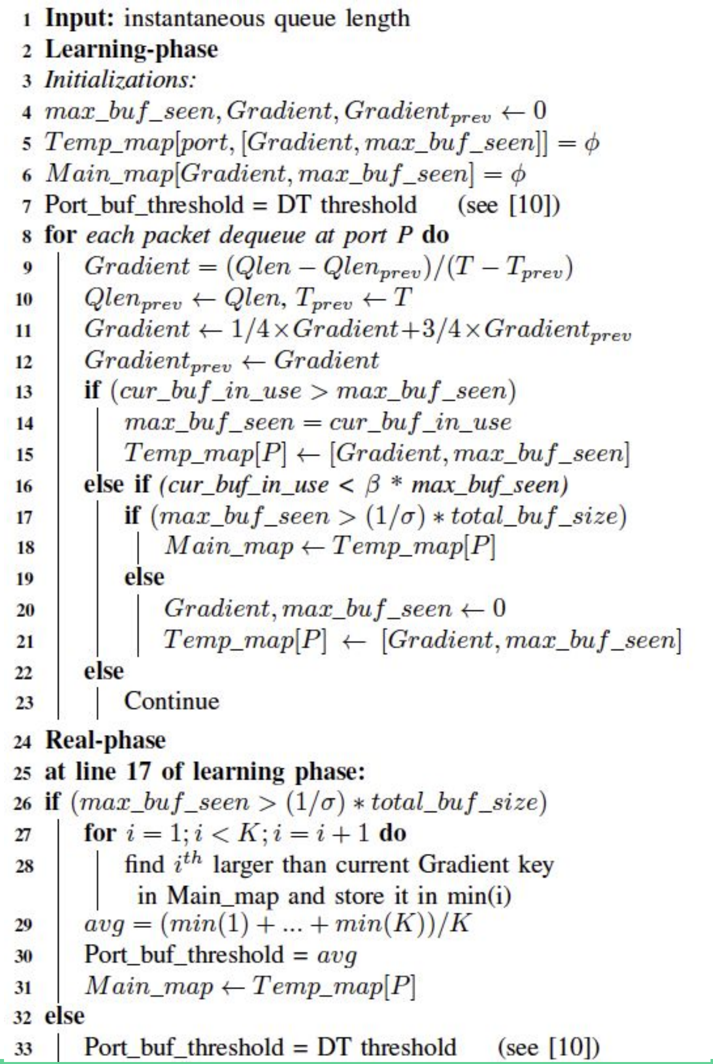
Implementation - Smartbuf algorithm for buffer allocation

9. **Real phase :** If line 17 of the algorithm holds, Smartbuf takes different actions that are shown in lines 27– 31 of algorithm 1.

10. When an incast situation is detected, the switch searches the Main_map for the K saved gradients that are larger than the current gradient and closest to it.

11. Smartbuf continues to insert new gradient and buffer occupancy pairs into the Main_map even during the real phase (line 31) to continuously improve the accuracy of the buffer assignment, even as the workload changes.

12. Smartbuf uses the K-NN approach to increase the accuracy of its buffer allocation algorithm



```
1  Input: instantaneous queue length
2  Learning-phase
3  Initializations:
4  $max\_buf\_seen, Gradient, Gradient_{prev} \leftarrow 0$
5  $Temp\_map[port, [Gradient, max\_buf\_seen]] = \phi$
6  $Main\_map[Gradient, max\_buf\_seen] = \phi$
7  Port_buf_threshold = DT threshold    (see [10])
8  for each packet dequeue at port P do
9      $Gradient = (Qlen - Qlen_{prev})/(T - T_{prev})$
10     $Qlen_{prev} \leftarrow Qlen$, $T_{prev} \leftarrow T$
11     $Gradient \leftarrow 1/4 \times Gradient + 3/4 \times Gradient_{prev}$
12     $Gradient_{prev} \leftarrow Gradient$
13     if $(cur\_buf\_in\_use > max\_buf\_seen)$
14         $max\_buf\_seen = cur\_buf\_in\_use$
15         $Temp\_map[P] \leftarrow [Gradient, max\_buf\_seen]$
16     else if (cur_buf_in_use < $\beta$ * max_buf_seen)
17         if $(max\_buf\_seen > (1/\sigma) * total\_buf\_size)$
18             $Main\_map \leftarrow Temp\_map[P]$
19         else
20             $Gradient, max\_buf\_seen \leftarrow 0$
21             $Temp\_map[P] \leftarrow [Gradient, max\_buf\_seen]$
22     else
23         Continue
24 Real-phase
25 at line 17 of learning phase:
26 if $(max\_buf\_seen > (1/\sigma) * total\_buf\_size)$
27     for $i = 1; i < K; i = i + 1$ do
28         find $i^{th}$ larger than current Gradient key
            in Main_map and store it in min(i)
29     $avg = (min(1) + ... + min(K))/K$
30     Port_buf_threshold = $avg$
31     $Main\_map \leftarrow Temp\_map[P]$
32 else
33     Port_buf_threshold = DT threshold    (see [10])
```

Testbeds

1. The experiment was conducted in leaf-spine data center topology in which 20 leaf switches are each connected to 20 servers. The leaf switches are connected upstream to 10 spine switches, thus creating an over-subscription factor of 2.

    a. link speed of leaf switches and spine switches is 10 Gbps across the network
    b. network round trip-time to 80 microseconds

2. They created models of shared memory switches with a shared buffer size of 4 MB.

3. Modeled a web search workload with short flows in the range of 1 KB to 32 KB and long flows varying from 1 MB to 10 MB in size

4. Use DCTCP as our congestion control method in which the retransmission timeout is set to 10 ms.

5. In the experiments, Smartbuf is compared to various policies including Static partitioning, DT [Dynamic queue length thresholds], Complete sharing, EDT, and a recent work called FAB

Results

1. Three evaluation aspects:
    a. Performance
    b. Overhead of algorithm
    c. Accuracy and parameter sensitivity of buffer-demand estimation

2. 99th percentile flow completion times (FCT) **metrics** is used to evaluate network performance, under different loads

Smartbuf outperforms all the other schemes across all loads.

EDT performs good in low loads and bad in high loads

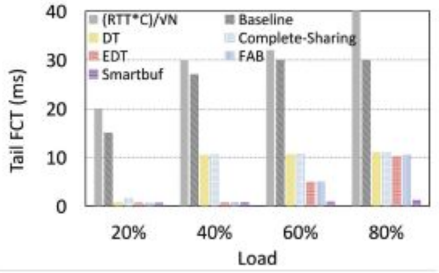


Fig. 3. 99^{th} percentile flow completion time

Results

2. In experiments with **varying buffer sizes**, the results show that as long as the buffers are not too smallor too large, Smartbuf ’s relative performance remains robust.

Smartbuf ’s proportional buffer allocation helps alleviate incast without adversely affecting the throughput and fairness for long flows.

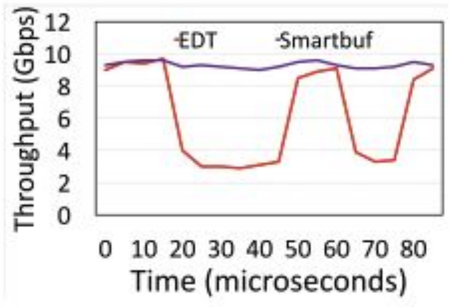


Fig. 4. Non-burst flows’ buffer share in EDT vs. *Smartbuf*

Results

3. The have also examined the **sensitivity of** their scheme to **different values of k** in the k-nearest neighbors algorithm.

In Figure 5, the impact of the variable "k" on the performance of Smartbuf is displayed for a high workload scenario.

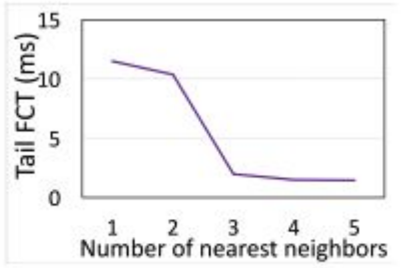


Fig. 5. Sensitivity study of k values

Results

4. They have presented the study of the **sensitivity of algorithm to different values of a σ** (threshold that classifies spikes in buffer usage as burst and non-burst.)

In Figure 6, better performance for larger values of σ . While larger σ provides high performance, it requires more processing as the size of the database is considerably larger.

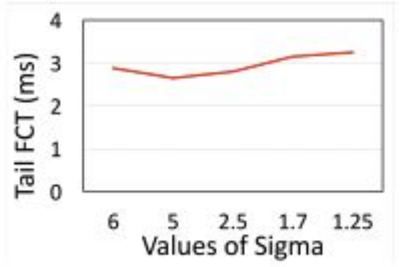


Fig. 6. Sensitivity to σ at 80% load

Results

5. There is an analysis of the state/**memory overhead** of the algorithm.

Figure 7 shows the maximum number of key-value pairs stored in their map under different loads (in a shared 4 MB buffer)

Observed that the number of entries in their database (i.e., map) scales proportionally to the load.

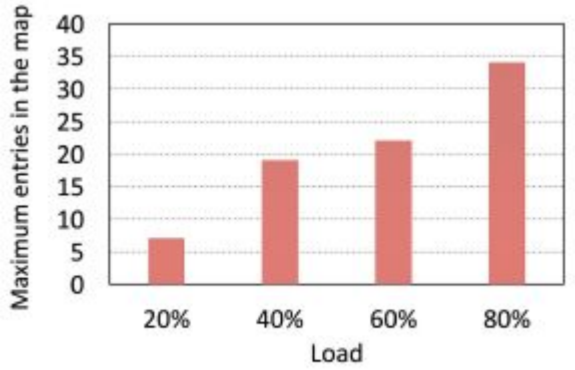


Fig. 7. Number of entries in the hash table

Results

6. In Figure 8, compares the **predicted demands** with the ideal demands for each workload level, with the difference between the two being the maximum error.

The most important finding is that the predicted demands are very close to the ideal buffer demands, with a difference of no more than 9%

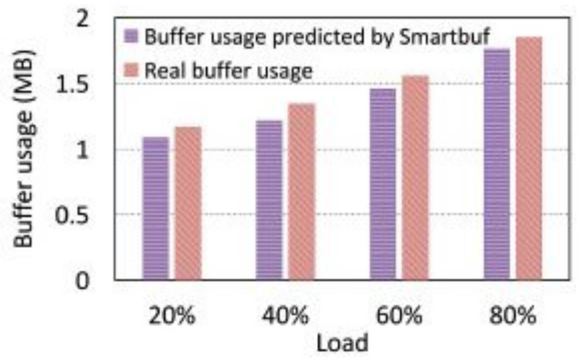


Fig. 8. Predicted vs. actual buffer demands: maximum prediction error at each load

Conclusion

1. Smartbuf is designed for shared-memory switches in data centers.

2. Smartbuf is able to estimate
   a. the memory requirements of each port based on information that is specific to the switch and
   b. then allocates memory to each port in proportion to its estimated needs.

3. Their results show that Smartbuf performs better than current state-of-the-art solutions when it comes to
   a. reducing tail latency,
   b. improving performance by as much as 8 times during high workloads while still ensuring fairness among ports.

4. Smartbuf's approach that relies solely on switch-local information may become increasingly attractive for managing resources.

Critique

1. Smartbuf is a clever adjustment of existing technologies that results in a remarkable increase in performance and minimization of network congestion

2. Considerable thought and effort is given into developing the Smartbuf algorithm.

3. The proposed solution aims to address a real and important problem in the field of networking,

4. The authors has provided a detailed analysis of their proposed method and its performance.

Critique

4. It helps to reduce network traffic to ensure that data transmissions can happen smoothly without any disruption.

5. Aims to address the issue of buffer overflow that can occur in shared-memory switches under heavy network traffic.

6. Situation;

When a port requires more memory, but there is no free memory left to be allocated, borrowing memory from another port allows the system to continue functioning.

In the scenario, the authors have decided to simply drop the packet instead of borrowing memory from another port. The reasoning behind this decision is not specified.

Critique

7. It has also provided the effect that varying values of k have on the sensitivity of our k-nearest neighbors algorithm, this makes the paper robust.

8. Smartbuf might not work Incompatible hardware or complex network configurations

9. Overall, the paper appears to be well-written and well-structured, and the authors make a strong case for their proposed solution

10. Whether or not the approach will prove to be effective and practical in real-world data centers remains to be seen, but the authors have laid a solid foundation for further research and development.

Thankyou!