

The Harmonic Policy for Online Buffer Sharing is $(2 + \ln n)$ -Competitive: A Simple Proof

VAMSI ADDANKI, Purdue University, USA
JULIEN DALLOT, TU Berlin, Germany
LEON KELLERHALS, TU Clausthal, Germany
MACIEJ PACUT, TU Berlin, Germany
STEFAN SCHMID, TU Berlin, Germany

The problem of online buffer sharing is expressed as follows. A switch with n output ports receives a stream of incoming packets. When an incoming packet is accepted by the switch, it is stored in a shared buffer of capacity B common to all packets and awaits its transmission through its corresponding output port determined by its destination. Each output port transmits one packet per time unit. The problem is to find an algorithm for the switch to accept or reject a packet upon its arrival in order to maximize the total number of transmitted packets.

Building on the work of Kesselman et al. (STOC 2001) on split buffer sharing, Kesselman and Mansour (TCS 2004) considered the problem of online buffer sharing which models most deployed internet switches. In their work, they presented the Harmonic policy and proved that it is $(2 + \ln n)$ -competitive, which is the best known competitive ratio for this problem. The Harmonic policy unfortunately saw less practical relevance as it performs n threshold checks per packets which is deemed costly in practice, especially on network switches processing multiple terabits of packets per second. While the Harmonic policy is elegant, the original proof is also rather complex and involves a lengthy matching routine along with multiple intermediary results.

This note presents a simplified Harmonic policy, both in terms of implementation and proof. First, we show that the Harmonic policy can be implemented with a constant number of threshold checks per packet, matching the widely deployed *Dynamic Threshold* policy. Second, we present a simple proof that shows the Harmonic policy is $(2 + \ln n)$ -competitive. In contrast to the original proof, the current proof is direct and relies on a 3-partitioning of the packets.

1 Introduction

In the online buffer sharing problem, we consider a network switch with n output ports labeled $1 \dots n$ and a shared buffer which can hold at most B packets. A stream of packets $\sigma = p_1, p_2, \dots$ arrives in the switch. Each packet p_i arrives at some time t_i and must be transmitted through a specific output port $q_i \in 1 \dots n$. We assume that the packets can be totally ordered by their arrival time — if two packets arrive at the same time, we assume that we break the tie arbitrarily.

When a packet p_i arrives, the switch decides to either accept or reject it. If p_i is accepted, it is stored in the buffer and queued for transmission through the output port q_i in a first-in-first-out (FIFO) manner. Each output port transmits up to one packet at each integer time $t = 1, 2, \dots$; the output port does not transmit anything if no packet is queued.

The task is to design an *acceptance policy* for the switch that decides whether to accept or reject each incoming packet. The objective is to maximize the number of accepted packets while respecting the shared buffer capacity at all time. The acceptance policy must be *online*, which means that the decision to accept or reject a packet must be made at the time of its arrival without knowledge of future packets.

Several buffer sharing policies have been established over the years. Among the most relevant are the Sharing with Maximum Queue Lengths (SMXQ) [3] and Dynamic Thresholds [7] algorithms, with the latter being the most used in practice. Both have a competitive ratio no better than $O(n)$,

Authors' Contact Information: Vamsi Addanki, Purdue University, West Lafayette, Indiana, USA; Julien Dallot, TU Berlin, Berlin, Berlin, Germany; Leon Kellerhals, TU Clausthal, Clausthal, Germany; Maciej Pacut, TU Berlin, Berlin, Berlin, Germany; Stefan Schmid, TU Berlin, Berlin, Berlin, Germany.

i.e., there exists a sequence of packets such that an optimal, offline acceptance policy accepts $O(n)$ more packets.

Following a series of results on switch buffer sharing [2, 4], Kesselman and Mansour [5] present the Harmonic policy for online buffer sharing in 2004. The main novelty of the Harmonic policy was to consider the queues sorted by their current size: it assigns roughly a $1/i$ fraction of the memory to the i -th largest queue, resulting in a competitive ratio of $(2 + \ln n)$. Their policy works as follows.

ALGORITHM 1 (THE HARMONIC POLICY). *Accept the incoming packet if, after accepting it, the following would hold for each $i \in 1 \dots n$: the total occupancy of the i queues with greatest occupancies is at most $\frac{B}{1+\ln n} \cdot \sum_{k=1 \dots i} \frac{1}{k}$.*

The Harmonic policy is the best known algorithm with respect to the competitive ratio metric – Kesselman and Mansour also showed a lower bound of $\Omega(\log n / \log \log n)$ for any online, deterministic policy, showing that Harmonic is close to optimal. Even though the alternative algorithms all have a worst-case competitive ratio of $O(n)$, the Harmonic policy however saw little practical relevance in modern network switches. This is arguably due to the initial presentation of the policy [5], which suggests that one must maintain a list of the output ports by queue occupancy and must perform n threshold checks upon each packet arrival, which is deemed too costly in practice. Moreover, the original proof of Harmonic’s competitive ratio is rather complex, involving a lengthy matching routine along with multiple intermediary results, which makes it harder for practitioners to adopt and understand the algorithm. Nowadays, buffer management algorithms are revisited through the lens of *learning-augmented* algorithms [1, 8, 9], where the acceptance policy uses machine-learned predictions to improve its performances. Most of the algorithms are based on one of the established algorithms and often use competitive analysis as a key metric to evaluate the prediction quality [1, 6]; The Harmonic algorithm therefore plays a central role as both a reference and a base to build new learning-augmented algorithms.

In this work, we first want to point out a quick observation that the Harmonic algorithm can be implemented to use only a constant number of operations per packet. Secondly, and mainly, we present a significantly simpler proof for its competitive ratio. We hope that this simpler proof makes Harmonic more accessible and possibly leads the way to more competitive learning-augmented algorithms for buffer sharing.

2 Efficiency of the Harmonic Policy

We briefly show that the Harmonic policy can be implemented very efficiently. Central to the Harmonic policy are the *thresholds*, which are proportional to the harmonic series.

Definition 2.1 (Thresholds). For all $k = 1 \dots n$, we define the threshold $T_k = \frac{B}{1+\ln n} \cdot \frac{1}{k}$.

ALGORITHM 2 (THE MODIFIED HARMONIC POLICY). *When packet p_i arrives, compute the smallest T_k such that the occupancy of queue q_i is strictly lower than T_k . Accept p_i if and only if accepting it results in at most k queues with occupancy no lower than T_k .*

For each queue, the smallest threshold T_k greater than its occupancy can be tracked in constant time per packet: each time a new packet is accepted or transmitted for a given queue, compare the new queue’s occupancy with the current threshold T_k and update it to T_{k+1} or T_{k-1} accordingly. Finally, it is possible to track the number of queues no lower than a given threshold in constant time per packet. For this, hold a list where values are associated with the thresholds, $T_1, T_2, \dots, T_n$ from left to right, and maintain the cumulative number of queues whose occupancy is above each threshold. When a packet p_i arrives, both the list update and the threshold check can be performed

in constant time (for the update: access the value of the previous threshold and increment the value on its left, for the threshold check: access the value and compare it with the corresponding threshold). Those structures can also be maintained efficiently when packets are transmitted from the queues: the number of operations is then in $O(n)$, a constant operation per queue, which is comparable with Dynamic Threshold [7] where the algorithm keeps track of all queue occupancies. All those changes require only two additional integer variables per queue and maximum two increments or decrements operations per packet arrival.

3 A Simpler Proof for the Competitive Ratio

We next give a simpler proof for the following theorem, originally proven by Kesselman and Mansour [5].

THEOREM 3.1. *The Modified Harmonic Policy is $(2 + \ln n)$ -competitive.*

PROOF. We refer to the Modified Harmonic policy as Harmonic, HAR is the set of packets accepted by Harmonic and OPT is the set of packets accepted by an offline, optimal policy.

We partition OPT in three subsets A , B and C as follows:

- (1) $A = \text{OPT} \cap \text{HAR}$
- (2) B contains the packets $p \notin \text{HAR}$ where p 's associated queue is strictly higher in HAR's cache than in OPT's cache at the time p arrives.
- (3) $C = \text{OPT} \setminus (A \cup B)$

We will first prove that $|\text{HAR}| \geq |A| + |B|$ by providing an injective mapping from $A \cup B$ to HAR. To obtain the desired mapping, we will construct injective sub-mappings restricted to the packets with output port q for each $q \in 1 \dots n$ and then merge all the sub-mappings. The sub-mappings are constructed iteratively as packets arrive. When packet $p_i \in A \cup B$ with output port $q \in 1 \dots n$ arrives, we extend the current sub-mapping as follows:

- If $p_i \in A$, then map p_i to itself in HAR.
- If $p_i \in B$, then find a packet $p \in \text{HAR}$ with the same output port q that already arrived and to which no packet was mapped yet, and map p_i to p .

We are left to prove that there always exists an unmapped packet like described in the second mapping rule. For a given packet p_i , we call $\text{Occ}(i, q, \text{HAR})$ the occupancy of queue q at time t_i in HAR (likewise for OPT) and define:

$$g_i(q) = \max \{ \text{Occ}(i, q, \text{HAR}) - \text{Occ}(i, q, \text{OPT}), 0 \}.$$

We also define $u_i(q)$ to be the number of packets of HAR in queue q without antecedent in the sub-mapping at time t_i . We prove by induction on the arriving packets that

$$\forall q, \forall i, \quad g_i(q) = u_i(q),$$

which would directly imply that the described mapping is possible as then $p_{i+1} \in B \implies g_i(q) > 0 \implies u_i(q) > 0$. The base case is clearly true at time 0 for all q . Assuming that this holds for some i , we prove that it also holds for $i + 1$ with a case distinction on the incoming packet p_{i+1} . If (1) $p_{i+1} \in B$ then both $g_{i+1}(q)$ and $u_{i+1}(q)$ are decremented by 1. Otherwise when (2) $p_{i+1} \in A$ then both $g_{i+1}(q)$ and $u_{i+1}(q)$ stay unchanged. Otherwise if (3) $p_{i+1} \in C$ then both $g_{i+1}(q)$ and $u_{i+1}(q)$ stay unchanged. Finally, if (4) $p_{i+1} \in \text{HAR} \setminus \text{OPT}$ then both $g_{i+1}(q)$ and $u_{i+1}(q)$ are incremented by 1.

We constructed an injective mapping from $A \cup B$ to HAR restricted to the packets with output port q ; we obtain an injective mapping from $A \cup B$ to HAR by merging the sub-mappings for each output port $q = 1 \dots n$, proving that $|\text{HAR}| \geq |A| + |B|$.

We then prove that $(1 + \ln n) \cdot |\text{HAR}| \geq |C|$. We prove it by exhibiting a *matching* between the packets of C and those of HAR such that:

- (1) each packet of C is matched to exactly one packet of HAR
- (2) each packet of HAR is matched to at most $1 + \ln n$ packets of C .

We construct the matching iteratively as packets of C arrive. We prove that, when OPT accepts a packet $p \in C$, there is always a packet in HAR's buffer available for matching that will be drained earlier, that is, matched with strictly less than $1 + \ln n$ packets and with a lower waiting queue than p . We do this by contradiction: let $p \in C$ be the first packet that is impossible to match without violating the wanted condition on the matching. We note T_k the threshold that triggered the rejection of p by Harmonic, that is, Harmonic's buffer would contain strictly more than k queues with occupancy no lower than T_k if p had been accepted. When OPT accepts p , its waiting queue is greater than T_k as $p \notin B$. Thus by assumption, all the packets in HAR's current buffer with position no greater than T_k are already matched with $1 + \ln n$ packets in C . Notice that the packets in HAR's buffer are matched to packets that are currently in OPT's buffer — this is true since, whenever some packet $p' \in C$ is matched, its position in OPT's queue is higher than that of its mate packet in HAR's queue and it will therefore be drained later. Harmonic's buffer contains at least $k \cdot \lceil T_k \rceil \geq \frac{B}{1 + \ln n}$ packets. Each of those packets are matched with $1 + \ln n$ packets of C in OPT's current buffer, which thus contains at least B packets without counting p : OPT cannot have accepted p , a contradiction.

Summing the two results obtained so far gives $(2 + \ln n) \cdot |\text{HAR}| \geq |A| + |B| + |C| = |\text{OPT}|$, proving the claim. $\square$

Acknowledgements. Research supported by the German Research Foundation (DFG), grant 470029389 (FlexNets), 2021–2025.

References

- [1] Vamsi Addanki, Maciej Pacut, and Stefan Schmid. 2024. Credence: Augmenting Datacenter Switch Buffer Sharing with ML Predictions. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 613–634. <https://www.usenix.org/conference/nsdi24/presentation/addanki-credence>
- [2] I. Cidon, L. Georgiadis, R. Guerin, and A. Khamisy. 1995. Optimal buffer sharing. *IEEE Journal on Selected Areas in Communications* 13, 7 (1995), 1229–1240. doi:10.1109/49.414642
- [3] M. Irland. 1978. Buffer Management in a Packet Switch. *IEEE Transactions on Communications* 26, 3 (1978), 328–337. doi:10.1109/TCOM.1978.1094076
- [4] Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. 2001. Buffer overflow management in QoS switches. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing* (Hersonissos, Greece) (STOC '01). Association for Computing Machinery, New York, NY, USA, 520–529. doi:10.1145/380752.380847
- [5] Alexander Kesselman and Yishay Mansour. 2004. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science* 324, 2 (2004), 161–182. doi:10.1016/j.tcs.2004.05.014 Online Algorithms: In Memoriam, Steve Seiden.
- [6] T. Lykouris and S. Vassilvitskii. 2021. Competitive Caching with Machine Learned Advice. *J. ACM* 68, 4 (2021), 24:1–24:25. <https://doi.org/10.1145/3447579>.
- [7] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. 2017. Analyzing and Enhancing Dynamic Threshold Policy of Data Center Switches. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2454–2470. doi:10.1109/TPDS.2017.2671429
- [8] Jiaxin Tang, Sen Liu, Yang Xu, Zehua Guo, Junjie Zhang, Peixuan Gao, Yang Chen, Xin Wang, and H. Jonathan Chao. 2022. ABS: Adaptive Buffer Sizing via Augmented Programmability with Machine Learning. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2038–2047. doi:10.1109/INFOCOM48880.2022.9796967
- [9] Mowei Wang, Sijiang Huang, Yong Cui, Wendong Wang, and Zhenhua Liu. 2022. Learning Buffer Management Policies for Shared Memory Switches. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 730–739. doi:10.1109/INFOCOM48880.2022.9796784