# *Starvation ratio*: letting applications drive datacenter congestion control

**Anderson Henrique da Silva Marcondes, Enzo Bello Boscatto, Guilherme Piêgas Koslovski**

[1]*Graduate Program in Applied Computing*
*State University of Santa Catarina (Udesc), Joinville, Brazil*
{*anderson.marcondes, enzo.boscatto*}*@edu.udesc.br, guilherme.koslovski@udesc.br*

***Abstract.*** *Datacenter performance is often limited by network-centric congestion controls relying on low-level metrics (e.g., packet loss, latency) that misinterpret applications needs. This work argues that applications should participate in congestion control decisions and introduces the starvation ratio (SR), a metric that detects when applications are truly limited by the network. Experimental evaluations within an 11-flow bottleneck scenario on a Linux-based prototype show that asynchronous applications can absorb network variations within a newly identified "silence zone" without degradation, proving conventional controls are overly restrictive. By deploying proactive and reactive mechanisms, our approach consistently reduces Flow Completion Time (FCT) for network-sensitive workloads. Notably, the proactive configuration eliminates micro-recovery delays, keeping the starvation ratio close to zero and reinforcing the baseline protocol through stable congestion window regulation. We conclude that shifting to application-driven signaling aligns network transmission with the receiver's processing pace, preventing computational underutilization.*

## 1. Introduction

The growing complexity of large-scale data processing applications has prompted a fundamental reinvention of modern datacenter (DC) architectures. The efficiency of DC is no longer defined solely by raw computational power, but by the seamlessness with which distributed applications can consume resources beyond local buses [Reed et al. 2022]. High-speed networks enables hardware disaggregation, redefining memory and storage as network-accessible shared resources rather than node-bound components.

However, this technological evolution also exposes the shortcomings of traditional protocols. In fact, DC network cannot differentiate between flows essential to application progress and those whose delivery can be delayed without impacting overall performance. Classical congestion control mechanisms, implemented through Transmission Control Protocol (TCP)-based algorithms, operate without awareness of the application's internal state. They respond exclusively to network-level signals, such as packet loss, round-trip time fluctuations, or explicit congestion notification markings, without considering whether these signals actually translate into degraded application performance.

Although recent proposals attempt to better capture the network dynamics [Bernárdez et al. 2025], and to decrease congestion by using software-defined networking [Wang et al. 2026, Diel et al. 2023], they still operate independently of the application's real processing needs. This strictly network-centric approach can be unnecessarily punitive [Bonato et al. 2024, Agarwal et al. 2023]. When congestion is detected,

protocols reduce the congestion window as a precautionary measure, and consequently, computational resources become underutilized, increasing the overall execution time.

To address these limitations, this work aims to mitigate computational resource underutilization and application-level performance degradation caused by overly conservative datacenter network transport policies. To achieve this, the central object of our proposal is the *Starvation Ratio* (SR), a novel, locally computable metric that advocates for an application-aware approach to congestion control in DCs. Inspired by the concept of resource starvation in operating systems, SR quantifies the condition in which a task remains ready to execute being prevented from making progress due to the unavailability of data. Unlike traditional network-centric metrics, SR provides an application's perspective. It enables to distinguish between harmless congestion, where network signals indicate load but application progress remains unaffected, and critical bottlenecks that genuinely impair execution. By shifting the decision-making from purely network-driven to application-observed impact, SR enables context-aware congestion management.

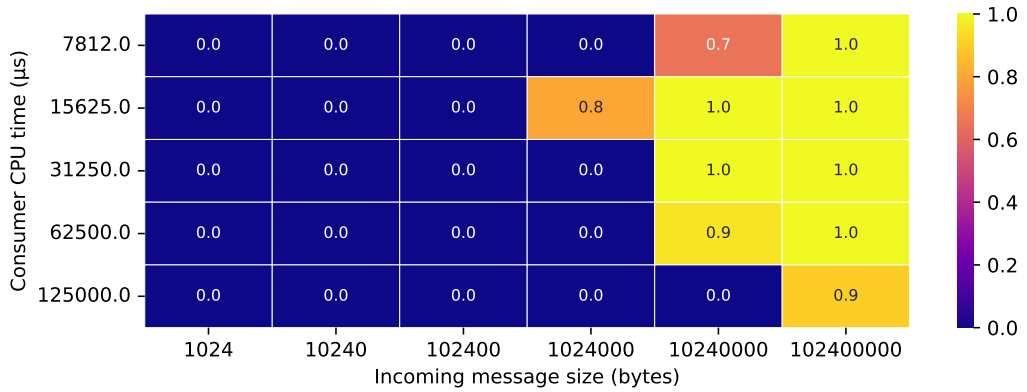



**Figure 1. SR of an application employing asynchronous processing and communication sharing a network bottleneck with ten TCP flows.**

The primary contribution of this work is based on identification of a silence zone in TCP congestion control (Figure 1). A broad spectrum of load and processing-time combinations in asynchronous applications can yield a starvation ratio of zero, which occurs either under network underload conditions or when severe network overload is effectively concealed by the application's processing dynamics. In the analyzed scenario, ten TCP flows share a communication link with an application employing asynchronous processing and communication, a design paradigm widely adopted in modern frameworks. Multiple configurations of incoming message size and CPU utilization were explored to represent diverse workloads. The silence zone reveals a limitation of TCP congestion inference: even under congestion conditions, network signals may be systematically misinterpreted without explicit application-level awareness. This observation challenges network-centric indicators when applied to asynchronous applications. In short, the contributions are threefold: *(i)* the SR metric to capture communication demands; *(ii)* dynamic transport mechanisms triggered directly by application starvation; and *(iii)* experimental evaluation demonstrating the effectiveness of application-driven DC congestion control guided by SR metric.

## 2. Starvation in distributed applications

### 2.1. Distributed and asynchronous applications

Datacenter efficiency is no longer measured solely by raw data transmission capacity, but by how seamlessly distributed applications consume these resources [Katebzadeh et al. 2023]. Application performance is frequently constrained by network conditions, rather than local hardware limits, over which the host operating system has no direct control. This interdependence creates a scenario where network bottlenecks propagate inactivity, directly causing starvation in communicating applications: even with an idle CPU, delayed data delivery prevents continuous stream processing, leading to periods of application-level inactivity.

While the asynchronous overlap of computation and communication is a key performance strategy [Reed et al. 2022], communication delays can easily disrupt this balance. The restriction of data delivery to the CPU may result in idle periods, followed by bursts of accumulated work, translating into costly resource underutilization in cloud environments. Furthermore, as communication networks exist within servers [Agarwal et al. 2023], a single delay can cascade inactivity across the entire execution chain, significantly increasing end-to-end latency.

### 2.2. Research hypothesis

Given that local components such as storage and main memory are now delivered over the network [Ewais and Chow 2023], the network acts as a critical data bus feeding the CPU, making network starvation as severe as traditional local memory bottlenecks.

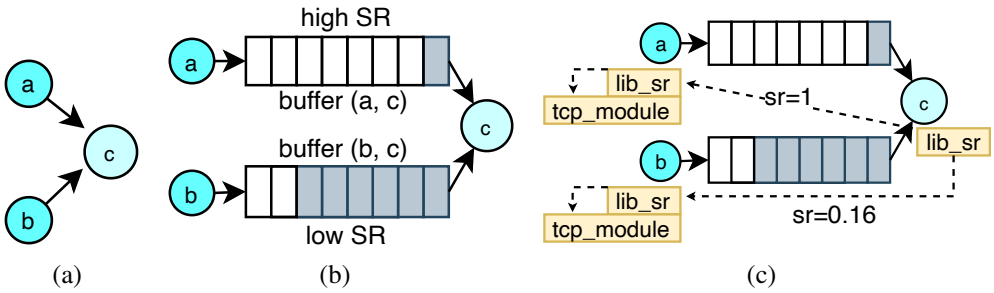



**Figure 2. Application's scenario. Fig. 2(a) denotes a DAG representing a distributed application ($a$ and $b$ are data producers while $c$ is a consumer). Fig. 2(b) represents producers and consumer communicating trough shared buffers (gray rectangles represent data). Fig. 2(c) positions `lib_sr` to support proactive or reactive congestion control.**

Figures 2(a) and 2(b) illustrate the components of a distributed application communicating through shared buffers. Such traditional producer-consumer model is commonly used for implementing large-scale communication frameworks. Whenever data is received or generated, it is placed into an intermediate buffer (termed $(a, c)$ and $(a, b)$), which is subsequently consumed for processing. Whenever the consumer attempts to process data and any buffer is empty, it indicates that the network is the limiting factor, and the starvation phenomenon occurs. The research hypothesis of this work seeks to determine the actual communication demand by monitoring and controlling the utilization of the distributed application's intermediate buffer.

### 2.3. Related work

The evolution of congestion control in DCs has been driven by the pursuit of metrics that minimize flow completion time [Dukkipati and McKeown 2006]. Initially, DCTCP [Alizadeh et al. 2010] leverages explicit congestion notification markings to react before losses occur, and it focuses strictly on maintaining shallow buffers in network switches, without considering whether the application can tolerate higher latency without performance degradation.

To address the dynamic nature of datacenter traffic, emerging approaches explore intelligent mechanisms; for instance, Tessler et al. [Tessler et al. 2022] introduce reinforcement learning frameworks to automatically learn and adapt congestion control policies, optimizing performance metrics directly from network feedback. In turn, Saba [Katebzadeh et al. 2023] uses application-level signals to prevent bandwidth underutilization, aligning with the starvation ratio philosophy. However, while Saba requires complex coordination between network and hosts, SR introduces a deterministic, locally computable metric based purely on CPU starvation. Furthermore, the identification of the silence zone through SR enables a more granular strategy: congestion control can be disabled not only when the network is uncongested, but whenever the application's asynchronous processing can absorb additional latency.

On the other hand, the Homa protocol [Montazeri et al. 2018] introduced a receiver-driven philosophy in which the receiver controls transmission rates through network priorities. Recently, Prasopoulos et al. [Prasopoulos et al. 2025] introduced a hybrid approach by delegating transmission control to the receiver through a credit-based mechanism. The protocol incorporates congestion signals from both the network and the senders, which report their effective transmission capacity across the network. Complementing receiver-driven paradigms, wide-topology solutions like IDCC [Geng et al. 2023] leverage fine-grained delay measurements and telemetry to regulate cross-datacenter traffic, optimizing flow completion times under variable propagation delays. The SR extends this logic by effectively turning the receiver into the arbiter of congestion.

## 3. The Starvation Ratio (SR)

### 3.1. Formal model

SR is defined as the proportion of time an application remains forcibly idle due to the lack of incoming network data relative to its total execution time. For a given data flow, let $T_{wait}$ denote the accumulated time the application is ready to execute but blocked waiting for data, and $T_{proc}$ the time spent processing. The metric is given by $SR = \frac{T_{wait}}{T_{proc}+T_{wait}}$, where the denominator approximates to the Flow Completion Time (FCT) [Dukkipati and McKeown 2006]. The SR ranges from 0 to 1, providing a normalized measure of communication-induced inefficiency: SR close to zero indicates that the network fully sustains processing demand with imperceptible latency, while SR $= 1$ denotes total starvation, where application progress is entirely halted by communication bottlenecks.

As the total execution time is often unknown in advance, the computation of SR is performed using an approximation based on the state of the application's receive buffer. The parameter $T_{wait}$ is measured at a fine-grained level whenever the consumer requests data and finds the buffer empty, recording the exact interval until the arrival of the next

packet. In this way, SR does not require prior knowledge, instead, it is calculated dynamically over temporal windows. This approach turns the metric into a real-time indicator, capable of capturing instantaneous variations in data delivery smoothness.

### 3.2. Control and prevention

We argue that congestion control should shift from being purely reactive to the network to being triggered by the application. In this sense, two phases are identified: *(i)* Transparency phase (SR $< \epsilon$): while below a given threshold $\epsilon$, the application resides in the silent zone. Network congestion is masked by asynchronous processing and communication, and the transmission rate remains at its maximum as allowed by the traditional TCP. *(ii)* Starvation phase: when SR reaches the critical threshold, flow rate should be adjusted proportionally to the local level of starvation. This approach aggressively uses network resources only when processing capacity requires it.

### 3.3. Linux-based prototype implementation

The practical feasibility of SR-based control is demonstrated through an implementation in a Linux environment using the developed `lib_sr` library and minor modifications to the Reno-based TCP module, as illustrated in Figure 2(c).

**`lib_sr`.** The application interacts with `lib_sr` to signal the producer that network prioritization is required. On the consumer side, the `monitor_send` function, transmits the corresponding criticality level to the producer. On the producer side, the `monitor_recv` function receives the SR and invokes the `setsockopt` primitive with the `SO_MARK` option. This operation marks packets at the socket level, allowing the Linux networking subsystem to identify flows operating in the silence zone or in a starvation state, and to adjust the congestion window according to the feedback provided by the application. While network prioritization can also be enforced at intermediate switches using framework architectures like Differentiated Services (DiffServ), our mechanism operates strictly under the end-to-end principle. By managing traffic dynamics directly at the end-hosts via socket markings, the proposed solution avoids the overhead of complex state coordination across the physical network fabric. The implementation on the consumer adopts a multi-threaded model to isolate packet reception from logical processing. A dedicated processing thread was designed to measure the duration it remains blocked while waiting for network data.

**Marking and congestion window adjustment.** Congestion control follows a consumer-driven model, where the consumer computes the SR and generates a control signal that reflects the application's health status. When the signal is received by the producer, the congestion control mechanism must be adjusted to prioritize impaired flows. Such flows may temporarily occupy the silence zone of other flows. Upon receiving the marking via socket configuration, the TCP module on producer adapts the congestion window (`cwnd`) according to the receiver's maximum capacity. In practice, `cwnd` is progressively increased, guided proportionally by the SR value toward the receiver window limit. Finally, once the SR returns to a level deemed acceptable by the application, the aggressive growth of the congestion window is suspended, and the native TCP congestion control algorithm (Reno in the prototype) is resumed.

# 4. Experimental Protocol

## 4.1. Environment and application

The experimental setup consists of two Dell Precision 3600 servers (Intel Core i7-13700K, 64 GB RAM, running Ubuntu 22.04.1 LTS with the Linux 5.15.0-143-generic kernel) connected via a 10 Gbps HPE switch. We deployed a producer-consumer application over this topology, transmitting data of varying sizes from the producer to the consumer, as illustrated in Figure 2(b). The tested data sizes were 1 KB, 10 KB, 100 KB, 1 MB, 10 MB, and 100 MB, reflecting typical datacenter traffic [Benson et al. 2010].

Communication starts with the producer sending data. On the consumer side, a dedicated thread receives incoming data and stores it locally, where a CPU-intensive thread consumes it. To differentiate application workloads, we evaluated five processing times: 7.812 $\mu$s, 15.626 $\mu$s, 31.250 $\mu$s, 62.500 $\mu$s, and 125.000 $\mu$s. The consumer requests new data only after processing the predefined time for the current batch. If no data is available, the application experiences starvation. This setup allows us to assess whether network traffic prioritization is necessary in scenarios where application-level processing constraints prevent immediate data consumption. Finally, the consumer-producer application is monitored by `sr_lib`, as depicted in Figure 2(c).

## 4.2. Experimental scenarios

We selected three metrics to evaluate the impact of the SR-driven congestion control: *(i)* FCT, the time required to transfer a flow of a given size; *(ii)* SR, as discussed in Sec. 3; and *(iii)* `cwnd` evolution, to track the internal progression of each flow. For each data size, we repeated the transfer 30 times to collect sufficient data for statistical analysis. Figures 3 and 5 present FCT results using cumulative distribution functions (CDFs), while Figures 4 and 6 show SR values. Figure 7 summarizes the `cwnd` evolution.

To explore application-oriented congestion control, we defined two scenarios: proactive and reactive configuration. Following the end-to-end principle in distributed systems, only the endpoints should directly assess application behavior [Saltzer et al. 1984]. This allows applications to implement sophisticated policies for notifying the producer about starvation events. In this study, we evaluate the applicability of the SR metric and leave the development of application-optimized policies as future work.

Proactive marking informs in advance, using `sr_lib`, that the application will require network resources to perform the data transfer. This scenario represents a selfish configuration, in which the application requests the maximum allowed capacity between the producer and the consumer (theoretically limited by the `rwnd`). Once the transfer is fully completed the marking is removed. In contrast, the reactive configuration waits for actual starvation to occur within the application. Once it is detected, the application uses `sr_lib` to send the marking. Similarly, the marking is removed once the transfers are completed. This configuration aims to mark only the periods in which starvation actually occurs in the application; however, the reaction time (sending the marking message and reconfiguring the TCP module) may impact the overall performance.

The standard Reno configuration serves as our baseline. Importantly, this approach only temporarily overrides congestion control to optimize specific applications, reverting to the default protocol set by the datacenter administrator whenever possible. In each run, 11 TCP flows share the congested channel: 10 flows generated with TCP

iperf representing background traffic, and 1 flow originating from the producer-consumer application. Only the producer-consumer application is monitored with `lib_sr`.

## 5. Results and discussion

To support the discussion, workloads are categorized into three operational regimes, defined by the CPU time available to the application: *(i)* High-sensitivity zone ($7.812\,\mu$s and $15.625\,\mu$s) represents scenarios in which CPU processing speed exceeds the network delivery rate, making the application highly sensitive to variations in the communication channel. *(ii)* Transition zone ($31.250\,\mu$s and $62.500\,\mu$s) encompasses intermediate times where the balance between computation and communication fluctuates, allowing the emergence of the so-called silent zone to be observed. *(iii)* CPU saturation zone ($125.000\,\mu$s), a CPU-bound regime with reduced network dependence. Crucially, increased network aggressiveness here introduces packet-processing and OS interrupt overheads that compete with execution threads, degrading overall performance.

### 5.1. Reactive control

The experimental results (Figures 3 and 4) demonstrate the effectiveness of reactive control for network-sensitive workloads. In scenarios within the high sensitivity zone, particularly for the smallest computation interval analyzed ($7.812\,\mu$s), monitoring SR led to a consistent reduction in FCT across most data flow sizes. The leftward shift of the CDF curves indicates that reactive control mitigates communication delays more efficiently than the baseline protocol (with SR off). This behavior supports the premise that, in applications characterized by high data demand and short CPU intervals, network sensitivity is amplified. As a result, starvation-based feedback emerges as an effective mechanism to prevent processor underutilization and forced idle periods.

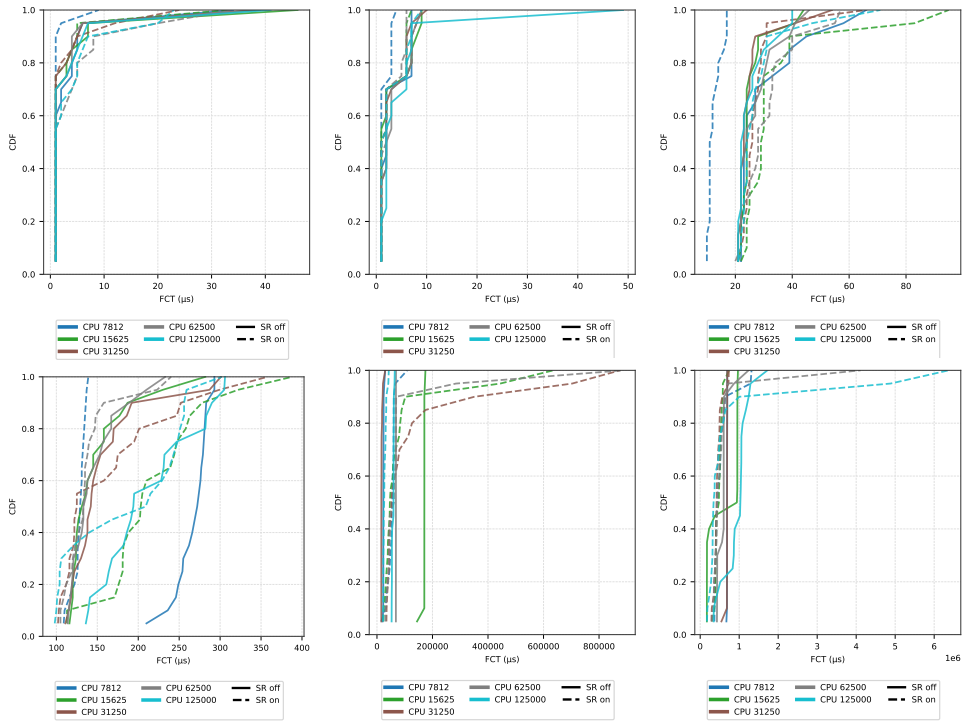
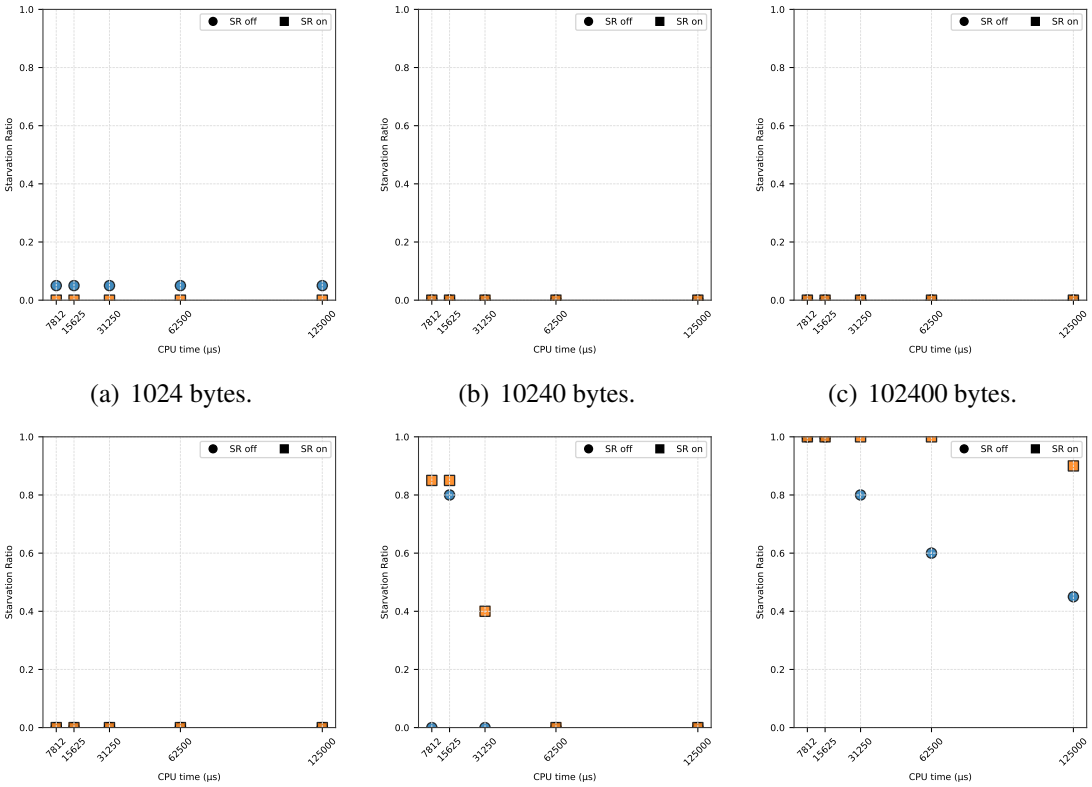



**Figure 3. FCT with reactive configuration of SR marks.**

As computational demand shifts into the transition zone ($31.250\,\mu$s and $62.500\,\mu$s), the benefits provided by SR signaling begin to change. The time the application spends on local processing becomes sufficiently long to absorb variations in network. As a result, the application operates predominantly within the silence zone, where starvation events are infrequent (Figure 4) and the reactive mechanism is invoked less often. At this stage, the difference in FCT between SR enabled and disabled starts to narrow, indicating that the network is no longer the dominant factor influencing application performance.




(a) 1024 bytes. (b) 10240 bytes. (c) 102400 bytes.

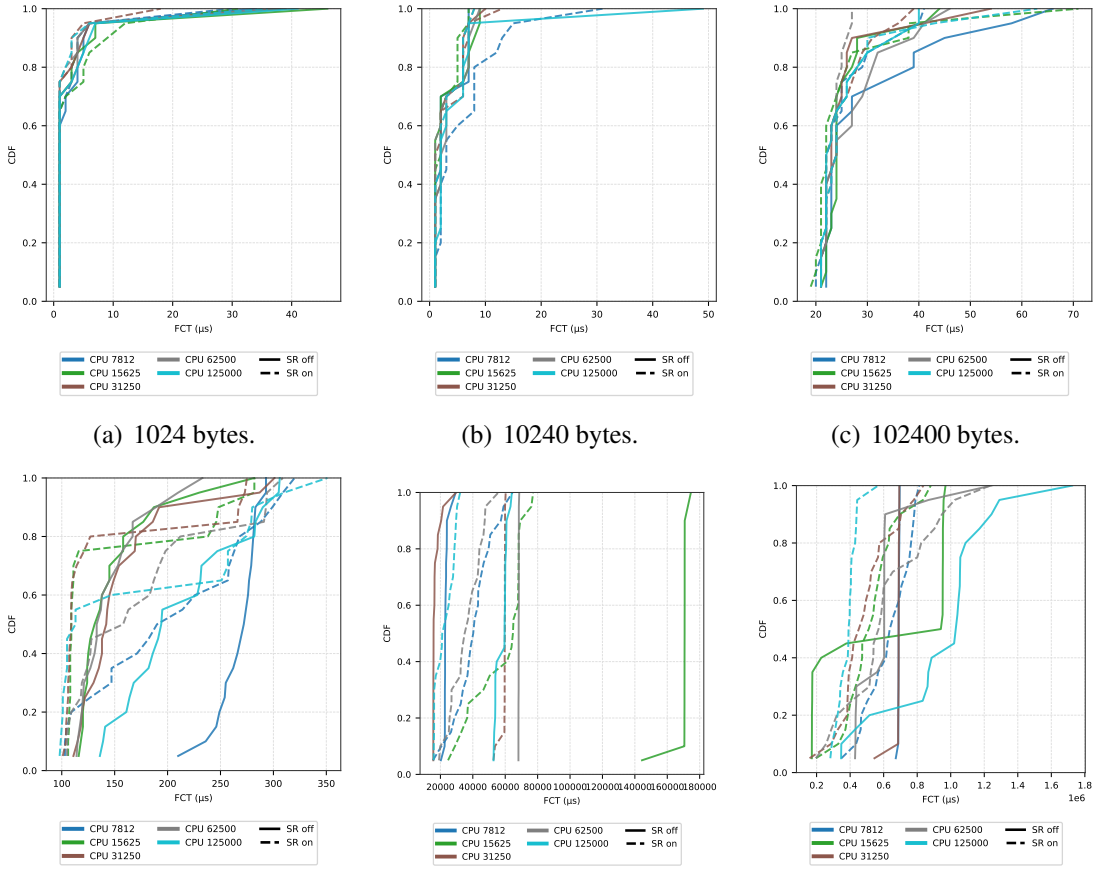
(d) 1024000 bytes. (e) 10240000 bytes. (f) 102400000 bytes.

**Figure 4. SR per CPU time ($\mu$s) of consumer with reactive configuration.**

Finally, in the saturation zone ($125.000\,\mu$s), the workload profile shifts into a CPU-bound regime. The reactive control becomes neutral or, in some cases, slightly detrimental in terms of FCT. Because the network already outpaces the processor's consumption, the previously mentioned packet-processing overhead negates any computational gains.

## 5.2. Proactive control

The experimental results (Figures 5 and 6) show that proactive control outperforms the reactive one, particularly in scenarios within the high sensitivity zone. By preventing SR from reaching high levels, the proactive control aims to keep the application operating continuously, eliminating the micro-recovery delays observed in the reactive model after buffer depletion. As a result, an even more pronounced leftward shift of the CDFs can be observed, yielding the lowest FCT values recorded for workloads of $7.812\,\mu$s and $15.625\,\mu$s. Even within the transition zone, the proactive control demonstrates superior




(a) 1024 bytes. (b) 10240 bytes. (c) 102400 bytes.

(d) 1024000 bytes. (e) 10240000 bytes. (f) 102400000 bytes.

**Figure 5. FCT with proactive configuration of SR marks.**

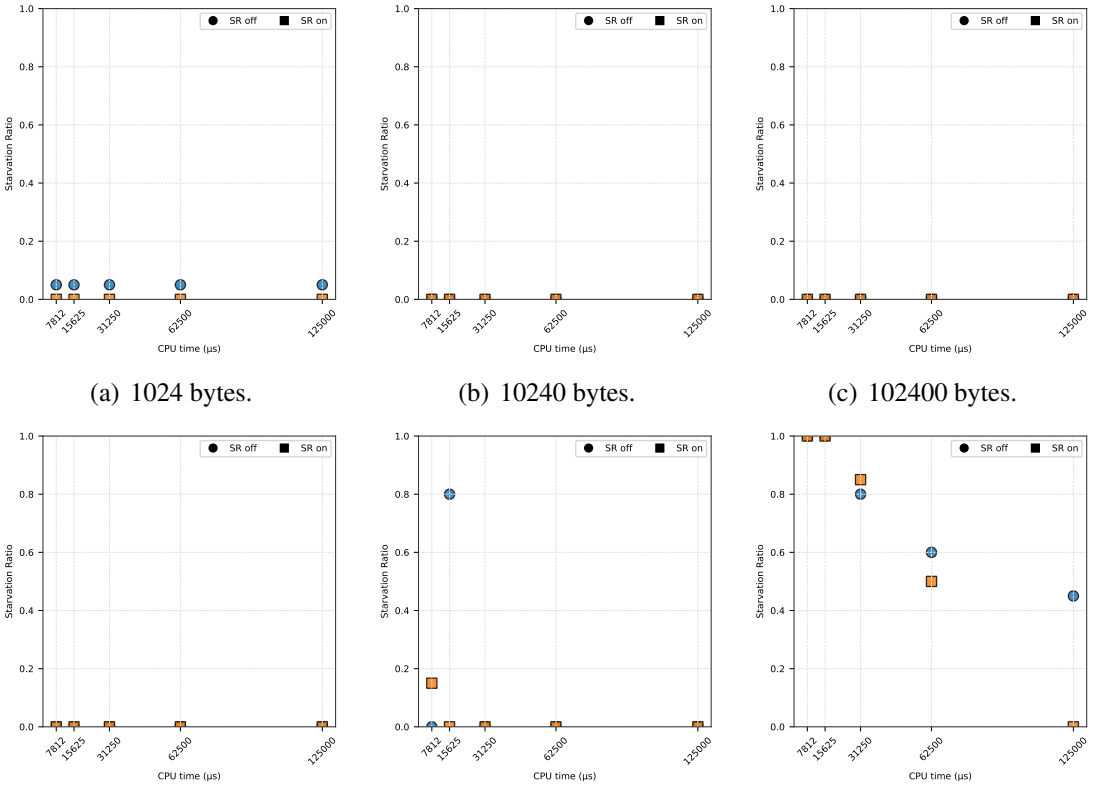
stability. While the reactive model exhibits sporadic starvation spikes before each correction, the proactive approach is able to keep SR consistently close to zero.

The effectiveness of the proactive approach diminishes as the workload reaches the saturation zone ($125.000\,\mu$s). For larger data flows (from 10 MB onward), enabling SR marking not only fails to provide gains but leads to an increase in FCT. In this CPU-bound regime, the extended processing interval allows the reception buffer to remain almost constantly filled, making the application insensitive to network fluctuations.

### 5.3. Key observations

The comparative analysis between the models allows for consolidating key insights into the effectiveness of the SR marking. First, the results confirm that adopting a reactive control already outperforms the standard congestion control protocol, showing that simply signaling starvation after its detection is sufficient to recover application performance. Within the high sensitivity zone, the proactive control emerges as the highest-performing solution, as its ability to anticipate buffer depletion eliminates CPU idle states. Nevertheless, the reactive model stands out as a robust and lower-complexity implementation alternative, particularly suitable for systems that can tolerate micro-delays.

The analysis of `cwnd` dynamics, presented in Figure 7, reveals the fundamental mechanism behind the observed FCT reduction. While background traffic flows drastically reduced their windows upon detecting congestion signals, the application demon-




(a) 1024 bytes. (b) 10240 bytes. (c) 102400 bytes.

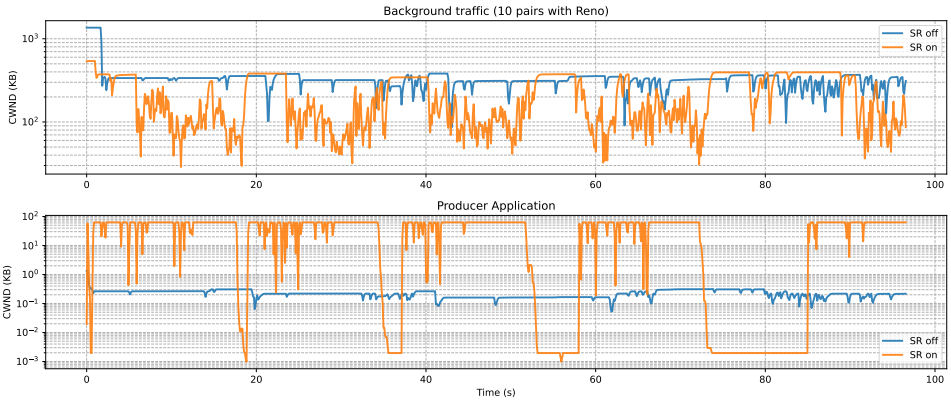
(d) 1024000 bytes. (e) 10240000 bytes. (f) 102400000 bytes.

**Figure 6. Starvation ratio distributed per CPU time ($\mu$s) of consumer with proactive configuration.**

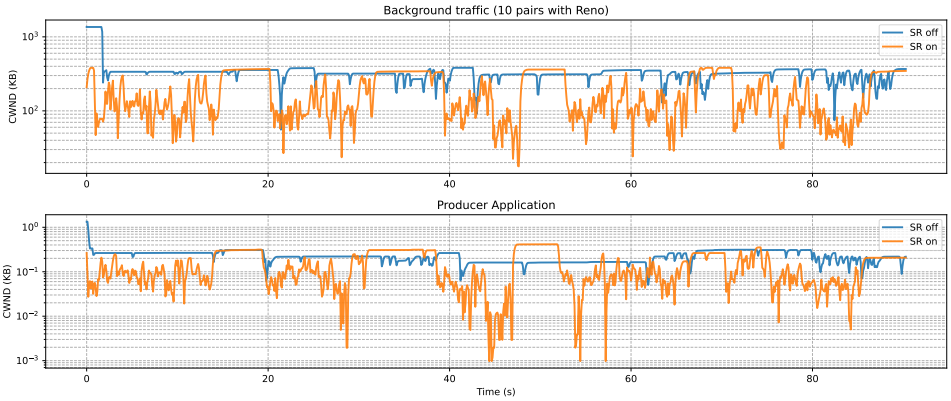
strates greater stability. Under reactive control (Figure 7(a)), the application's `cwnd` recovers more rapidly after starvation events, effectively reclaiming bandwidth left unused by traditional protocols. In contrast, under proactive control (Figure 7(b)), the gains manifest through flow stability: although differences in amplitude are more subtle, the mechanism prevents the micro-oscillations and preemptive drops typically observed in the baseline protocol. Maintaining a stable `cwnd` close to the receiver's limit ensures a more continuous and predictable data supply to the CPU.

Results demonstrate that integrating the SR into congestion control significantly reduces FCT in network-intensive scenarios. The execution time savings are consistent across different flow sizes, reinforcing the premise that SR identifies network opportunities overlooked by purely network-centric metrics. The experimental evidence supports the central hypothesis of this work: the application can and should actively participate in congestion control decisions. In this sense, shifting toward a SR-based congestion control ensures that DC operates in alignment with the processing pace of the flow's receiver.

While demonstrating framework viability, certain limitations apply. Regarding *internal validity*, host-stack overheads might introduce micro-delays in reactive signaling, and the behavior of administrative control traffic decoupled from traditional TCP mechanisms remains uncertain in production fabrics. In terms of *external validity*, our scenarios operate within a controlled 11-flow bottleneck topology using standard Reno. Consequently, evaluating the starvation ratio under larger fabrics, dynamic background




(a) Reactive control.




(b) Proactive control.

**Figure 7. Congestion control window of all TCP flows.**

workloads, and advanced protocols (e.g., DCTCP or BBR) remains an open challenge.

## 6. Conclusion

This work demonstrated that application-driven congestion control, guided by the Starvation Ratio (SR), effectively addresses the limitations of network-centric metrics. By acting only when starvation is detected, we can optimize network-sensitive applications. As future work, we will integrate machine learning techniques to predict SR before the degradation of overall DC performance. Additionally, given that the starvation ratio is structurally decoupled from transport-layer semantics, we plan to evaluate its applicability to alternative protocols, such as UDP and QUIC, as well as to fully disaggregated memory architectures where the network effectively assumes the role of the primary system bus within the DC.

## Acknowledgments

This work was partially funded by the National Council for Scientific and Technological Development (CNPq), the Coordination for the Improvement of Higher Education Personnel - CAPES - Brazil (PROAP/AUXPE), FAPESC and developed at LabP2D.

## References

Agarwal, S., Krishnamurthy, A., and Agarwal, R. (2023). Host congestion control. In *Proc. of the ACM SIGCOMM 2023 Conference*, page 275–287. ACM.

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data center tcp (dctcp). In *Proc. of the ACM SIGCOMM 2010 Conference*, pages 63–74.

Benson, T., Akella, A., and Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proc. of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, page 267–280. ACM.

Bernárdez, G., Suárez-Varela, J., Shi, X., Xiao, S., Cheng, X., Barlet-Ros, P., and Cabellos-Aparicio, A. (2025). GraphCC: a practical graph learning-based approach to congestion control in datacenters. *Computer Networks*, 257:110981.

Bonato, T., Kabbani, A., De Sensi, D., Pan, R., Le, Y., Raiciu, C., Handley, M., Schneider, T., Blach, N., Ghalayini, A., et al. (2024). FASTFLOW: flexible adaptive congestion control for high-performance datacenters. *arXiv preprint arXiv:2404.01630*.

Diel, G., Miers, C. C., Pillon, M. A., and Koslovski, G. P. (2023). RSCAT: Towards zero touch congestion control based on actor–critic reinforcement learning and software-defined networking. *Journal of Network and Computer Applications*, 215:103639.

Dukkipati, N. and McKeown, N. (2006). Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1):59–62.

Ewais, M. and Chow, P. (2023). Disaggregated memory in the datacenter: a survey. *IEEE Access*, 11:20688–20712.

Geng, Y., Zhang, H., Shi, X., Wang, J., Yin, X., He, D., and Li, Y. (2023). Delay based congestion control for cross-datacenter networks. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pages 1–4.

Katebzadeh, M. S., Costa, P., and Grot, B. (2023). Saba: rethinking datacenter network allocation from application's perspective. In *Proc. of the Eighteenth European Conference on Computer Systems*, pages 623–638.

Montazeri, B., Li, Y., Alizadeh, M., and Ousterhout, J. (2018). Homa: a receiver-driven low-latency transport protocol using network priorities. In *Proc. of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 221–235.

Prasopoulos, K., Kosta, R., Bugnion, E., and Kogias, M. (2025). SIRD: A sender-informed, receiver-driven datacenter transport protocol. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 451–471.

Reed, D., Gannon, D., and Dongarra, J. (2022). Reinventing high performance computing: challenges and opportunities. *arXiv preprint arXiv:2203.02544*.

Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984). End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288.

Tessler, C., Shpigelman, Y., Dalal, G., Mandelbaum, A., Haritan Kazakov, D., Fuhrer, B., Chechik, G., and Mannor, S. (2022). Reinforcement learning for datacenter congestion control. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):43–46.

Wang, H., Lu, Y., and Wang, Z. (2026). GFCC: a global fast congestion control mechanism based on software-defined networking. *IEEE Transactions on Networking*, 34:2746–2761.