SRAP: Sender-side Receiver-Aware Port selection for High-Speed Multi-Flow TCP

Shingo Hattori[1], Osamu Tatebe[2] (advisor)

1. Graduate School of Science and Technology, University of Tsukuba
2. Center for Computational Sciences, University of Tsukuba

筑波大学 University of Tsukuba

SC25 St. Louis, MO | hpc ignites.

Abstract

Achieving **400 Gbps** requires aggregating **multiple flows** across cores rather than pushing single-flow limits. However, with **16×25 Gbps TCP** flows, random ephemeral ports cause receiver-side packet steering (RSS) to concentrate flows on single CPU cores, degrading throughput from **25 Gbps** to below **5 Gbps.**

Unlike receiver-side solutions that suffer from cache misses and state migration overhead, **SRAP** transparently controls source ports at senders, ensuring **collision-free 1:1 mapping** without runtime remapping costs or application modification.

With **16×25 Gbps** flows, random assignment achieves optimal distribution with probability 1.04%, causing throughput to vary from 44.8 to 395 Gbps, while our approach consistently delivers **23.3-25 Gbps per flow** with guaranteed 1:1 flow-to-core mapping.

1. Introduction

How to achieve 400G throughput with pacing...? Aggregating multiple flows, but there are difficulties.

Single-flow throughput is limited by sequential processing and CPU clock speeds. Aggregating multiple flows across multiple cores provides a scalable path—achieving 400 Gbps requires distributing load across multiple paced flows (e.g., 16×25 Gbps).

Pacing prevents microbursts and packet loss, but when RSS maps multiple paced flows to the same CPU core, maintaining target-rate pacing becomes impossible.

We propose Source Port Assignment Protocol (SRAP), a sender-side approach that prevents flow collisions through intelligent source port selection, achieving 9× better worst-case throughput than random assignment while avoiding dynamic remapping overhead.

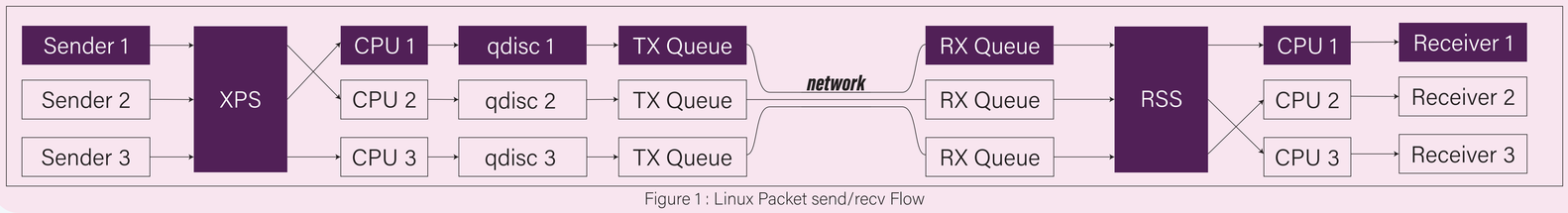

Figure 1 : Linux Packet send/recv Flow

2. Problem Analysis

RSS's random flow-to-core mapping destroys packet pacing.

RSS distributes packets across CPU cores by hashing 4-tuples to map flows to receive queues. Random ephemeral ports create unpredictable flow-to-core mappings - with 16 flows and 32 cores, collision-free probability is only 1.04%.

Comparison: RSS++ suffers from cache misses and state migration overhead. ARFS requires kernel modifications. SRAP avoids these issues.`

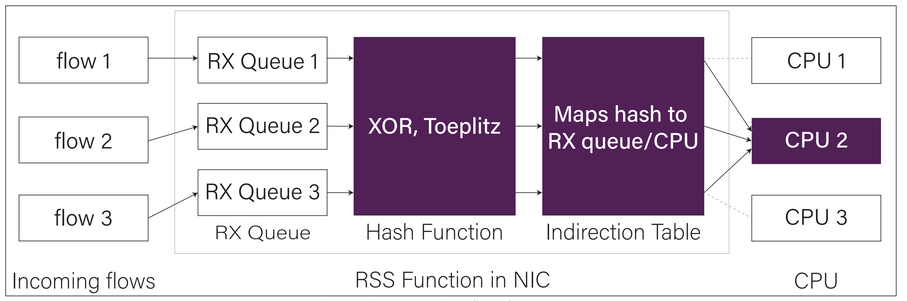

Figure 2 : Packet reception flow from NIC to CPU

Flow collisions limit throughput—worst case: 16 flows on one core, <5 Gbps each. Random ports cause unpredictable collisions, making aggregation unreliable. Sender CPUs are controllable but receiver RSS is opaque. We enable predictable mappings via source ports.

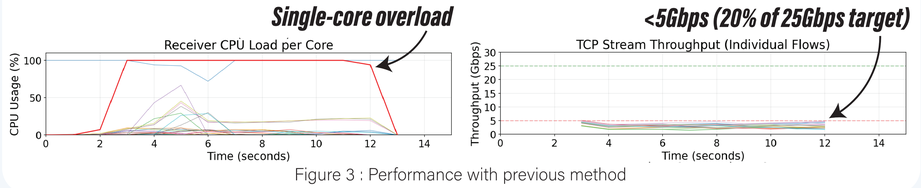

Figure 3 : Performance with previous method

3. Proposed Approach

Select source ports at sender to guarantee collision-free mapping.

SRAP controls receiver-side flow distribution by selecting source ports that hash to different CPU cores through three coordinated components.

1. **RSS Information Exchange**: Exchanges RSS configurations (indirection tables, hash functions) with receivers—no app core binding needed, only RSS table matters.
2. **Port-to-CPU Predictor**: Computes RSS hashes and returns collision-free ports.
3. **Transparent Port Binder**: Intercepts connect() calls transparently, requests collision-free ports, and calls bind() to control the RSS 4-tuple without application modification.

By controlling source ports, SRAP prevents receiver-side flow collisions, achieving predictable multi-flow pacing without modifying applications or receivers.

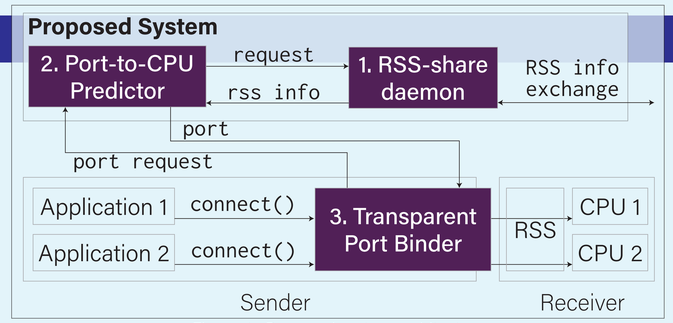

Figure 4 : Proposed system architecture

4. Evaluation

Successfully achieve 400 Gbps aggrigate with pacing.

Setup : AMD EPYC 9354P, 32 cores with DDR5 256GB and ConnectX-7 400GbE NICs running Linux 6.8.0. 16×25 Gbps TCP CUBIC flows with Linux FQ[3] pacing, iperf3 zero-copy optimizations, and 100MB windows.

Results : Figure 4 demonstrates SRAP's effectiveness—all 16 flows maintain stable 23-25 Gbps throughput while CPU load is evenly distributed at 50-65% across cores. This contrasts sharply with random assignment where flows dropped below 5 Gbps and CPUs reached 100% utilization due to collisions.

SRAP achieves perfect 1:1 flow-to-core mapping, eliminating the 99% collision probability inherent in random ports. Aggregate throughput reaches 380-395 Gbps, compared to 44.8-395 Gbps variance without control, enabling reliable multi-flow aggregation.

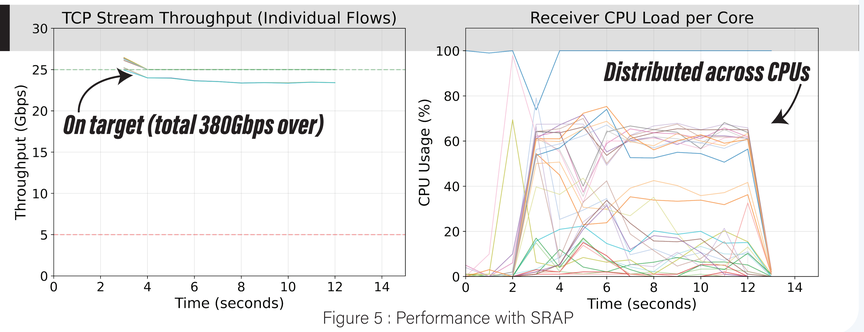

Figure 5 : Performance with SRAP

5. Conclusion & Discussion

Scalable pacing for 400G aggregate and beyond.

We demonstrated that random port assignment causes severe RSS flow collisions, with only 1.04% probability of achieving optimal distribution for 16 flows on 32 cores, resulting in throughput varying from 44.8 to 395 Gbps.

SRAP (Sender-side Receiver-Aware Port selection) transparently controls source ports, achieving consistent 1:1 mapping and 380-395 Gbps throughput without modifying applications or receivers.

This demonstrates that source port engineering transforms unpredictable multi-flow performance into reliable high-throughput communication essential for 400G+ environments, achieving 9× faster worst-case transfers while preventing network-wide congestion from pacing breakdown.

References [1] Tom Barbette, Georgios P. Katsikas, Gerald Q. Maguire, and Dejan Kostic. 2019. RSS++: load and state-aware receive side scaling. In Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (Orlando, Florida) (CoNEXT '19). Association for Computing Machinery, New York, NY, USA, 318–333. doi:10.1145/3359989.3365412 [2] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. Commun. ACM 60, 2 (Jan. 2017), 58–66. doi:10.1145/3009824 [3] Jake Edge. 2013. TSO sizing and the FQ scheduler. LWN.net (August 2013). https://lwn.net/Articles/564978/ [4] Nathan Hanford, Brian Tierney, and Dipak Ghosal. 2015. Optimizing data transfer nodes using packet pacing. In Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science (Austin, Texas) (INDIS '15). Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. doi:10.1145/2830318.2830322 [5] Tom Herbert and Willem de Bruijn. 2015. Scaling in the Linux Networking Stack. https://www.kernel.org/doc/Documentation/networking/scaling.txt. Linux Kernel Documentation. [6] Marcos Schwarz, Brian Tierney, Kiran Vasu, Eli Dart, Christian Esteve Rothenberg, Jeronimo Bezerra, and Italo Valcy. 2025. Recent Linux Improvements that Impact TCP Throughput: Insights from R&E Networks. In Proceedings of the SC'24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (Atlanta, GA, USA) (SC-W '24). IEEE Press, 775–784. doi:10.1109/SCW63240.2024.00111 **Acknowledgements** This work is based on results obtained from the project, ``Research and Development Project of the Enhanced infrastructures for Post-5G Information and Communication Systems" (JPNP20017), commissioned by the New Energy and Industrial Technology Development Organization (NEDO), and was supported by JSPS KAKENHI Grant Number JP22H00509, JST CREST JPMJCR24R4, the Multidisciplinary Cooperative Research Program in CCS, University of Tsukuba, and Fujitsu Limited.