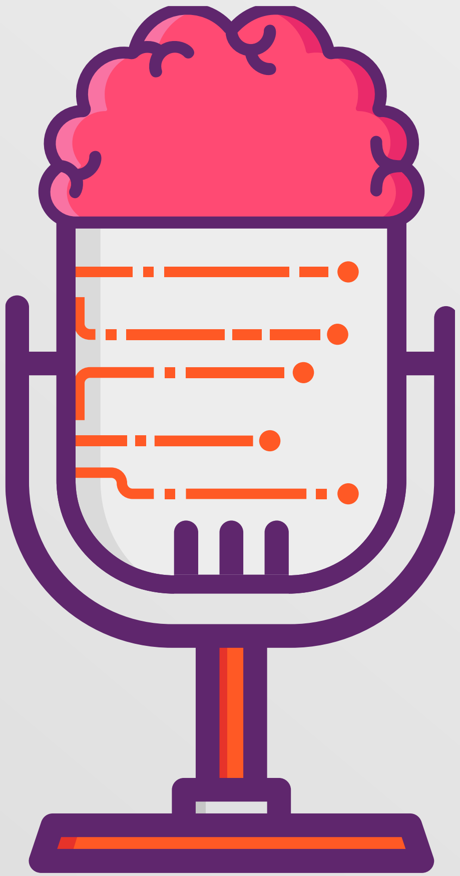
Towards Scalable and Expressive Stream Packet Processing

Relatore: Alessandra Fais

NeRdTalks S03E03 @ NetResults - 28/01/2022



Introduction and Motivations

Introduction

- Modern networks are *softwarized*
  - Specialized hardware replaced by general purpose devices
  - Network functions implemented as software applications
- And they are shared
  - Accommodate a variety of services on the same infrastructure
  - Adapt to requests for service (de-)activation
  - Always guarantee Quality of Service (QoS) requirements

- Network operators need tools that ease the task of network programming
  - Rapid and easy network (re-)configuration and management
  - Continuous real-time monitoring for detecting security issues and performance degradation

A new era of programmable networks

- In-network data plane processing
  - Programmable software switches
  - Programmable abstractions (e.g., P4)

- Commodity hardware for data plane operations (general-purpose servers)
  - Cheaper alternatives to specialized hardware
  - High flexibility
  - Good maturity level
    - Multi-core architectures to enable parallel computations
    - Multi-queue NICs for parallel tapping of packets from the wire
    - Software accelerated packet capture frameworks

The need to process live packet streams on network end hosts

- Perform accurate packet analytic tasks in end-host servers depends on
  - Ability to capture all the packets arriving to the NIC at speeds of 10+ Gbps
  - Design processing applications to handle incoming data fast enough not to lose packets

<u>STRENGTHS:</u>

- Great flexibility, fully programmable platforms
- Multi-gigabit packet handling with huge volumes of traffic can be achieved
  - Use accelerated packet capture frameworks

<u>PROBLEM:</u>

- How to design network applications to scale with the speed of received data and not get overwhelmed?

The need for parallelism

- Design network applications which take advantage of multiple cores

  COMPLEX TASK:

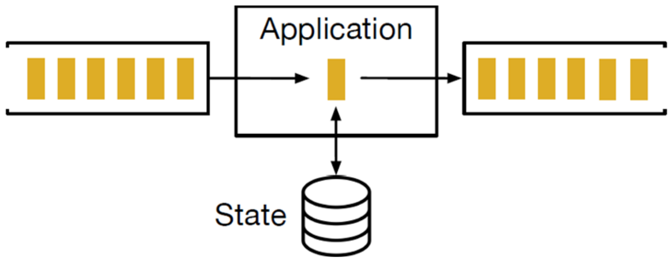
  - Network programmers must build their applications from scratch by
    - Implementing the interfaces towards the lower hardware level
    - Implementing the proper mechanisms for parallel programming

  SOLUTION:

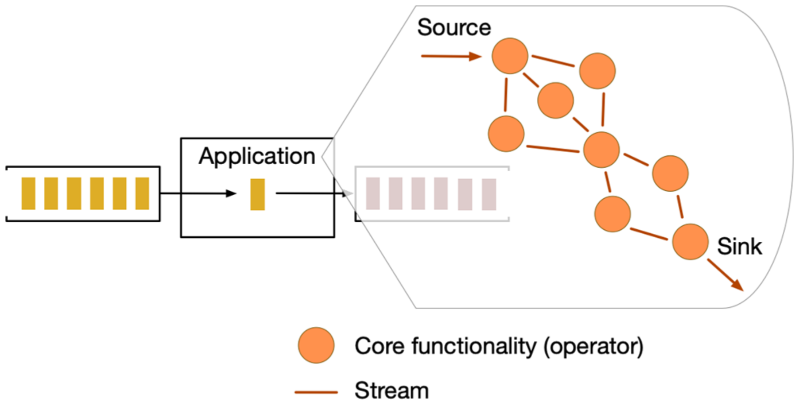
  - Leverage the power of Data Stream Processing (DaSP) frameworks
    - Manage parallelism in an effective way
    - Offer high-level programming abstractions
  - Let network programmers only focus on the application logic

Background: Data Stream Processing (DaSP)

- Useful everywhere there is the need for continuous and real-time analysis of streaming data
  - Networking, smart cities, smart mobility, smart logistics, ...



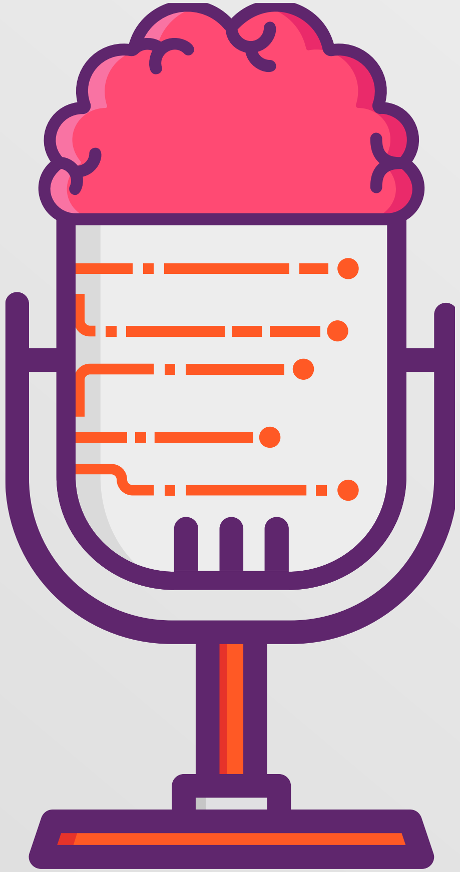
- DaSP frameworks
  - Simplify the implementation of efficient streaming computations
  - Designed to implement general-purpose applications
  - Offer programming abstractions of multi-stage pipelines
    - Every stage (operator) can be seamlessly parallelized
    - Lower-level details hidden to the programmer



Background: DaSP frameworks

- Mainstream DaSP frameworks
  - Based on the Java Virtual Machine
  - Support implementation of streaming applications on distributed environments
  - Many overheads impact performance
    - Not possible to keep up with common traffic rates in networking scenario

- A recent proposal: WindFlow
  - Targets single multi-core machines with GPU support
  - C++17-based
  - High performance + high expressiveness



Proving the feasibility of using the DaSP approach in the networking scenario

The need for suitable extensions

- The WindFlow DaSP framework
  - Offers a modern design and high performance
  - However, it is conceived to implement general streaming computations
  - It lacks mechanisms for binding source operators to multiple sources of data (NIC hardware queues)
  - Source operators must be tuned for packet capturing and parsing

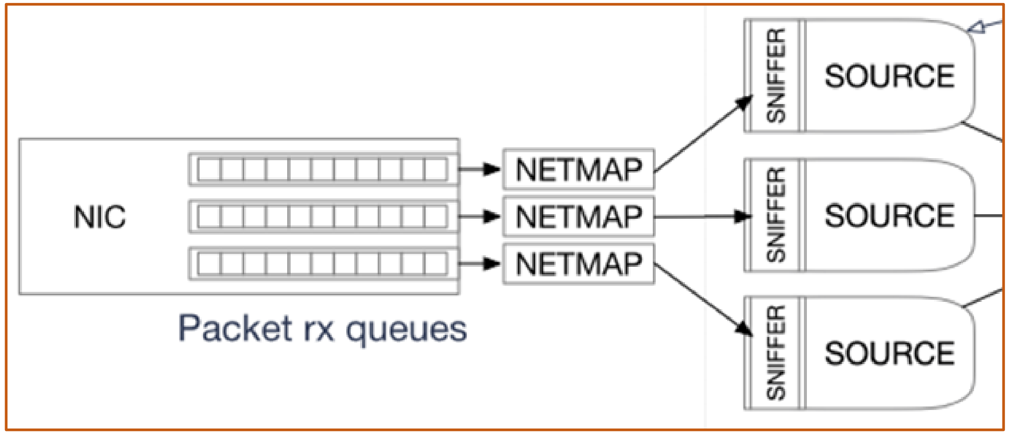
- We implemented proper optimizations for the networking domain
  - Full support for parallel source entities
  - Each one attached to a different NIC hardware queue
  - Acceleration of source operators
  - Fully generic memory pool of tuples

Support for multi-source

- The SOURCE oversees the following tasks
  - Receives captured packets from the NIC
    - Using fast packet capturing tools (e.g., Netmap, AF_XDP, libpcap, AF_PACKET)
    - Module called SNIFFER
  - Parses packet headers and creates accordingly all the tuples that will be processed in the graph

- The SOURCE is the bottleneck in the graph
  - Given its considerable workload

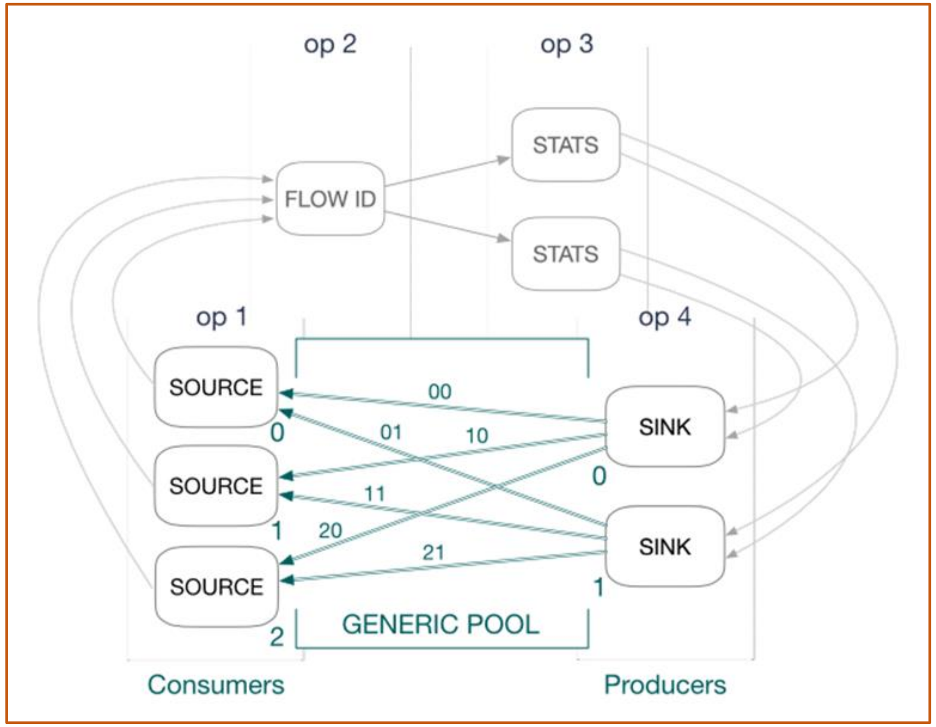
- We removed the performance bottleneck by parallelizing the SOURCE operator



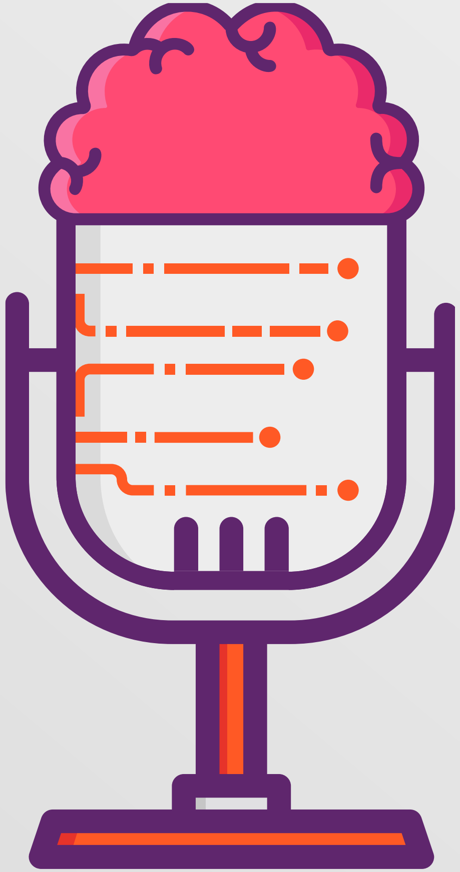
Generic memory pool

- Tuples are the stream elements flowing in the application graph
  - SOURCE nodes initialize tuples from packet contents
  - SINK nodes collect tuples at the end of the processing

- Memory pool and tuple recycling in the application graph to avoid expensive memory allocations
  - Pre-allocated pool of empty tuples
  - Matrix of communication channels
    - Implemented as SPSC* queues
  - Support for generic number of consumer and producers
    - SOURCE consumes tuples from the pool
    - SINK produces tuples into the pool



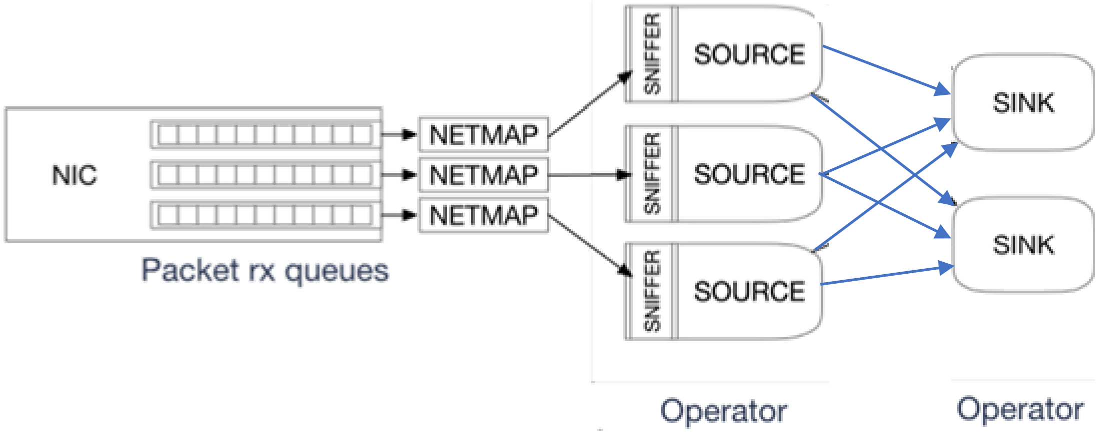
* SPSC = Single Producer Single Consumer



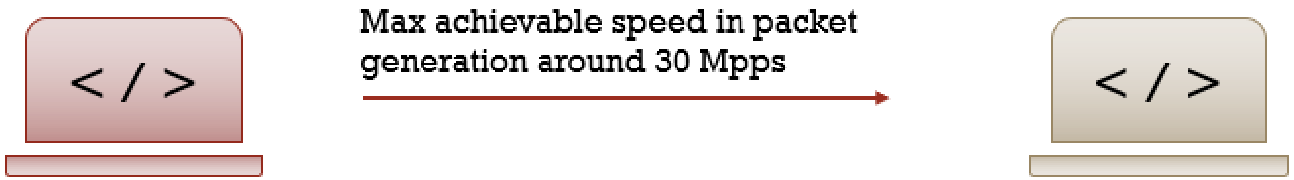
Experimental evaluation

Raw speed test

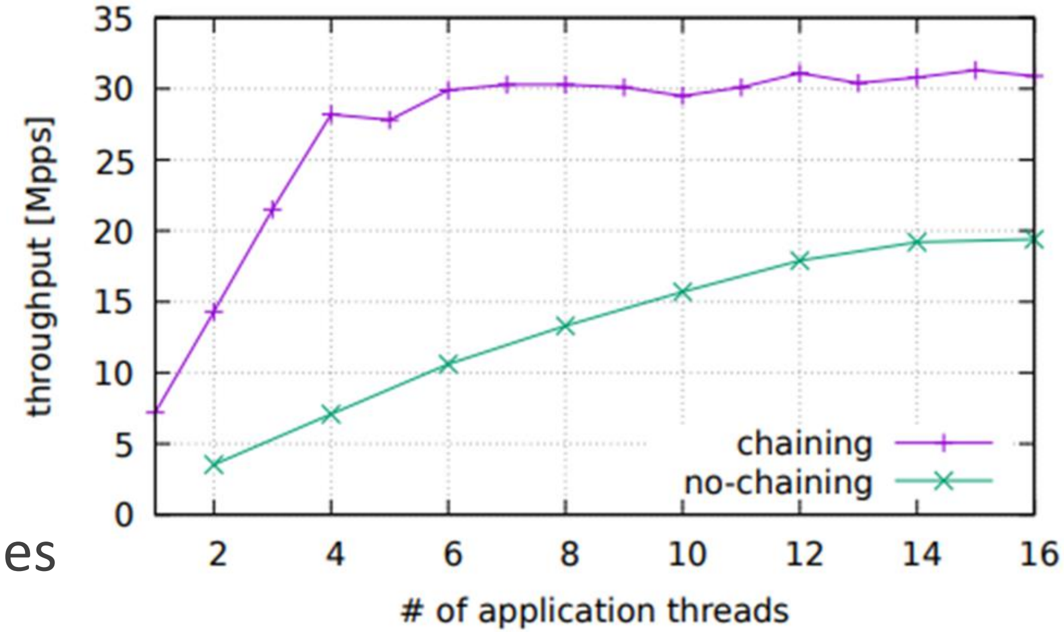
- Assess the scalability of the basic framework
- Modules in the system
  - Packet capturing sub-system
  - Inter-thread communication mechanisms
  - Tuple memory pool
- Minimal configuration of the application
  - SOURCE and SINK operators only
  - Tested configurations
    - Chained execution (on the same thread)
    - Non-chained execution (on separated threads)



Raw speed test



- Workload characteristics
  - Synthetic traffic
  - Minimally sized packets (64 bytes, UDP)
  - Traffic generator runs pkt-gen (Netmap suite)



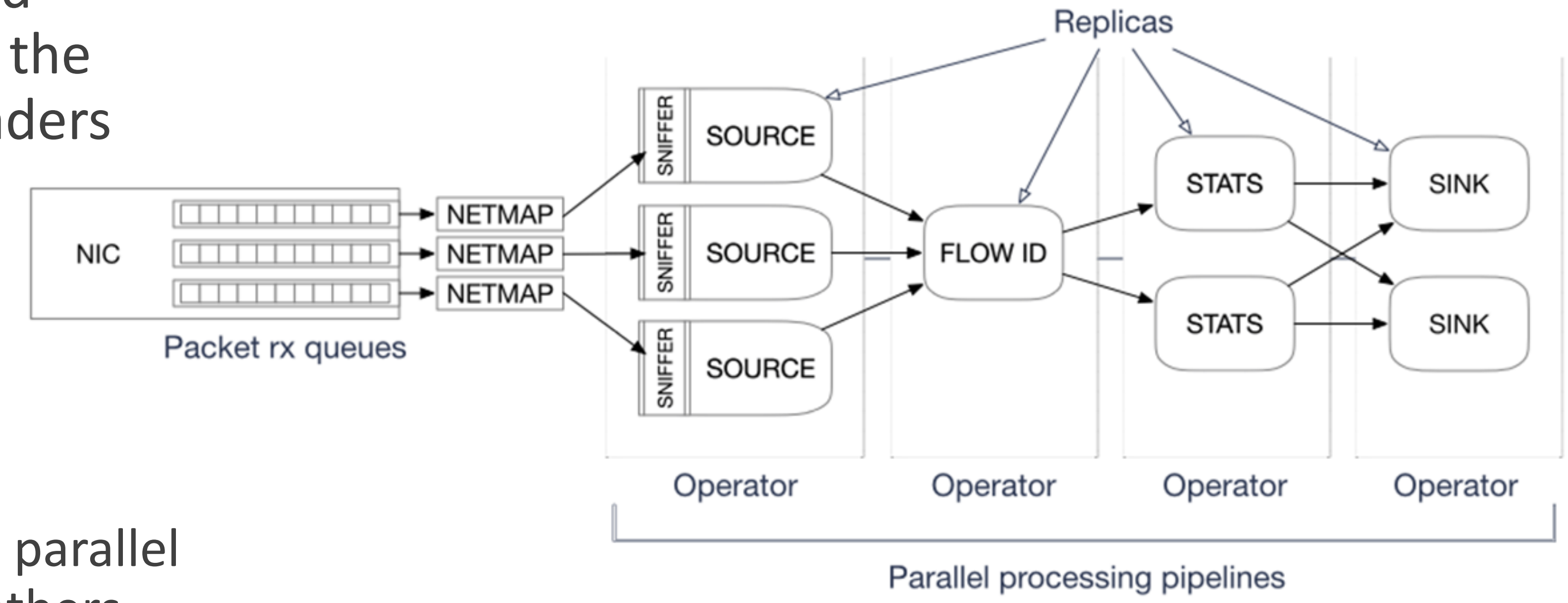
- Packet rate (millions of packets per second) sustained by the minimal packet-counter application for increasing number of source nodes

- Given X application threads, the number of source nodes in the chaining case is X, while in the no-chaining case is X/2

Flow counter application test

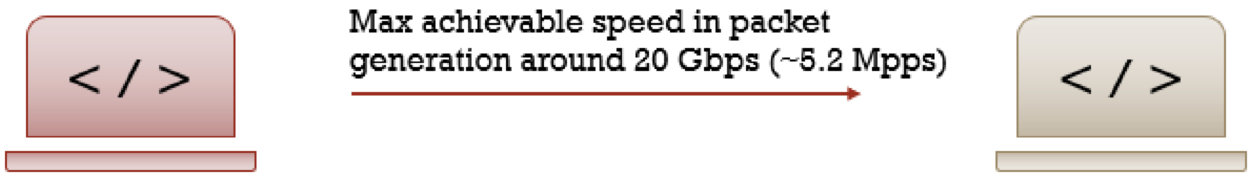
More complex application:

- Produce some packet and flow level analytics from the processing of packet headers

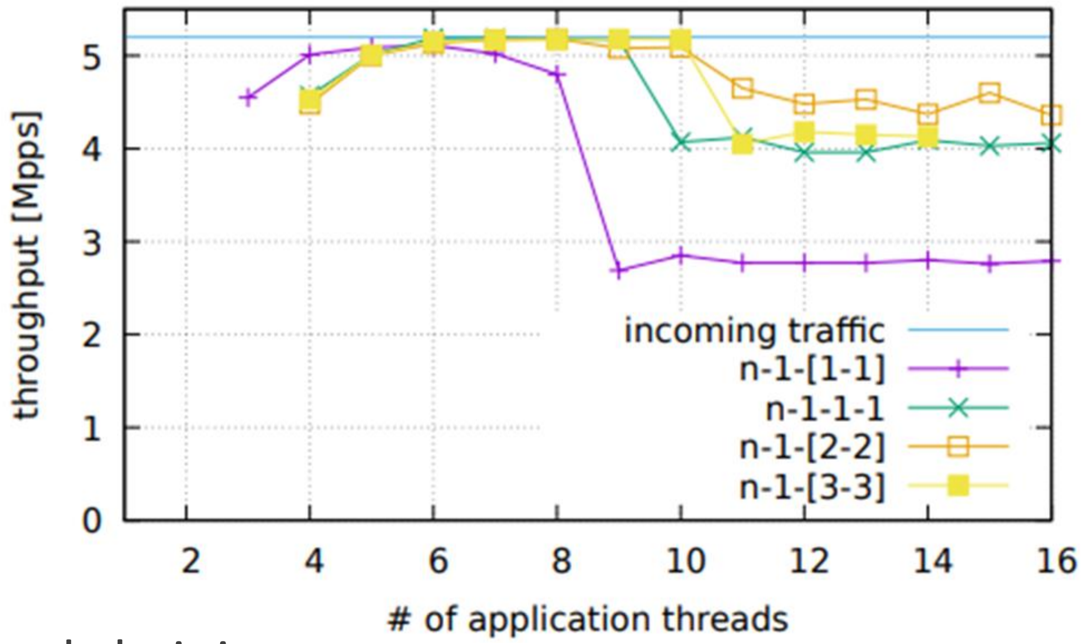


- Pipeline of 4 nodes
    - All can be replicated in parallel independently of the others

Flow counter app test

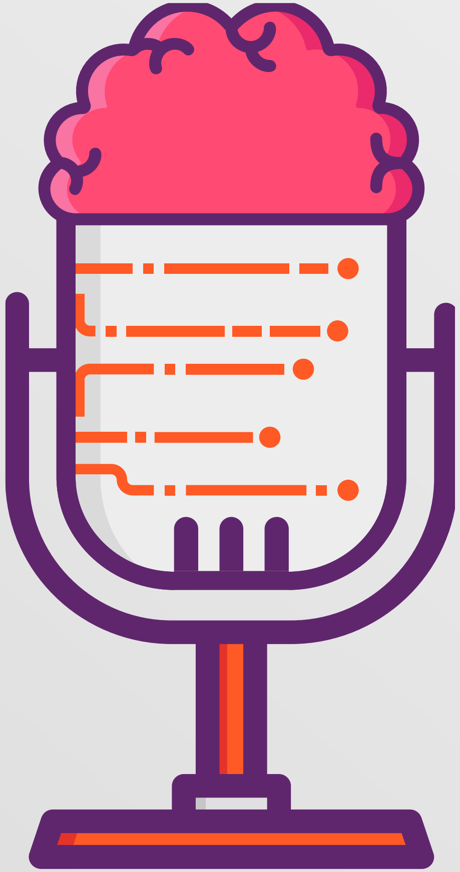


- Workload characteristics
  - Realistic traffic
  - pcap file replayed at max speed using nmreplay (Netmap suite)
  - packets of different lengths (generally larger than 64 bytes)



- Packet rate (millions of packets per second) sustained by the test monitoring application with three different combinations of replication and chaining

n-1-1-1	n SOURCE nodes, no other replicated nodes, no chaining
n-1-[1-1]	n SOURCE nodes, no other replicated nodes, STATS node is chained with SINK node
n-1-[2-2]	n SOURCE nodes, one FLOWID node, two nodes each of both STATS and SINK chained pairwise



Conclusions

- App throughput increases linearly with the number of sources
  - Full data elaboration sustained up to nearly 20 Gbps in different configurations
- Proved feasibility of using DaSP approach to compute packet and flow level analytics from live data
  - Common computational pattern for network applications
  - Network monitoring is one of the largest domains of interest
- WindFlow can be a convenient development platform in the networking domain
  - <u>FUTURE DIRECTIONS:</u>
integrate these results into the WindFlow framework
to provide a new native networking module with *ad hoc*
ready-to-use network-oriented operators