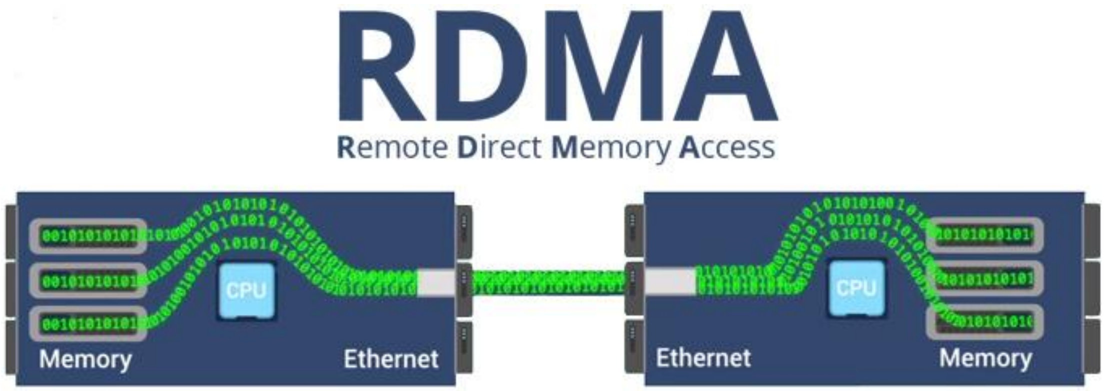
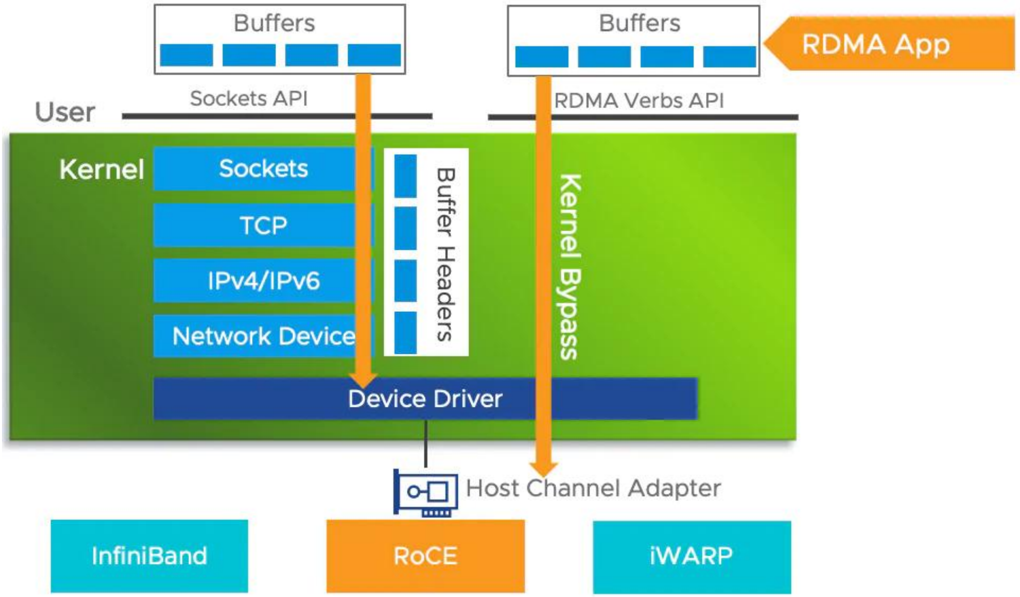
# Status and plans for RDMA based DAQ on eFEC

Sorin Martoiu, Matei-Eugen Vasile, Gabriel Stoicea
Nayib Boukadida
Costin-Emanuel Vasile, Andrei-Alexandru Ulmamei,
Voichita Iancu, Calin Bira, Radu Hobincu

# RDMA Explained (I)




Buffers

Buffers

RDMA App

User

Sockets API

RDMA Verbs API

Kernel

Sockets

TCP

IPv4/IPv6

Network Device

Buffer Headers

Kernel Bypass

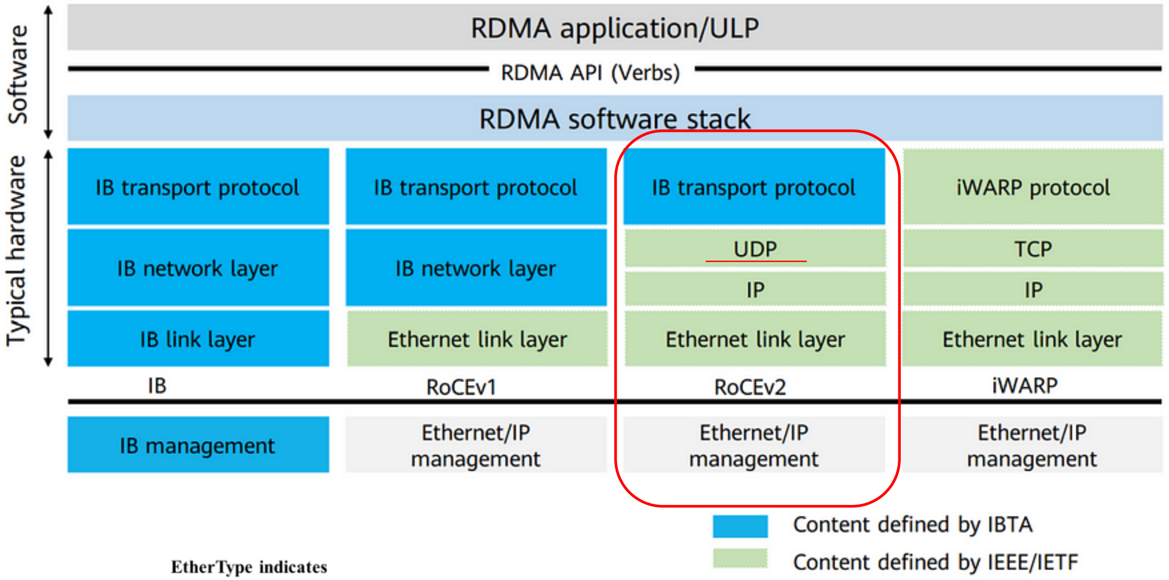
Device Driver

Host Channel Adapter

InfiniBand

RoCE

iWARP


Software

Typical hardware

RDMA application/ULP

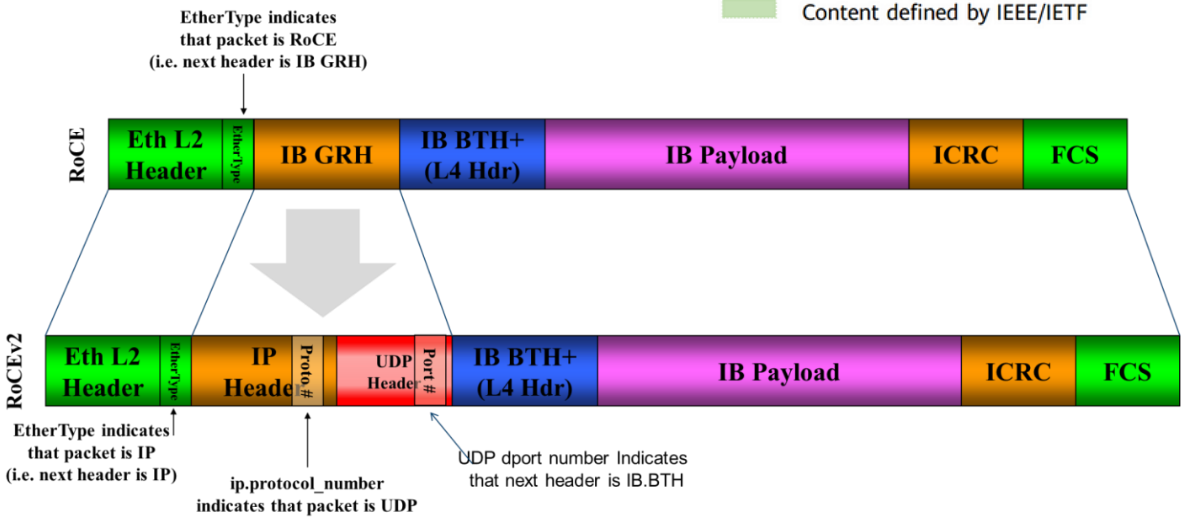
RDMA API (Verbs)

RDMA software stack

| IB | RoCEv1 | RoCEv2 | iWARP |
|---|---|---|---|
| IB transport protocol | IB transport protocol | IB transport protocol | iWARP protocol |
| IB network layer | IB network layer | UDP<br>IP | TCP<br>IP |
| IB link layer | Ethernet link layer | Ethernet link layer | Ethernet link layer |
| IB management | Ethernet/IP management | Ethernet/IP management | Ethernet/IP management |

🟦 Content defined by IBTA

🟩 Content defined by IEEE/IETF


EtherType indicates that packet is RoCE (i.e. next header is IB GRH)

RoCE: Eth L2 Header | EtherType | IB GRH | IB BTH+ (L4 Hdr) | IB Payload | ICRC | FCS

RoCEv2: Eth L2 Header | EtherType | IP Header | Proto # | UDP Header | Port # | IB BTH+ (L4 Hdr) | IB Payload | ICRC | FCS

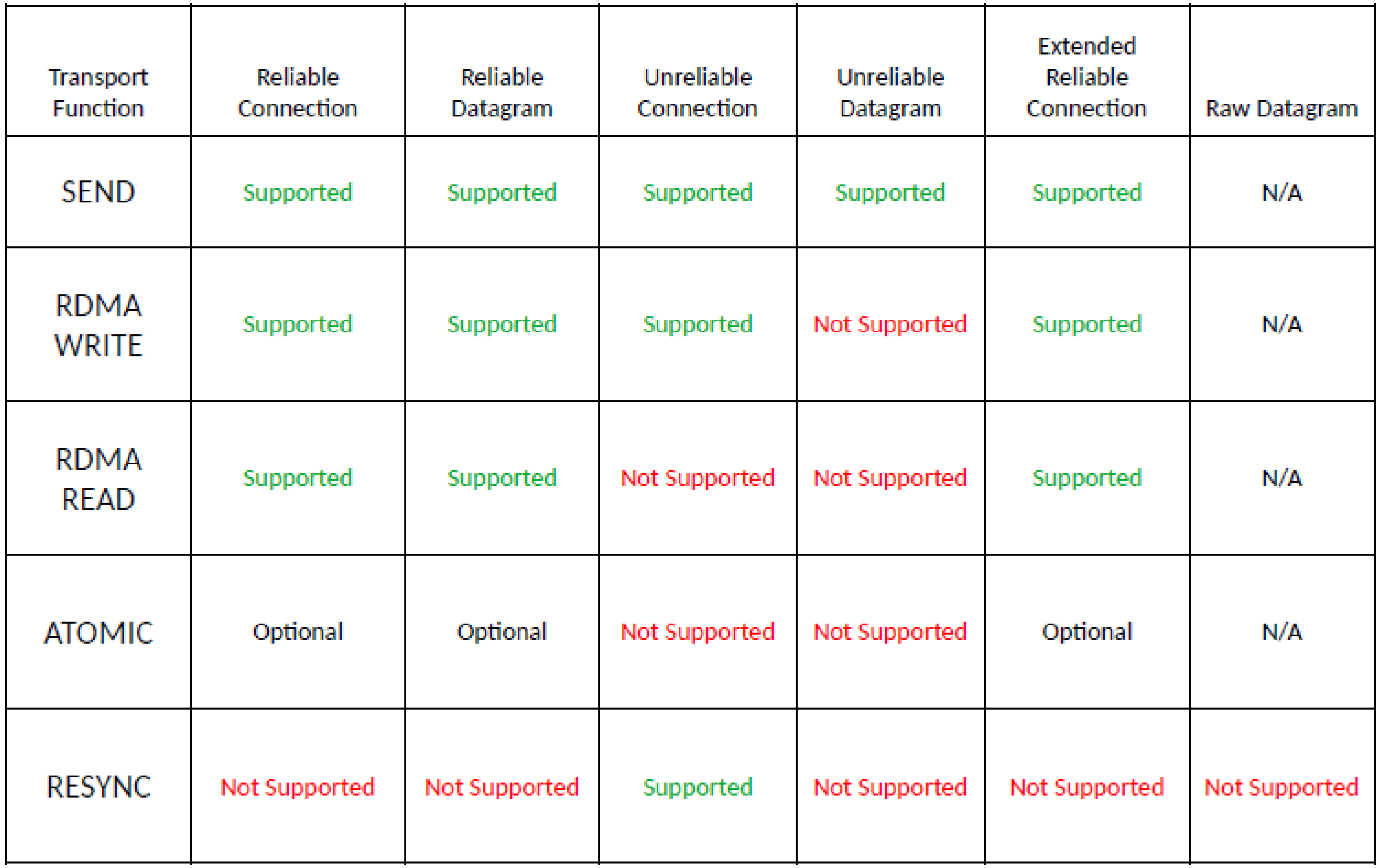
EtherType indicates that packet is IP (i.e. next header is IP)

ip.protocol_number indicates that packet is UDP

UDP dport number Indicates that next header is IB.BTH

# RDMA Explained (II)

- **RDMA (Remote Direct Memory Access)** is a means of transfering data between devices with little CPU involvement
- RDMA supports several **Transport Services**, the relevant one for the current implementation being *Reliable Connection (RC)*
- and several **Transport Functions**, the most relevant ones for the current implementation being SEND and WRITE

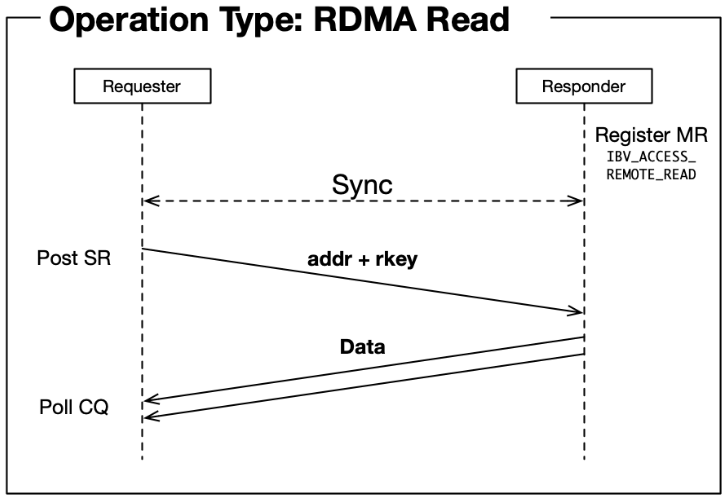


| Transport Function | Reliable Connection | Reliable Datagram | Unreliable Connection | Unreliable Datagram | Extended Reliable Connection | Raw Datagram |
|---|---|---|---|---|---|---|
| SEND | Supported | Supported | Supported | Supported | Supported | N/A |
| RDMA WRITE | Supported | Supported | Supported | Not Supported | Supported | N/A |
| RDMA READ | Supported | Supported | Not Supported | Not Supported | Supported | N/A |
| ATOMIC | Optional | Optional | Not Supported | Not Supported | Optional | N/A |
| RESYNC | Not Supported | Not Supported | Supported | Not Supported | Not Supported | Not Supported |

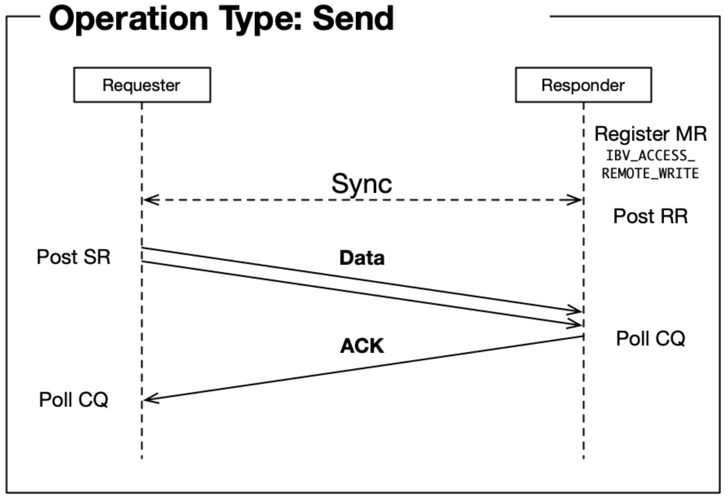
InfiniBand Architecture Specification Volume 1, Release 1.4

# RDMA Explained (III)

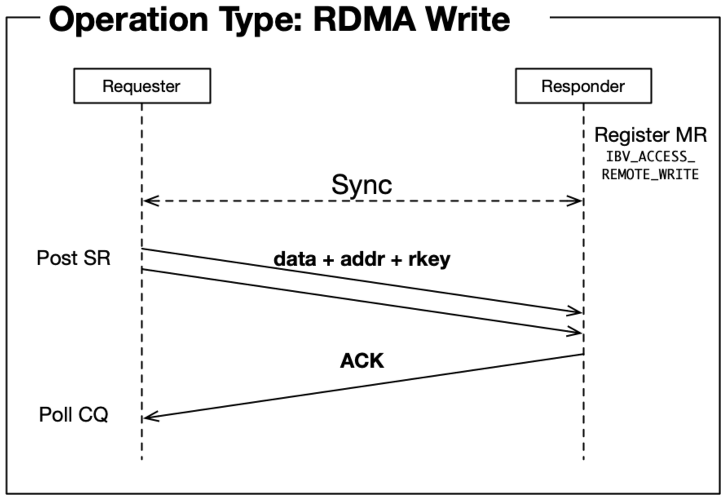
### Operation Type: RDMA Read



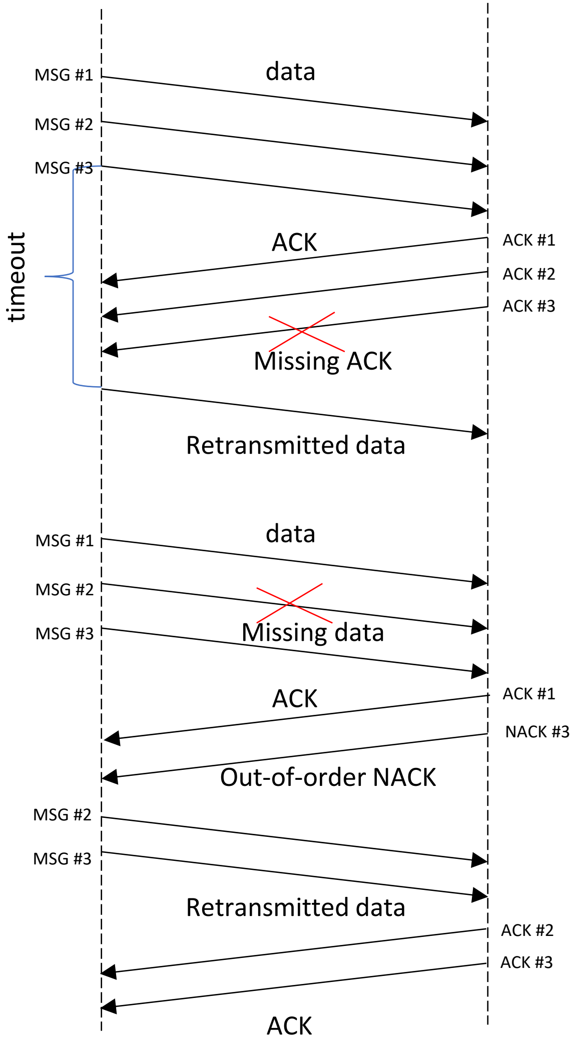
### Operation Type: Send



### Operation Type: RDMA Write



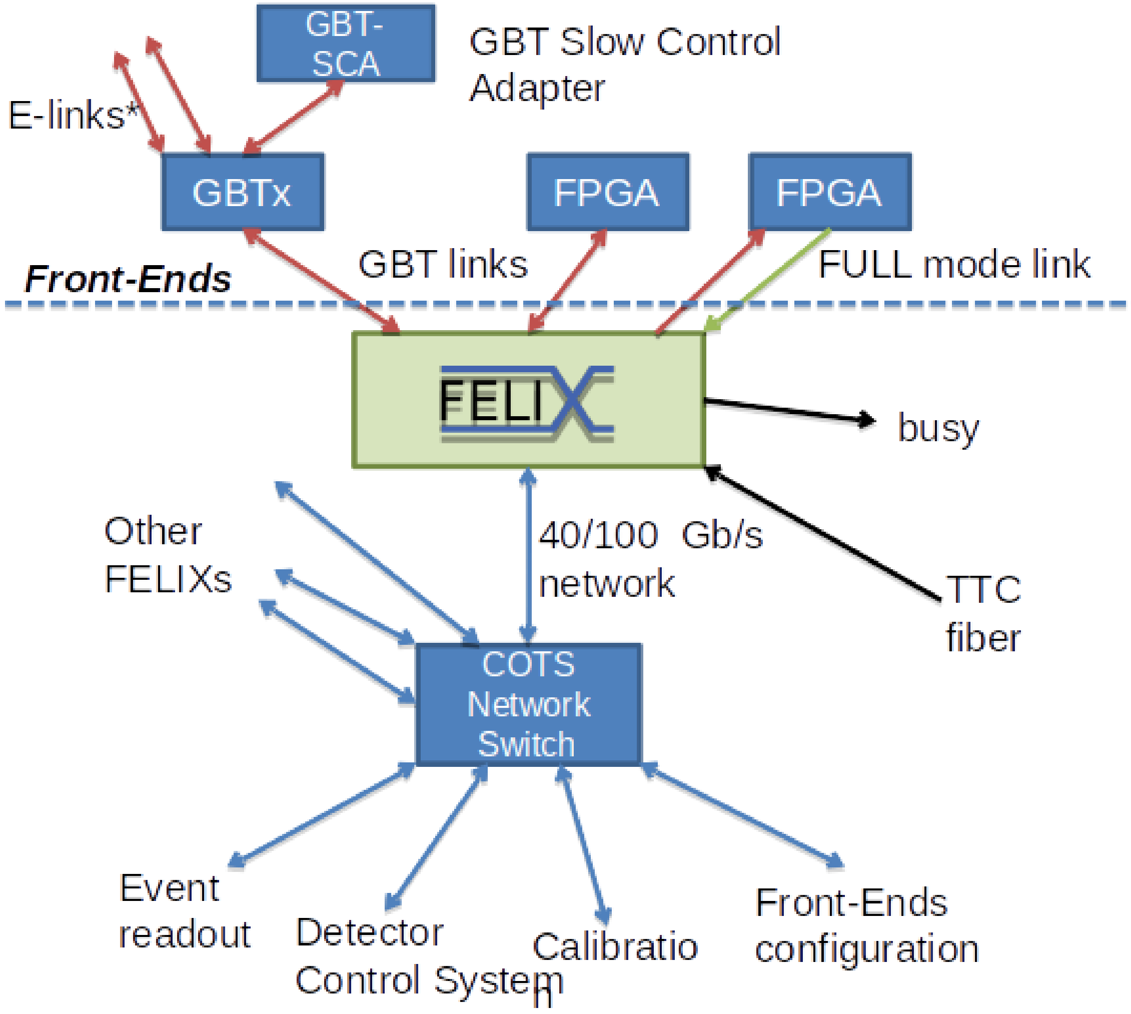
## Reliable Connection (RC)



# ATLAS FELIX & netio-next

- **FELIX** routes event, detector control, configuration, calibration and monitoring data
- Aggregates e-links
- Connects bidirectionally ATLAS detector Front-Ends and the DAQ system
- Detector independent
- It is built using:

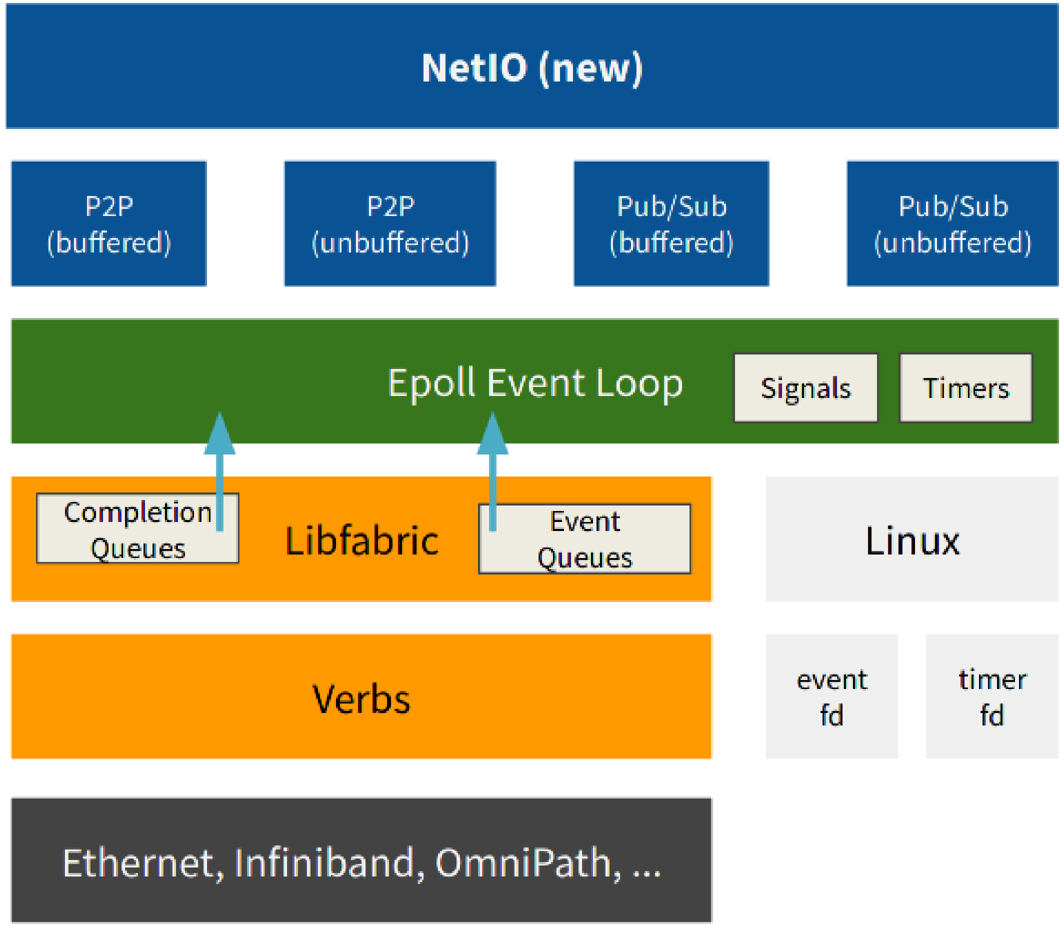
- **netio-next**[1] is a CERN-developed communication library based on libfabric
- Supports various RDMA implementations such as *Infiniband/RoCE*
- Implements a publisher/subscriber paradigm
- Uses *RDMA Send* messages for all its communication
- It is the network interface layer of FELIX ATLAS data acquisition in LHC Run 3 (2022)



[1] *Event-driven RDMA network communication in the ATLAS DAQ system with NetIO*, Jorn Schumacher (CERN), CHEP 2019, https://indico.cern.ch/event/773049/contributions/3473244/

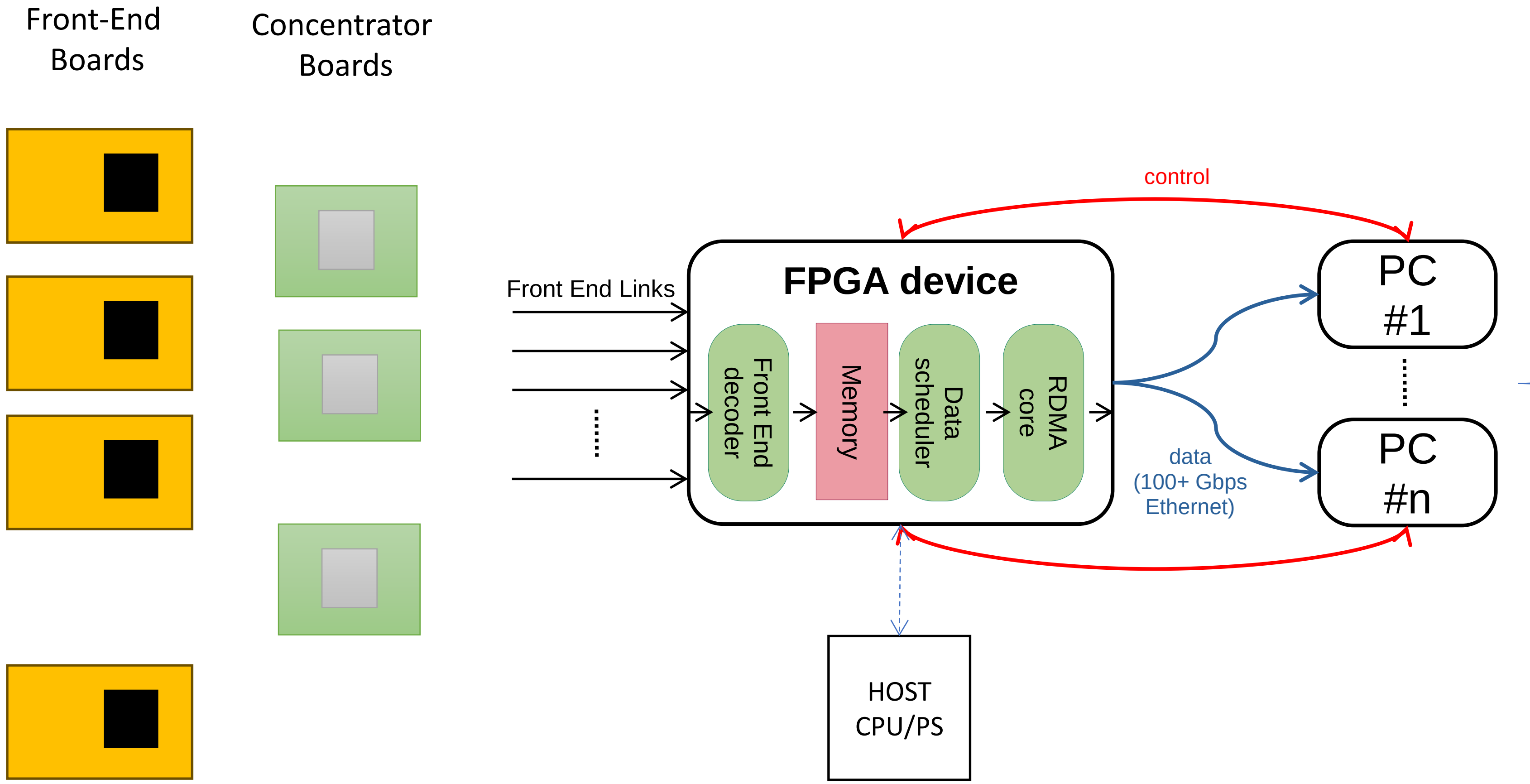
# NetIO

- CERN-developed communication library:
  - based on *libfabric*
  - supports various RDMA implementations:
    - Infiniband
    - RoCE
    - OmniPath
  - publisher/subscriber paradigm
  - RDMA Send messages for:
    - control plane
    - data plane
- network interface layer of FELIX
- ATLAS data acquisition in LHC Run 3 (2022)



*Event-driven RDMA network communication in the ATLAS DAQ system with NetIO*, Jorn Schumacher (CERN), CHEP 2019
https://indico.cern.ch/event/773049/contributions/3473244/

# FPGA-RDMA DAQ Implementation

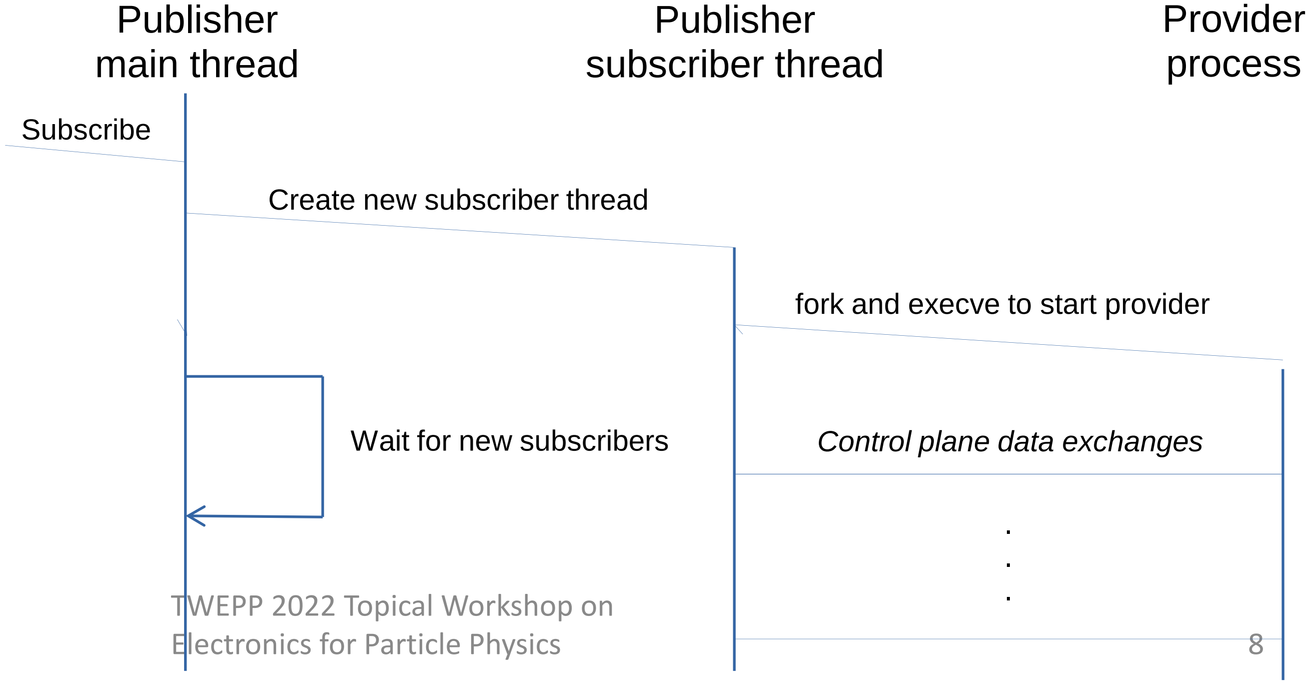


- DAQ PCs
- Storage Nodes
- On-line Monitoring PCs
- Data Quality Node
- …

# New software features

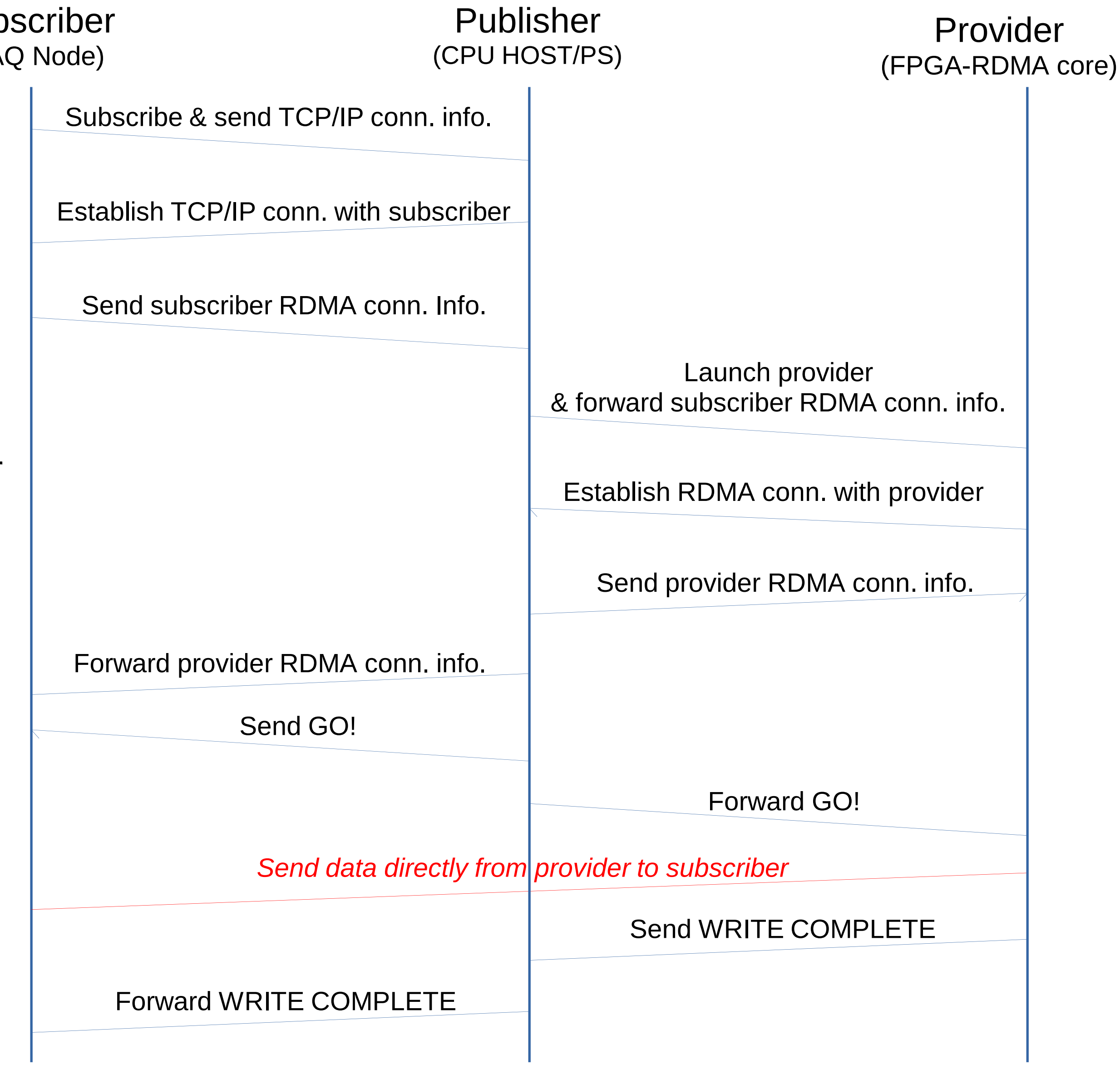
- Full 3-way handshake needed before any data can be moved between any 2 given endpoints
- TCP/IP connections between:
    - **subscriber & publisher**
    - **provider & publisher**
- in order to provide a way for the **subscriber & provider** to perform the 3-way handshake
- multi-threaded solution: a **thread** for each **subscriber**

## subscriber & provider management:



Publisher main thread
Publisher subscriber thread
Provider process
Subscribe
Create new subscriber thread
fork and execve to start provider
Wait for new subscribers
*Control plane data exchanges*
.
.
.

## connection establishment with full handshake:



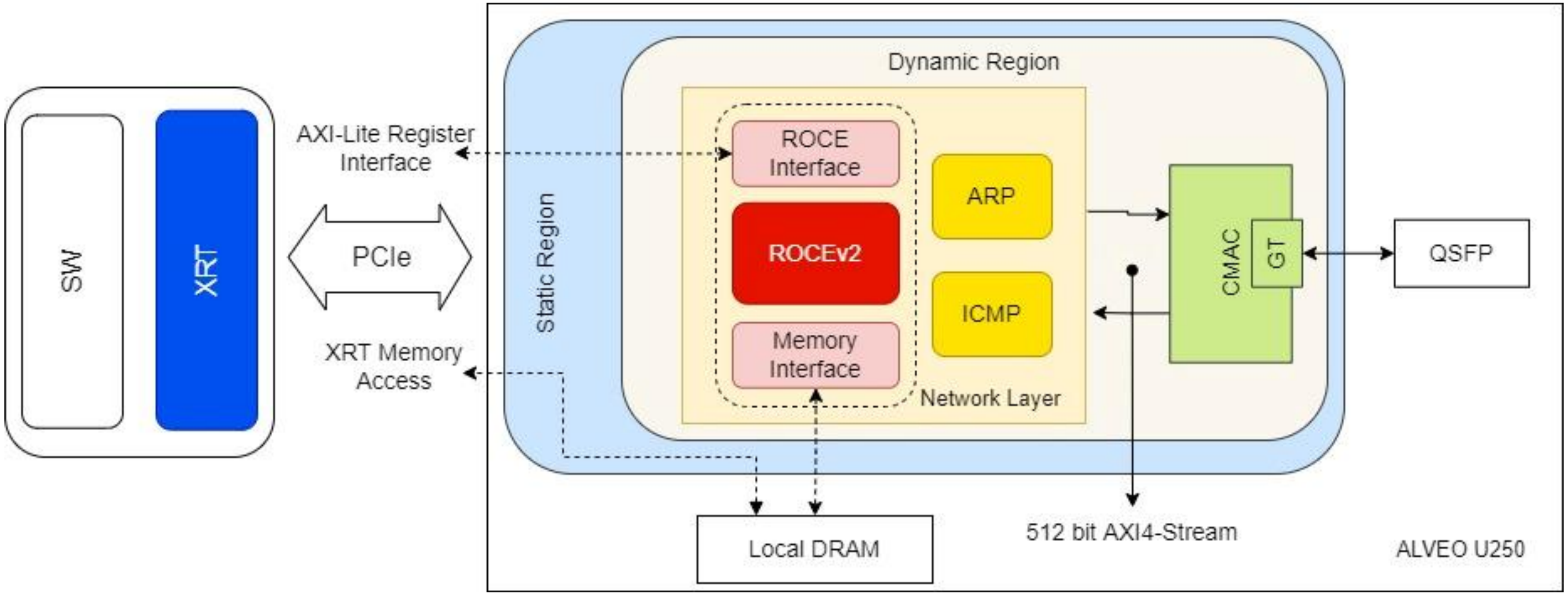
Subscriber (DAQ Node)
Publisher (CPU HOST/PS)
Provider (FPGA-RDMA core)
Subscribe & send TCP/IP conn. info.
Establish TCP/IP conn. with subscriber
Send subscriber RDMA conn. Info.
Launch provider & forward subscriber RDMA conn. info.
Establish RDMA conn. with provider
Send provider RDMA conn. info.
Forward provider RDMA conn. info.
Send GO!
Forward GO!
*Send data directly from provider to subscriber*
Send WRITE COMPLETE
Forward WRITE COMPLETE

# Existing RDMA implementation (hardware)

- Starting point:
  - **FPGA Network Stack**
  - Developed by **ETH Zurich**
  - Fully written in **HLS** (High-Level Synthesis)
  - Contains modules required for setting up network communication:
    - ARP
    - ICMP
    - UDP/IP
    - TCP/IP
    - ROCEv2 (RDMA over Converged Ethernet)
  - Available as **Open Source** on GitHub
    - https://github.com/fpgasystems/fpga-network-stack
    - https://github.com/fpgasystems/coyote

- RDMA implementation uses is **RoCEv2 (RDMA over Converged Ethernet, version 2)**
  - allows the Infiniband-based RDMA to work over any routed IP-based network
- RoCEv2 core contains support for:
  - RDMA Write
  - RDMA Read
  - Retransmission
- Functionality added:
  - RDMA Send
  - Updated retransmission
  - Invariant CRC calculation
  - Connection management (work in progress)

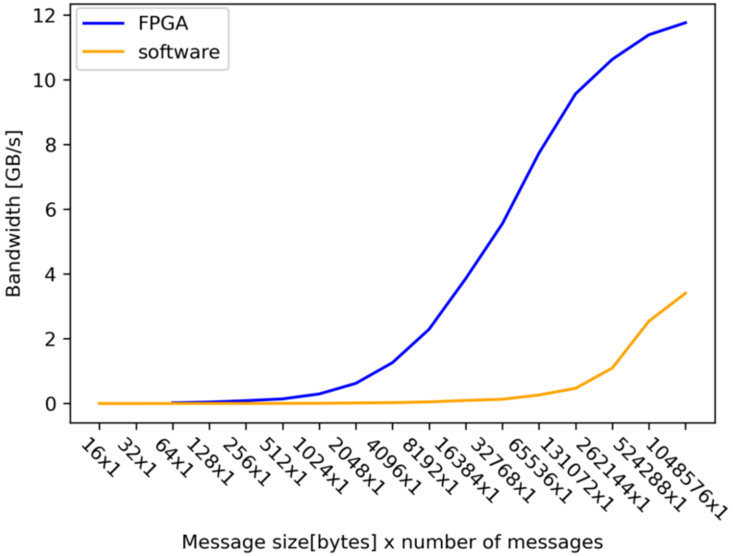
# Alveo Test System: Resource Usage



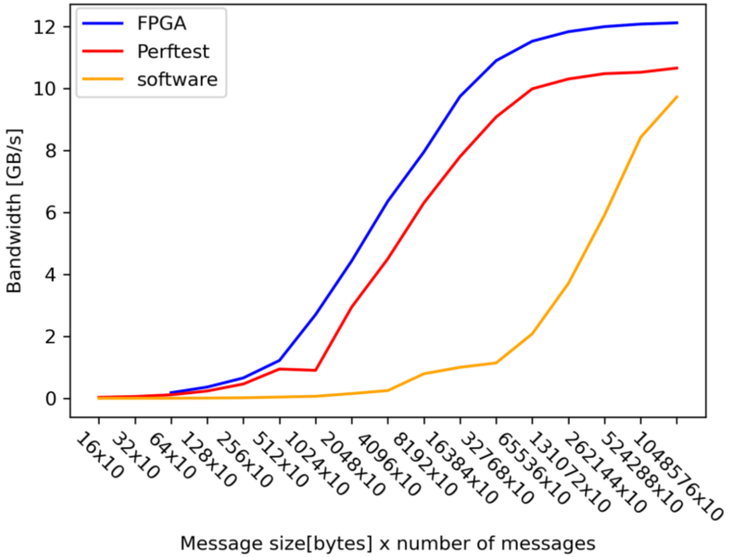
| | **CLB LUTs** | | **CLB Registers** | | **Block RAM Tile** | | **URAM** | |
|---|---|---|---|---|---|---|---|---|
| *ROCEv2 IP* | 37224 | 2.16% | 78435 | 2.27% | 45 | 1.67% | 0 | 0.00% |
| *ROCE Infrastructure* | 22836 | 1.32% | 17593 | 0.51% | 16 | 0.60% | 18 | 1.41% |
| *Network Layer (ARP, ICMP,…)* | 15762 | 0.91% | 31705 | 0.92% | 3.5 | 0.13% | 0 | 0.00% |
| *CMAC* | 13876 | 0.80% | 41244 | 1.19% | 22 | 0.82% | 0 | 0.00% |
| *TOTAL* | 89698 | 5.20% | 168977 | 4.89% | 86.5 | 3.22% | 18 | 1.41% |
| *Alveo U250 Total* | 1726216 | | 3456000 | | 2688 | | 1280 | |

# RDMA implementation performance (1)

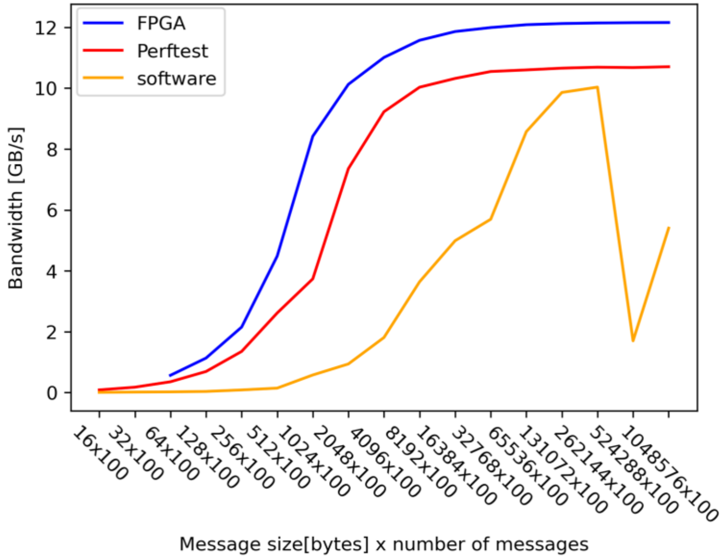
1 message / RDMA Write



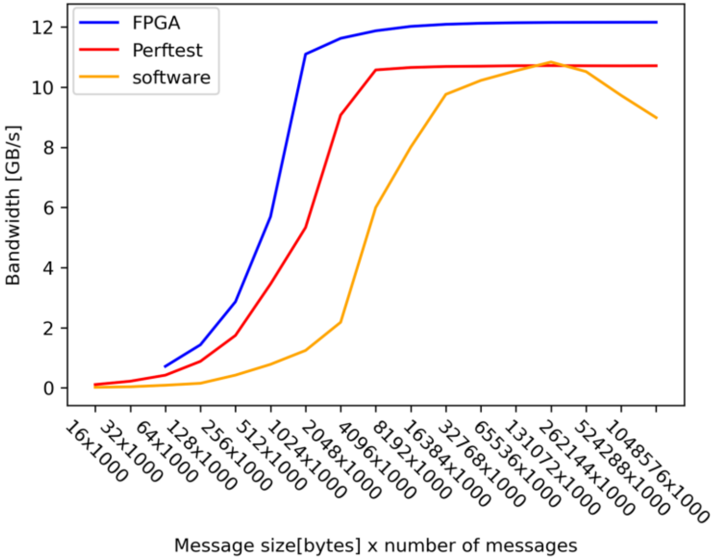
10 messages / RDMA Write



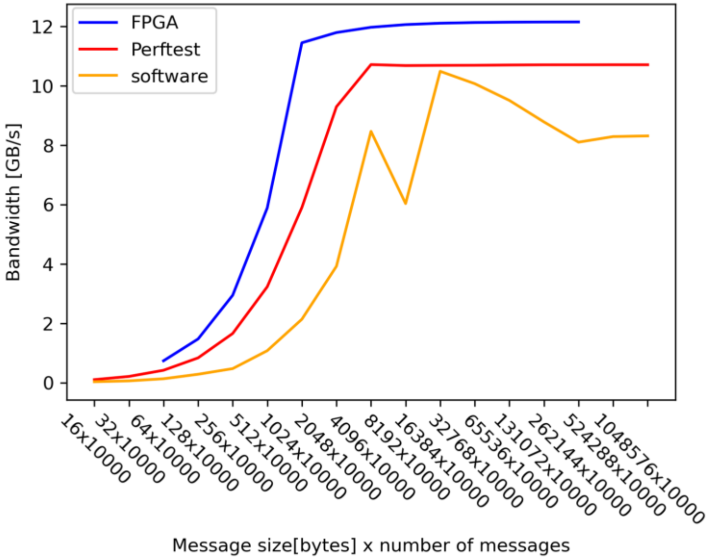
100 messages / RDMA Write



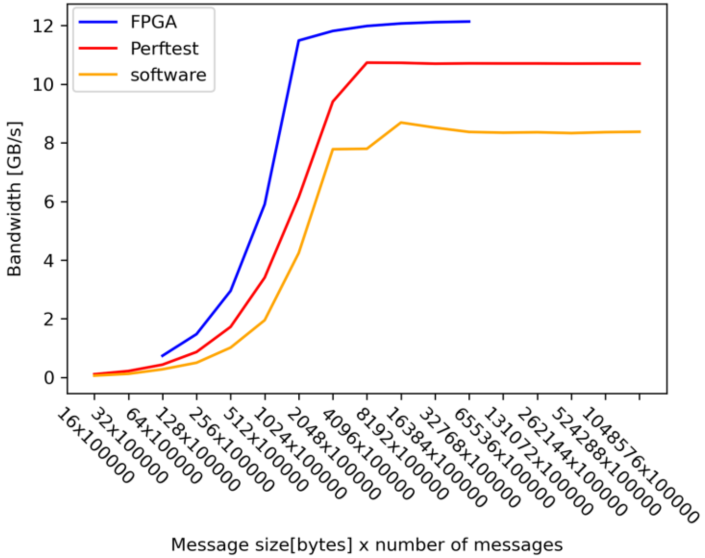
1k message / RDMA Write



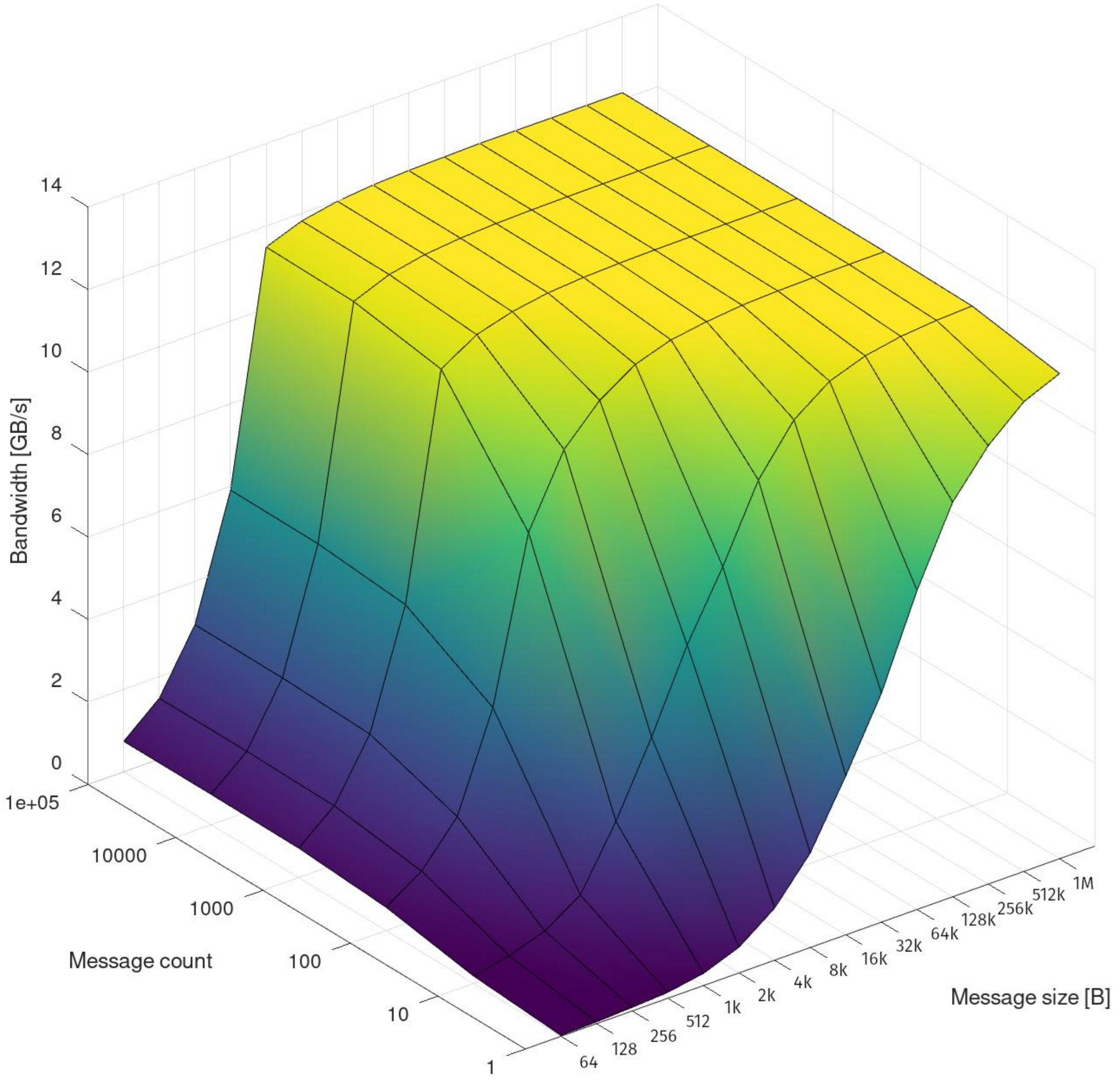
10k messages / RDMA Write

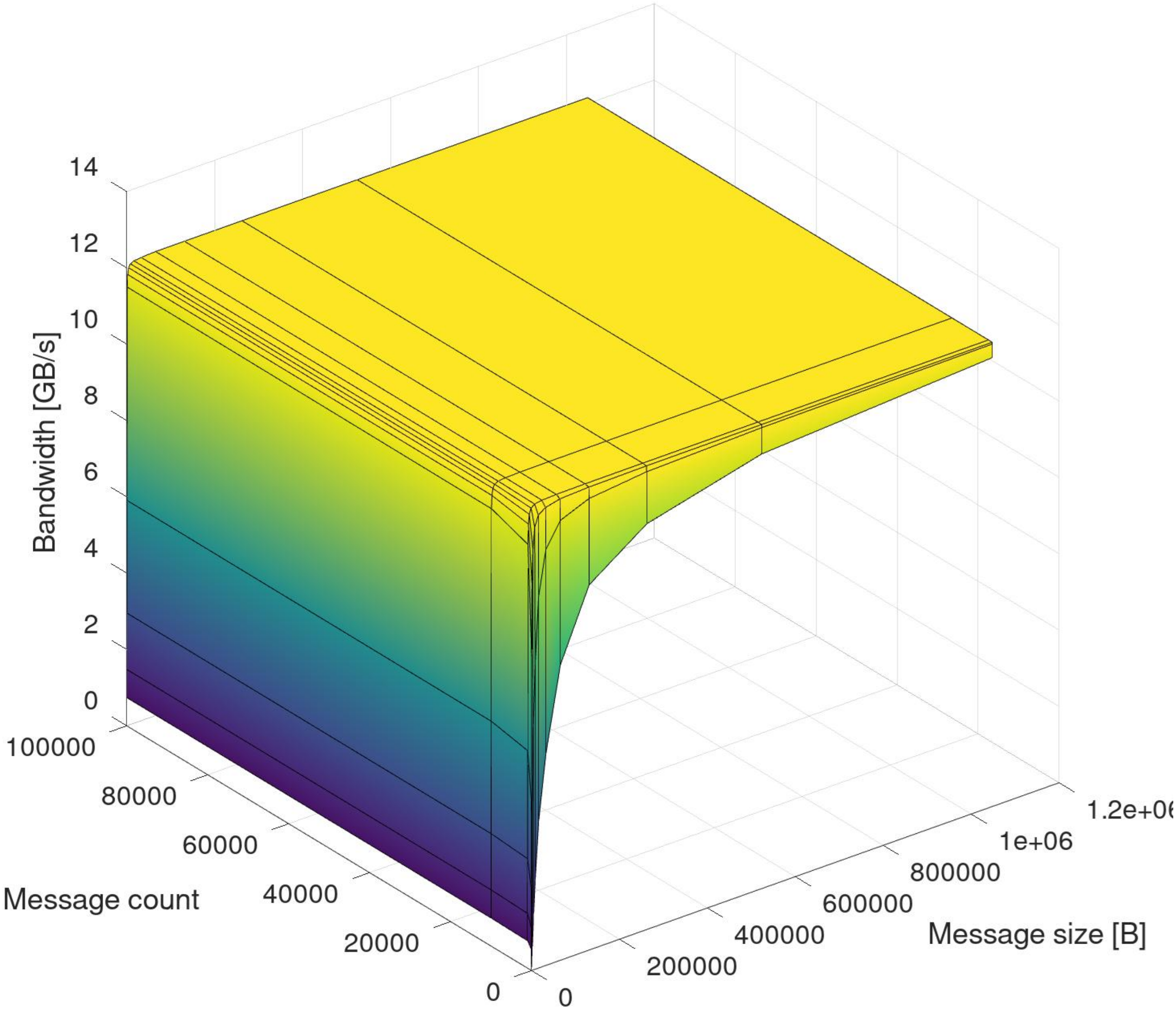


100k messages / RDMA Write

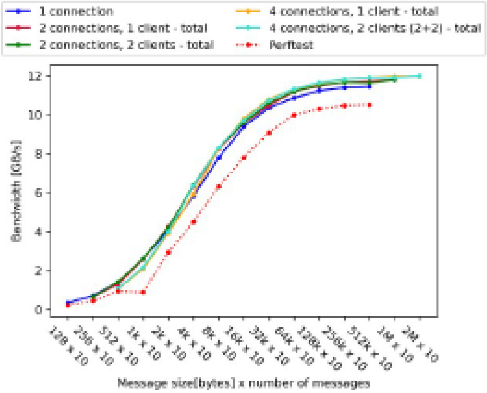


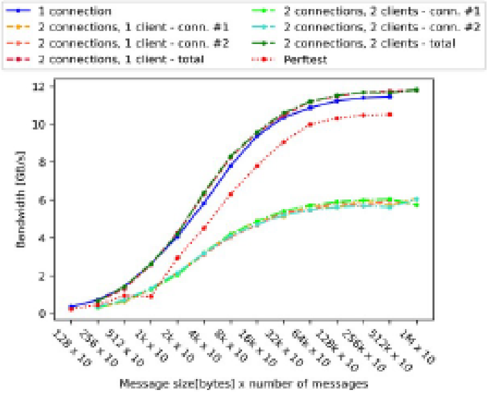
# RDMA implementation performance (2)

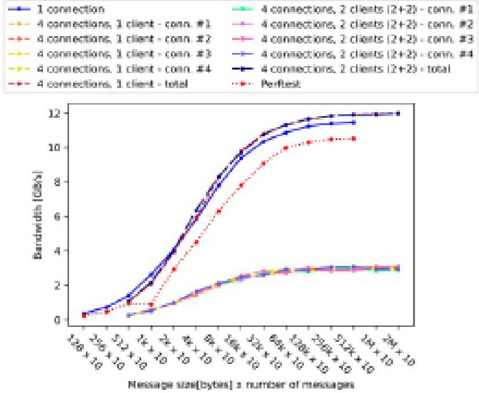


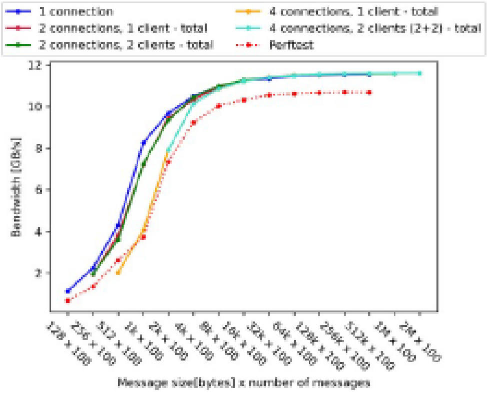


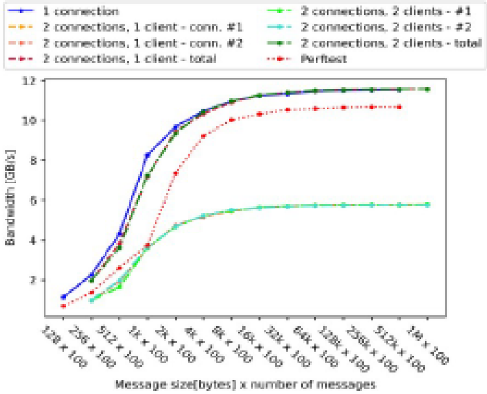
# RDMA implementation performance (3) – single burst, multiple clients

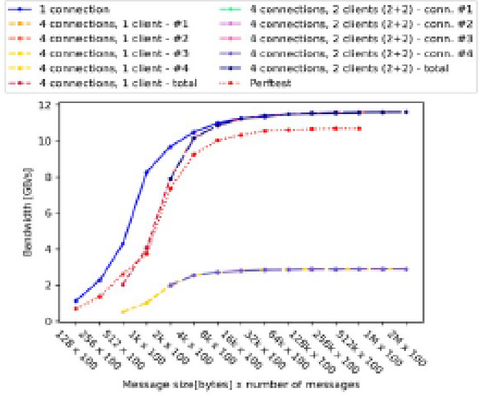


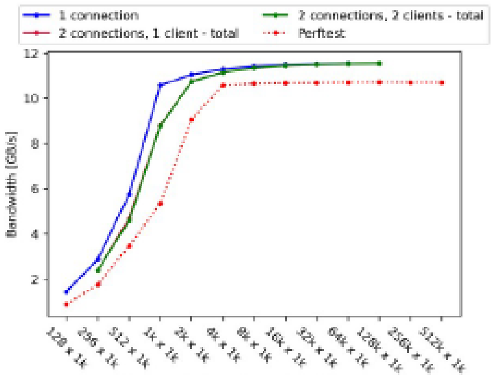


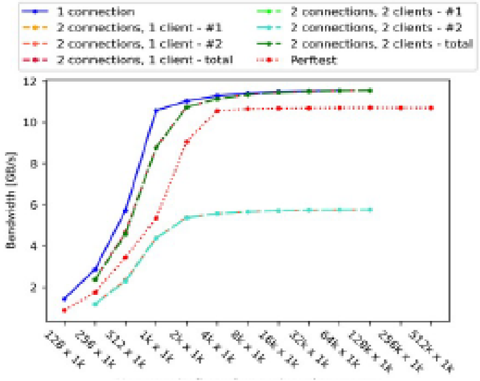










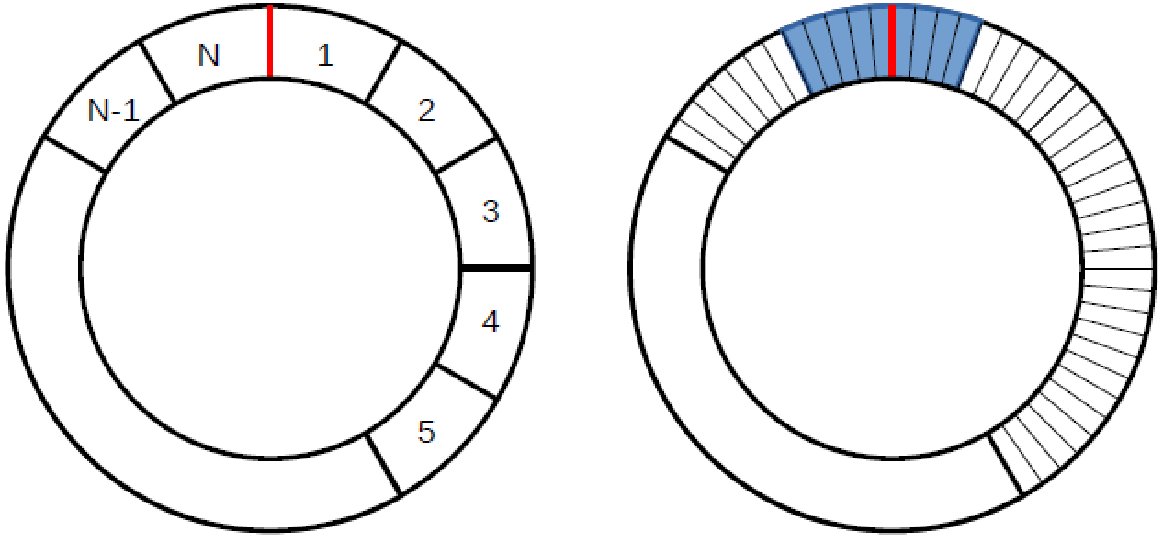


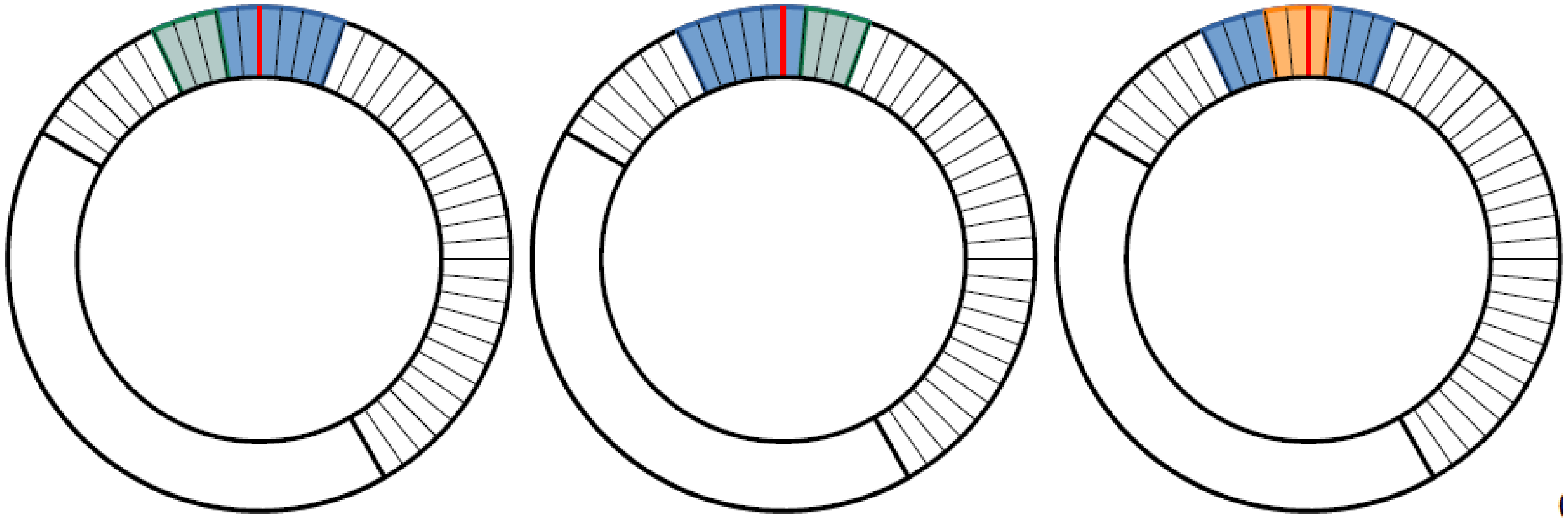
- the theoretical maximum bandwidth of the used links is **100Gb/s** (i.e. **12.5GB/s**)
- a software implementation, both ours and what can be measured with *Perftest*, can reach up to **10.5GB/s**
- our hardware implementation has been measured to reach up to **11.54GB/s** with a single connection and up to **11.98GB/s** total with multiple connections

- **10** msg. count, tested with:
  - two connections distributed as:
    - both conn. on the same client
    - two clients, each with a single conn.
  - four connections distributed as:
    - all four conn. on the same client
    - two clients, each with two conn.
  - at **link saturation**, the total send bandwidth is **higher** than the single connection setup by:
    - **2.41% for 2 connections, 1 client**
    - **1.76% for 2 connections, 2 clients**
    - **4.05% for 4 connections, 1 client**
    - **4.06% for 4 connections, 2 clients**
- **100** msg. count, tested with:
  - two connections distributed as:
    - both conn. on the same client
    - two clients, each with a single conn.
  - four connections distributed as:
    - all four conn. on the same client
    - two clients, each with two conn.
  - at **link saturation**, the total send bandwidth is higher than the single connection setup by:
    - **0.23% for 2 connections, 1 client**
    - **0.22% for 2 connections, 2 clients**
    - **0.34% for 4 connections, 1 client**
    - **0.34% for 4 connections, 2 clients**
- **1000** msg. count, tested with:
  - two connections distributed as:
    - both conn. on the same client
    - two clients, each with a single conn.
  - all four connections test setups are currently overloading the resources of the FPGA RDMA core implementation
  - at **link saturation**, the total send bandwidth is **lower** than the single connection setup by:
    - **0.31% for 2 connections, 1 client**
    - **0.31% for 2 connections, 2 clients**

# Hardware Sender Streaming

- when sending bursts (single or streaming):
  - the circular buffer capacity was measured in burst
  - no burst could have data on both sides of the buffer capacity limit
- when sending using full streaming, *data sent* notifications are sent at a fixed time interval (0.1s)
  - consequently, the number of messages sent is not always the same
    - received data could end up on both sides of the buffer capacity limit
- with a **single worker**, it is enough to take into account where the limit is within the received data
- with **multiple workers**, it is also necessary to take into account **which worker** has its data on both sides of the limit

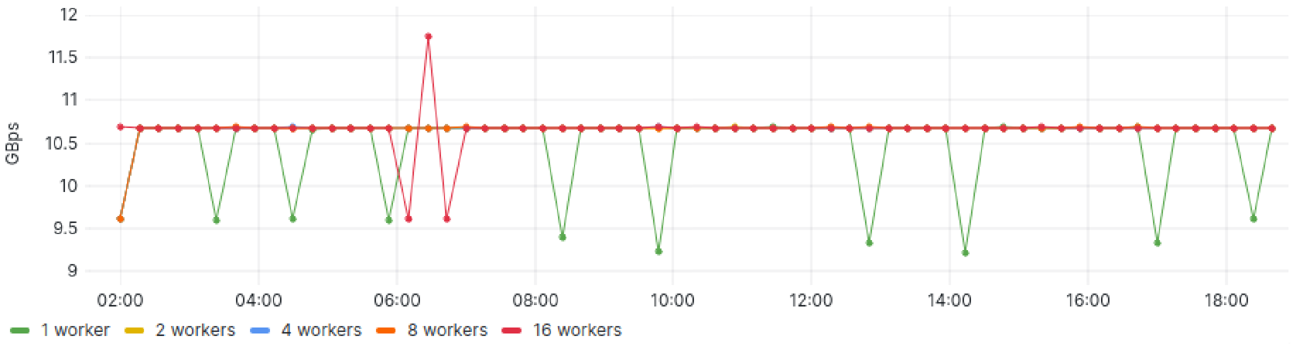




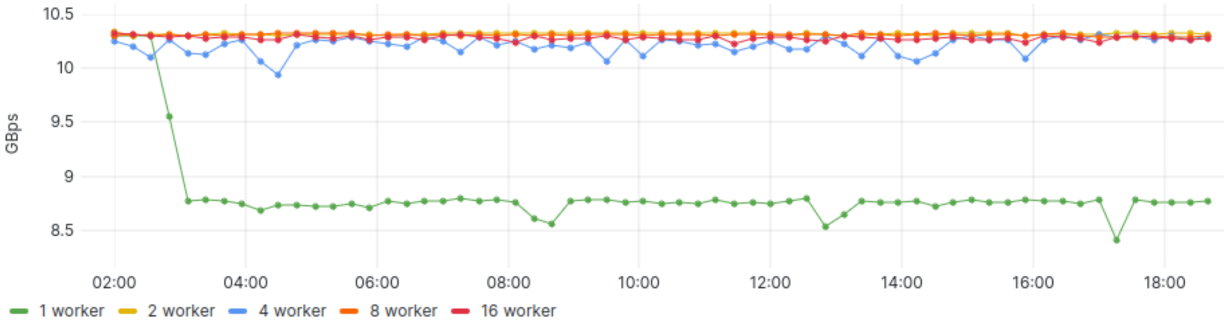
# Multi-worker receiver testing – 1 receiver

- receiver running on fast PC

receiver (hw sender) - uv

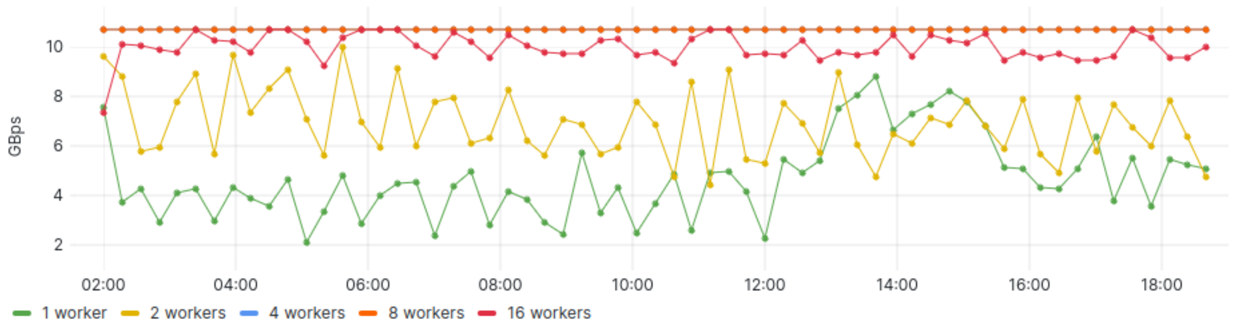


receiver (sw sender) - uv

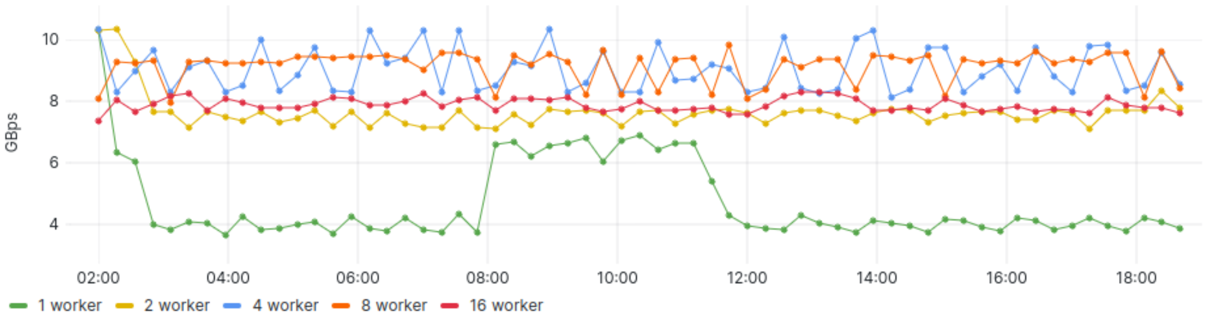


- receiver running on regular PC

receiver (hw sender) - uw

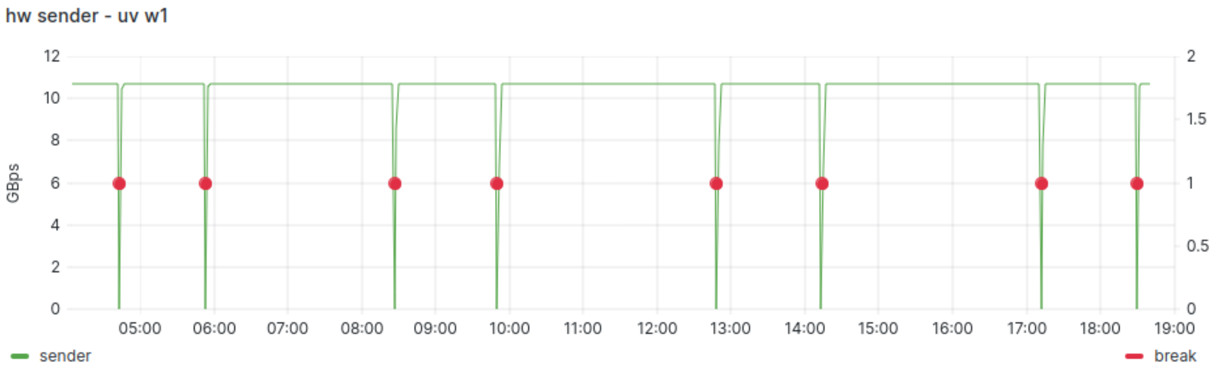


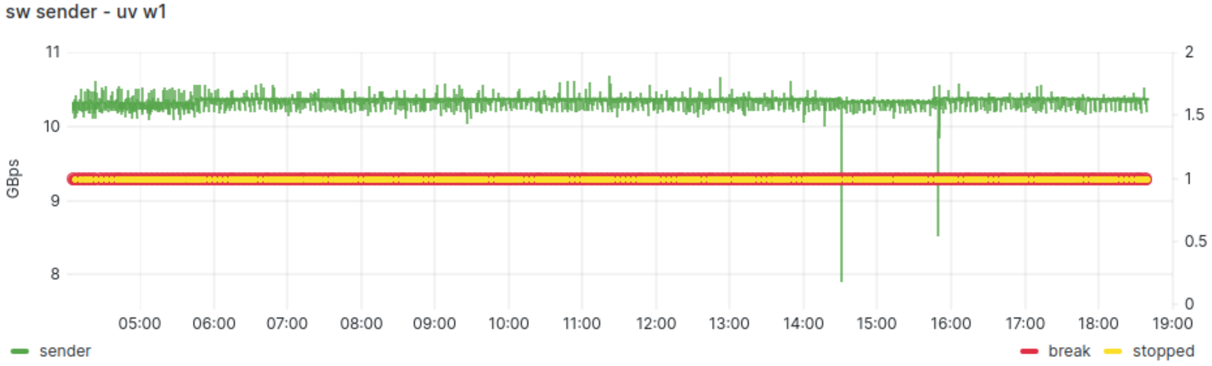
receiver (sw sender) - uw



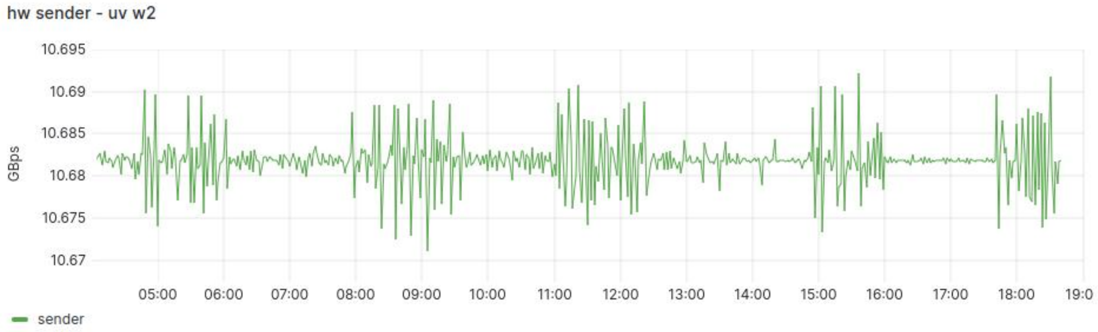
# Hardware sender streaming testing – 1 receiver (1)

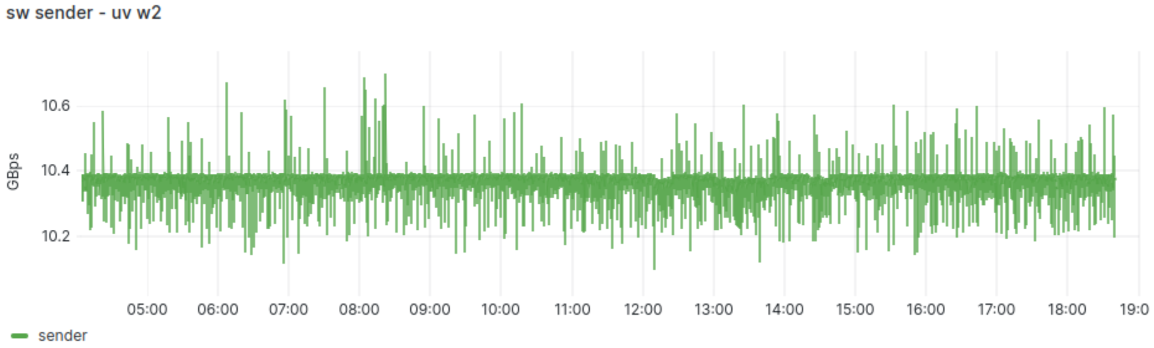
- receiver running on fast PC, 1 worker


hw sender - uv w1


sw sender - uv w1

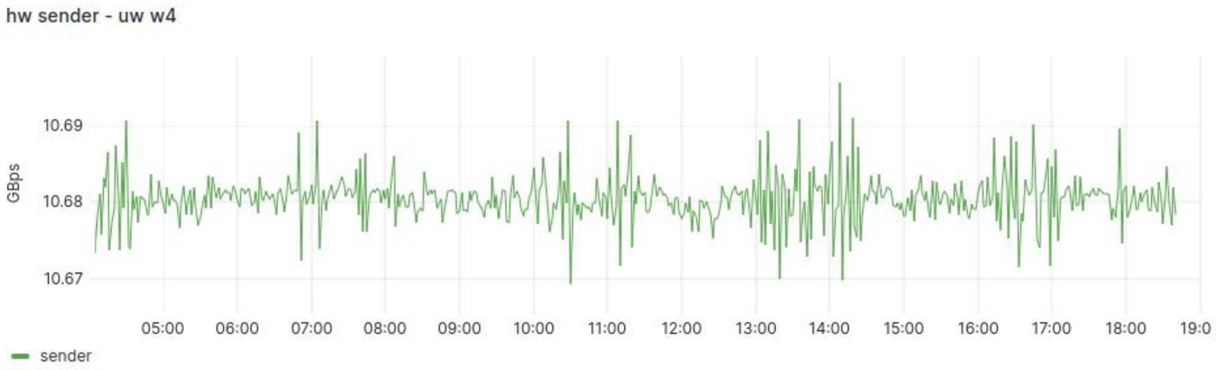
- receiver running on fast PC, 2 workers

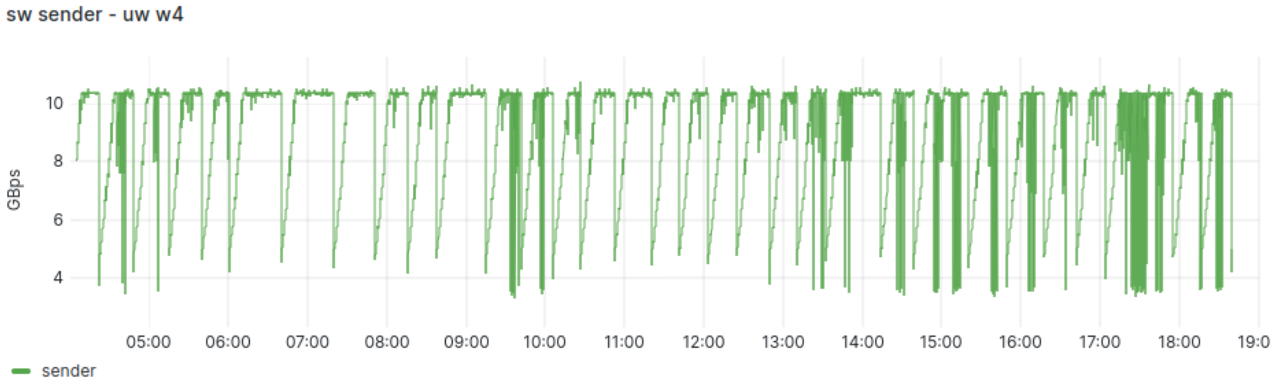

hw sender - uv w2


sw sender - uv w2

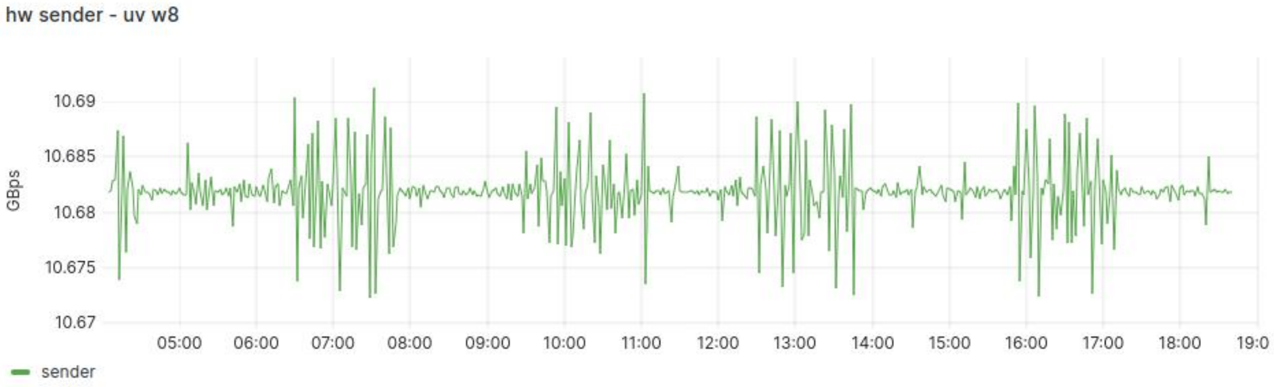
# Hardware sender streaming testing – 1 receiver (2)

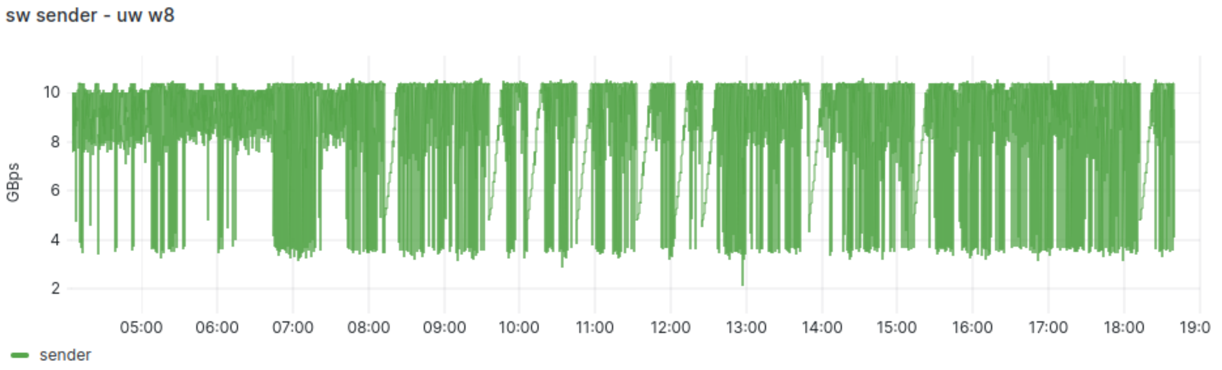
- receiver running on regular PC, 4 worker





- receiver running on regular PC, 8 workers

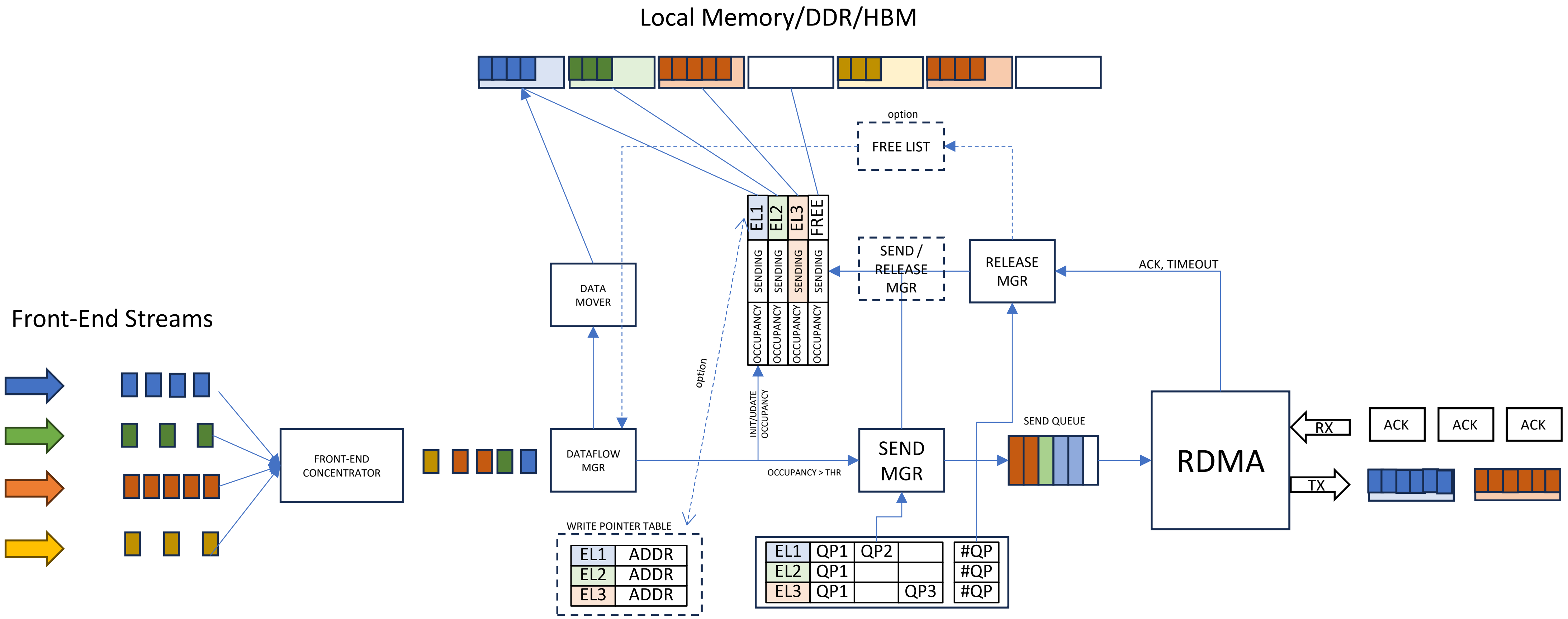




# Hardware sender streaming testing conclusions

- The hardware and software senders’ bandwidth are almost identical
- The hardware implementation offers more stable bandwidth across all tests:
  - Hardware sender bandwidth measured at sender (runs with backpressure excluded):
    - Average: 10.653 GBps
    - Standard deviation: 0.119
  - Hardware sender bandwidth measured at receiver (runs with backpressure excluded):
    - Average: 10.549 GBps
    - Standard deviation: 0.142
  - Software sender bandwidth measured at sender (runs with backpressure excluded):
    - Average: 10.351 GBps
    - Standard deviation: 0.784
  - Software sender bandwidth measured at receiver (runs with backpressure excluded):
    - Average: 10.080 GBps
    - Standard deviation: 0.840

# DAQ Streaming Implementation



Local Memory/DDR/HBM

Front-End Streams

FRONT-END CONCENTRATOR

DATAFLOW MGR

DATA MOVER

option

FREE LIST

| EL1 | EL2 | EL3 | FREE |
|---|---|---|---|
| SENDING | SENDING | SENDING | SENDING |
| OCCUPANCY | OCCUPANCY | OCCUPANCY | OCCUPANCY |

INIT/UDATE OCCUPANCY

option

SEND / RELEASE MGR

RELEASE MGR

ACK, TIMEOUT

OCCUPANCY > THR

SEND MGR

SEND QUEUE

RDMA

RX

TX

ACK

ACK

ACK

WRITE POINTER TABLE

| | |
|---|---|
| EL1 | ADDR |
| EL2 | ADDR |
| EL3 | ADDR |

| | | | | |
|---|---|---|---|---|
| EL1 | QP1 | QP2 | | #QP |
| EL2 | QP1 | | | #QP |
| EL3 | QP1 | | QP3 | #QP |

# Conclusions

- Development of FPGA-RDMA DAQ implementation is well advanced
- RDMA transport over Ethernet Fabric with hardware FPGA RDMA core validated with burst or streaming
- R&D on optimal software RDMA DAQ receiver ongoing
- Front-end DAQ streaming scheme on FPGA is under development
- Integration in SRS eFEC is next

# Further Resources

- FPGA implementation of RDMA for ATLAS Readout with FELIX in High Luminosity LHC (TWEPP 2021)
- Integration of FPGA RDMA into the ATLAS Readout with FELIX in High Luminosity LHC (TWEPP 2022)
- Performance profiling and design choices of an RDMA implementation using FPGA devices (TWEPP 2023)
- Improvements for the implementation of RDMA on FPGA devices (TWEPP 2024)