# Improvements for the implementation of RDMA on FPGA devices

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

**TWEPP 2024 Topical Workshop on Electronics for Particle Physics**

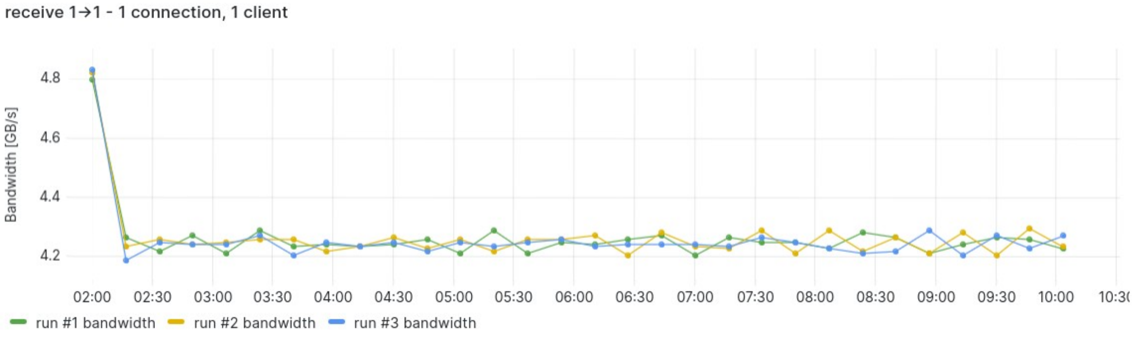
# Previous work

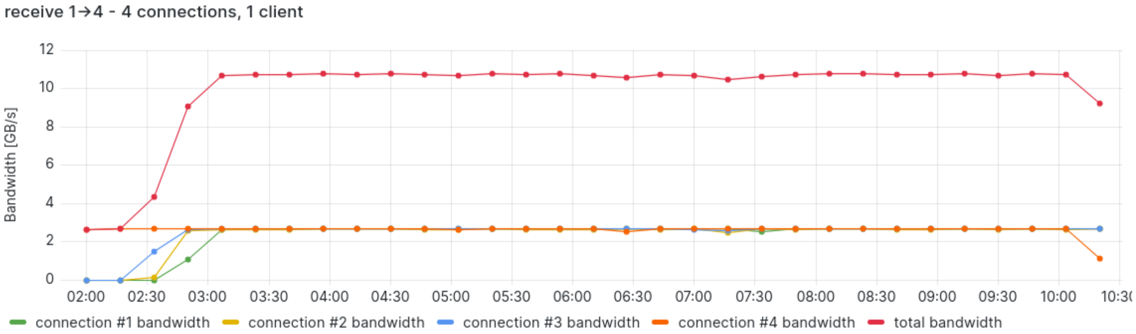
- The features already implemented at the beginning of this work[1]:
  - **Software sender streaming** to one or more **single-worker receivers**
  - **Single burst hardware sender** to one or more single-worker receivers
- Identified issues:
  - A single thread (single-worker) receiver has difficulties receiving all the data sent by a sender in real time
  - Support for hardware streaming missing

[1] *Performance profiling and design choices of an RDMA implementation using FPGA devices*, Matei-Eugen Vasile (IFIN-HH), TWEPP 2023, https://indico.cern.ch/event/1255624/contributions/5445303/

# Multi-worker receiver (1)

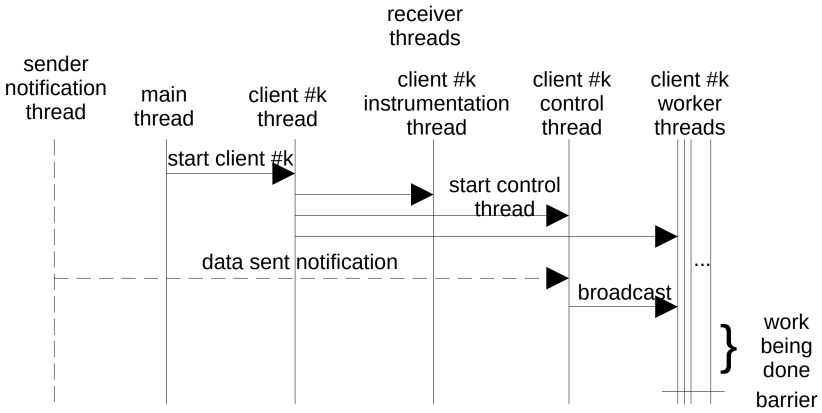
- In previous testing:
  - Software sender in streaming mode
  - Single client running streaming receiver
- The receiver could sustain only an average of about **4.2 GBps**
  - The sender was able to sustain about **10.5 GBps**
- If using a setup with a single sender and multiple receivers, even if all receivers were running on the same device:
  - The link bandwidth was being split among all receivers
  - The client device running all receivers had no problem sustaining the full sender bandwidth of about 10.5 GBps




- What would happen if the receiver, instead of receiving the data on a single thread, would use multiple threads (multiple workers) to process the incoming data? Would it behave similar to running multiple receivers on the same device?

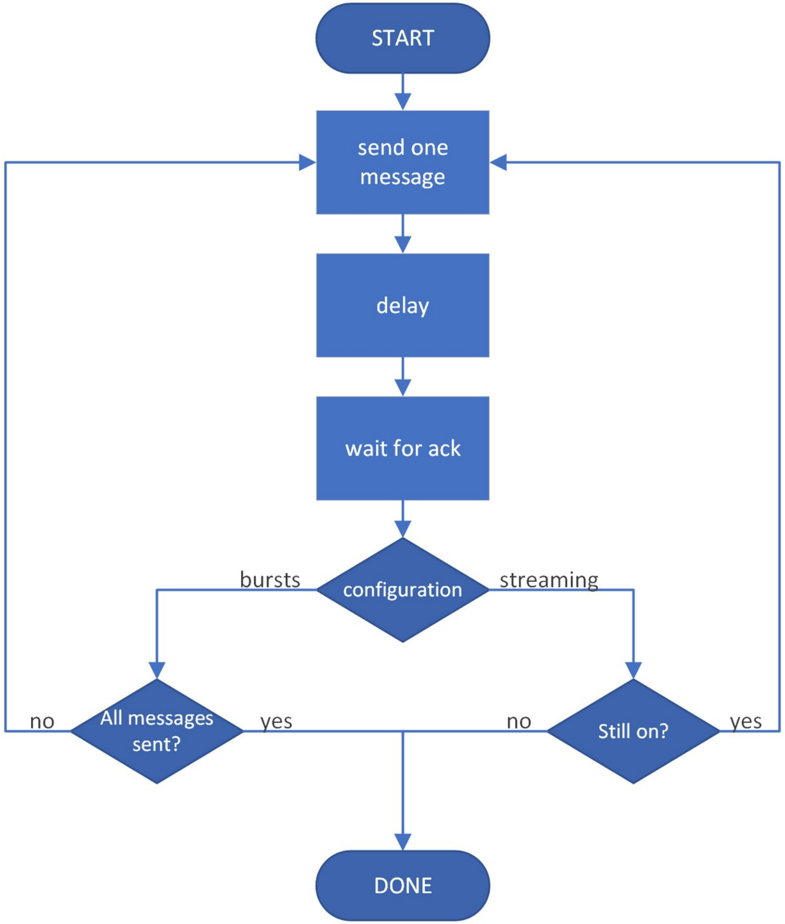
# Multi-worker receiver (2)




- implementation of multi-worker receiver:
  - the receiver starts a client thread
    - the client thread starts:
      - instrumentation thread
      - control thread
      - worker threads (workers)
  - all **workers** are **started at the beginning** of the run and they are **never destroyed** during the run
  - all workers block waiting on a **conditional variable** when there is no data to be processed
    - the conditional variable is unlocked by a **broadcast** emitted by the **control thread**, when a data is received
  - all workers are held at a **barrier** until all of them reach the barrier
  - the first worker manages the consumption of the received data by all the workers
- backpressure:
  - enabled on the control thread (when crossing the upper threshold of buffer occupancy)
  - disabled by the first worker thread (when crossing the lower threshold of buffer occupancy)

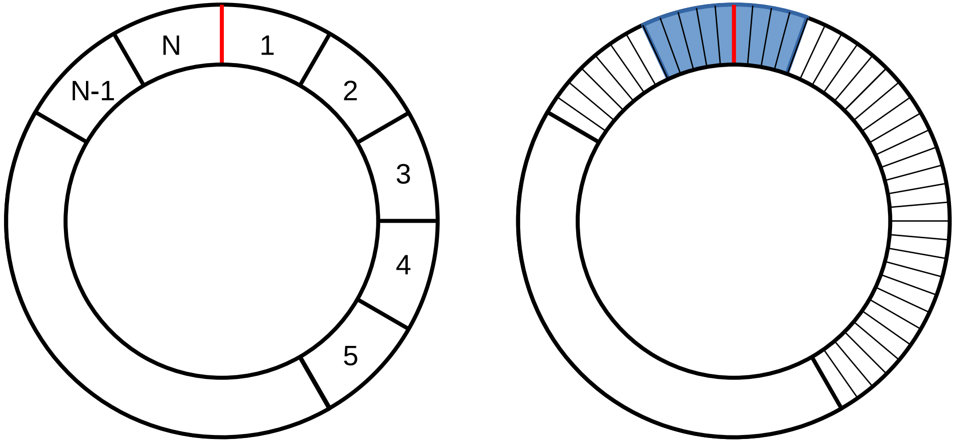
# Hardware sender streaming (1)

- First, implemented a streaming solution directly combining:
  - the hardware burst-based sender (message size * message count bursts)
  - the software streaming sender
    - which in turn is sending a stream of bursts
- This solution had two problems:
  - was not optimal from the point of view of the hardware design implementation
  - could not be instrumented correctly in order to measure the sender bandwidth
- Second hardware streaming solution (full streaming):
  - messages are streamed continuously (only message size counts)
  - the sender software is much simplified
    - but it creates complications on the receiver side

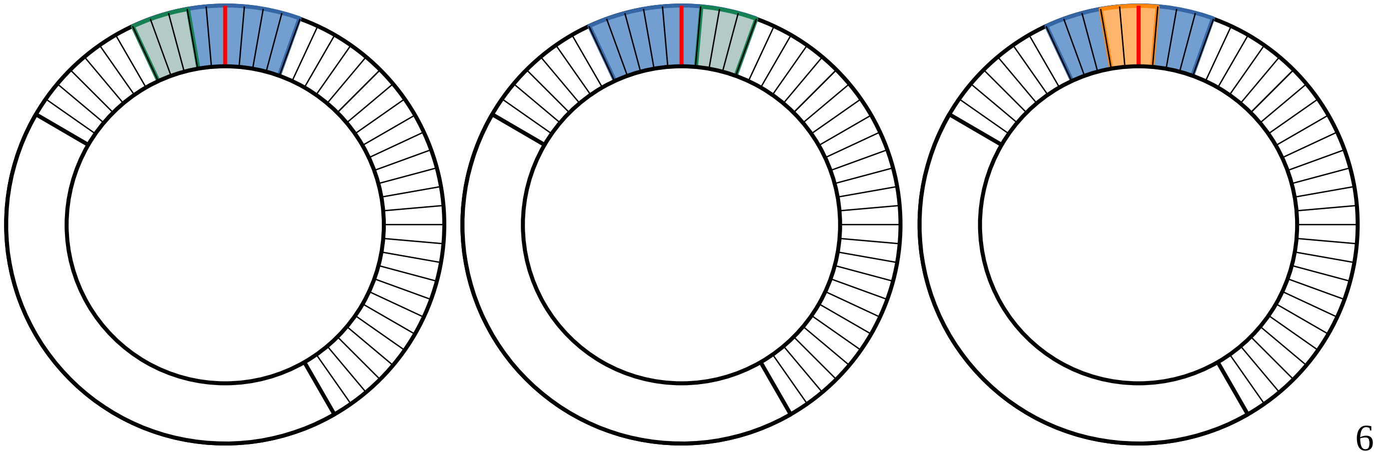


# Hardware sender streaming (2)

- 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

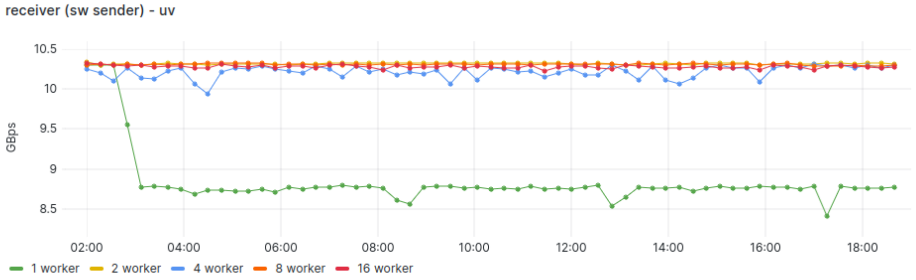


# Multi-worker receiver testing

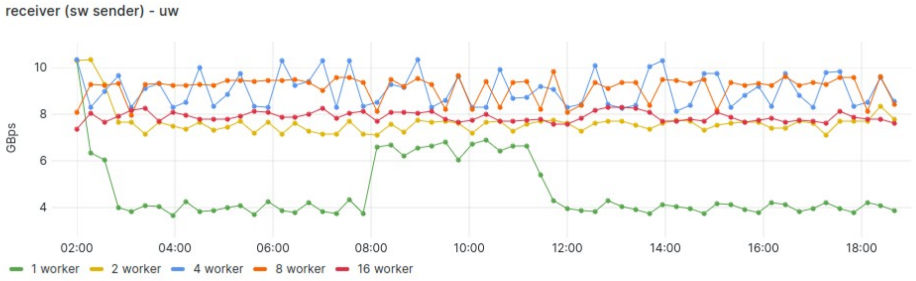
- For testing, 5 devices have been used:
  - 1 sender PC (designated “u”)
    - used both for the software sender and for the hardware sender using the Alveo U50 board installed in it
    - AMD EPYC 9354P, 64 cores, 192 GB DDR5 RAM
  - 4 receiver PCs (designated “v”, “w”, “y” and “z”)
    - device “v” – Intel Xeon Gold 6416H, 144 cores, 128 GB DDR5 RAM – fast PC
    - device “w” – Intel Xeon Silver 4215R, 32 cores, 128 GB DDR4 RAM – regular PC
    - device “y” – Intel Xeon Silver 4214Y, 48 cores, 320 GB DDR4 RAM – regular PC
    - device “z” – Intel Xeon Gold 6234, 32 cores, 320 GB DDR4 RAM – regular PC
- Tests have been run with 1, 2, 4, 8 and 16 workers on each sender/receiver configuration
- The sender/receiver configurations tested are:
  - 1 sender – 1 receiver
    - u→v
    - u→w
  - 1 sender – 2 receivers
    - u→vv (both receivers running on the same, fast, PC)
    - u→ww (both receivers running on the same, regular, PC)
    - u→vw (one receiver running on the fast PC, one running on one of the regular PCs)
  - 1 sender – 4 receivers
    - u→vvvv (all four receivers running on the same, fast, PC)
    - u→wwww (all four receivers running on the same, regular, PC)
    - u→vwyz (each receiver running on a different PC)
- Software sender tests were run with:
  - bursts of 1000 messages of 8192 bytes
  - receiver buffer capacity of 1000
- Hardware sender tests were run with:
  - messages of 8192 bytes
  - receiver buffer capacity of 1000000

# Multi-worker receiver testing – 1 receiver

- receiver running on fast PC

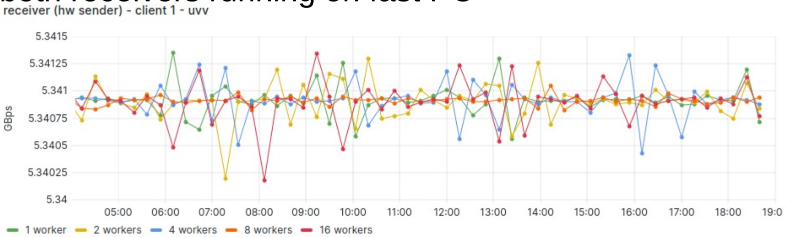




- receiver running on regular PC

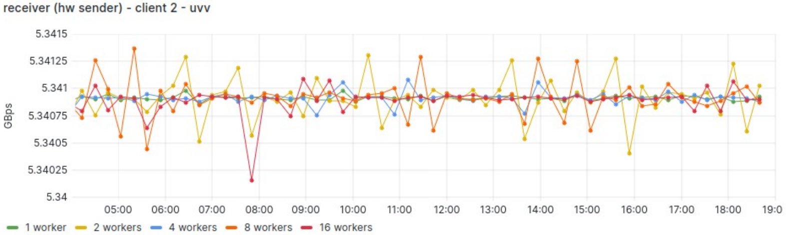



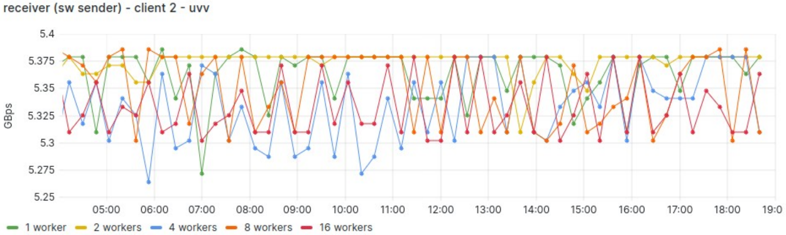
# Multi-worker receiver testing – 2 receivers (1)

- both receivers running on fast PC

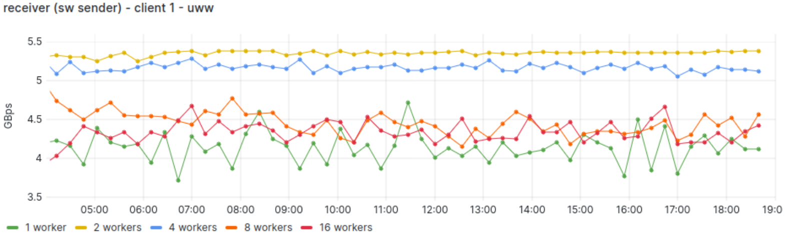




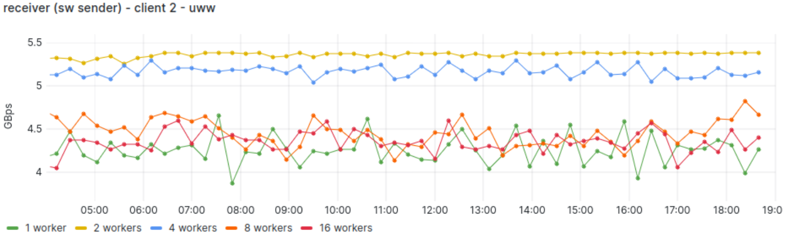



- both receivers running on regular PC

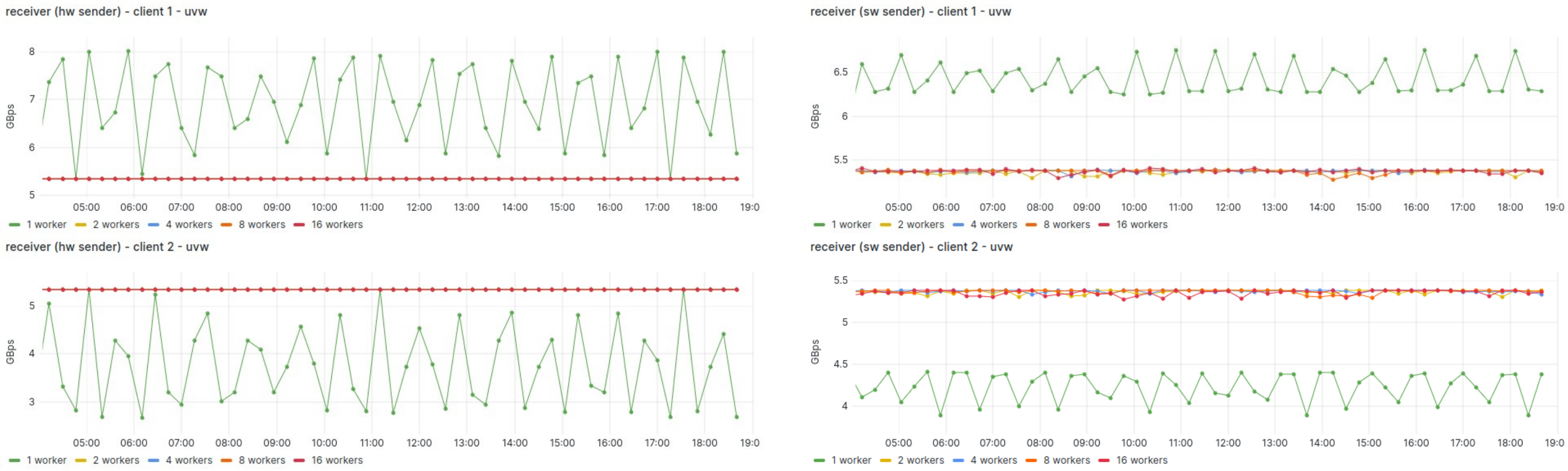







# Multi-worker receiver testing – 2 receivers (2)

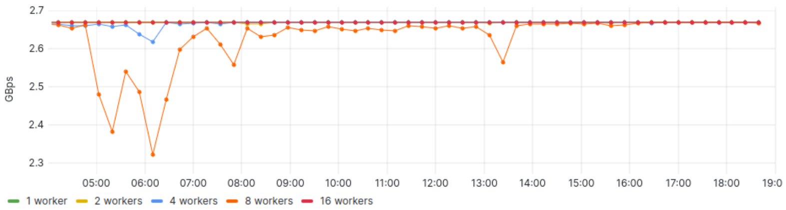
- first receiver running on fast PC, second running on regular PC



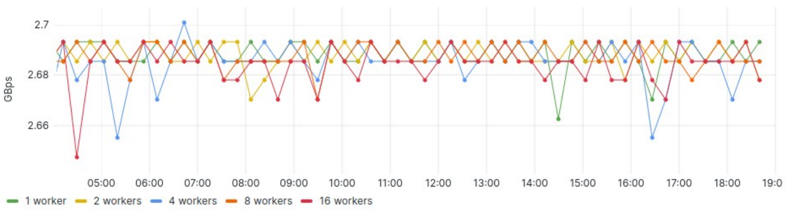
# Multi-worker receiver testing – 4 receivers (1)

- all receivers running on fast PC

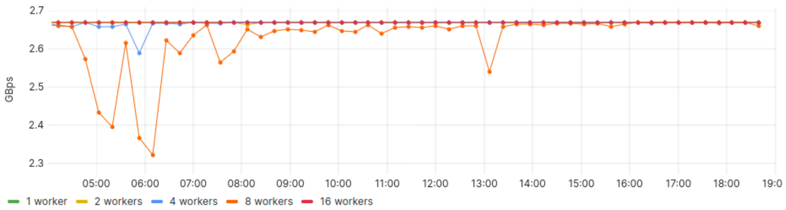
receiver (hw sender) - client 1 - uvvvv



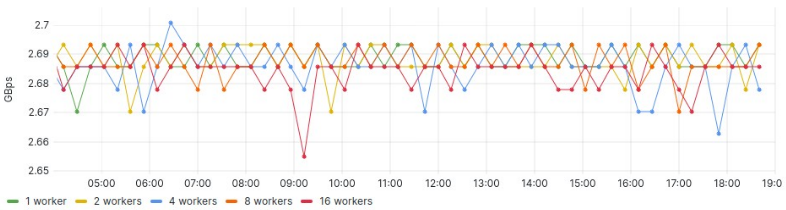

receiver (sw sender) - client 1 - uvvvv



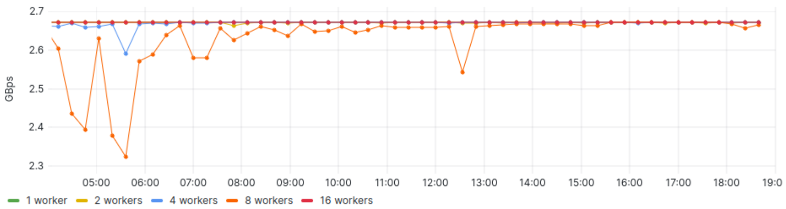

receiver (hw sender) - client 2 - uvvvv



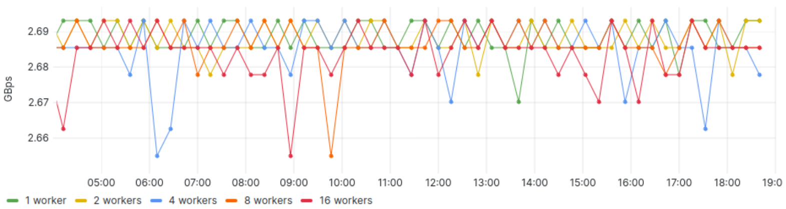

receiver (sw sender) - client 2 - uvvvv



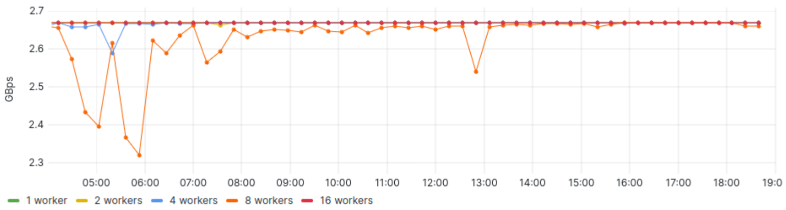

receiver (hw sender) - client 3 - uvvvv



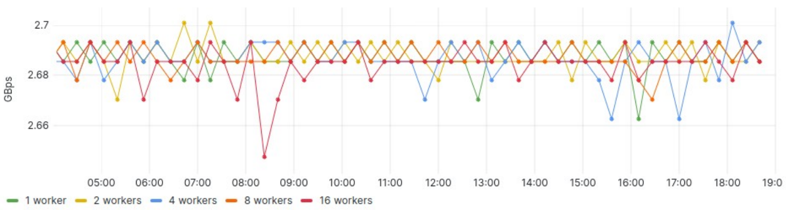

receiver (sw sender) - client 1 - uvvvv




receiver (hw sender) - client 4 - uvvvv




receiver (sw sender) - client 1 - uvvvv

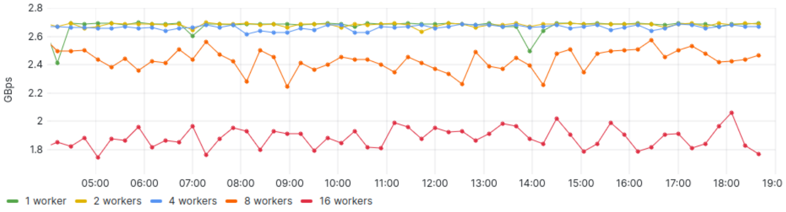


# Multi-worker receiver testing – 4 receivers (2)

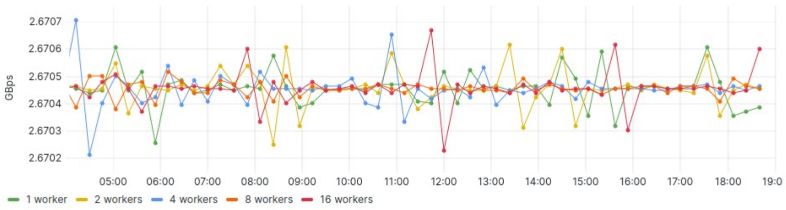
- all receivers running on regular PC

receiver (hw sender) - client 1 - uwwww

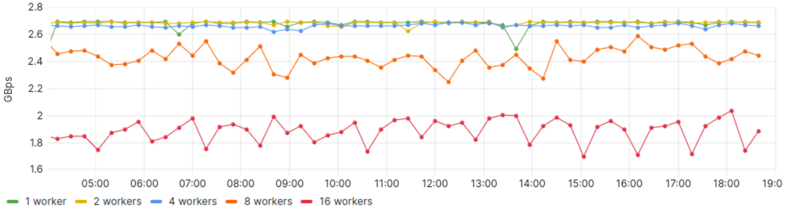

receiver (sw sender) - client 1 - uwwww



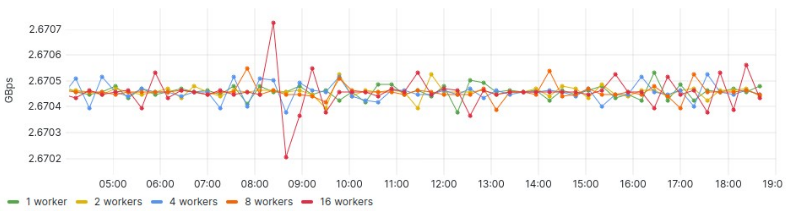

receiver (hw sender) - client 2 - uwwww



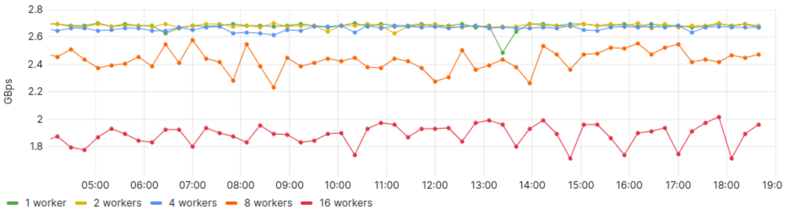

receiver (sw sender) - client 2 - uwwww




receiver (hw sender) - client 3 - uwwww

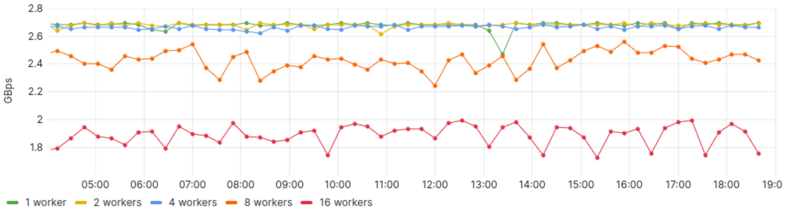



receiver (sw sender) - client 1 - uwwww




receiver (hw sender) - client 4 - uwwww

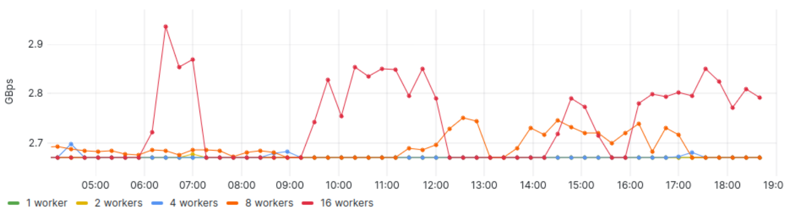

receiver (sw sender) - client 1 - uwwww



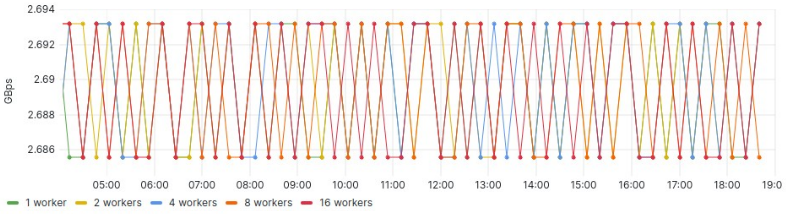
# Multi-worker receiver testing – 4 receivers (3)

- each receiver running on a different PC

receiver (hw sender) - client 1 - uvwyz

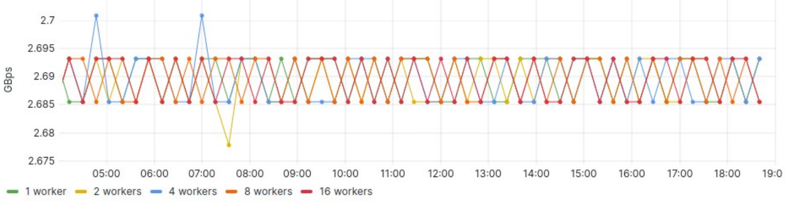



receiver (sw sender) - client 1 - uvwyz

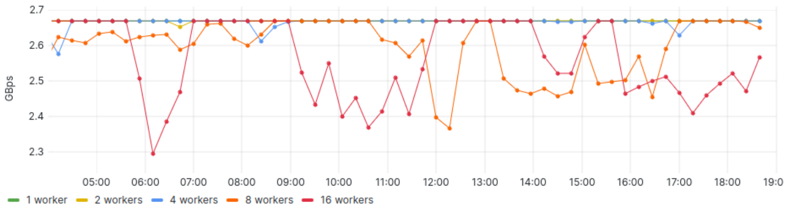



receiver (hw sender) - client 2 - uvwyz

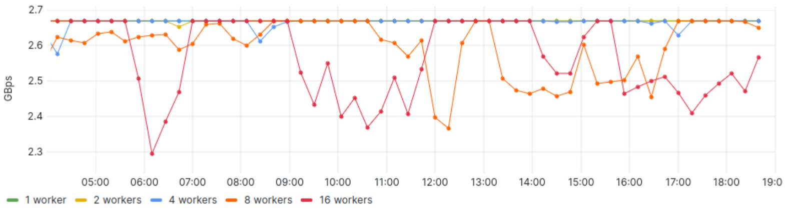

receiver (sw sender) - client 2 - uvwyz




receiver (hw sender) - client 3 - uvwyz




receiver (hw sender) - client 3 - uvwyz




receiver (hw sender) - client 4 - uvwyz


receiver (hw sender) - client 4 - uvwyz

# Multi-worker receiver testing conclusions

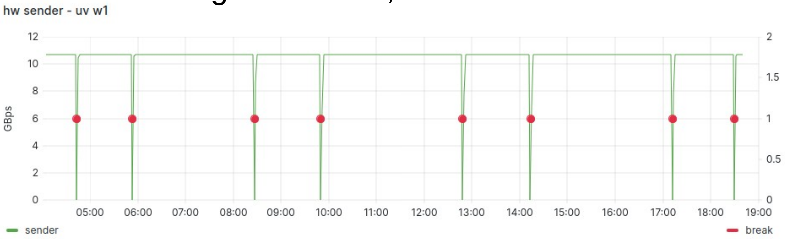
- The multi-threaded receiver implementation improves the performance of the receiver
- The performance depends greatly on the performance of the device running the receiver
- 2 or 4 workers is the option offering most performance gains

# Hardware sender streaming testing

- The same 5 devices have been used
- Tests have been run with 1, 2, 4, 8 and 16 workers on each sender/receiver configuration
- The same sender/receiver configurations have been used
- The point of this testing was to compare the performance of the hw sender with that of the sw sender
  - The receiver software was the same in all test cases

# Hardware sender streaming testing – 1 receiver (1)

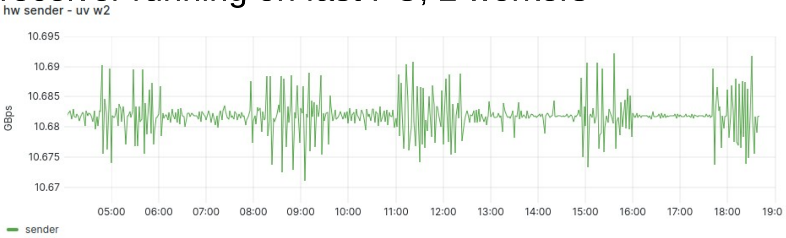
- receiver running on fast PC, 1 worker



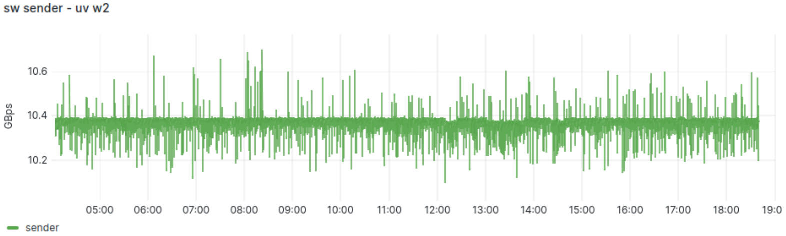


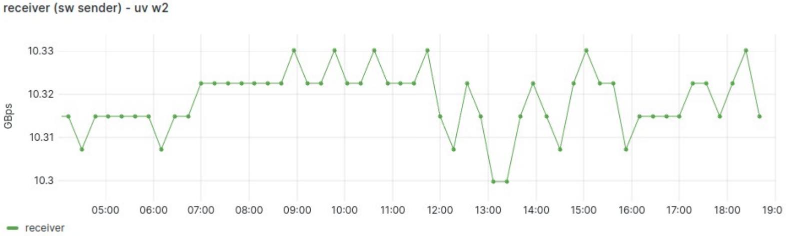


- receiver running on fast PC, 2 workers



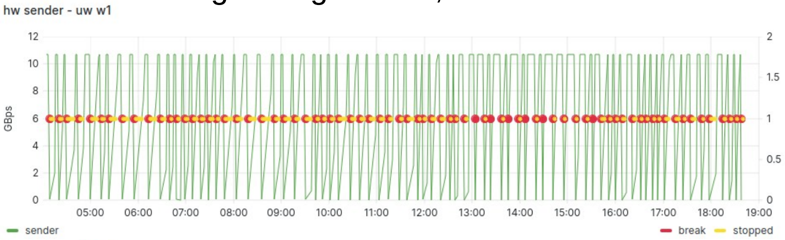


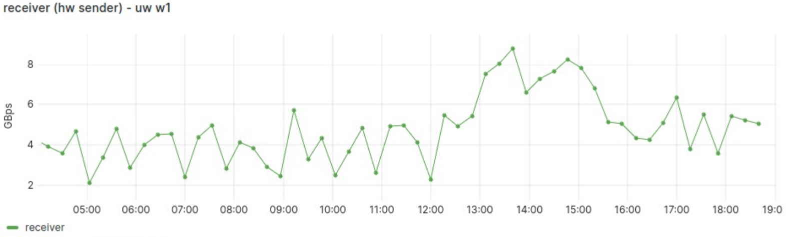


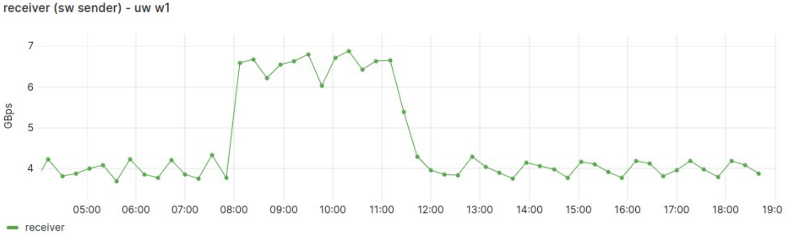
# Hardware sender streaming testing – 1 receiver (2)

- receiver running on regular PC, 1 worker

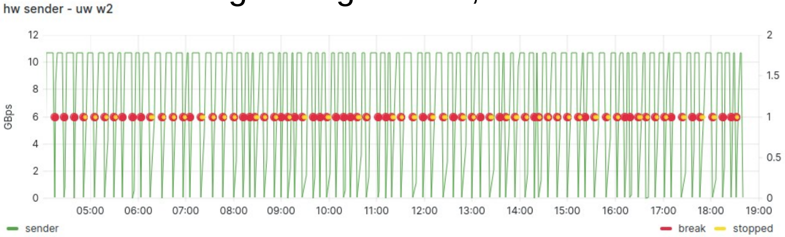


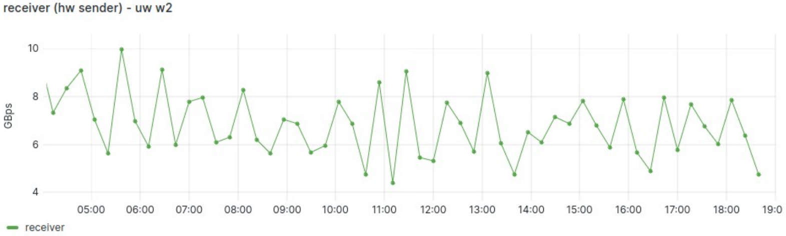



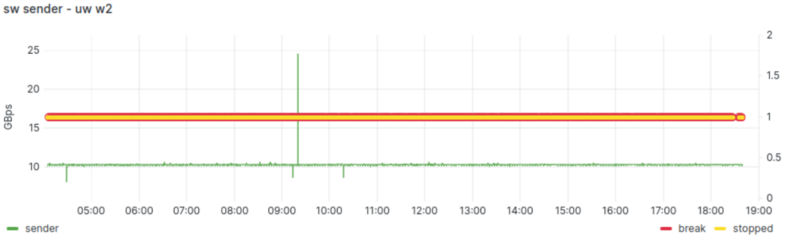


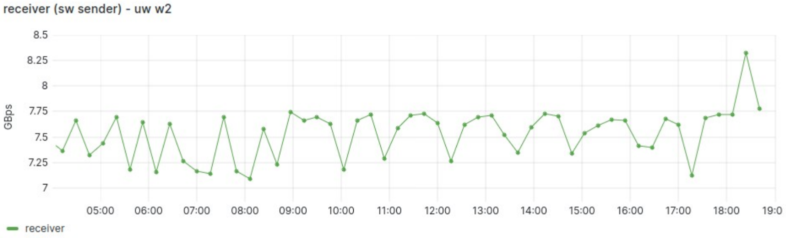



- receiver running on regular PC, 2 workers



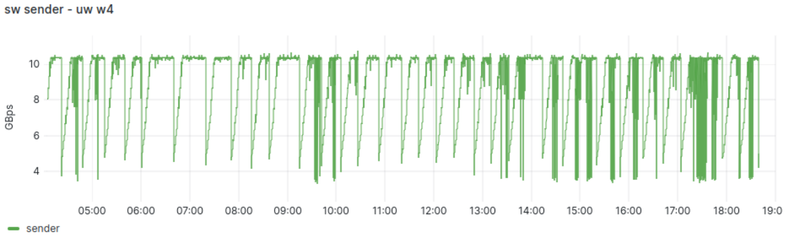



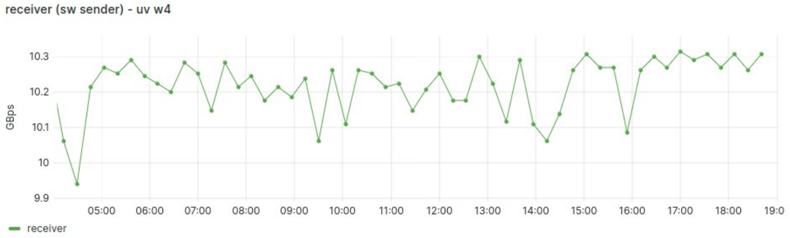



# Hardware sender streaming testing – 1 receiver (3)

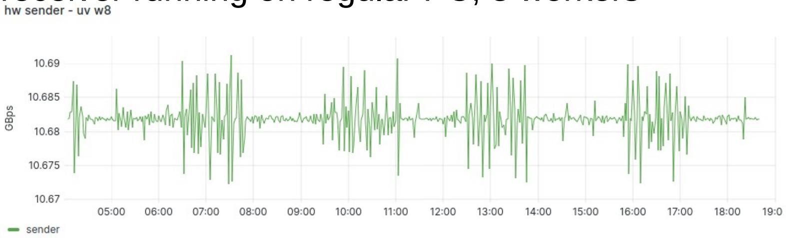
- receiver running on regular PC, 4 worker



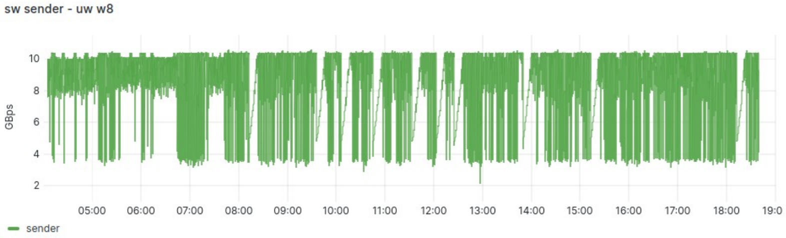


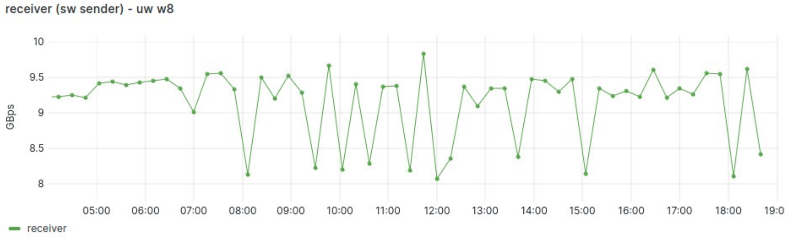




- receiver running on regular PC, 8 workers



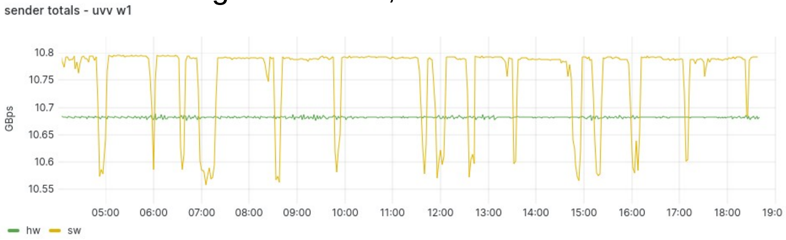


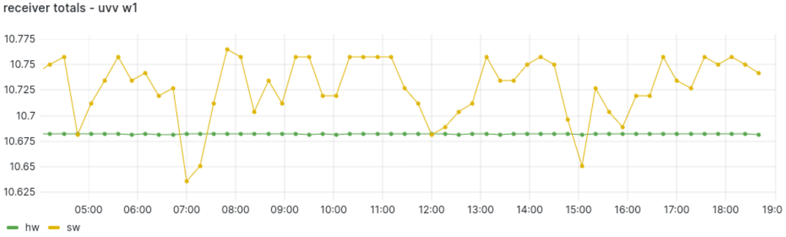


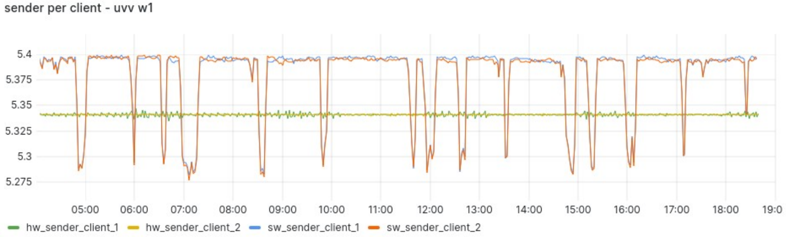



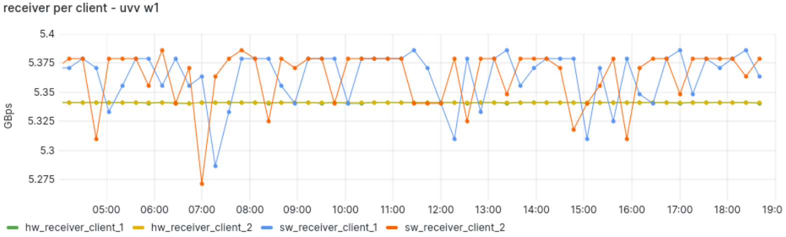
# Hardware sender streaming testing – 2 receivers (1)

- receiver running on fast PC, 1 worker

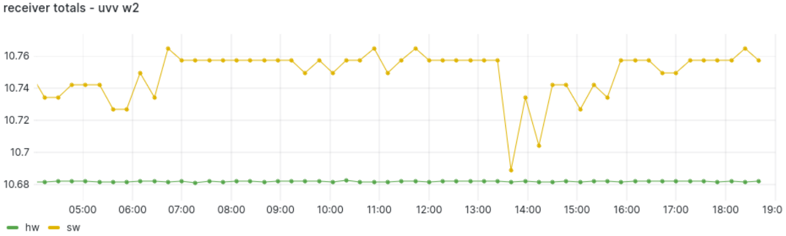


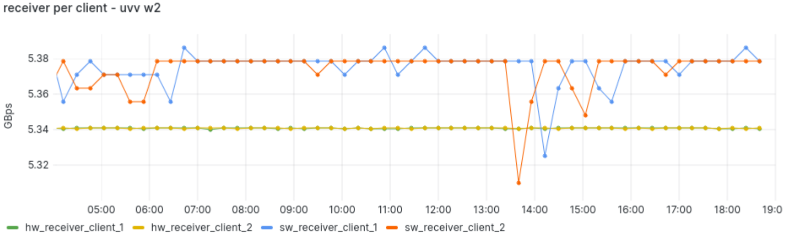




- receiver running on fast PC, 2 workers

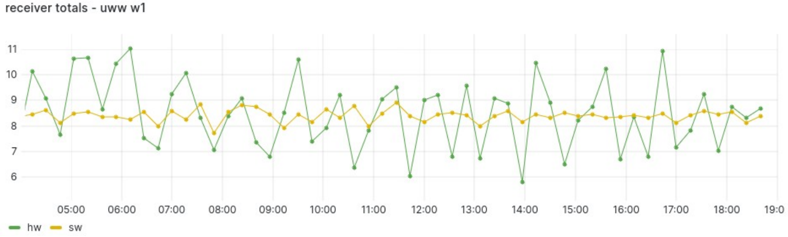





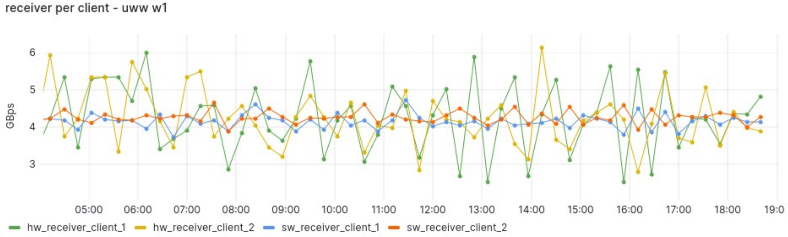


# Hardware sender streaming testing – 2 receivers (2)

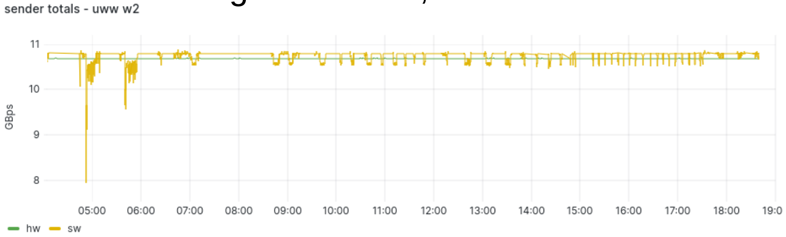
- receiver running on fast PC, 1 worker

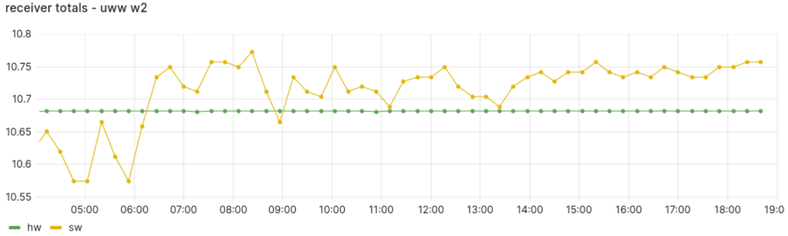



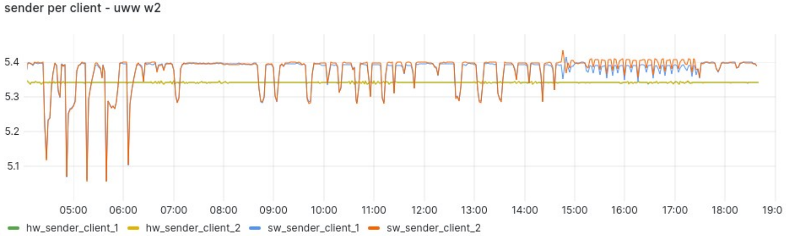





- receiver running on fast PC, 2 workers

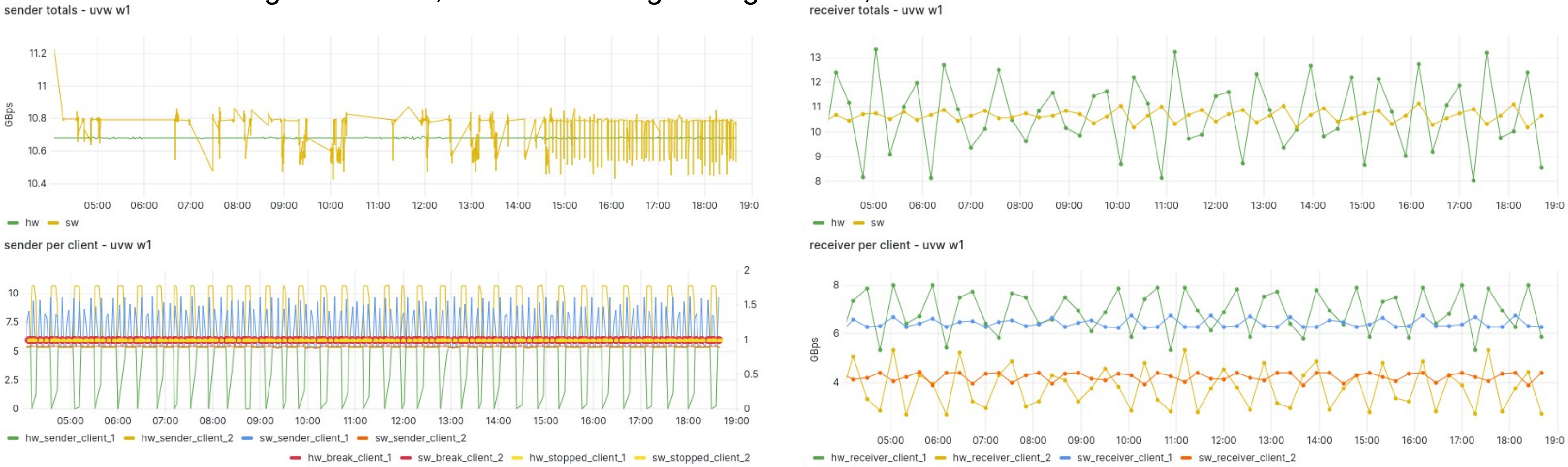


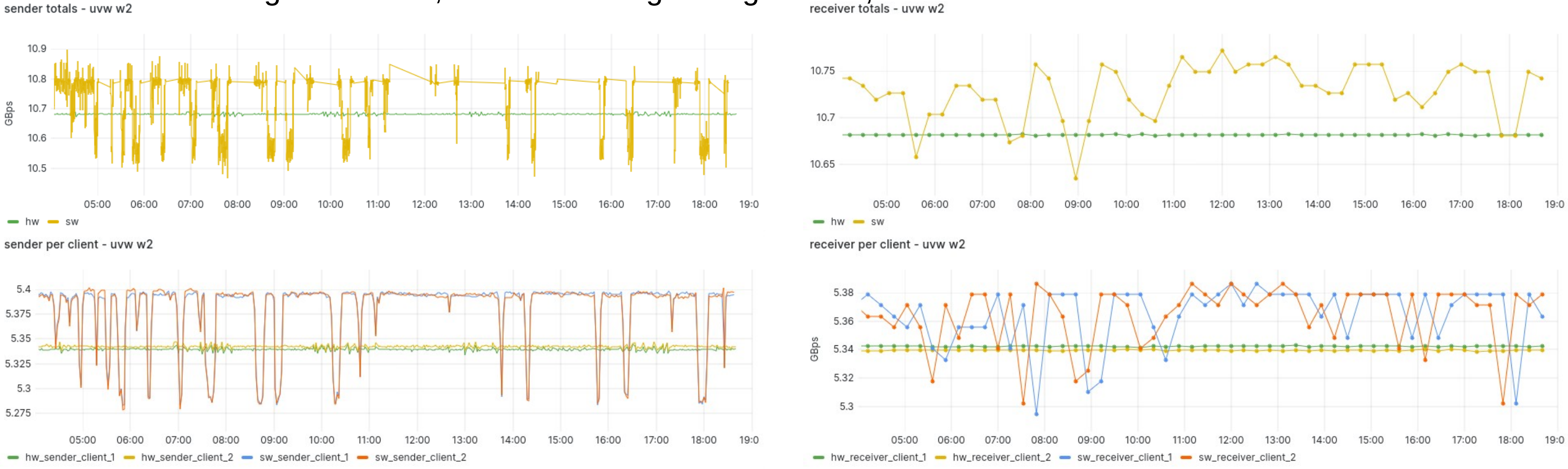




# Hardware sender streaming testing – 2 receivers (3)

- first receiver running on fast PC, second running on regular PC, 1 worker

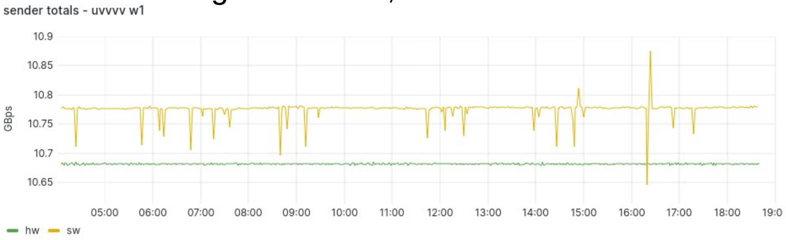


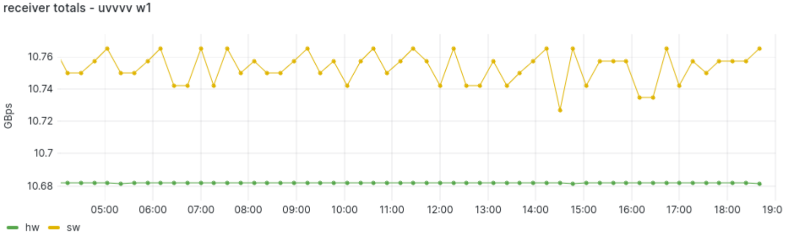

- first receiver running on fast PC, second running on regular PC, 2 workers

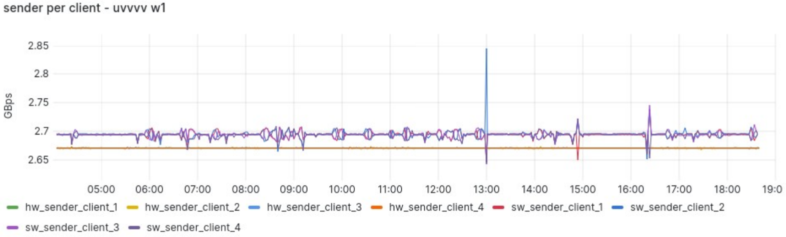


# Hardware sender streaming testing – 4 receivers (1)

- receiver running on fast PC, 1 worker

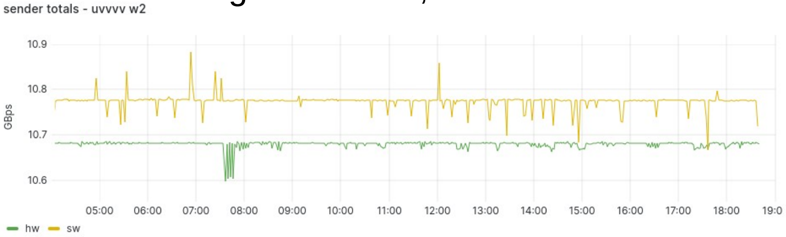


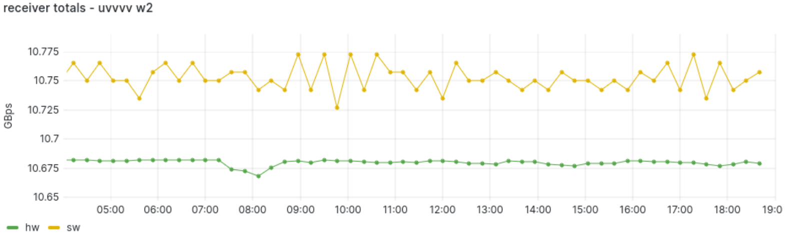



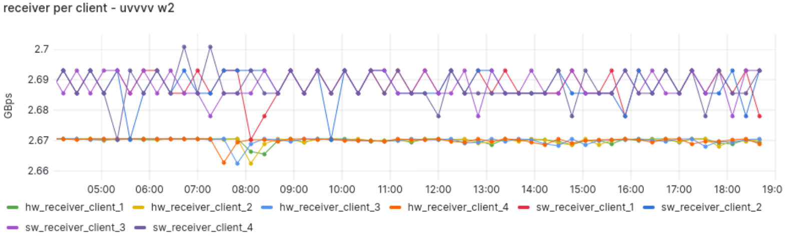


- receiver running on fast PC, 2 workers



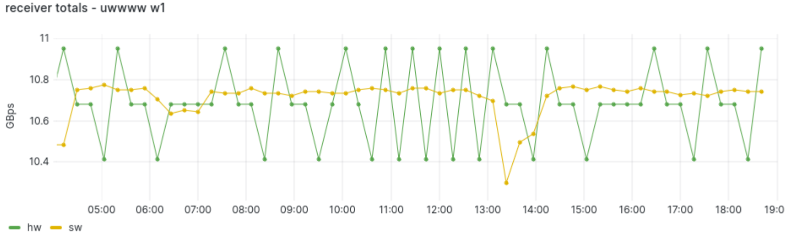



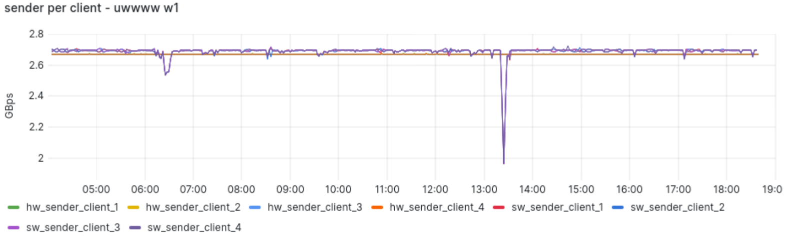




# Hardware sender streaming testing – 4 receivers (2)

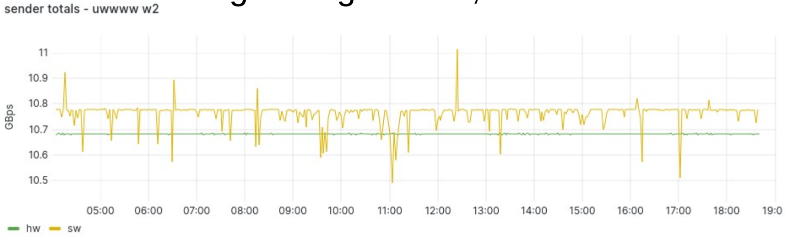
- receiver running on regular PC, 1 worker

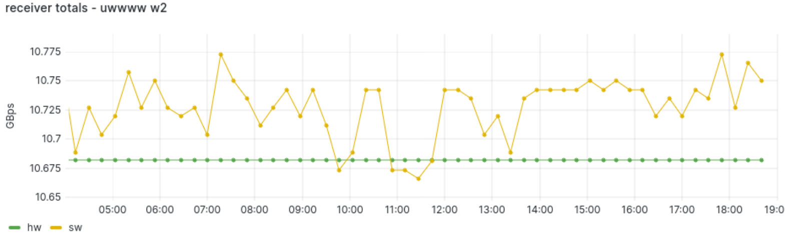





- receiver running on regular PC, 2 workers

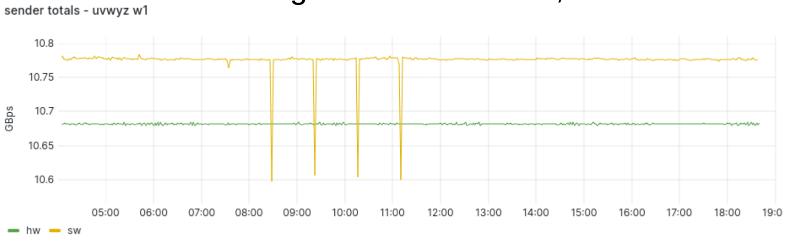


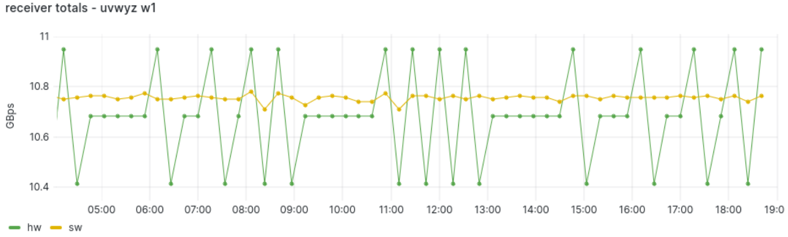





# Hardware sender streaming testing – 4 receivers (3)

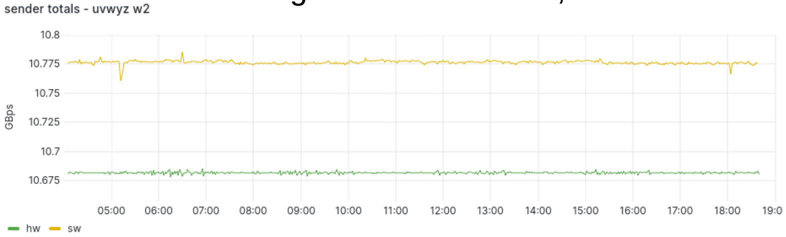
- each receiver running on a different PC, 1 worker

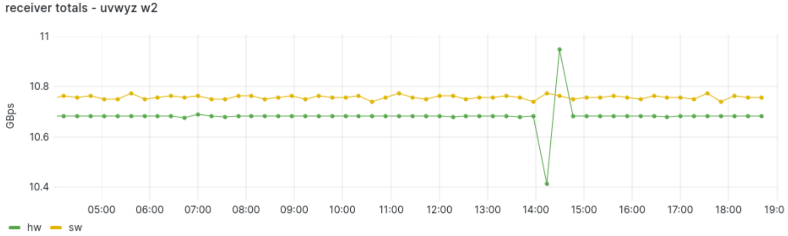


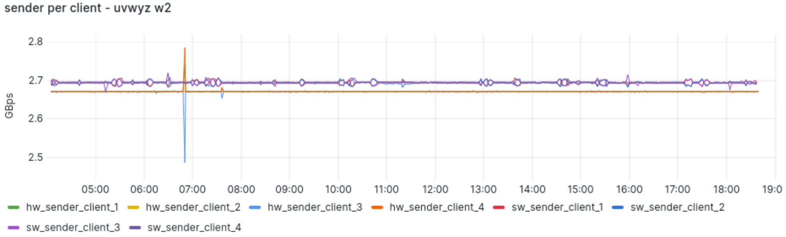



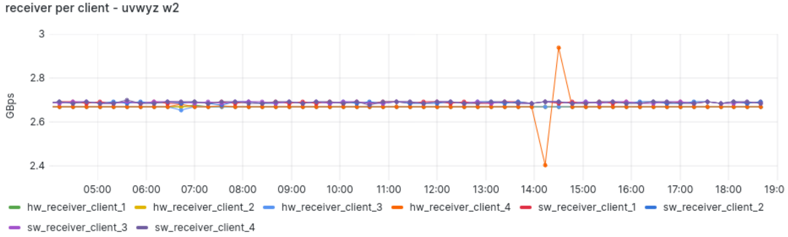



- each receiver running on a different PC, 2 workers



# 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

# Conclusions

- The multi-threaded implementation improves the performance of the receiver
  - The performance depends greatly on the performance of the device on which the receiver is running
  - 2 or 4 workers are the options offering most performance gains
- The hardware and software senders’ bandwidth are almost identical
  - The hardware implementation offers very stable data transfer rates
- The RDMA on FPGA implementation is close to being feature complete and ready for use as a communication subsystem in other projects

# Q&A