

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

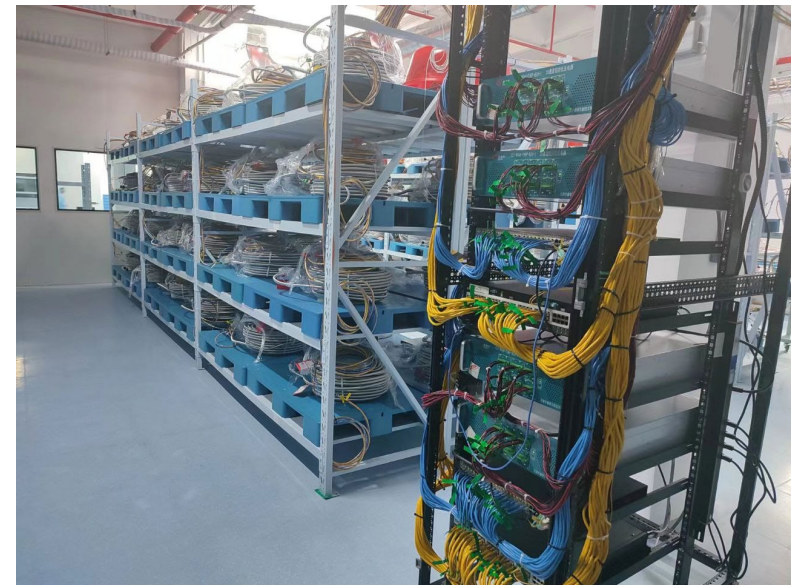
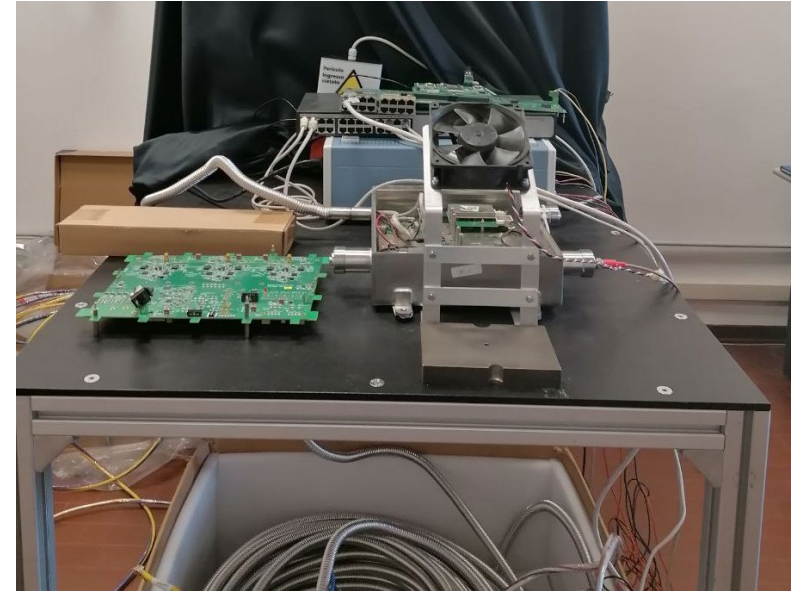
Rate and bandwidth measurements

Antonio Bergnoli, Riccardo Brugnera, Vanessa Cerrone, Alberto Coppi,
Alberto Garfagnini, Marco Grassi, Beatrice Jelmini, Ivano Lippi,
Andrea Serafini, Andrea Triossi, **Riccardo Triozzi**, Katharina von Sturm

6 May 2022

Overview

1. IPbus and its implementation implementation
2. Data acquisition performances for a single GCU
3. Parallel acquisition with multiple GCUs



The IPbus suite

JUNO challenge: **acquire** and transfer **data** in parallel to the **remote monitoring** and control of the electronics.

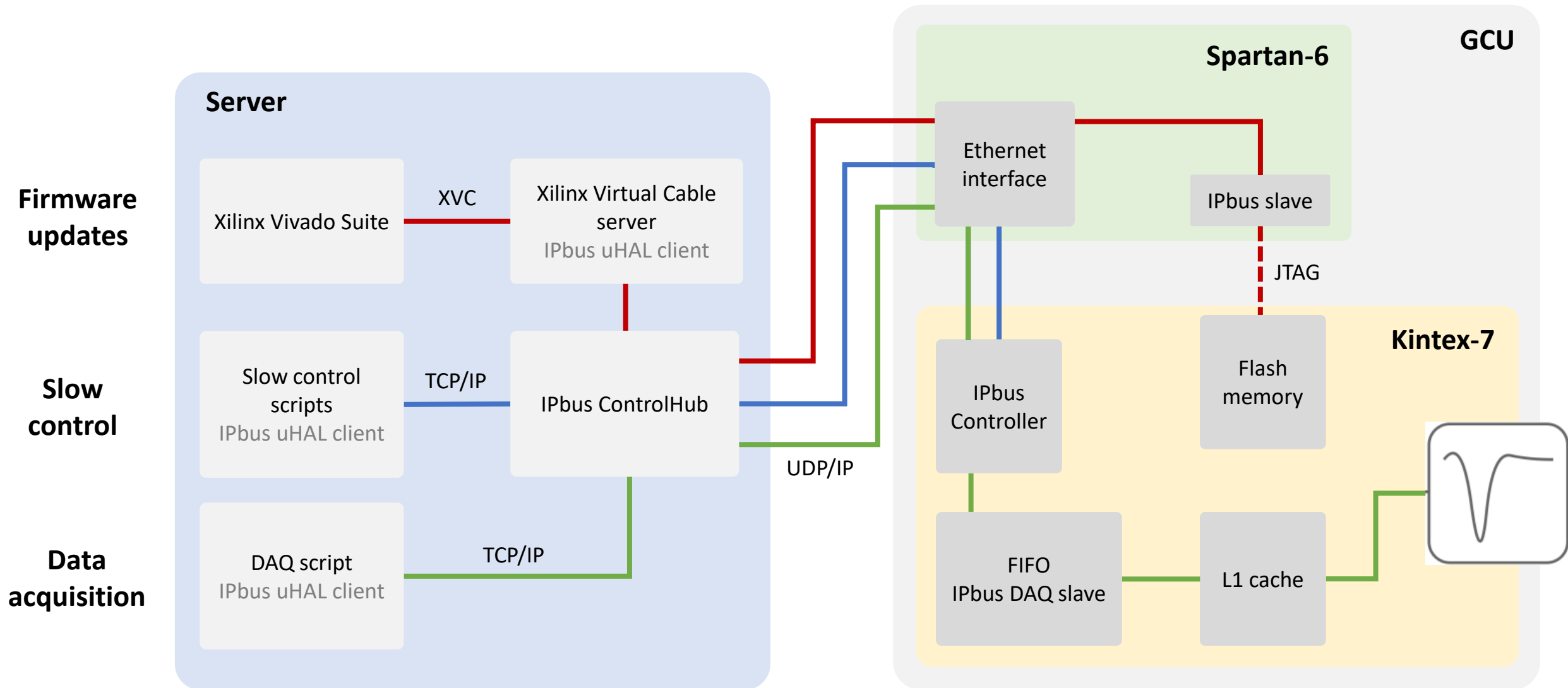
IPbus suite of software and firmware implements a reliable high-performance control link specifically suited for particle physics electronics.

IPbus is a hardware and firmware solution that communicates over Ethernet using **UDP/IP** and consists of:

- (i) A **firmware module** implementing the IPbus protocol within end-user hardware (e.g. JUNO's Kintex-7 and Spartan-6 FPGAs)
- (ii) A **micro Hardware Access Library (uHAL)** providing an end-user C++/Python library for read/write operations on IPbus.
- (iii) A **software application** called **ControlHub**, which mediates simultaneous hardware access from multiple uHAL clients.

While the UDP protocol does not include any native reliability mechanism, the use of **ControlHub** assures the duplication and re-ordering of any lost IPbus UDP packet, **providing a reliability mechanism** at software level.

IPbus implementation



IPbus implementation

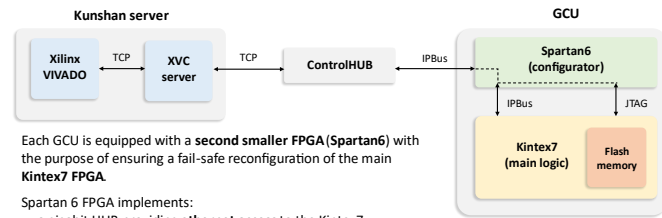
Firmware updates

Slow control

Data acquisition

My talk at Jan 2022 Collaboration Meeting ([DocDB 7827](#))

Simplified scheme of server-FPGA connection



Each GCU is equipped with a second smaller FPGA (Spartan6) with the purpose of ensuring a fail-safe reconfiguration of the main Kintex7 FPGA.

Spartan 6 FPGA implements:

- a gigabit HUB providing ethernet access to the Kintex7
- the JTAG access to the Kintex7 via XVC (Xilinx Virtual Cable) proprietary protocol and IPBUS;

The XVC server bridges vendor tools (via XVC-tcp protocol) to the IPBUS (via the TCP link with the ControlHUB)

18/01/2022

A. Serafini - Firmware flashing update

3

Flashing Procedure for a single GCU

Volatile flashing

- An instance of Vivado (Xilinx) connects to the Kintex7 through a virtual cable running on ethernet (XVC -> TCP -> IPBus -> JTAG).
- A temporary firmware called "programmer" is flashed on the volatile memory of the Kintex7 FPGA via Vivado. The programmer is accessible via a static and hardcoded IP address.

Permanent flashing

- The FPGA_prog utility is then used to write the desired firmware on the permanent flash memory of the Kintex7.
- The Kintex7 automatically reboots, loading the updated firmware from its flash memory.
- The Spartan6 FPGA provides the Kintex7 its static IP based on its GCU_ID number.

18/01/2022

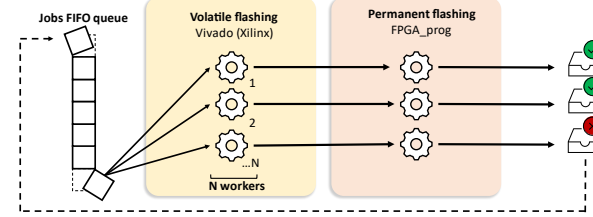
A. Serafini - Firmware flashing update

4

The solution: workers and jobs queuing

For each GCU, a "volatile" job is submitted to a FIFO queue.

N workers are initialized. Whenever a worker is available, it runs the first job of the queue. As soon as the worker ends the flashing of the programmer, it launches the permanent flashing of the GCU in background and moves on to the next job.



18/01/2022

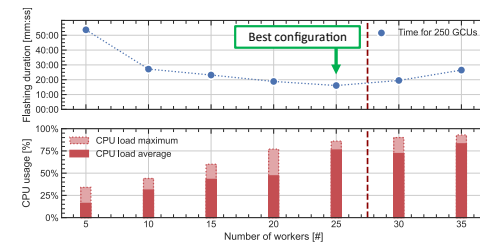
A. Serafini - Firmware flashing update

5

Optimization of allocated resources

Logging the CPU usage confirmed our guess: CPU starts to saturate for >25 workers.

Counter-intuitively, for >25 workers the total time needed to flash 250 GCUs returns to increase.



Total: 250 GCUs		
N. workers [#]	Duration [mm:ss]	
5	53:40	
10	27:10	
15	23:10	
20	18:50	
25	16:10	Best configuration
30	19:30	
35	26:30	

18/01/2022

11

IPbus implementation

Server

Xilinx Vivado

Slow control scripts
IPbus uHAL

DAQ script
IPbus uHAL

Firmware updates

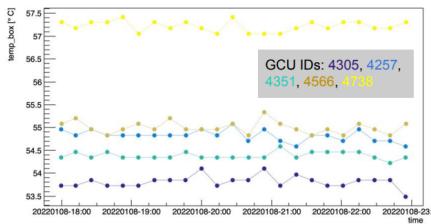
Slow control

Data acquisition

Beatrice's talk at Jan 2022 Collaboration Meeting ([DocDB 7829](#))

Test Protocol: Test4 - slow control

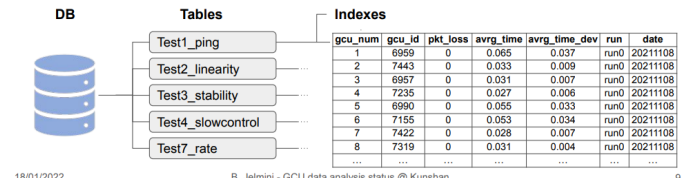
- Monitored quantities: temp_box, temp_ch0/1/2, HV_ch0/1/2, VCCINT, VCCAUX, VREFP, VREFN, VCCBRAM
- Output text file with: mean value, max value, min value per GCU
- Summary plots



18/01/2022 B. Jelmini - GCU data analysis status @ Kunshan 7

Indexing tests results: an SQL database

- Results, setup and configuration information from Test 1, 2, 3, 4 and 7 are indexed and included in a SQL database (DB).
- GCUs and tests are uniquely identified by run number, date, time and GCU ID.
- The DB has a table for each Test type. Each table has an index for each information or plot available for that particular test.

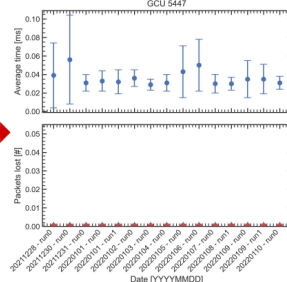


18/01/2022 B. Jelmini - GCU data analysis status @ Kunshan 9

Querying the DB to investigate GCUs' properties

```
SELECT * FROM Test1 WHERE gcu_id =5447;
```

gcu_num	gcu_id	pkt_loss	avrg_time	avrg_time_dev	run	date
1	51	5447	0	0.039	0.035	run0 20221228
2	50	5447	0	0.056	0.048	run0 20221230
3	59	5447	0	0.031	0.009	run0 20221231
4	59	5447	0	0.033	0.011	run0 20220101
5	59	5447	0	0.032	0.013	run0 20220101
6	59	5447	0	0.036	0.009	run0 20220102
7	59	5447	0	0.029	0.006	run0 20220103
8	59	5447	0	0.031	0.009	run0 20220104
9	59	5447	0	0.028	0.009	run0 20220105
10	59	5447	0	0.035	0.028	run0 20220106
11	59	5447	0	0.03	0.01	run0 20220107
12	59	5447	0	0.03	0.007	run0 20220108
13	57	5447	0	0.035	0.02	run0 20220109
14	57	5447	0	0.035	0.016	run0 20220109
15	57	5447	0	0.031	0.007	run0 20220110



18/01/2022 B. Jelmini - GCU data analysis status @ Kunshan 10

Test Protocol: Test1 - ping

- Flood each GCU with 100 56-byte packets
- Output text file with: average response time + std dev, fraction of packet loss per GCU
- Summary plot



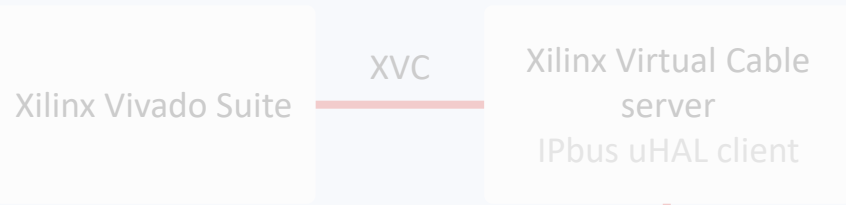
18/01/2022 B. Jelmini - GCU data analysis status @ Kunshan Test date: 9th Jan 2022 4

Quantities in green are stored in the database

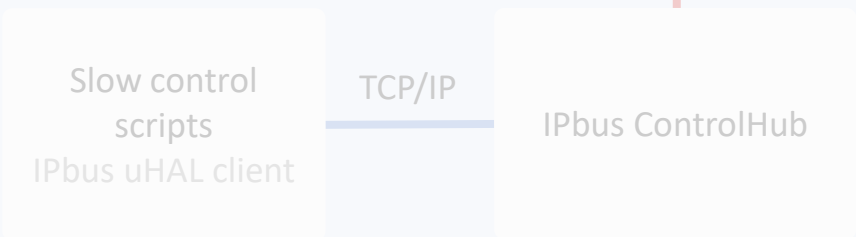
IPbus implementation

Firmware updates

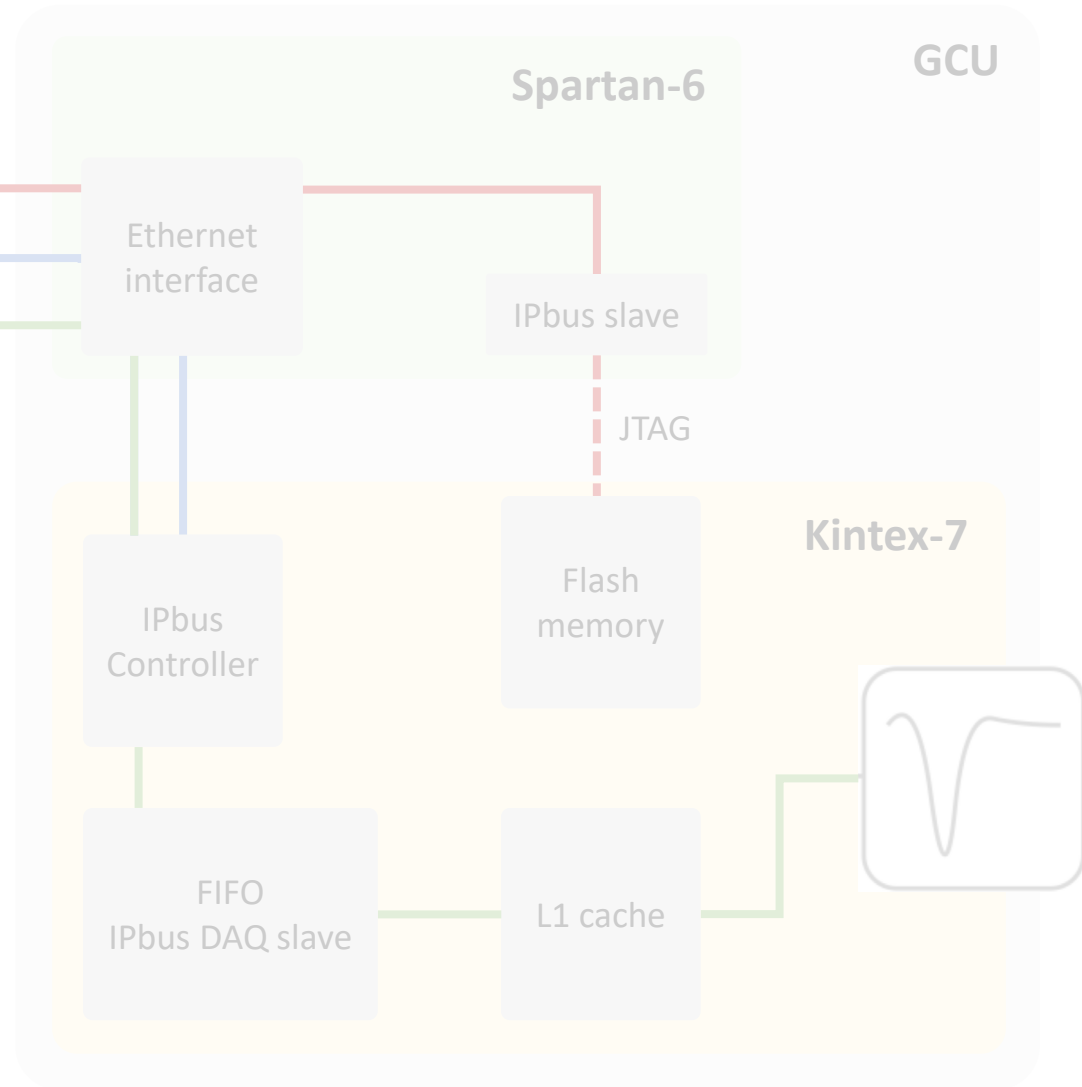
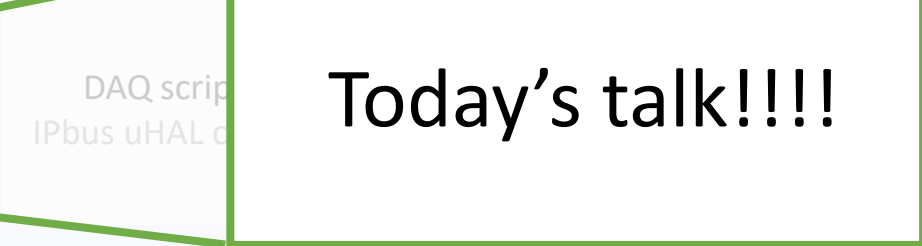
Server



Slow control

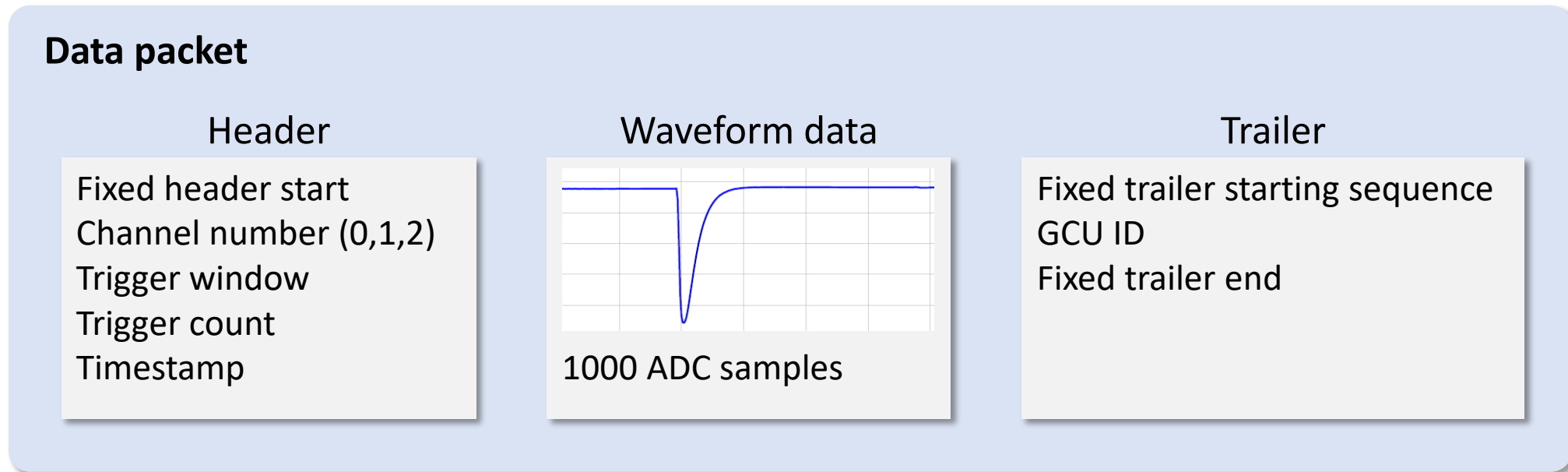


Data acquisition



The structure of a waveform data packet

A waveform data packet is composed of a **header**, a **trailer** and the actual **waveform** data. Each waveform contains **1 μ s** of data sampled with **1GHz frequency**, for a total of **1000 samples**.



1 waveform packet = 16 header bytes + 16 trailer bytes + $1 \mu\text{s} \times 1000 \text{ ADC}/\mu\text{s} \times 2 \text{ bytes}/\text{ADC} = \mathbf{2032 \text{ bytes}}$
The **FIFO** has a dimension of **8192 bytes** and can therefore contain **~4 waveforms**.

Maximum transferrable bandwidth for 1 GCU

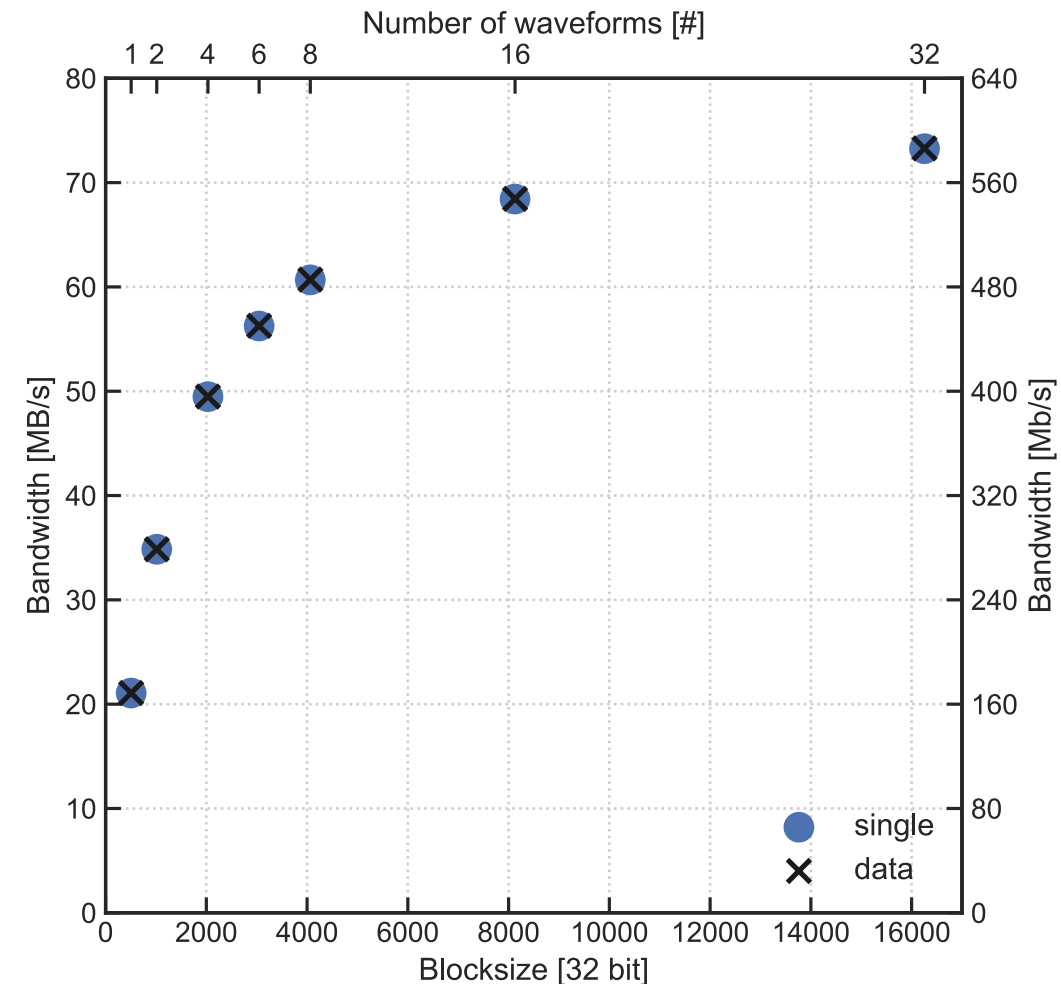
Via IPbus it is possible to **specify** the “**blocksize**” with which read the waveform data from the GCU.

Higher blocksize → less data packets → higher bandwidth

We performed a scan of the transferrable bandwidth as a function of the blocksize.

We logged both the network bandwidth (single) and the bandwidth used by valid waveform data (data).

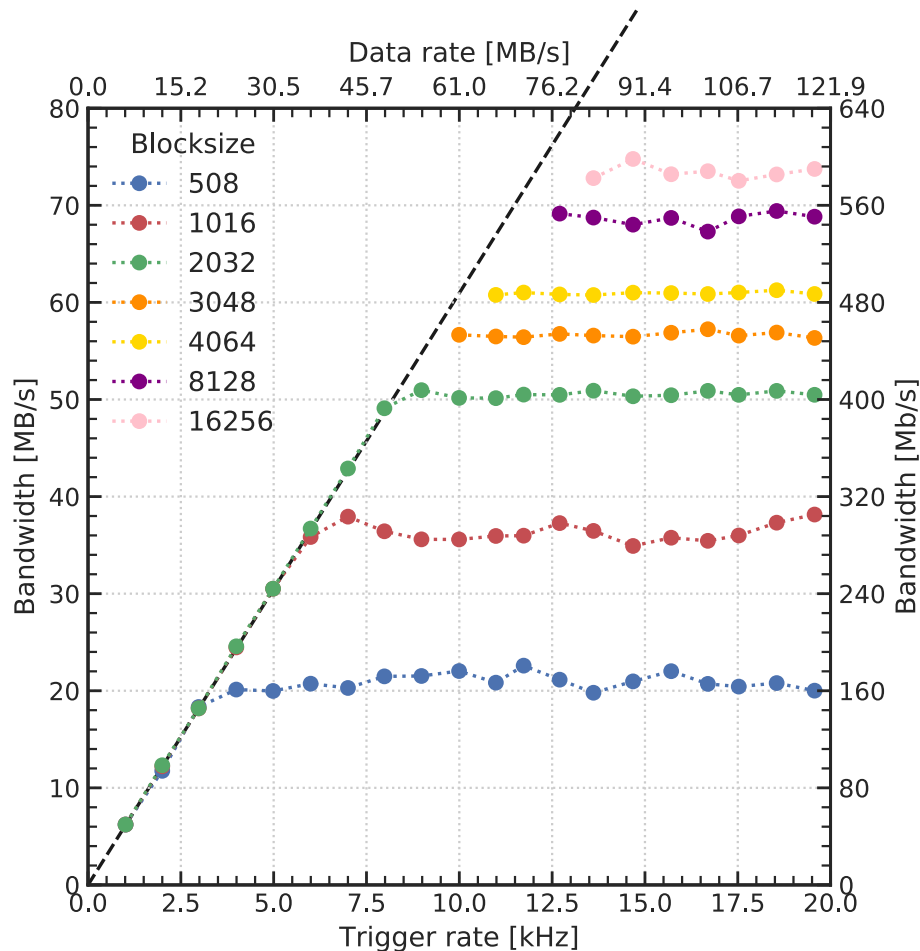
- For large payloads it is possible to reach **~0.5 Gbit/s**
→ maximum permitted by IPbus protocol
- “single” bandwidth and “data” bandwidth correspond
→ all transferred packets are waveform data packets



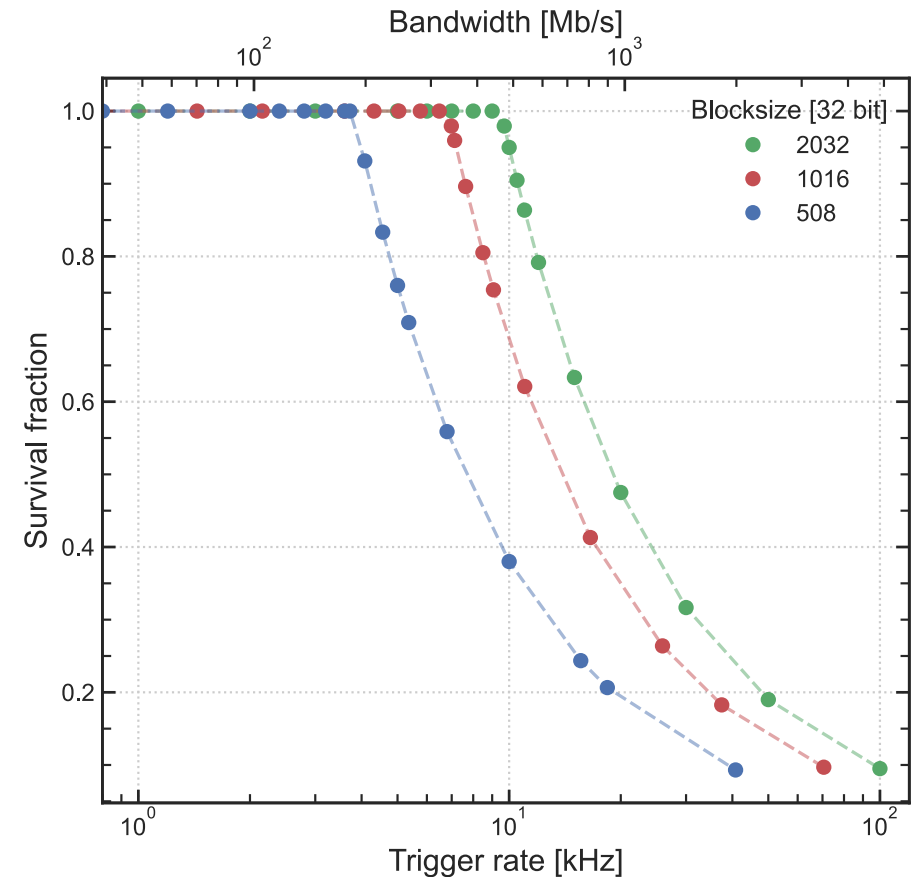
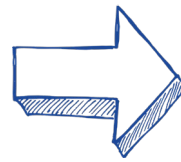
Maximum transferrable bandwidth for 1 GCU

Blocksize determines the maximum transferrable bandwidth for a GCU.

When **bandwidth saturates**, we are **not able to transfer all waveforms (wf)** collected by the GCU.



$$\text{survival fr.} = \frac{\# \text{ wf transferred}}{\# \text{ wf generated}}$$

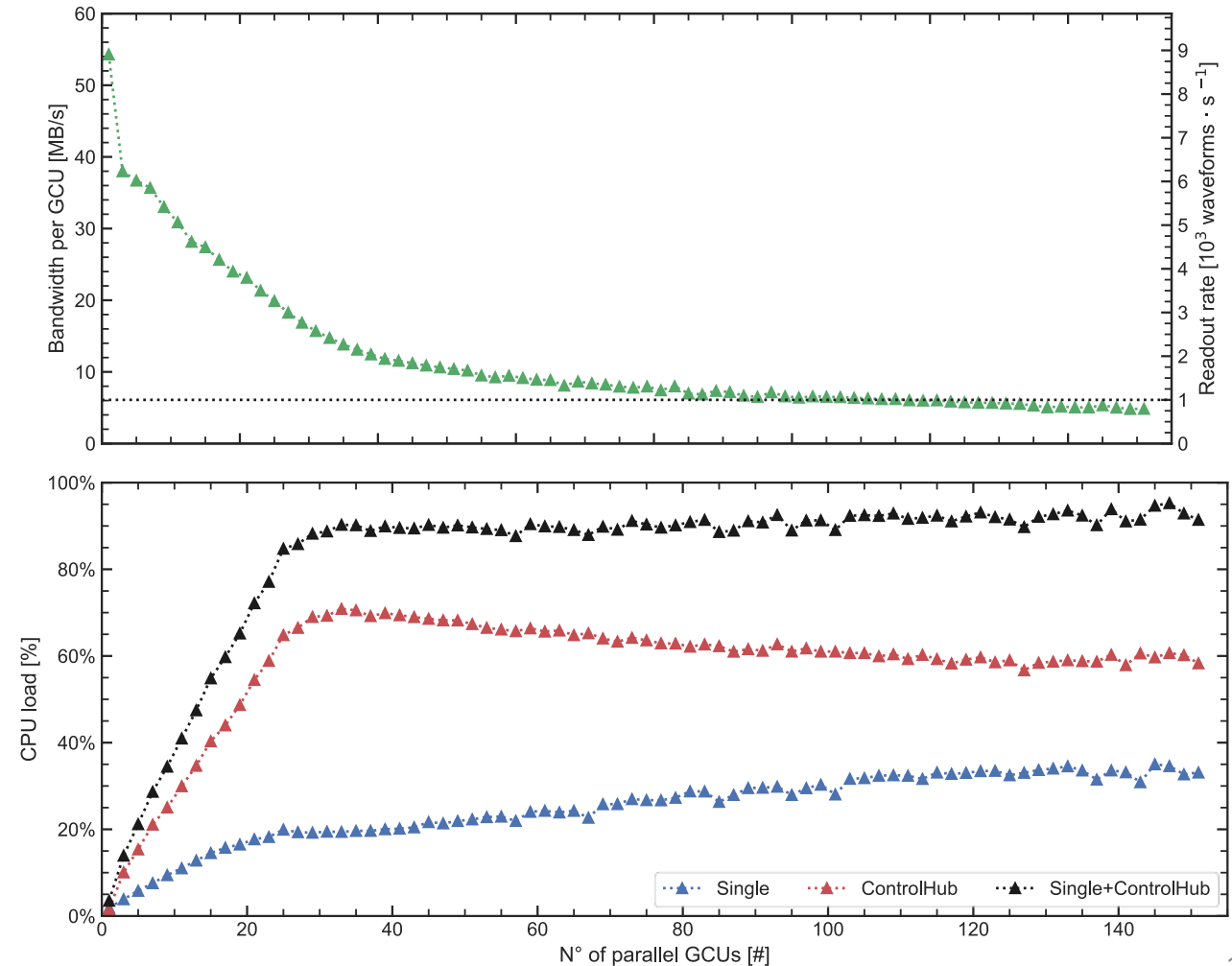


Managing several GCUs in parallel

At **Kunshan** we have a CentOS server having an Intel Xeon Gold 6226 CPU @ 2.70GHz for a total of **24 cores** (48 threads). We can test up to 344 GCUs in parallel.

We logged the transferred bandwidth as a function of the number of GCUs read in parallel for a blocksize of 2048.

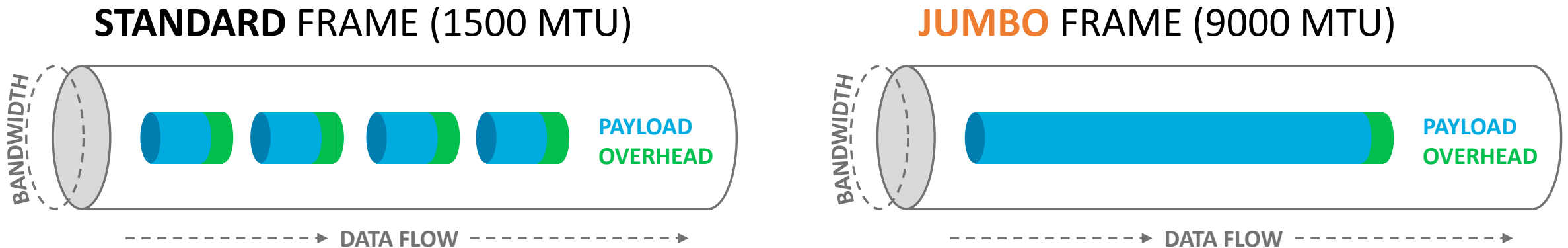
- Server resources saturate for ~ 30 GCUs. Afterward, transferred bandwidth slowly continue to grow.
- Most resources are used by the ControlHub.
- Looking at the bandwidth transferred per GCU (total bandwidth/ # GCUs), it is still possible to manage a 1kHz trigger rate with 90-100 GCUs.



Reducing CPU usage with Jumbo Frames

Standard ethernet packets are designed to carry **1500 bytes of payload**. This payload is also called **Maximum Transferrable Unit (MTU)**.

Most modern network interfaces can support **MTUs up to 9000 bytes**. Such ethernet packets are usually referred to as **Jumbo Frames**.

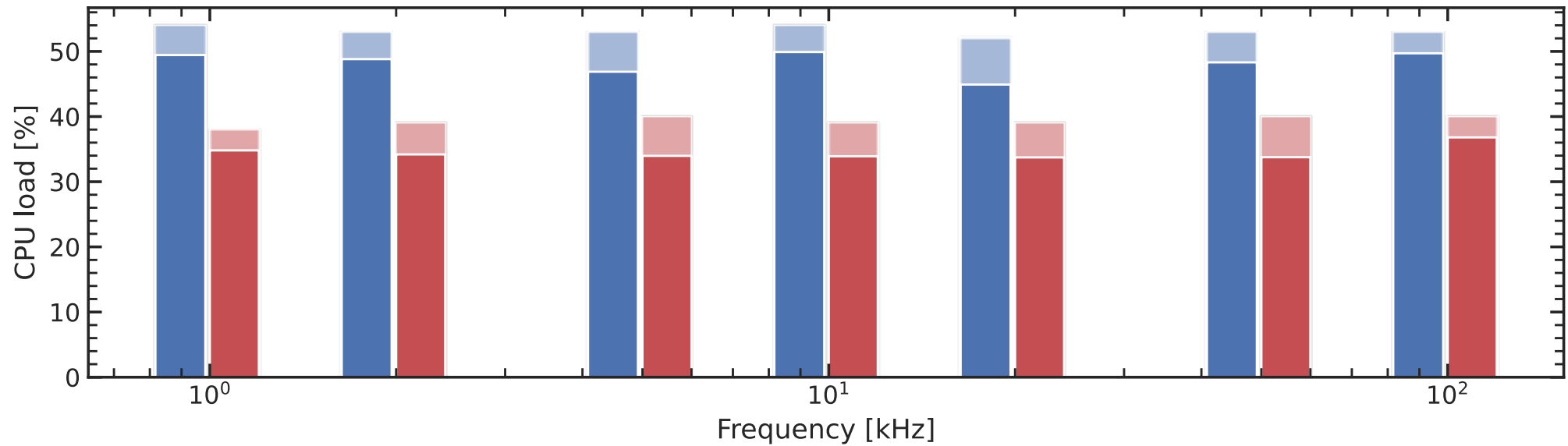
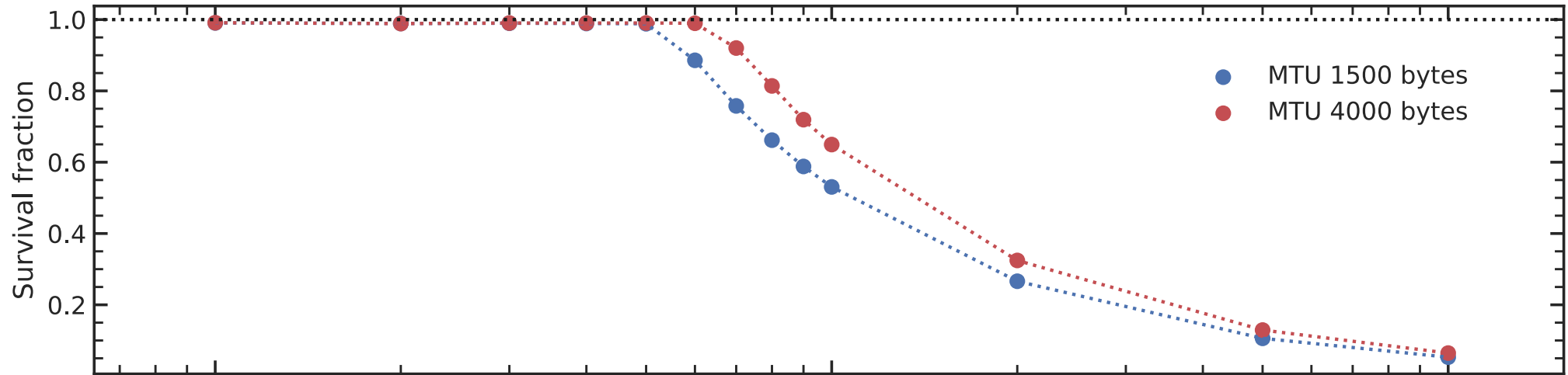


The rationale:

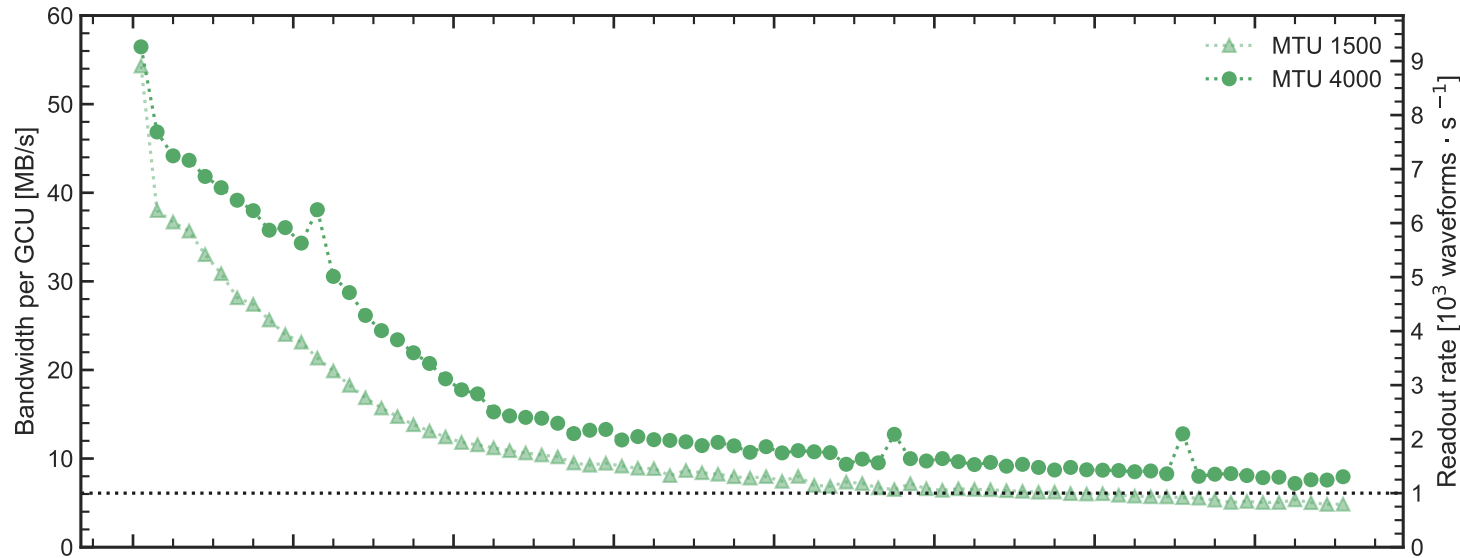
higher payloads → less packets → lower CPU utilization

Testing Jumbo Frames @ LNL

9 GCUs

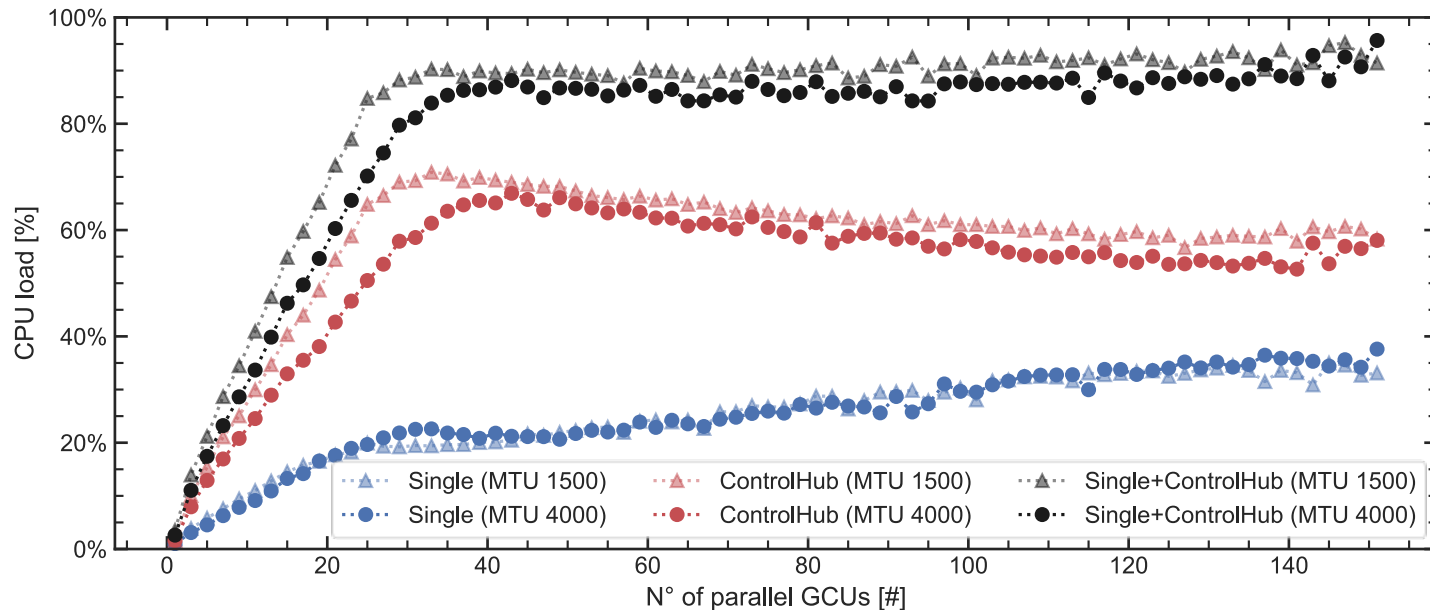


Testing Jumbo Frames @ Kunshan



Jumbo Frames permit to lower ControlHub's CPU utilization by 20%

As a consequence, it is now possible to manage a 1kHz trigger rate with 150 GCUs!



A rough estimate for JUNO setup:

17612 L-PMTs

→ $17612/3 = 5871$ GCUs

→ $5871/150 = 40$ servers

Needs to be carefully checked!

Needs additional measurements!!

Final remarks

- At GCU level we can manage trigger rates up to 10 kHz.
- When dealing with multiple GCUs, the bottleneck is represented by the CPU usage of the ControlHub. However, keep in mind that ControlHub is crucial to permit the simultaneous access to data and slow control parameters.
- Jumbo Frames can help in reducing ControlHub's CPU usage. It is possible to sustain a trigger rate of 1kHz when managing 150 GCUs in parallel.

We are condensing all our tests and measurements in an article under preparation!

Implementation and performance of IPbus in JUNO
Large-PMT data acquisition stream

collaboration^a

^a*Institutions will be indicated here*

