Telepresence-Enhanced Network Music Performance (TENEMP)

(SPIRIT OC1)

Experiment setup

Experiments

I. Performance comparison of different NMP tools.
   - Assess feasibility of Network Music Performance (NMP) in 5G networks.
   - P2P transmissions / various musician placements

II. Performance gains of edge-computing.
   - Can NMP benefit from the placement of an SFU server at the edge?

III. Integration of future telepresence techniques in NMP tools.
   - Integrate 3D streaming with NMP.
   - Assess the feasibility with and without SFU

Objectives - keywords

- OBJ1: Compare the performance of NMP tools over 5G. Measure the latency and bandwidth requirements of different NMP tools, that use different protocols over 5G.
- OBJ2: Assess the impact of SFU placement on NMP tools. Measure the latency and bandwidth requirements of NMP with no SFU or with an SFU, at the 5G edge or outside it.
- OBJ3: Assess the feasibility of NMP tools over 5G. Cross-examine the findings of the experimentation campaign with the individual requirements of different NMP use cases.
- OBJ4: Explore the integration of telepresence with NMP tools. Extend an open-source NMP tool to convey streams of holographic and/or avatar-based telepresence.
- OBJ5: Explore the performance of NMP with telepresence support. Measure the additional bandwidth requirements and latency overhead of the telepresence stream.
- OBJ6: Explore the in-network cost of NMP with telepresence support. Measure bandwidth requirements, latency overhead, computational and memory consumption at the SFU due to the extra telepresence stream.
- OBJ7: Assess the feasibility of telepresence-enabled NMP applications. Cross-examine the findings of the experiments with the requirements of different NMP use cases.
- OBJ8: Implement a prototype NMP tool with telepresence support. Design, develop and test a prototype NMP application supporting both audio and telepresence streams.

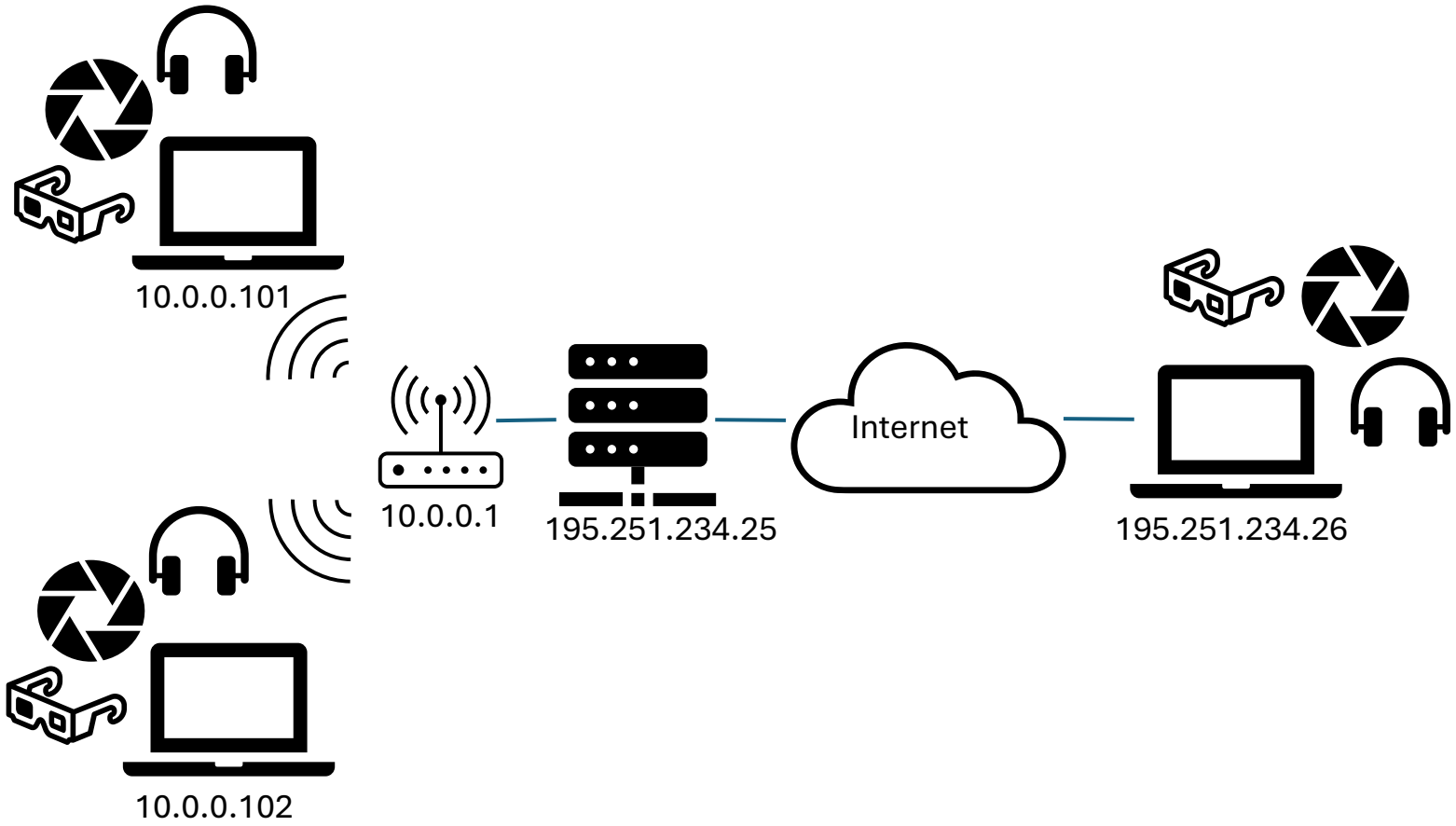
Survey of NMP tools (specs)

Apps	Platform	Arch	UI	Protocol	Dev Lang	Video Codec	Audio Codec	Network	Latency (ms)	Parallel Streams	SFU support
Jitsi	Windows, Linux, OSX	Client server, SFU	browser	Webrtc	JS	H264	Opus	WAN	>80	N web tab	native
Gstreamer (Alsa, Jack)	Linux	P2P	CLI	RTP, UDP	C, python	H264	PCM, Opus	LAN	20-40	N inherent	Need dev new app
Aretousa (Pulse-audio)	Linux	P2P	UI and CLI	RTP UDP	C	N/A	PCM	LAN	~ 20	N processes	Not inherently
Jamulus (Alsa, Jack)	Linux, Windows	Client server, SFU	UI CLI for server	RTP UDP	C++ (JUCE)	N/A	Opus	WAN	20-40	1	native
Jacktrip (Alsa, Jack)	Linux, OSX	P2P	UI + CLI	RTP UDP	C++(Qt)	N/A	PCM	WAN	< 20	1	Not inherently
Sonobus (Alsa, Jack)	Linux, OSX	P2P Server for ip's	UI	RTP UDP	C++ (JUCE)	N/A	PCM, Opus	WAN	< 20	1	Not inherently
UltraGrid (System Audio, Jack)	OSX, Linux	P2P	UI	RTP-based		H264, DXT1 (**3d**)	PCM	WAN	~80	2	Not inherently
ELK	ELK OS running on Pis	undefined	UI	Aloha (UDP)	Linux tools, LV2	N/A	undefined	WAN	~2 (audio processing)	1	Undefined

NMP use cases (requirements)

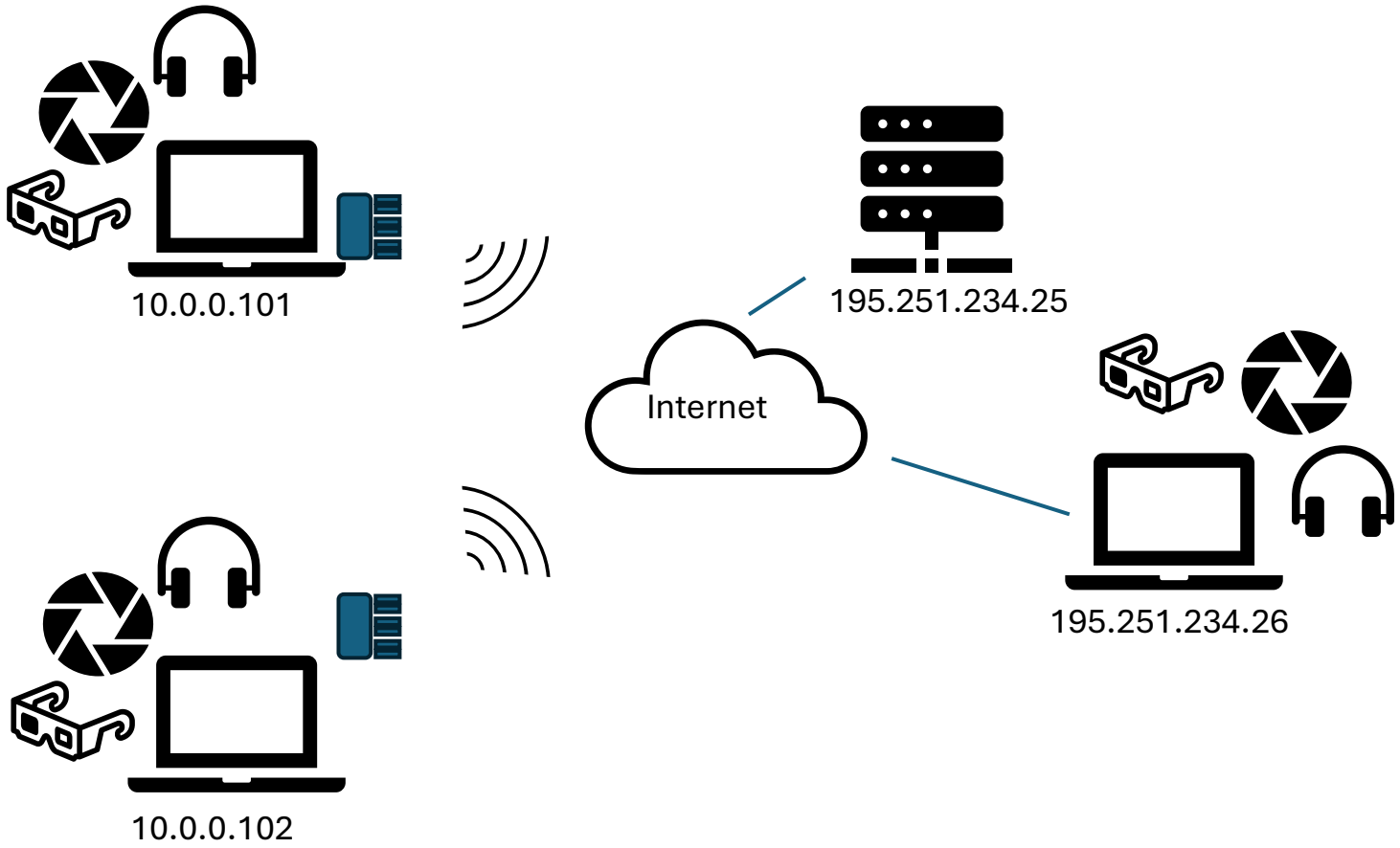
Use-case scenario	Audio Latency (ms)	Video Support & Latency (ms)	Audio Video synchronization/offset
Remote Music performance (synchronous)	<30 mouth-to-ear	Optional / < 30 ms motion-to-photon	audio lead to video: <15ms video lead to audio: <45ms (ITU for TV broadcast)
Remote Music recording (pseudo-synchronous, metronome)	< 120 ms	Optional / <200 ms	
Remote Music Education (asynchronous)	< 120 ms	Important / <200 ms	

TENEMP testbed @ AUEB (for development)



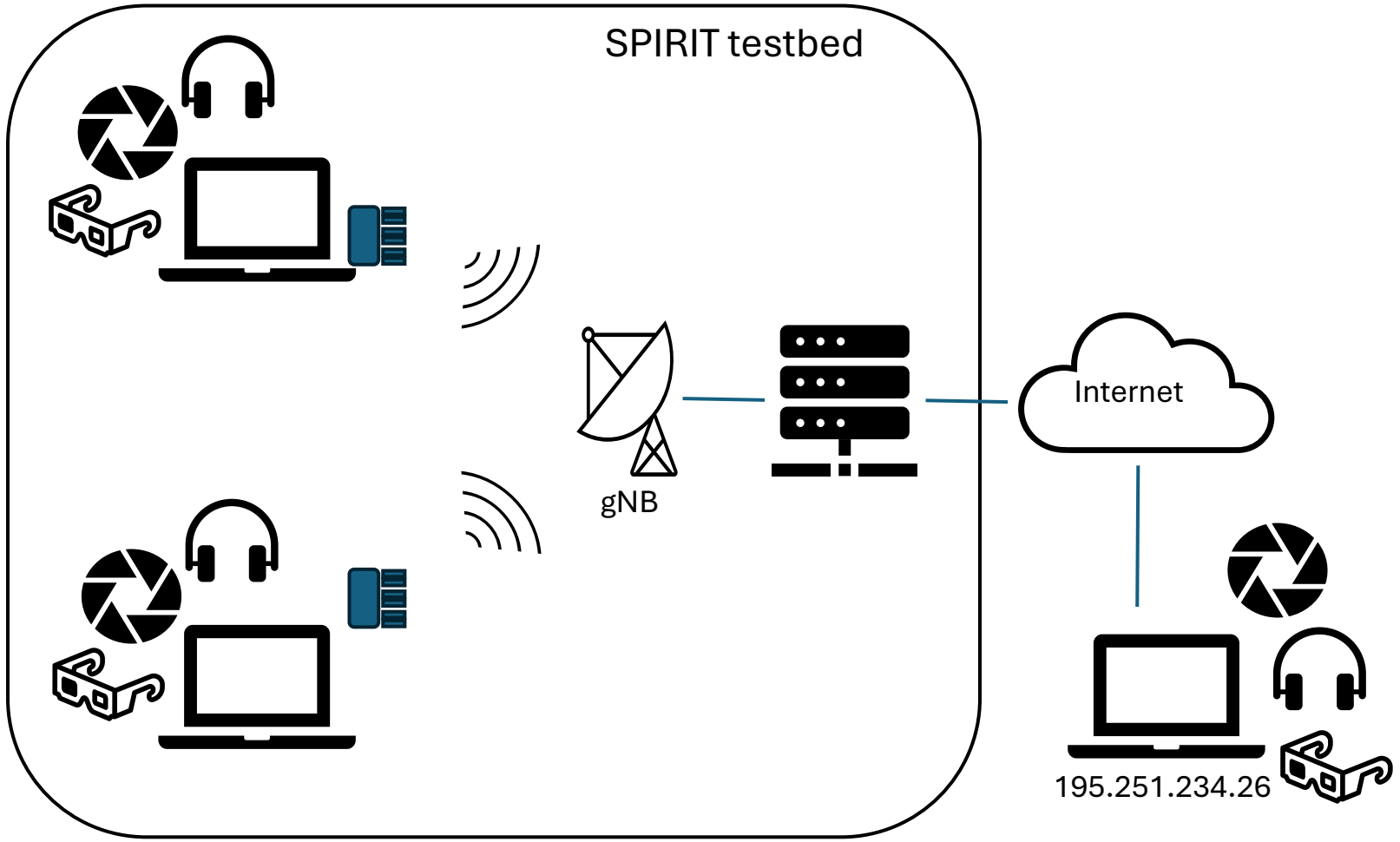
	SFU/ MCU (edge server)
	3D glasses
	3D camera
	Headphones
	Musician's computer, Audio interface, WiFi and Ethernet Network interface (configured by netem)
	Radio antenna (Wi-Fi router), NAT
	Wireless link, bi-directional
	Wired link (Gbit Ethernet), bi-directional

TENEMP testbed @ AUEB (for verification)



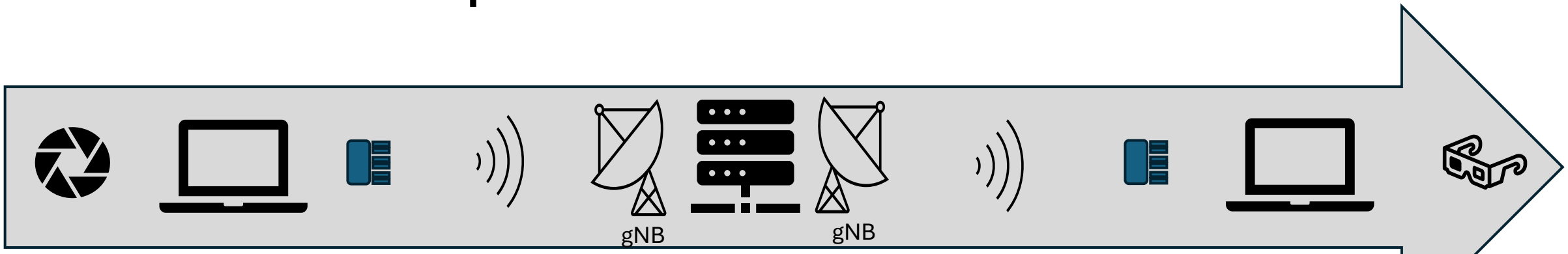
	SFU/ MCU (edge server)
	3D glasses
	3D camera
	Headphones
	Musician's computer, Audio interface, WiFi and Ethernet Network interface (configured by netem)
	5G antenna
	5G link to commercial provider, bi-directional
	Wired link (Gbit Ethernet), bi-directional

TENEMP @ SPIRIT testbed



	SFU/ MCU (edge server/Kubernetes)
	3D glasses
	3D camera
	Headphones
	Musician's computer, Audio interface, WiFi and Ethernet Network interface (configured by netem)
	5G antenna (usb-dongle)
	5G link to SPIRIT NG-RAN, bi-directional
	Wired link (Gbit Ethernet), bi-directional

3D Stream Pipeline



1.	2.	3.	4.	5.	6.	7.
Depth camera captures video	Musician's laptop/RPi creates the data stream	5G dongle transmits the (RTP) pkts to gNB	MEC server applies frame synch & merging	5G dongle receives the (RTP) pkts fr om gNB	Musician's laptop/RPi forwards data stream	Glasses present the 3D scene

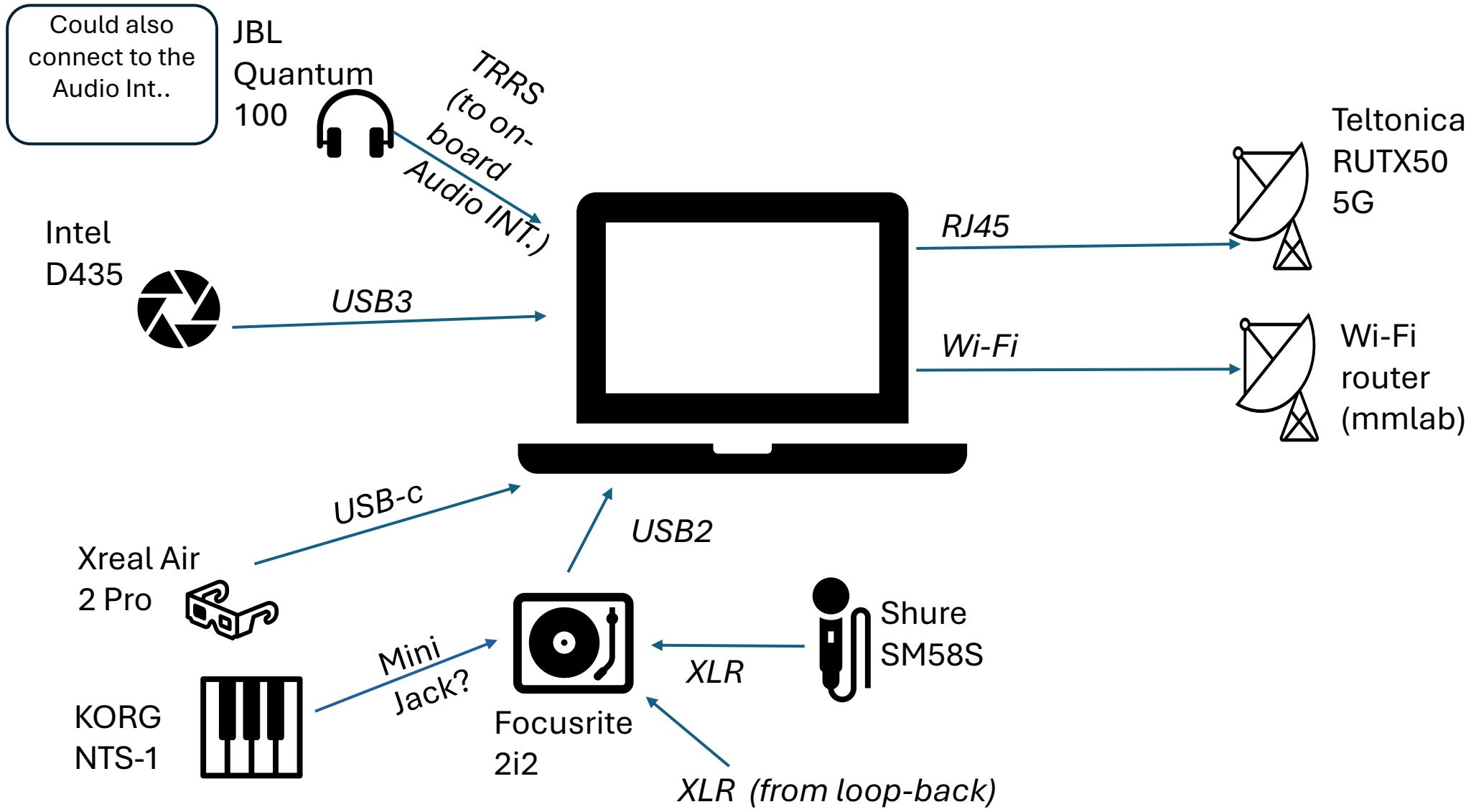
Metrics

- Latency:
  - Measure mouth-to-ear & motion-to-photon delay
  - Send impulse signals, loopback
  - Use Audacity (DAW) to measure difference
- Bandwidth:
  - Measure packets (or bytes) received/sent at node network interfaces
  - Several Linux-based tools create traffic
    - nload, bmon, iftop, vnstat, ifstat
- Computing cost:
  - Measure CPU and memory usage of the SFU service
    - `kubectl top node` command (available to cluster admin)

Integration Questions

- What is the procedure of booking the testbed for experiments?
- Interfacing with Frame Render and Synchronization Function
- Alternative: Interfacing with the Producer app
  - Is WebRTC the only option for producer-consumer communication?
  - Can we deploy multiple producer apps?
- Can we get the SPIRIT S/W to experiment with (at AUEB testbed)?
- Is it possible to access the MEC services remotely?
- What is the H/W setup (smartphones, antennas, cameras)?
- What S/W and H/W is available at each testbed? Is the setup identical?
- What H/W should we carry with us when visiting the testbed?

End-user (musician node) setup



TENEMP services/apps (6 applications)

Holographic in: 3D stream encoding service

Holographic out: unity app

Measurement: latency/band width measurement app?

gstreamer

Control plane (init.py): SDP/ICE msg preparation

Data plane (webRTCbin): connect to "SFU"

NMP: Jamulus, sonobus, others

Gstreamer sub-apps:

- Init.py: creates "room"
- Caller.py: joins room
- WebRTCbin: streams data (peer-to-peer)