CloudCom 2026
Oct 27-29, Paris, France



Optimized Task Offloading Using NMSR Priority-Queues for Scientific Stream Processing

Ben Blamey
Malmö University
ben.blamey@mau.se
https://benblamey.com/

Ida-Maria Sintorn, Salman Toor,
& Andreas Hellander,
Uppsala University

MALMÖ UNIVERSITET



UPPSALA UNIVERSITET

Slides © Ben Blamey et al. 2026 (Figures from Paper)

Topics

- Telemedicine
- Microscopy
- Stream Processing
- Edge Computing
- Task Offloading
- Bandwidth Constraints
- Optimization
- Benchmarking

MiniTEM™ (Vironova)

25keV transmission electron microscope.

Microscopic and Sub-Microscope imaging.

Compact, semi-automated, intended for non-experts.

Healthcare Applications:
- Disease Diagnosis
- Drug Development
- Medical Research

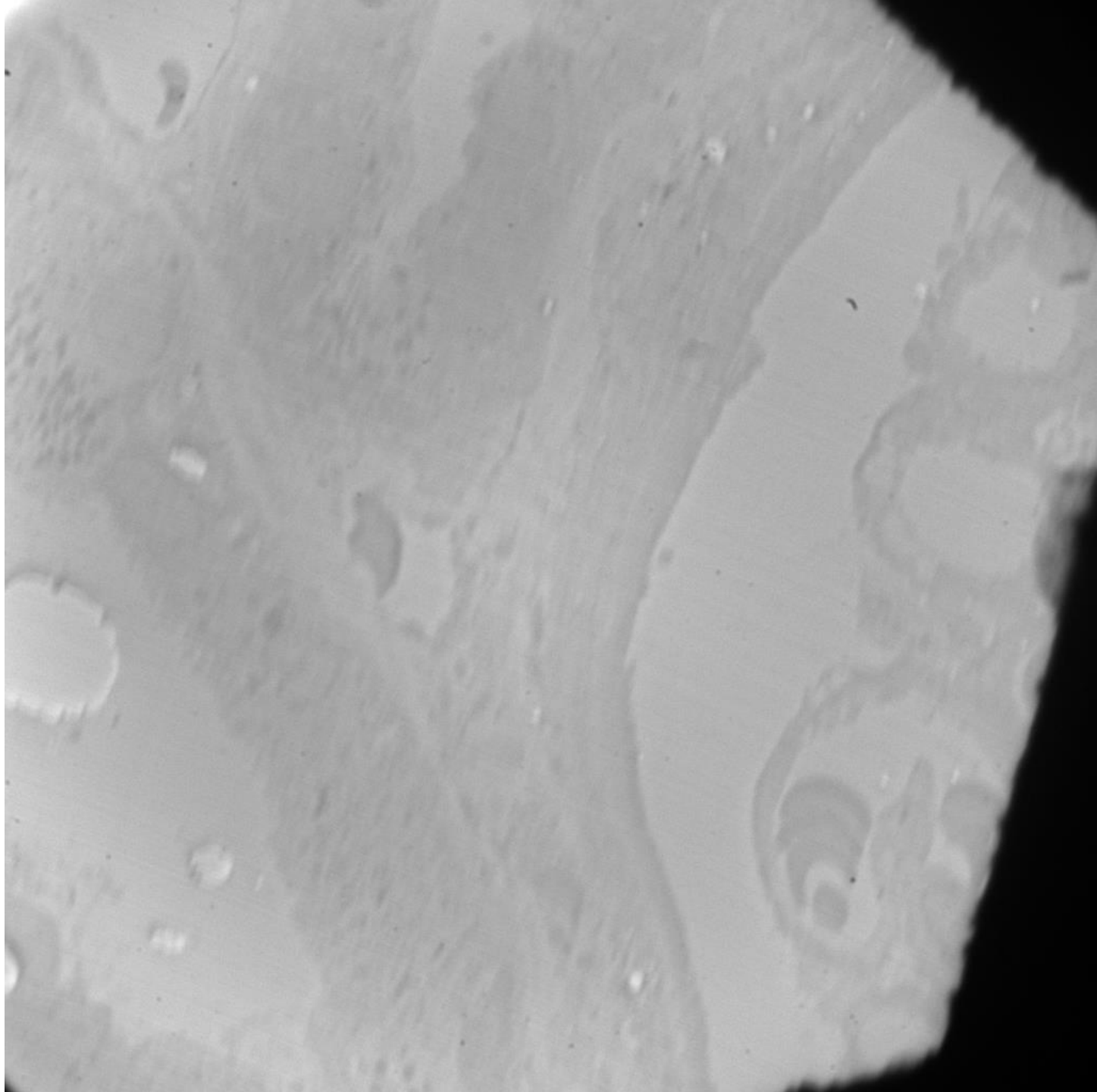


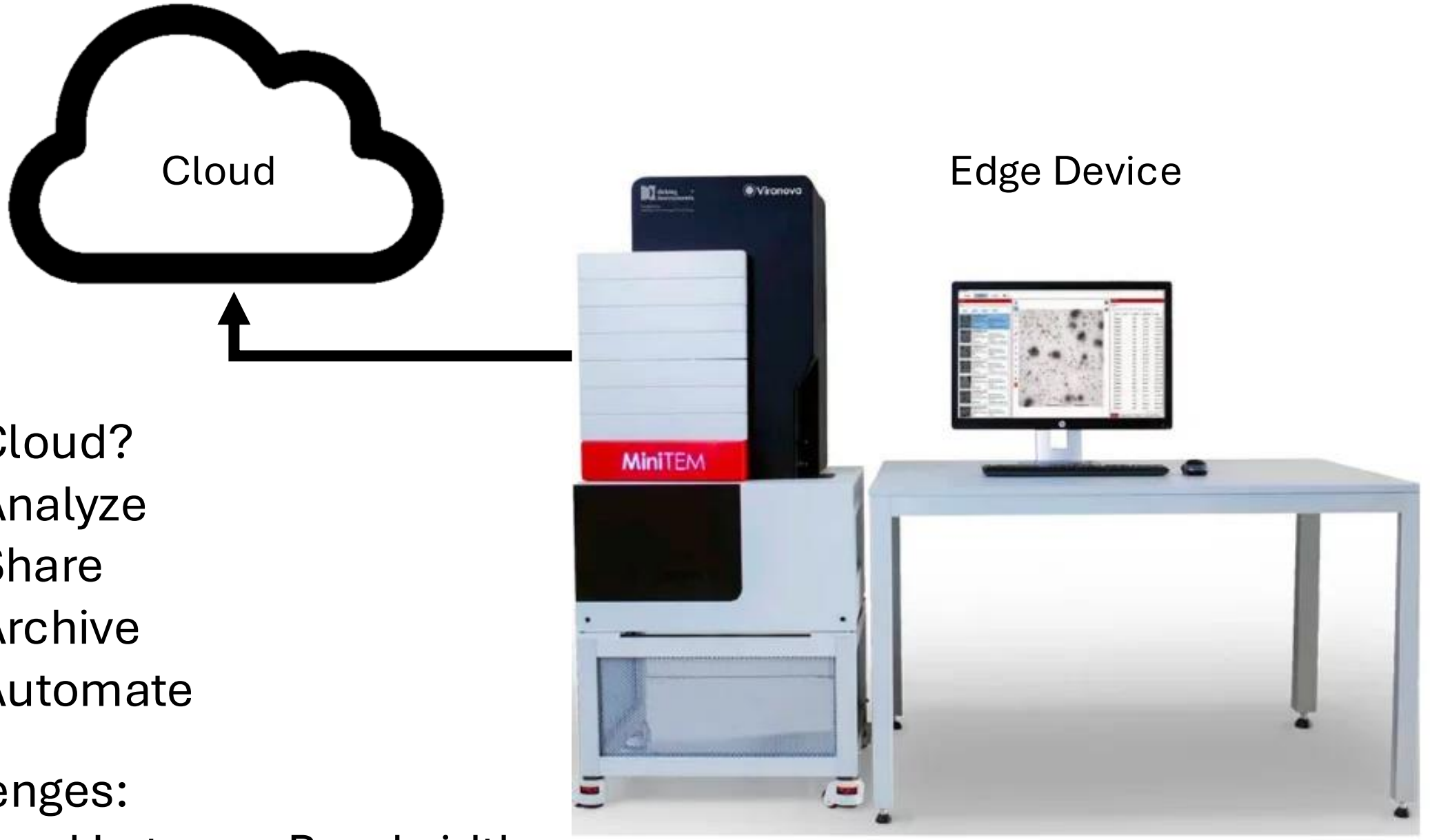
Thin kidney tissue sample, apx. 2mm ⌀, resting on honeycomb mesh.

Before high magnification, operator scans the sample to find a region of interest (759 PNG image dataset).

Goal: use cloud analysis for automated real time control.

⇒ timely image upload





Why Cloud?

✓ To Analyze

✓ To Share

✓ To Archive

✓ To Automate

Challenges:

◆ Upload Latency, Bandwidth

Broadband Access & Health

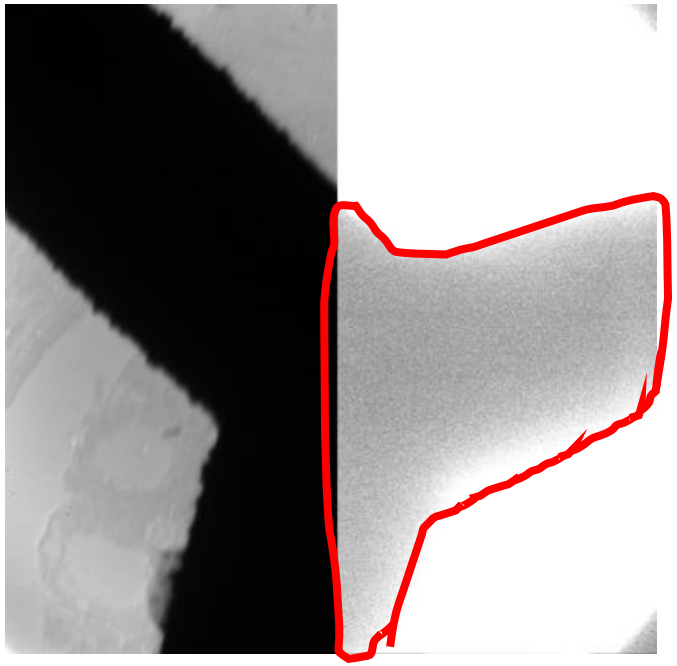
- High-Income Countries
  - Correlation between broadband access and diagnostic lab availability.
  - Non-communicable diseases where broadband is weak.
- Middle-Income Countries
  - Poorly Maintained Infrastructure → low network speeds.
- Low-Income Countries
  - Lack of broadband infrastructure a widespread barrier to telemedicine.
  - Low rate of Healthcare Information Exchange (HIC)

<u>Overall: benefits of healthcare AI don't reach everyone!</u>

- Images are PNG compressed (lossless), don't want lossy compression.
- What else for faster transfer? What about the shadow areas?

Left:

original image -
honeycomb grid
obscuring the sample.



Right:

increased brightness,
noise visible.

- Operator to remove these regions' noise ⇒ ↑ compressibility ⇒ ↓ upload latency

- Edge device ⇒ may not have enough computing power to process all messages.

…. a novel, hybrid **Task Offloading** and **Stream Processing** Optimization Problem

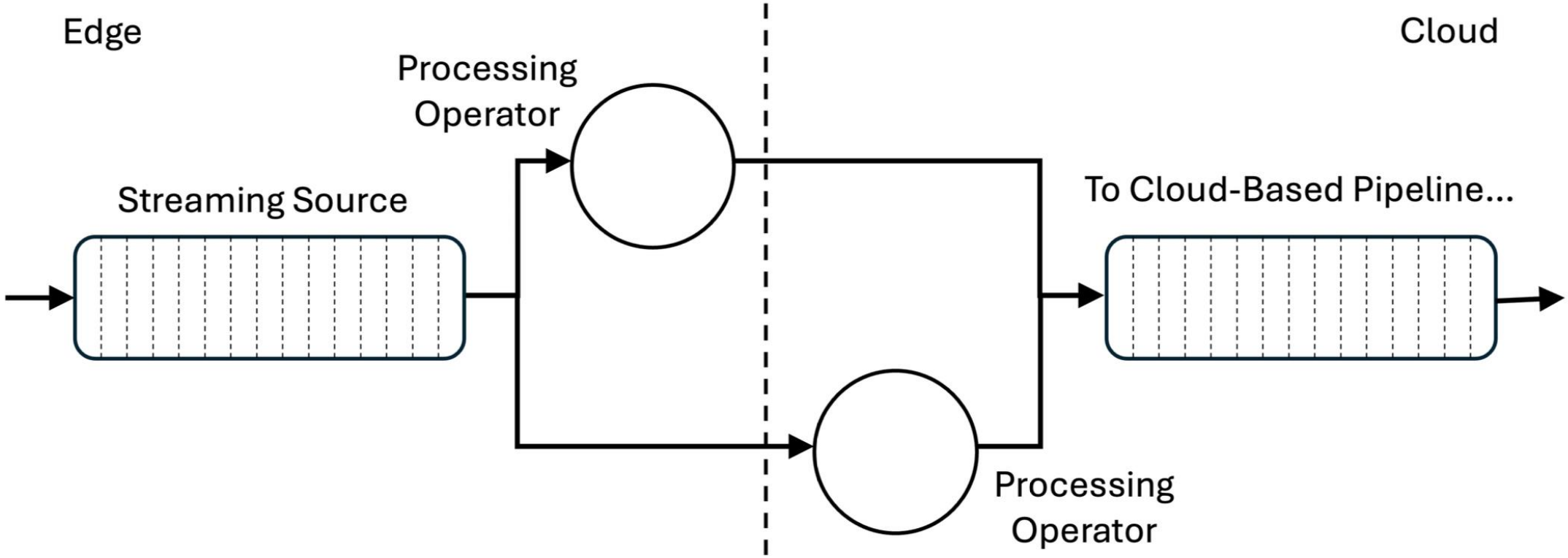


Fig. 1. Logical streaming pipeline used in this study.

Characteristics of both…

Stream Processing:

✓ Messages form a Stream

✓ All Messages are Streamed

✗ Optimization: Operator Placement

✗ Messages are handled homogenously

✗ Messages tend to be small (enterprise)

Task Offloading:

✗ Tasks are Independent

✗ Only Some Tasks Offloaded/Uploaded

✓ Optimization: Task/Message Placement

✓ Per-Task Routing/Decision

✓ Tasks/Messages can be large

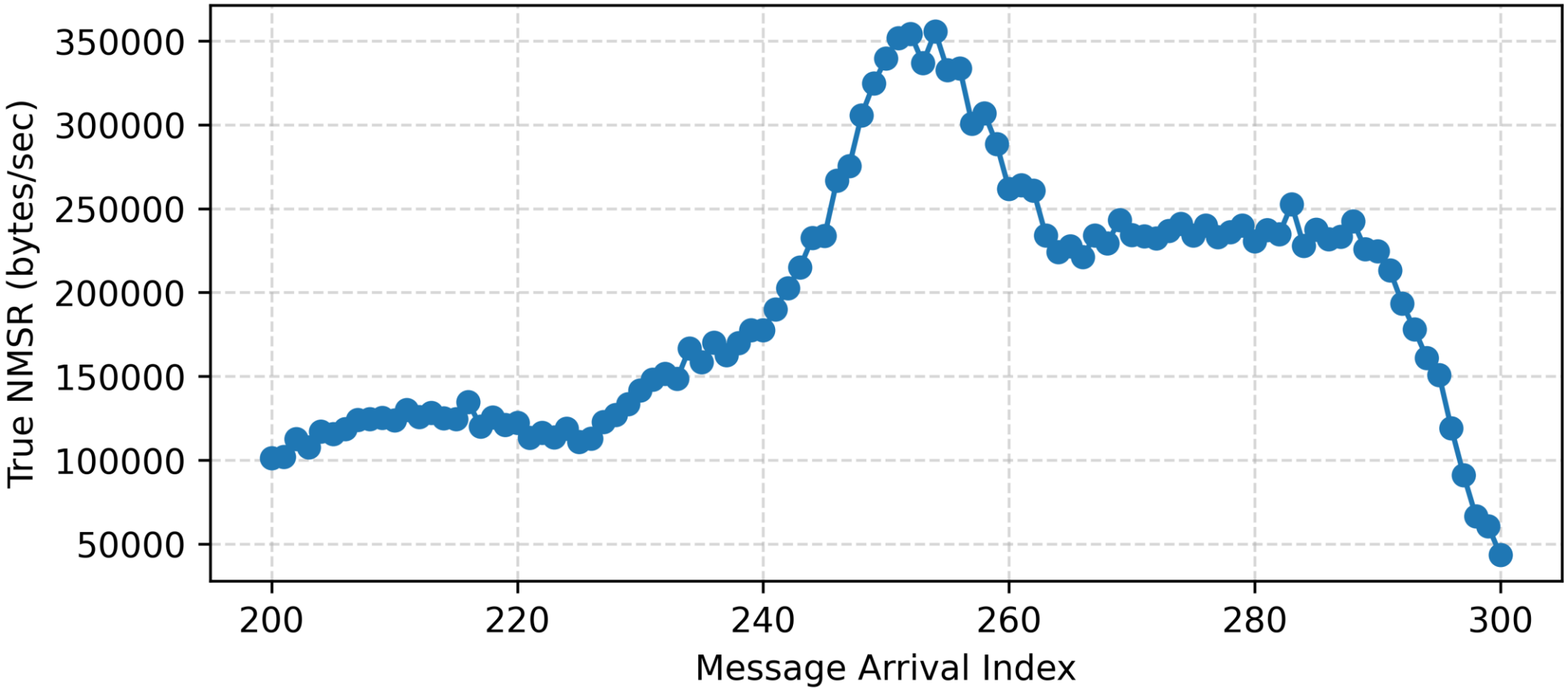
Which messages to proritize for Edge Processing?

....those where limited edge CPU processing power has the most effect on reducing size (and improving upload speed).

Key Idea: ***Normalized Message Size Reduction***

$$\mathrm{NMSR}(i) = \frac{\mathrm{MessageSize}_i^{\mathrm{before}} - \mathrm{MessageSize}_i^{\mathrm{after}}}{\mathrm{ProcessingTime}_i}$$

True NMSR (bytes/sec), selected interval of streamed dataset



Normalized Message Size Reduction (NMSR) Ratio

1. **Dependent** on the data + processing operator.
2. **Observable** - by executing the operator.
3. **Variable**
4. **Noisy**
5. **Required -** for makespan Optimization
6. **Unknown** a-priori ⇒ the ***Task Parameter Prediction Problem***
7. **Smooth** - can be exploited….
8. **Predictable** - using (e.g.) linear splines.

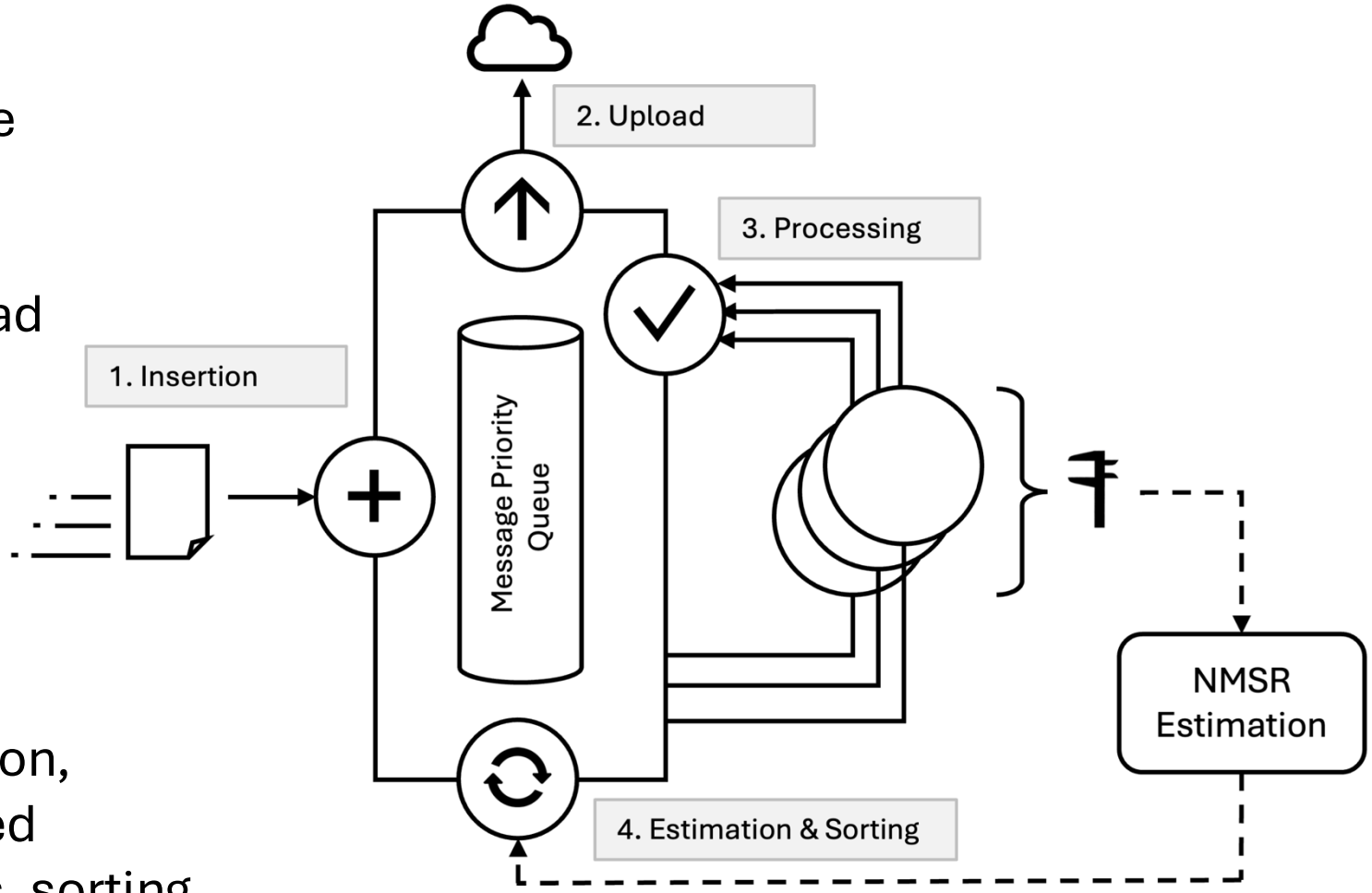
Concurrent Operations on a Priority Queue at the Edge

1. Insertion - messages are added to the queue.

2. Upload - messages selected for upload according to **ascending** NMSR.

3. Processing - messages selected for edge processing based **descending** NMSR.

4. Estimation and Sorting – size reduction, compute time are recorded for processed messages, prediction with linear splines, sorting.



Python Implementation (asyncio):
https://github.com/HASTE-project/haste-agent

Benchmarking Results

TABLE II
EXPERIMENTAL CONFIGURATIONS

Key	Description
`(uponly)`	No processing, original images uploaded. Control.
`(1,r)`	1 core, random prioritization. Baseline.
`(2,r)`	2 cores, random prioritization. Baseline.
`(3,r)`	3 cores, random prioritization. Baseline.
`(1,s)`	1 core, splines priority estimation.
`(2,s)`	2 cores, splines priority estimation.
`(3,s)`	3 cores, splines priority estimation.
`(all)`	All images processed offline a priori. Control.

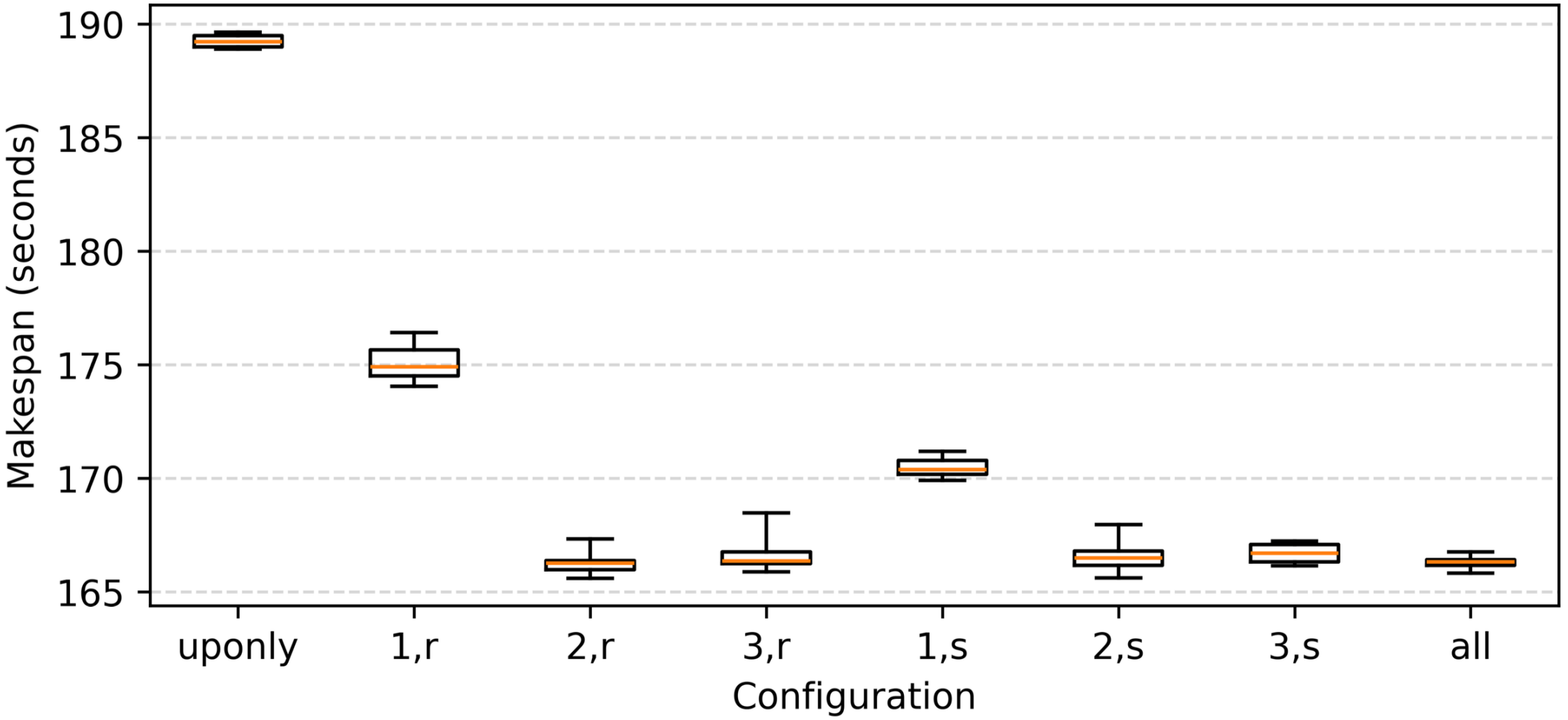


Fig. 5. Makespans (end to end latency, seconds), by experimental configuration

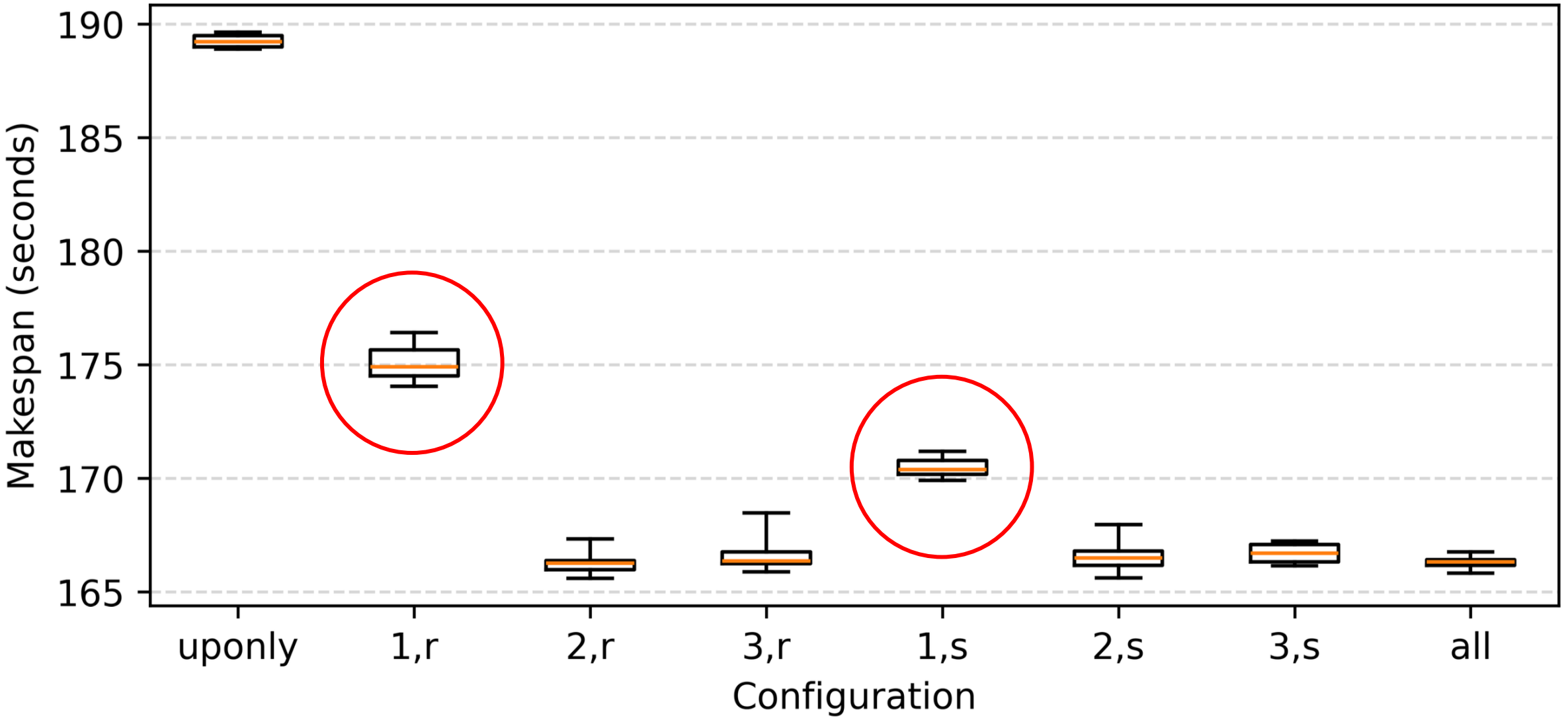


Fig. 5. Makespans (end to end latency, seconds), by experimental configuration

TABLE III

MAKESPANS (IN SECONDS), BY EXPERIMENTAL CONFIGURATION.

Configuration	Min (s)	Max (s)	Mean (s)	SD (s)
(uponly)	188.90	189.63	189.24	0.28
(1,s)	169.90	171.19	170.48	0.44
(2,s)	165.61	167.95	166.58	0.73
(3,s)	166.14	167.24	166.69	0.43
(1,r)	174.04	176.42	175.09	0.83
(2,r)	165.58	167.34	166.29	0.55
(3,r)	165.88	168.47	166.69	0.85
(all)	165.82	166.76	166.30	0.29

Results for (1,s) vs. baseline, (1,r), with Welch's t-test shows statistically significant difference ($p < 10^{-5}$)

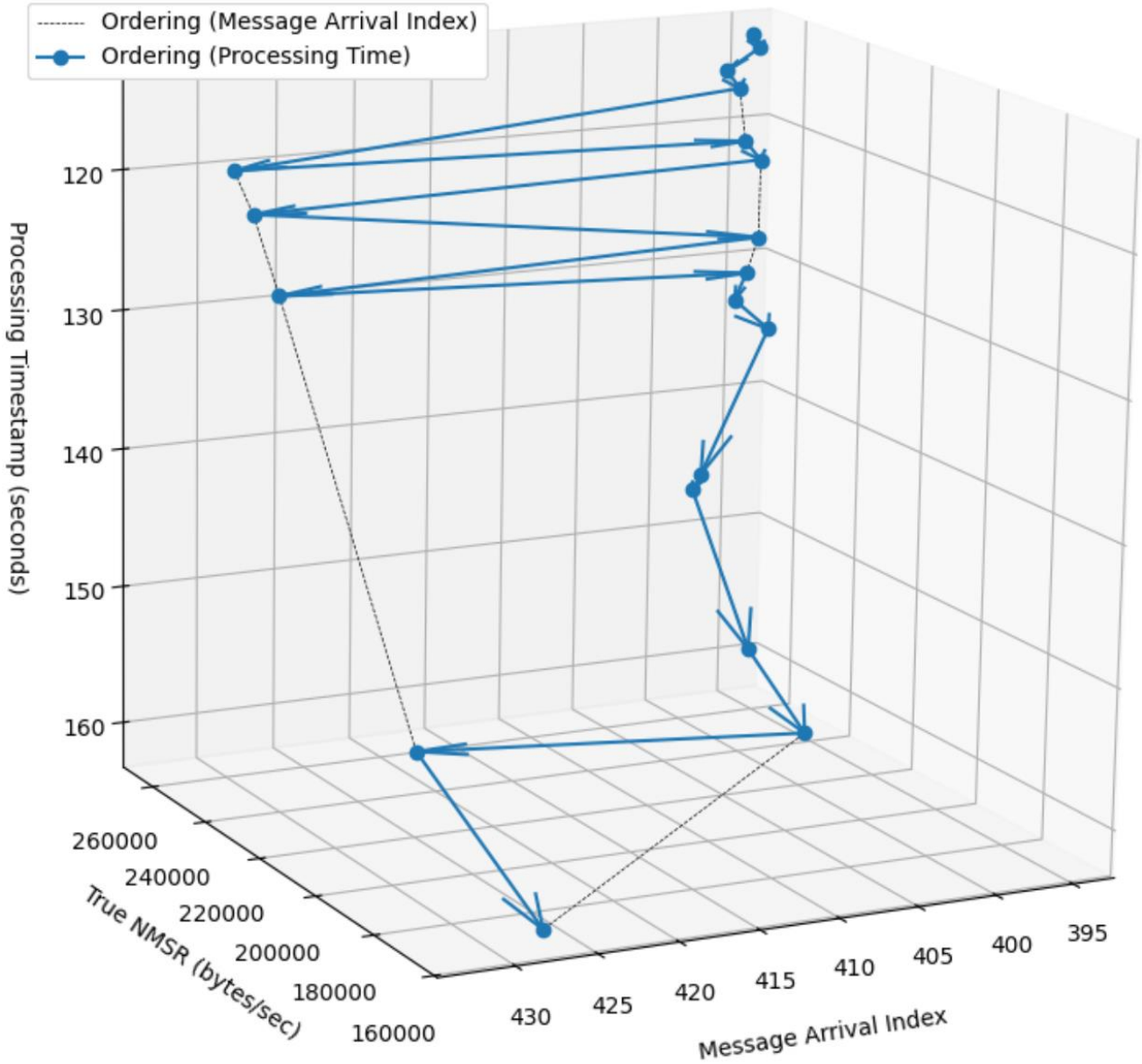


Fig. 6. Message arrival and processing orders, timestamps and arrival index, selected intervals, experimental configuration `(1,s)`.

Applicability of the Approach

- ✓ **Large** Messages (e.g. Scientific Computing Applications)
- ✓ **Constrained** Bandwidth
- ✓ **Processing Operator** which is computationally intensive.
- ✓ **Variance** in parameters (size reduction, message size).
- ✓ **Smoothness** w.r.t. stream index, or other metadata.

Scientific Computing – Message Sizes

Application	Size	Source
Microscopy (tile/chunk)	131KB	$256 \cdot 256 \cdot$ 16-bit [40]
Microscopy (2D frame)	1.76MB	$512 \cdot 1800 \cdot$ 16-bit [41]
Microscopy (multi-channel)	16MB	$(5 + 3) \cdot$ 2MB [42]
Radiography (X-ray)	8MB	$2000 \cdot 2500 \cdot$ 16-bit [43]
Computed Tomography (CT, Slice)	0.5MB	$512 \cdot 512 \cdot$ 16-bit [43]
Magnetic Resonance Imaging (Slice)	0.5MB	$512 \cdot 512 \cdot$ 16-bit [43]
Mass Spectrometry (MS1, profile)	100KB	$25000 \cdot$ 32-bit [44]

TABLE I
TYPICAL UNIT DATA SIZE, VARIOUS INSTRUMENTS

Existing Work often…	**Our Approach…**
Model-Based, Engineering Complexity Ignored	Implementation + Benchmarking
Message Parameters assumed known A-Priori	*Task Parameter Prediction Problem* is part of the system
Assume Complete Knowledge of a Distributed System	No Knowledge of Distributed System
Optimization: Static & Global	Optimization: Dynamic & Local
Optimization Scaling & Cost Ignored	Optimization Computationally Cheap, included in Benchmarking
Unproven Resilience / Error Handling, Dynamic Environment	Simplicity of Queue-Based Approach is Resilient and Flexible
No Data and Source Code	Public Dataset & Source Code

Conclusions

- Task Offloading is relevent to Scientific Computing / Microscopy
- *Task Parameter Prediction Problem* is neglected
- Parameter smoothness in stream can be exploited
- Queue-based approach: effective & robust; no global optimization
- NMSR-priority queue yields a statistically significant reduction in makespan under benchmarking
- Dataset & Source code and are open-source

Questions? Feedback?

ben.blamey@mau.se
www.benblamey.com

Listing 1. NMSR estimation and message selection.

```
on_new_message(index):
 info[index].state ← NOT_PROCESSED

on_message_uploaded(index):
 info[index].state ← UPLOADED
 // choose next message to upload:
 next_index ← any index i | state(i) = PROCESSED
 if !next_index:
   next_index ← i | info[i].state = NOT_PROCESSED and
                    info[i].predicted_NMSR is minimal
 info[next_index].state ← UPLOADING
 return next_index

on_message_processed(index, delta_bytes, cpu_time_sec):
 info[index].state ← PROCESSED
 info[index].true_NMSR ← -delta_bytes / cpu_time_sec

 // re-predict NMSRs using linear splines.
 info[].predicted_NMSR ← compute_splines(info[])

 if (counter++ % N != 0):
  next_index ← i | info[i].state = NOT_PROCESSED and
                   info[i].predicted_NMSR is maximal
 else:
  // sample for new regions of high/low NMSR.
  next_index ← random i | info[i].state = NOT_PROCESSED
 info[next_index].state ← PROCESSING
 return next_index
```