A High-Rate Data Acquisition Protocol with Relative Topology Reconstruction

Blaž Jerman[2], Niki Hrovatin[1,2], and Aleksandar Tošić[1,2]

[1]University of Primorska, Faculty of Mathematics, Natural Sciences and Information Technologies, Glagoljaška 8, 6000 Koper, Slovenia

[2]University of Primorska, Andrej Marušič Institute

May 2026

1 Introduction

Wired sensor networks are used in systems where reliable communication, low delay, and stable timing are important. They are common in areas such as structural health monitoring, smart buildings, industrial monitoring, medical sensing, and dense embedded sensor installations. In these applications, many sensors may need to send measurements frequently and with predictable timing. Wireless networks are often easier to install, but they can suffer from interference, limited battery life, packet loss, and less predictable delay. Because of this, wired networks are still a strong choice when stable and continuous data acquisition is required.

A dense sensor network contains many sensor nodes placed close together. Each node may measure temperature, pressure, acceleration, force, light, or another physical value. When the number of nodes increases, the communication system becomes more important. It is not enough that each sensor works correctly. The network must also collect all sensor data fast enough. If the network is too slow, the data logger receives old data or cannot sample the full system at the required frequency.

Traditional wired communication solutions, such as Modbus RTU over RS-485, are simple and widely used. They are popular because they are known, robust, and supported by many devices. However, these systems usually work with a request-response model. In this model, one master device asks one sensor for data, waits for the response, and then asks the next sensor. This communication style is easy to understand and implement, but it becomes slower as the number of nodes increases. If every device must be contacted separately, the total acquisition time grows with the number of devices.

Another issue is the physical structure of the network. Bus systems can have problems when wiring becomes complex. Long cables, branches, and bad termination can create signal integrity problems. These problems may lead to reflections, distorted signals, and communication errors. In a dense deployment, where many devices must be placed inside a physical structure, it is not always practical to keep the wiring ideal.

A further problem is that many existing systems do not automatically know the physical position or connection structure of the sensors. A device may have an identifier, but the system does not necessarily know where that device is located in the real installation. This means that the user often has to manually map sensor identifiers to

physical locations. For small systems this is acceptable, but for larger systems it becomes difficult and error-prone. If a node is replaced, moved, or connected differently, the map can become wrong.

The work described in this seminar focuses on a high-rate data acquisition protocol with relative topology reconstruction for wired sensor networks. The goal is to provide a simple communication method that can collect data quickly while also reconstructing the relative structure of the network. The system uses UART communication and requires only two UART interfaces per node. It supports both linear and tree-like network structures. Each node can forward data from its child nodes while also adding its own measurement data.

The main idea is that data is collected through a cascading process. A main node triggers the first node in the network. That node sends its already prepared message upward, triggers its child nodes, receives their data, and prepares a new message for the next cycle. This creates a pipeline where data from deeper nodes moves upward step by step. Because the data includes information about child relationships, the logger can reconstruct the relative parent-child topology.

The main goals of the presented approach are:

- to support high-rate data acquisition in dense wired sensor networks,
- to keep the hardware simple by using UART communication,
- to allow flexible linear and tree-like topologies,
- to automatically reconstruct the relative network topology,
- to reduce the need for manual mapping of sensor identifiers to physical positions.

2 Related Work

Several wired communication approaches are used in sensor networks, each with different trade-offs in speed, wiring complexity, cost, and topology support.

Modbus RTU over RS-485 is a common industrial solution for connecting sensors, actuators, and controllers [1]. Its main advantage is simplicity, but it usually follows a request-response model. The master queries devices one by one, so the acquisition time increases as the number of nodes grows [2].

Hermanis et al. [3] present a real-time wired sensor network based on a daisy-chain line topology. This approach is efficient for synchronized data collection, but it is mainly limited to linear layouts. Sun [4] proposes a UART-based bidirectional daisy-chain architecture that supports linear and ring-like communication, but such approaches can require additional communication logic or interfaces.

Loureiro et al. [5] introduce XDense, a wired grid sensor network for dense distributed sensing. Grid-based systems support more complex layouts, but they usually require more communication channels, routing, and synchronization logic.

Table 1 compares these approaches with respect to communication basis, topology support, interface requirements, and topology reconstruction.

Table 1: Comparison of representative wired sensing approaches with respect to communication basis, topology support, interface requirements, and topology reconstruction.

Approach	**Physical / link basis**	**Network topology**	**Per-node interface**	**Topology reconstruction**
Modbus RTU over RS-485 [1, 2]	RS-485	Bus	One RS-485 interface	No
SPI daisy chain [3]	SPI	Linear	One SPI interface	Yes, position in line
Bidirectional UART daisy chain [4]	UART	Linear / ring	Two UART interfaces	Yes, position in line
XDense [5]	UART	Grid	Four full-duplex UART interfaces	No
Presented approach	UART	Linear / tree-like	Two UART interfaces	Yes, relative parent-child reconstruction

The presented approach focuses on a different design point: simple UART-based hardware, support for linear and tree-like networks, predictable communication, and automatic relative topology reconstruction.

3 Protocol

The system consists of a data logger, a main node, and several sensor nodes. The data logger is connected to the main node through a serial connection. The main node starts the data acquisition process and sends the selected acquisition frequency to the network.

Each sensor node can have up to two child nodes. One child is considered the right child and the other the left child. A node receives data from its children, appends its own data, and sends the combined message to its parent. In this way, data moves upward through the network until it reaches the main node and then the data logger.

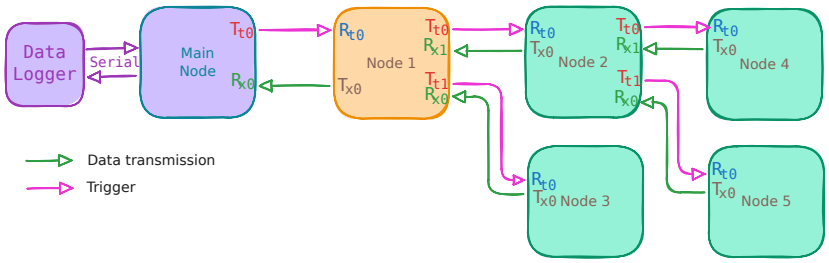


Figure 1: Example wiring of the data logger, main node, and sensor nodes.

The protocol is triggered using a digital signal. When a node is triggered, it first sends the message that it has already prepared. After that, it triggers its child nodes and reads their responses. Then it creates a new message for the next polling cycle. This means that every node always sends data from the previous cycle and prepares new data for the next one.

This behavior creates a pipeline. Data from a deep node does not reach the logger immediately. Instead, it moves upward one level at each polling step. For example, data from a node at depth two first reaches its parent, then the parent's parent, and only after that the main node. Because of this, data from deeper nodes is older than data from nodes closer to the root. The data logger must therefore realign the samples. A sliding window can be used for this purpose. The size of the sliding window depends on the maximum depth of the network.

The basic message format is:

$$M = [ID,\ D,\ R,\ L,\ CRC]$$

Here, ID is the node identifier, D is the sensor payload, R and L indicate the index offsets from the current node to the right and left children (0 indicates no child), and CRC is used for error detection.

At startup, a node has not yet received data from its child nodes, so the message is:

$$M = [ID,\ D,\ R = 0,\ L = 0,\ CRC]$$

If a node has a right child, the message becomes:

$$M = [ID_i,\ D_i,\ R_i = 1,\ L_i = 0,\ CRC_i,\ ID_{i+1},\ D_{i+1},\ R_{i+1} = 0,\ L_{i+1} = 0,\ CRC_{i+1}]$$

For a chain of three nodes, the message expands recursively:

$$\begin{aligned} M = [&ID_i,\ D_i,\ R_i = 1,\ L_i = 0,\ CRC_i,\\ &ID_{i+1},\ D_{i+1},\ R_{i+1} = 1,\ L_{i+1} = 0,\ CRC_{i+1},\\ &ID_{i+2},\ D_{i+2},\ R_{i+2} = 0,\ L_{i+2} = 0,\ CRC_{i+2}] \end{aligned}$$

The fields should be kept small because every byte reduces the maximum acquisition frequency. In a simple implementation, identifiers, flags, and the CRC can each be stored in one byte. The payload size depends on the sensor type and the number of measured values. For example, a node with several analog channels may send a larger payload than a node with one digital sensor.

Algorithm 1 shows the simplified operation of one node.

Algorithm 1 Pseudo-code of the node operation

```
1: M ← []
2: M.append(ID, readSensors(), 0, 0)
3: M.append(crc(M))
4: while true do
5:     if nodeTriggered then
6:         send(M)
7:         rightM ← triggerAndReadRight()          ▷ Returns right child subtree message
8:         leftM ← triggerAndReadLeft()            ▷ Returns left child subtree message
9:         rightIndexOffset ← 0
10:        leftIndexOffset ← 0
11:        if rightM ≠ null then
12:            rightNodeCount ← nodeCount(rightM)
13:            rightIndexOffset ← 1
14:        end if
15:        if leftM ≠ null then
16:            leftIndexOffset ← 1 + rightNodeCount
17:        end if
18:        M ← []                                  ▷ Clear old message
19:        M.append(ID, readSensors(), rightIndexOffset, leftIndexOffset)
20:        M.append(crc(M))
21:        if rightM ≠ null then
22:            M.append(rightM)
23:        end if
24:        if leftM ≠ null then
25:            M.append(leftM)
26:        end if
27:    end if
28: end while
```

4 Relative Topology Reconstruction

The protocol sends all node data as one serialized stream. Because of this, the data logger must reconstruct the network structure from the received sequence. The reconstruction is based on the information inside each node message.

Each node message contains information about whether the node has a right or left child. The data logger reads packets until it detects that the root node has appeared again. This means that one complete network frame has been received. The root node therefore works as a synchronization point.

After a full frame is collected, the logger uses relative indexes to rebuild the network. If a node has a right child, the index tells how far ahead in the received list the right child is located. The same idea is used for the left child. In this way, the algorithm can rebuild the parent-child structure without needing a manually prepared map.

This is useful because the physical network may change. A node may be added, removed, or replaced. In a traditional system, the user would need to update the mapping manually. With relative topology reconstruction, the software can detect the structure from the communication stream. This does not give an absolute physical position in meters, but it gives the connection structure. For many applications, this is already enough because the relative structure explains how nodes are connected.

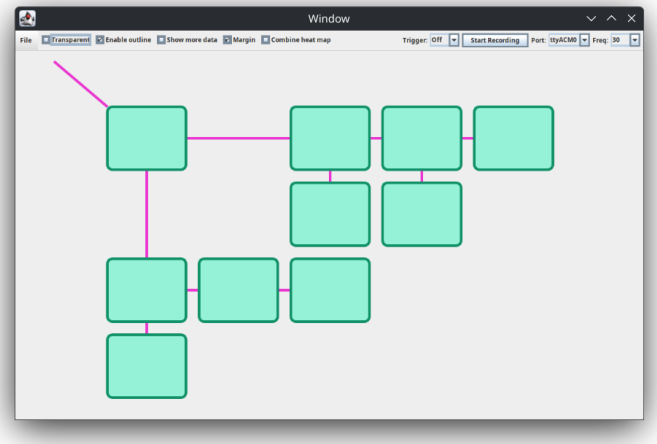


Figure 2: Example of reconstructed parent-child topology. Replace this path with the real figure later.

CRC checking is important during reconstruction. If one packet is corrupted, the reconstructed topology could become wrong. For this reason, the algorithm discards the current frame when a CRC error is detected. It then waits for synchronization again. This prevents invalid data from being used to build the network structure.

Algorithm 2 shows the simplified synchronization and reconstruction process.

Algorithm 2 Synchronization and relative topology reconstruction

```
1: M_L ← length of one node message in bytes
2: nodes ← [ ]                                          ▷ List of received nodes
3: network ← {}                                         ▷ Tree structure of the network
4: while true do

5:     Setup frequency and read the root node
6:     if frequencyUpdated or crcFailed then
7:         serial.write(frequency)                      ▷ Frequency is sent in two bytes
8:         root ← read_packet(M_L)
9:         crcFailed ← false
10:    end if
11:    nodes ← [ ]
12:    nodes.append(root)

13:    Read nodes until the root node appears again
14:    while true do
15:        node ← read_packet(M_L)
16:        if CRC(node) is invalid then
17:            crcFailed ← true
18:            break
19:        end if
20:        if node.id = root.id then
21:            root ← node
22:            break
23:        end if
24:        nodes.append(node)
25:    end while

26:    Construct the relative network topology
27:    for i = 0 to |nodes| − 1 do
28:        if nodes[i].rightIndexOffset > 0 then
29:            r ← i + nodes[i].rightIndexOffset
30:            network.addRightChild(nodes[i], nodes[r])
31:        end if
32:        if nodes[i].leftIndexOffset > 0 then
33:            l ← i + nodes[i].leftIndexOffset
34:            network.addLeftChild(nodes[i], nodes[l])
35:        end if
36:    end for

37:    output: network                                  ▷ Used for storage and visualization
38: end while
```

5 Evaluation

The protocol was evaluated using sensor nodes based on a microcontroller with UART communication. Each node had connectors for the parent node and for up to two child nodes. The experiments tested both linear and tree-like topologies.

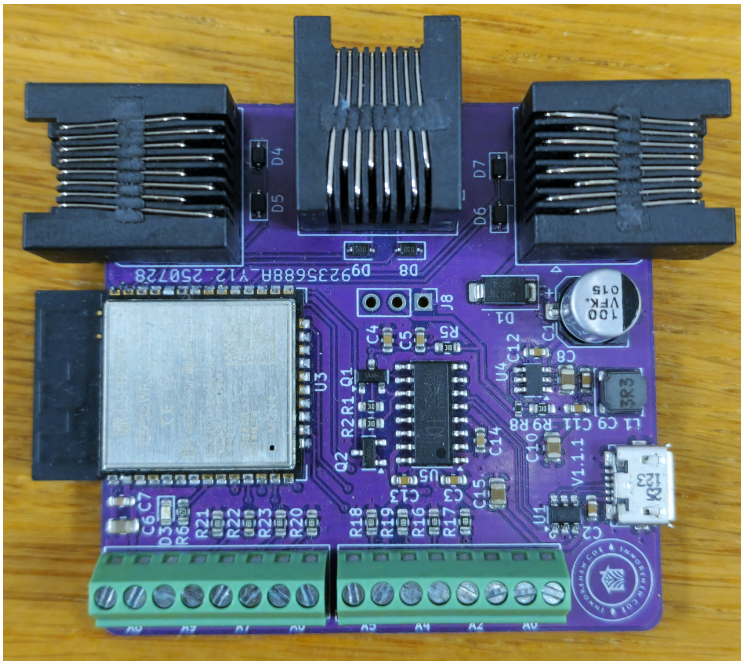
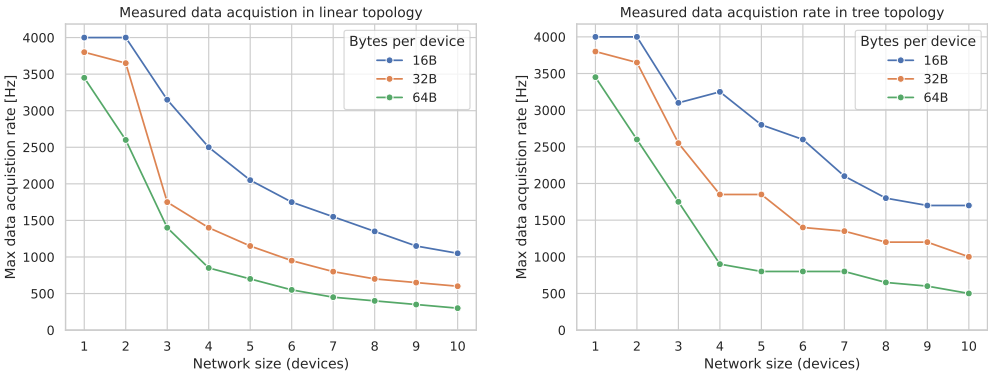


Figure 3: Sensor node used for the experimental setup.

Two main topology types were considered. The first was a linear topology, where every node has only one child. This is the worst case for communication delay, because all data must pass through many intermediate nodes. If the network has ten nodes in a line, the data from the last node must be forwarded through all previous nodes before it reaches the logger.

The second topology was a tree topology, where the network is more balanced. This can improve throughput because data from different branches can be collected in parallel. In a balanced tree, the maximum depth is smaller than in a linear chain with the same number of nodes. This reduces the number of forwarding steps for the deepest nodes.



(a) Measured results of the linear topology.

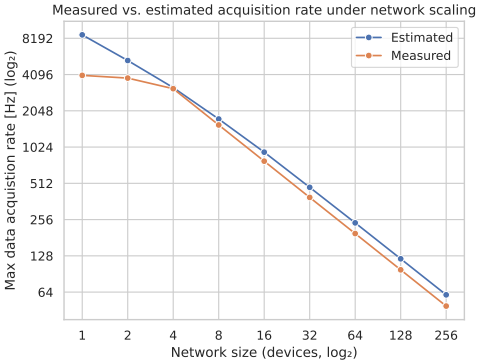
(b) Measured results of the tree topology.

Figure 4: Linear and tree-like testbed topologies.

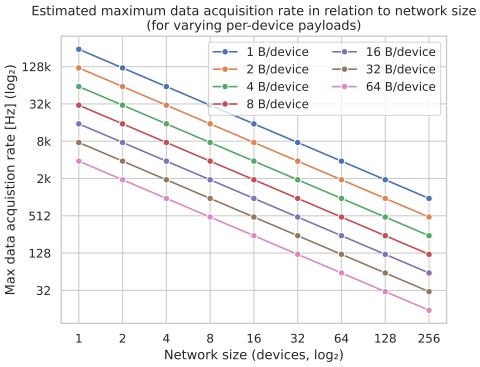
The results showed that the maximum data acquisition rate decreases as the num-

ber of nodes or the payload size increases. This is expected because more bytes must be transmitted through the network. The linear topology had the lowest performance because all data travels through one long chain. The tree topology performed better because both branches can send data in parallel.

The payload size has a strong effect on the maximum sampling frequency. If each node sends only a few bytes, the network can be sampled more quickly. If each node sends many bytes, the transmission time increases. This means that the protocol is especially useful when the payload is small or medium sized, which is common in many sensor systems.



(a) Measured and theoretical maximum data acquisition rates for a linear topology of size n, with each virtual device contributing a 16-byte payload.



(b) Estimations of maximum frequencies per number of devices and varying per-device payloads.

Figure 5: Measured maximum data acquisition rate for different network sizes and payload sizes.

The measurements also showed that the behavior of the protocol can be predicted from the network size, the payload size, and the UART speed. This is useful because it allows a system designer to estimate whether the protocol is suitable for a given sensor network before building the full system. For example, if the required sampling rate is known, the designer can estimate how many nodes can be connected and how large the payload can be.

A practical advantage of this approach is that the system does not need complex networking hardware. Since UART is available on many microcontrollers, the protocol can be implemented using low-cost components. This makes it suitable for school projects, research prototypes, and low-cost sensor installations.

6 Conclusion and Future Work

The seminar described a high-rate data acquisition protocol with relative topology reconstruction for wired sensor networks. The protocol uses UART communication and supports both linear and tree-like structures while requiring only two UART interfaces per node.

The main advantage of the system is that it combines simple hardware with automatic topology reconstruction. This reduces the need for manual mapping of sensor positions and makes the system easier to deploy and maintain. The protocol is especially useful in dense sensor networks where many nodes must be connected with predictable timing.

The evaluation showed that the protocol provides predictable performance. The maximum sampling rate depends mainly on the number of nodes, the payload size, and the network topology. Linear networks are slower because data must pass through every node in the chain, while tree-like networks can achieve better performance because branches can operate in parallel.

The topology reconstruction method also improves usability. Instead of requiring the user to manually describe the network structure, the data logger can reconstruct the relative parent-child connections from the received stream. This makes the system more flexible when nodes are added, removed, or rearranged.

Future work could include testing the protocol in real smart-building or floor-sensor applications. Another possible improvement is differential data transmission, where nodes only send new data when a meaningful change is detected. This could reduce bandwidth usage in systems where measurements change slowly. More advanced error recovery could also be added, so the system can recover faster after a corrupted packet or disconnected node.

References

[1] G. Thomas, "Introduction to the Modbus protocol," *The Extension*, vol. 9, no. 4, pp. 1–4, 2008.

[2] N. C. Gaitan, I. Zagan, and V. G. Gaitan, "Evaluation of the performance of the Modbus RTU communication protocol for consecutive address transactions and improvements related to ModbusE," *The Journal of Supercomputing*, vol. 82, no. 2, p. 71, 2026.

[3] A. Hermanis, R. Cacurs, K. Nesenbergs, and M. Greitans, "Efficient real-time data acquisition of wired sensor network with line topology," in *2013 IEEE Conference on Open Systems*, pp. 133–138, 2013.

[4] X. Sun, "Bidirectional multi-device daisy chain communication architecture and verification based on UART serial communication protocol," *Highlights in Science, Engineering and Technology*, vol. 81, pp. 583–591, 2024.

[5] J. Loureiro, R. Rangarajan, and E. Tovar, "Distributed sensing of fluid dynamic phenomena with the XDense sensor grid network," in *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, pp. 54–59, 2015.