Utilizing Cost-effective NB-IoT-based Sensors for Detecting Water Temperature and Flow

1st Petri Rantanen
Tampere University
Pori, Finland
petri.rantanen@tuni.fi

2nd Jarkko Mäkivaara
AQVA.IO Oy
Pori, Finland
jarkko@aqva.io

3rd Mika Saari
Tampere University
Pori, Finland
mika.saari@tuni.fi

4th Pekka Sillberg
Tampere University
Pori, Finland
pekka.sillberg@tuni.fi

5th Hannu Jaakkola
Tampere University
Pori, Finland
hannu.jaakkola@tuni.fi

***Abstract*—Improperly turned off sink and shower faucets, leaking toilets, and faults in pipes can cause significant expenses in increased water bills, and cause even more serious problems if water finds its way inside building structures. Today, many kinds of sensor systems can be used to send data to the internet from a multitude of environments, and the collected data can be processed with algorithms to find anomalies. This paper presents a prototype for measuring ambient room temperature and water pipe temperatures. As an example case, this paper shows how the wireless temperature measurement prototype can be used to detect water flow within the pipes. The flow detection could be used, for example, for finding faults in water applications and devices. This papers describes the basic operating principle for the prototype, the initial findings, challenges and future directions, and introduces a technical solution for executing the prototype software within an NB-IoT module without the need to utilize a separate microcontroller or small computer.**

***Index Terms*—NB-IoT, water usage, sensors, prototypes**

I. Introduction

Today, different kinds of sensor systems send data to the internet from various environments. The collected data could be processed using algorithms to find anomalies. This study presents one way to collect, process, and visualize simple data to help real property control and maintenance.

The main idea of the research is to collect data from the surface of water pipes by measuring the temperature changes within a certain timeline. The collected data is sent to the cloud service without modification or processing. In the cloud service the data are processed with the main purpose of finding water leaks. Further, the gathered data are visualized for testing purposes.

This study belongs to Internet of Things (IoT) related research, carried out by the Software Engineering and Intelligent Systems (SEIntS) group at Tampere University (TAU), Pori. In our earlier study [1], we focused on rapid prototyping with off-the-shelf devices and open source software. IoT prototypes with wireless sensor networks (WSN) have been built for testing the prototypes and collecting the data.

The Internet of Things (IoT) is the expansion of Internet services, which connects everyday physical objects to a network. The areas of IoT have been clarified in a survey [2]. A crucial part of IoT is the sensor networks and especially Wireless Sensor Networks, whose basic features have been collected in a study [3]. WSN and conventional sensor networks use several possible technologies for data transfer. The low bit rate environmental data (Temperature, humidity e.g.) can use the Low Power Wide Area Network (LPWAN) technologies [4], such as LoRa [5] or NB-IoT [6].

This research project uses cost-effective off-the-shelf devices to build data gathering prototypes. The devices for the prototype are selected with low power features - the sensor nodes are designed to work with batteries. This feature has also affected the software which has been developed for data gathering and transfer. The data transfer from the sensors uses Narrowband Internet of Things (NB-IoT) Low Power Wide Area Network (LPWAN) radio technology. The research study uses new IoT technology for testing with the aim of gathering data. In addition, data processing and the visualization aspect are addressed.

The rest of the paper is structured as follows: Section II discusses the basic operating principle of the prototype system and the collected data. Section III describes the developed data gathering prototype system and its hardware and software components. Section IV discusses the findings made during the research. The testing environment is also described. Finally, section V summarizes the main conclusions of the research.

II. Data

Improperly turned off sink and shower faucets, leaking toilets, and faults in pipes can cause significant expenses in increased water bills, and can even cause more serious problems if water finds its way inside building structures. In our use case, the goal was to detect water usage in the campus of Satakunta University of Applied Sciences located in Rauma, Finland. Nowadays, water leak detectors that can be installed in the main water meters are available from various commercial companies, but these can, in general, only detect overall leaks, and cannot pinpoint the location of the leak - as was the case with the main campus building. Especially in larger building complexes, even a rough estimate of the location of the leak can be helpful. Thus, we started to design a prototype system that would require as few modifications as possible to the existing water systems, and which could be used to estimate where in the building the water consumption (or leak) was taking place. To the best of our knowledge, the building did not have any leaks per se, but these could be simulated through test setups.

The basic operating principle is quite simple, and the technical details for the prototype are described in more detail in section III. The water systems in buildings consist of hot and cold water pipes, which in general run in close proximity to each other. When water flows within the pipes, the temperature of the external surface of the pipe is close to the actual temperature of the water flowing inside the pipe. When the flow stops, the temperature of the pipe will start to rise (in cold water pipes) or drop (in hot water pipes) and will eventually reach the ambient temperature (e.g., room temperature). Thus, by collecting temperature data from both the cold and the hot pipes, and combining it with ambient temperature measurements, it is possible to estimate whether water is flowing inside the pipe. Or, as in this case, to detect water leaks. Accurate temperature sensors are relatively cheap by today's standards, making it possible to install sensors in multiple locations (or on multiple pipes). This would in turn allow splitting larger buildings (or building complexes) into smaller sections, which would help to locate possible leaks - or sensors could be installed in locations that have known water usage problems. Based on discussions with the campus management personnel, the most common problems in the campus were sink faucets in public restrooms that had been only partly turned off. The second most common issues were leaking showers in the gymnasium. Thus, our tests were concentrated in the parts of the building complex where the restrooms were located and in the gymnasium. Unfortunately, because of the ongoing COVID-19 situation, the premises were in minimum use hindering actual usage tests, but on the other hand it allowed for easier implementation of the simulated scenarios.

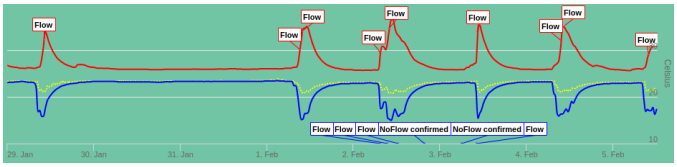

Fig. 1. Visualization of water flow.

Fig. 1 illustrates the water temperature changes within a one-week period. The blue line (bottom) shows the cold water temperature variances (as measured from the surface of the pipe), the yellow line (middle) shows the ambient temperature, and the red line (top) shows the hot water pipe temperature. In the example shown in the figure, the ambient temperature sensor is close to the cold water pipe (within 2 cm), causing the ambient temperature to follow the cold water temperature more closely. The minor changes in the water flow can also be seen in the smaller dips and peaks in the hot and cold water temperature lines (for example, between *2. Feb* and *3. Feb*). The *Flow* and *NoFlow confirmed* markers show automatically detected changes, and in the current implementation they are configurable temperature threshold values.

Parts of the campus building use continuous hot water circulation. In this case, the temperature changes can also be

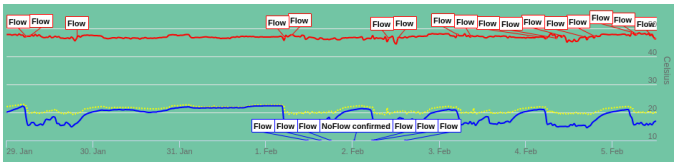

Fig. 2. Visualization of water flow in a system with continuous hot water circulation.

seen, but the changes are much smaller, and require more tweaking of the threshold values. Also, the size (volume) of the pipe could have a bigger impact on the detection accuracy, as larger pipes and smaller flows make detection more challenging. Depending on the case, it could also be possible to measure the cold water pipe temperatures only, as in practice cold water is mixed in the faucet when hot water is used. In other words, the hot water usage could be detected from the cold water usage, and this relation can also be seen in Fig. 2. Of course, this kind of approach would not allow the detection of leaks in the hot water pipes.

Based on our tests, it takes approximately 15 to 60 minutes for the pipe temperature to reach the maximum change after the flow has started (tested with 1 dl/min, 1 l/min, and 3 l/min), and a fast change in the temperature gradient is detectable within the first five to 20 minutes after turning the flow on. The difference in flow volume causes changes in both the time to reach the maximum temperature change and in the final temperature maximum/minimum. For example, the cold water temperature dropped from the stagnant water temperature of 21 degrees to a minimum of 13 to 15 degrees (with a larger flow causing more significant temperature change maxima). After turning the flow off the gradient was much smoother, but still detectable within five to 20 minutes. The water temperature returned to within two degrees of the ambient temperature in about two hours and fully reached the ambient temperature in four to six hours if no further flow was induced.

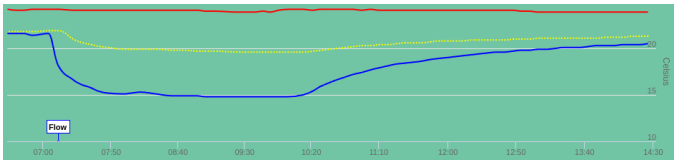

Fig. 3. Visualization of cold water temperature change.

The change in cold water temperature is visualized in more detail in Fig. 3. In the example case shown in the figure, the flow was started at 7:00 and stopped at 10:00. The flow was approximately one liter/min (or 0.95l/min as reported by the main water meter) and the temperature was measured every five minutes from the external surface of the 20 mm diameter copper pipe.

Thus, assuming that the start of a water flow can be detected within 20 minutes and the end of the flow within two hours, it should be possible to detect leaks by observing the temperature changes during the time of day when there should not be any

water usage (outside office/work hours, nighttime, weekends, etc.). Theoretically, normal usage could be separated from the measurements, for example, by utilizing a learning algorithm; however, in our case, we had knowledge of the times of day when there should not have been any water usage in the building.

III. Prototype

The only purpose of the prototype device was to gather sensor readings at predetermined intervals and to upload the measurements to a cloud service. In our use case, only temperature sensors were required. We had previously utilized Raspberry Pi and Arduino for similar measurements [7], [8] and also had a working prototype that used the Microchip AVR ATtiny. For this prototype we decided to attempt an implementation that does not use a separate microcontroller, but where the application code is executed entirely within the NB-IoT modem. This would allow for a smaller device size and theoretically lower power consumption. In our current use cases, mains power was available, but the possibility to run the device entirely on battery power for several years, and the generally lower power consumption were both deemed worthwhile goals.

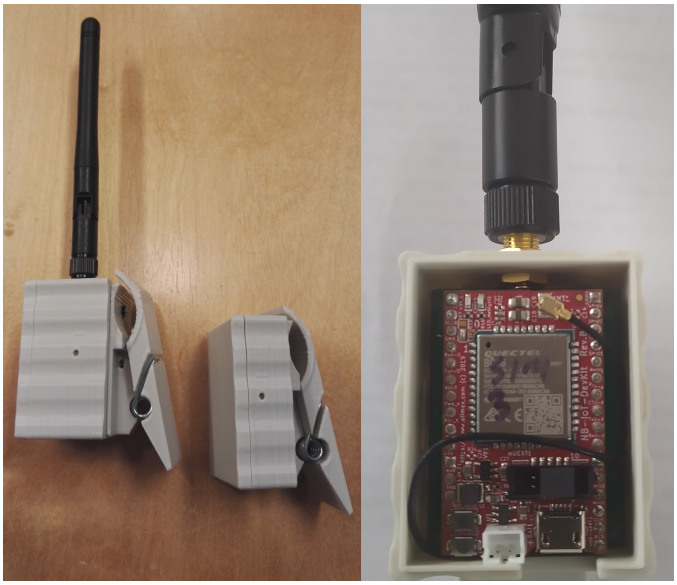

Fig. 4. The 3D-printed cases for the prototype on the left and the inside of the case on the right.

The final prototype can be seen in Fig. 4. The device is approximately 39 x 65 x 50 mm in size (including the pipe connector clip, but not the antenna). The cases are 3D printed. The NB-IoT module is located inside the case, which has the antenna. The clips are used to attach the cases onto water pipes (hot and cold water) and the temperature sensors are located inside the clips and thus, are pressed against the outer surface of the pipes. If the pipes are insulated, this could affect the temperature measurements, so preferably the sensors should be installed onto a part of the pipe where heat can be transferred between the pipe and the surrounding air. The cases are connected to each other by wire. The ambient temperature sensor is located inside the same case that contains the NB-IoT module. The ambient sensor could also be placed in a separate location, but in practice the temperature inside the case is close to the outside temperature. The case is not airtight or insulated, and the temperature produced by running the NB-IoT module is negligible. The close proximity to the water pipe affects the ambient sensor readings, but in practice the effect is only a minor one.

A. Hardware

The hardware inside the casing consists of an NB-IoT DevKit manufactured by Olimex [9]. The DevKit includes connecters commonly found in Raspberry Pi and Arduino (e.g., GPIO, I2C, SPI, UART, USB power) and the embedded multi-band NB-IoT module is a Quectel LTE BC66. The module is controlled either directly by AT commands or by using the Quectel OpenCPU API. The DevKit costs approximately 20 euros, making it very cost-effective even when compared to Arduino. The hardware requires an NB-IoT compatible SIM card. Network operator fees vary between countries, but in terms of total operating costs an NB-IoT can be cheaper than the competing LPWAN (Low Power Wide Area Network) technologies such as SigFox and LoRa. Another advantage of NB-IoT is the direct access to TCP/IP networks (i.e., the internet) without proprietary or licensed middleware. In our area the coverage for NB-IoT was also measured and found to be excellent, with decent transfer rates achieved in all piloting locations (inside and outside buildings).

For temperature measurements we used three DS18B20 sensors. The sensors were connected to the UART (Universal Asynchronous Receiver-Transmitter) pins on the DevKit, and the communication was performed over the 1-Wire protocol. Our original plan was to use the GPIO connectors directly, but after trial and error and lengthy discussions on the Quectel forum we discovered that the OpenCPU's GPIO API performance was too slow to communicate properly with the DS18B20 sensors, requiring re-implementation of the communication protocols [10]. The specifications for DS18B20 promise ±0.5°C accuracy from temperatures from -10°C to +85°C, which was sufficient for our use. Naturally, other comparable temperature sensors could be used as well. The advantage of DS18B20 would be the 1-wire protocol, which in theory allows the use of only a single data wire (and voltage and ground wires) for communication, but unfortunately the issues with GPIO API forced us to implement a more complex setup.

B. Software

The software is executed directly on the NB-IoT module and programmed using the OpenCPU SDK created by Quectel. The code is written in C, although the supported core libraries are slightly more limited than those commonly available with standard C language. The OpenCPU is available for all modules manufactured by Quectel so at least in theory the same code could be run on multiple devices, although testing

this was beyond the scope of our current research focus. The OpenCPU is basically a wrapper for the lower level functions and in many cases a knowledge of AT commands is required even when using the higher level API. The maximum user application size (about 200kB ROM) and RAM (100kB) are quite limited on the BC66 module, and the program code is executed as a single-threaded application. This requires a fairly straightforward application procedure. In our case, the operation is run in eight simple steps:

1) The device wakes up from deep sleep (or starts up for the first time).
2) The device is registered on the NB-IoT network. This may sometimes take a while (up to 20 minutes for the first registration), but in general it takes from three to 20 seconds. The platform has limited support for timers, but as the platform cannot truly run multiple threads, interrupting network registration may cause unforeseeable issues. In practice, this means that if deep sleep is used, real-time measuring can be problematic. In our case, the temperature measurements are read at five-minute intervals, and thus, the minor delay in network registration is not an issue.
3) The IMEI (International Mobile Equipment Identity) and signal quality are read. IMEI is used to identify the devices in the cloud service.
4) The temperature measurements are read from each of the three sensors.
5) The voltage is read.
6) The TCP/IP socket is opened to the cloud service. Temperature, IMEI, voltage, and signal quality are sent to the service. The TCP/IP socket is closed. The data is transferred using a simple HTTP POST with data encoded as URI parameters.
7) The RTC (real-time clock) timer is set up to wake up the device after a predefined interval.
8) The device enters deep sleep (power save) state to conserve power.

In our case, voltage measurements and deep sleep states are not necessarily required as mains power is available (and the power consumption is relatively low even without deep sleep). However, in the future the plan is to support a battery-based approach, and thus, we wanted to test the functionalities in our prototype. A simple skeleton of the application code is also available as open source [10].

IV. Discussion

In our use cases, the primary goal was to detect leaking toilets, and showers and sink faucets that had been improperly turned off, as these were the most common problems in the target buildings. The building complex also had a water leak detector installed in the main water meter, which could detect leaks happening within the entire complex, but could not pinpoint the leaking locations within the complex. Based on our tests, the prototype is able to detect water leaks of about two liters per hour - as tested with polyethylene (PEX-Ax, EN ISO 15875) and (20 mm and 35 mm) and copper pipes. It was found that the system could detect the problem cases we were aiming to resolve, but the accuracy did not enable the reliable detection of minor leaks (very slowly dripping faucets, small leaks in pipes, etc.). In practice, it is challenging to create a leak smaller than two liters per hour by using a properly working faucet, so the system should, at least in theory, detect improper use, but not necessarily minor problems within pipes and applications. The pipes (by size/volume) in our test scenarios were those commonly found in Finnish households and building complexes, so the detection should work in most Finnish buildings, but we did not perform extensive studies on the relation of pipe size to detection accuracy. The detection accuracy could be improved by using more accurate temperature sensors, lowering the measurement interval, and tweaking the temperature thresholds. Also, small leaks may or may not be detected by observing the main water meter, depending on the volume of the pipes, the size of the building complex, and the general consumption of water in the building. In general, the smaller the flow and the larger the pipes, the more difficult the detection, whether our prototype or the main water meter observation is utilized. In our case, the building complex had fairly regular usage, and normally the building would be closed during nighttime - i.e., there should not be significant water usage outside operating hours, making leak detection easier. In buildings that have a constant (changing) water consumption around the clock, detecting leaks can be a fairly complex problem, and utilizing simple temperature detection may not be adequate.

We used a fairly simple algorithm based on pre-configured temperature thresholds discovered by case-specific experimentation. Our target building had air conditioning, which kept the room temperature (and the ambient temperature within the pipe ducts) fairly stable. A more complex algorithm might be required if there are large fluctuations in the ambient temperature, or if the ambient temperature rises close to the temperature of the hot water or drops close to the cold water temperature. Furthermore, in our use case, we were only interested in whether leaks were present, and not in the size of the leak. Whether the precise water flow (l/h) could be detected in practice by measuring the variances in the three temperatures (ambient, hot pipe, cold pipe) is doubtful even if theoretically possible. Nevertheless, sensors installed on pipes could also help to track the water temperature in cases where certain minimum or maximum temperatures are required within the water system, for example, to prevent unwanted bacterial growth in drinking water. However, in our case, we were only interested in detecting water flow.

The NB-IoT module was found to perform sufficiently well to run a simple application code that gathers sensor readings and submits them to a cloud service. Unfortunately, most of the existing wireless modules devices do not include easy-to-use libraries or SDKs for implementing applications directly on the hardware level, requiring the use of a separate microcontroller (e.g., Arduino) or external computer unit (e.g., Raspberry Pi). The OpenCPU APIs are similar to Arduino APIs, but not necessarily identical. There is also a certain

lack of documentation (e.g., error codes) and examples, which can make application design, implementation, debugging, and testing a challenge. Directly using code examples and libraries designed for Arduino may or may not work. Similarly, circuit examples should be carefully studied if adapting from existing Arduino and Raspberry Pi material. For example, after a round of oscilloscope debugging, interference was found in the NB-IoT Devkit's serial data lines, requiring additional resistor filters, even though an identical circuit had worked fine on Raspberry Pi. Furthermore, reading the temperatures occasionally caused the NB-IoT module to become stuck in the wake-up state and refuse to enter the deep sleep state, requiring a module reset to restore normal operation.

Ultimately, all of the issues with reading the temperatures were resolved, but some minor issues still remain with the 1-wire implementation. Namely, as the sensor order in the 1-wire interface can vary (for example, after power outages), it is crucial to identify which sensor is which. Unfortunately, reading the internal identifiers of the dallas sensors does not work reliably - the identifiers occasionally change slightly, perhaps because of unresolved timing issues or other unknown interference in the data lines. Fortunately, detecting the sensors through data analysis is simple - the lowest temperatures are in the cold water pipe sensors, the hottest in the hot water pipe, and the third one is the ambient temperature sensor. Nevertheless, this creates unnecessary complexity for the analysis algorithms. Regardless of the issues, implementing the application directly on the NB-IoT module level seemed like an interesting option. This is especially true if the modules become more powerful in the future and the availability of interfaces that support the writing of code in standard C/C++ or Arduino-C become more common.

V. Summary

This paper presented a cost-effective temperature measurement prototype, which was implemented as a code running directly in an NB-IoT module without an external microcontroller. The prototype was used to detect fluctuations in the temperatures of hot and cold water. The pipe temperatures were compared with the ambient temperature to detect water flow within the pipes. The basic operating principle, an example of data visualization, and a simple program flow for the application were presented. Despite certain technical issues and limitations, the approach was found to be able to detect common water leakage problems - sink and shower faucets that were improperly turned off and leaking toilets.

Acknowledgment

This work is part of the KIEMI project and has been funded by the European Regional Development Fund and the Regional Council of Satakunta.

References

[1] M. Saari, P. Sillberg, J. Grönman, M. Kuusisto, P. Rantanen, H. Jaakkola, and J. Henno, "Reducing energy consumption with IoT prototyping," Acta Polytechnica Hungarica, vol. 16, Issue Number 9, 2019.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Com- puter Networks, vol. 54, issue 15, pp. 2787–2805, 2010.

[3] I. Akyildiz, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," IEEE Communications Magazine, vol. 40, issue 8, pp. 102–114, 2002.

[4] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low Power Wide Area Networks : An Overview," IEEE Commun. Surv. Tutorials, vol. 19, no. 2, pp. 855–873, 2017.

[5] M. Saari, A. M. bin Baharudin, P. Sillberg, S. Hyrynsalmi, and W. Yan, "LoRa - A Survey of Recent Research Trends," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2018.

[6] R. S. Sinha, Y. Wei, and S.-H. Hwang, "A survey on LPWA technology: LoRa and NB-IoT," ICT Express, vol. 3, no. 1, pp. 14–21, 2017.

[7] P. Rantanen and M. Saari, "Towards the utilization of cost-effective off-the-shelf devices for achieving energy savings in existing buildings," Proceedings of 2020 IEEE 10th International Conference on Intelligent Systems (IS20), Varna, Bulgaria, 2020.

[8] M. Saari, A. M. bin Baharudin, P. Sillberg, P. Rantanen, and J. Soini, "Embedded Linux controlled sensor network," 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, pp. 1185-1189, 2016.

[9] Olimex, NB-IoT Development Board with BC-66 Module and Programmer, https://www.olimex.com/Products/IoT/NB-IoT/NB-IoT-DevKit/open-source-hardware, Retrieved March 18, 2021.

[10] KIEMI Project GitHub Repository, https://github.com/otula/kiemi, Retrieved March 18, 2021.