Temperature Monitoring Network
02220 Distributed Systems

Justas Poderys
s120944

Hans-Jacob Sørensen Enemark
s122509

May 27, 2013

Contents

1 Introduction

Sometimes grasping the big picture of a situation means piecing it together from very small parts. This could be as diverse as how a structure is reacting to an earthquake, the currents movements in the oceans, the change of weather in an entire country or something as volatile as the conditions inside a tornado. Even though the conditions and situations are different it all boils down to collecting specific data from individual locations at certain time intervals and then process it for further study.

In order to deliver each datapoint it is necessary to set up a certain type of infrastructure and software architecture. This comprises a medium through which the data is to be send and a piece of software to handle the sending and recieving of the data - the middleware.

This project is focused on the middleware. With a layered approached this middleware will seperate the data transfer and the coordination into two different parts. This way the transport medium could be replaced to fit the context of the application. In the given situation the data transfer layer will use TCP as a reliable channel.

In the context of monitoring the temperature in a given situation the data is the temperature value and its origin - time and place. The 'place' in this project is the ID of the sensor that captured the temperature value.

Each sensor node will be able to take one of two roles - normal or admin. In *normal* mode the node sends out the data periodically, and in *admin* mode the data is collected and organised. A 3^{rd} role in the system is the *user* who can request the average temperature from the current admin node and promote a normal node to admin.

The goal of this project is to get a current status of the temperature in a fictive scenario using distributed concepts. The proposed sulution is designed for a Linux context and implemented using the C programming language.

2 Proposed solution

With the objective of monitoring the temperature of a given area - e. g. a piggery, the scenario is as follows: A number of similar sensor nodes is distributed around the area each connected to the same network. This network is in this project capable of providing a reliable channel through TCP. A *user*-entity chooses one sensor node to be the admin node.

As depicted in Figure 1 the execution of the system can be devided into 3 main categories:

Reporting Regular nodes reports their temperature to the admin node periodically.

User request Upon request, the admin node evaluates the average temperature from the available data that is not too old and returns the result.

Promote new master When an admin node crashes or other wise fall out of service, the user then selects and promotes a regular node, which in turn notifies the remaining regular nodes of the promotion.

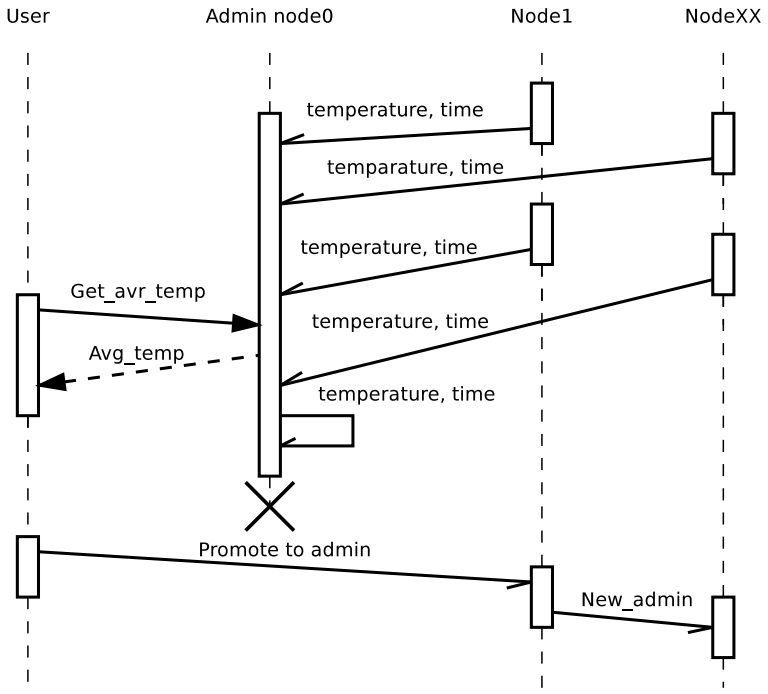


Figure 1: The most prominent messages in the system

2.1 Assumptions

The following assumptions are made about the context of the system:

- All nodes are syncronized within a certain margin. This is done to be able to timestamp each temperature value and provide the user with an up to date picture of the temperature situation. [INSERT SYNC METHOD]
- Reliable channels. Using the TCP protocol each data point is assumed delivered after it has been sent.
- Secure channels. No data is corrupted or generated by outsiders with malicious intend.
- Data integrity. The data received is assumed to be the data that is sent. No data is generated out of nowhere.
- IP addresses of all nodes are known to the user in before deployment of each node.
- Channels are asynchronous and message can take arbitrary time to arrive.

3 Design

In the overall design a modular approach has been chosen. This is done to devide the code into logic pieces, each with a clear cut role - much similar to the object oriented style from C++ or Java. Following this pattern makes the code easier to grasp - and to re-use for future use.

To keep the system going the messages are passed between the sensor nodes and the user. These messages are showvn in Figure 2.

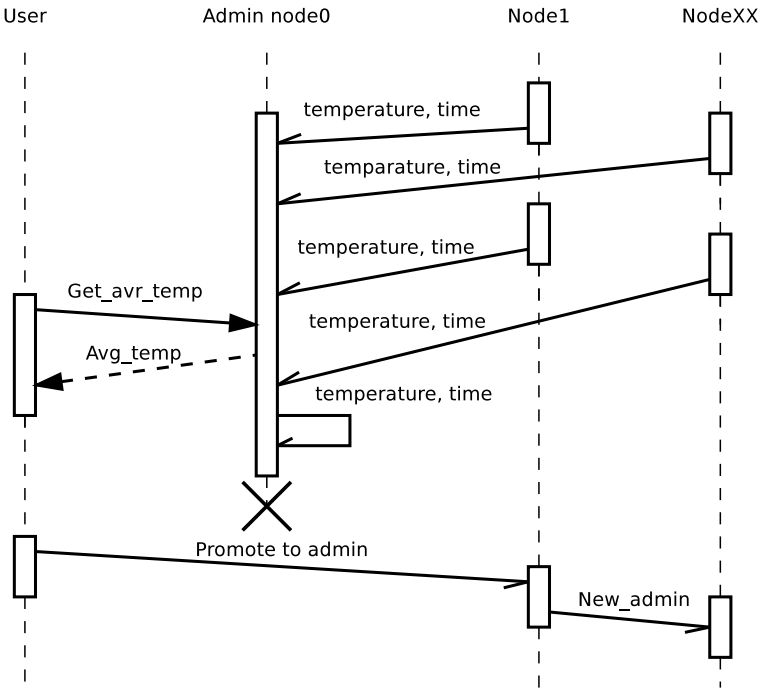


Figure 2: Messages in the system

These messages are sent using *exactly once* semantics, through TCP. This way the application level software only has to hand over the message to the transport layer and it will be sent reliably.

As it is shown in Figure 2 every node has to fill the role as a sensor. That means that the admin node has to have two roles. To achieve this the sensor software is devided into three parts - one for each of the jobs mentioned in section 2. With a clear seperation it is possible for the admin node to send temperature data to itself and thus keep working with two roles.

4 Implementation

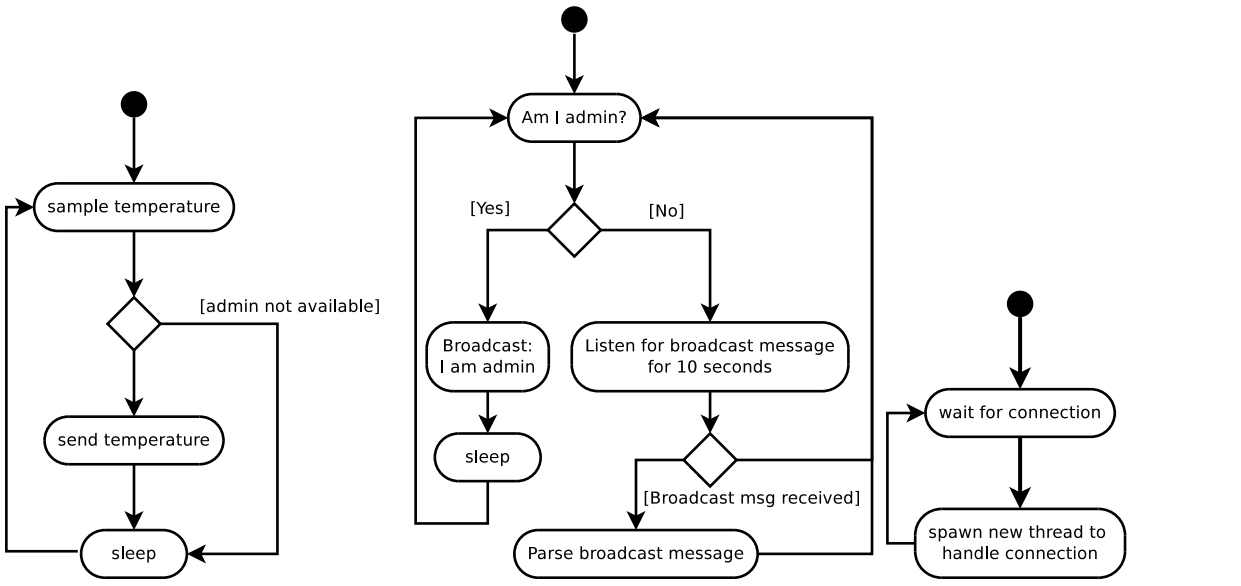
In the following relevant implementation details will be discussed for each of the main parts of the system.

The software is to be written using the C programming language. This choice is made based on the familiarity with the language and freedom it provides. Using this language makes the transition into a real world scenario with an embedded platform as the heart of the sensor rather trivial.

4.1 Sensor node with dual roles

The dual role functionality requires threads, with each thread to handle a particular task. The functionality of each thread is described in section 4.1 and will be implemented using pthreads.

One main thread spawns the other threads and is left out. With this structure a node can take two roles - both collecting temperature data and send it to the admin, and receiving data as an admin and process it. Thus sending data to itself as needed.



(a) Temperature thread (b) Administrator thread (c) Data handler thread

Figure 3: Flow charts on each thread in the sensor

4.2 User entity

The user piece of software is a straight forward command line tool. Listing the available options with numbers provides the user with a way to interact with it. Interaction is done by entering a number corresponding to the function and pressing the return key. Main functions are:

1. **Quit:** Ends the program.
2. **Connect to node:** Asks for an IP address and opens a connection to that IP.
3. **Promote to Admin node:** Initiates the promotion process on the selected node. The admin node is provided a key, that is unique to the admin.
4. **Get average T from node:** Requests the average temperature from the available data on the admin node.

The main structure of this program is a simple while()-loop that prints out the menu-options and waits for input on *stdin*. When an input is provided, a switch case evaluates the input and acts accordingly.

4.3 Message Passing

To transfer messages between the node IP protocol is used. On top of IP protocol, TCP and UDP protocols are used for connection management. TCP protocol is used to provide reliable message passing between user and nodes in the system and between nodes themselves. UDP protocol is used to provide broadcast communications. Due to the fact that UDP protocol does not guarantee delivery of messages, broadcast messages are repeated at set time intervals. UDP broadcast solutions was chosen due to the simplicity of implementation.

4.4 IP Version agnosticism

Implementation of the sensor system is mostly IP version agnostic. It was implemented this way due to the fact that networks are transitioning from the usage of IP version 4 to IP version 6 protocol. Converting the system to be fully compliant with IP version 6 would require minimal changes in source code. The only change that has to be done is to change IP address that is used for broadcasting. At the moment broadcasts are sent to the local broadcast address. This broadcasting method is not supported in IP version 6. In IPv6 native systems, broadcast messages should be sent to FF02:0:0:0:0:0:0:1 address which is reserverd for the multicast "All nodes address" communication. Decision to leave IP version 4 method of communication was done mainly due to the fact that IPv4 systems were used for testing.

4.5 Messages format

To provide scalable solution for message passing, Type-Length-Value (TLV) messages format was used. In TLV messages, each message is encoded in a tuple {Type, Length, Value}. Type and Length are integers, first one indicating what kind of message it is and second how long is the Value. TLV encodded messages are easy to serialize and deserialize for sending and later can be easily parsed as a linked list, becuase all types are known beforehand and length of each message is 2 times length of integer plus value of Length integer. If new message type has to be added, it is enough to add new Type value and update message parsing code with new message type. Serialization, deserialization and parsing code is same regardless of place where it is used - user node, sensor node or master node.

4.6 Time offset calculation

It is assumed in the system model used, that the communication channels are asynchronous. In temperature monitoring system this means that temperature value read in one node can take arbitrary time to reach master node. To solve this issue, time offset calculation is done. When node learns about new master node, it calculates the time offset from node local clock to the master local clock. This time offset then is used to adjust the local timestamp when reporting time when temperature reading was done to the master node. Further more, because all temperature values are timestamped with admin local time, it is easy for admin node to discard old messages. Time offset calculation is done using Network Time Protocol on-wire protocol. Ilustration of this protocol is provided in Figure 4

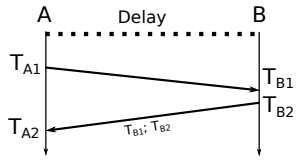


Figure 4: Offset calculation protocol

Protocol operation starts when peer A sends packet at time T_{A1} to peer B. Upon reception of this packet, node creates timestamp when packet was received T_{B1}. Then it constructs the reply packet. Receive timestamp is time when packet was received at a server T_{B1}. Finally transmit timestamp T_{B2} is added just before sending packet back to peer A. Upon reception of this reply message, peer A creates a timestamp T_{A2} to indicate when packet was received. At this point peer A will calculate offset

$\theta = \frac{(T_{B1}-T_{A1})+(T_{B2}-T_{A2})}{2}$

This offset is later used to adjust local timestamp when sending messages to master node.

4.7 Threading

Temperature sensors in this project are implemented using POSIX-threading (Pthreads). Pthreads are used to provide logical separation of tasks in each node. There are separate threads for data messages handling, temperature handling and broadcast messages handling. Using pthreads allowed to achieve error separation (error in one thread logic, will not affect other thread) and also easier workload distribution between programmers. Second place where pthreads are used is handling of multiple connections. Many application written in C use fork() system call to handle connection, but this was not an option for this implementation, because of the need of global variables visible from all threads.

5 Test of the system

In the following first the key test setup and then the main test objectives is described.

The sensor network is tested in a Linux environment on two laptops running Ubuntu. These two laptops are interconnected with an ethernet cable. For further details on the setup see A.1.

The testing is done to document that the functionality is as expected and acts as outlined in the project description. To test this, the following steps is executed:

1. A number of instances of the sensor program is started
2. The user program is started seperately.
3. Promote one sensor to admin.

This scenario will test that everything works initially and that every part of the software runs. After this startup procedure a random non-admin node is stopped and started again to show that it will find the admin node again. Then the admin node is stopped and a new one is promoted,

to show that the sensor nodes can switch admin node without userinteraction. Terminal output is shown in section A.3.

This will show a working system that fulfills the requirements outlined in the project description.

6 Conclusion

During a time of this project, a temperature monitoring and data gathering solution was implemented. Solution is designed to work in a distributes monitoring enviroment and to cope with simple failures - like delayed messages or several admin nodes working at the same time. Solution was implemented using a C programming language and a PThreads library for workloads separation. Command line user interface program was also implemented to promote nodes to master state and later to get average temperature readings. Solution was tested on two computers connected by Ethernet connection and each computer running 3 sensor nodes. No deviation from specification was noticed during a testing runs of a system.

6.1 Responsibilities

Program was written by both students. Data storrage, access, filtering and updating functions were written mostly by Hans-Jakob and network sockets processing was mostly written by Justas. Final report was a joint work of both Hans-Jakob and Justas.

A Test setup and execution

A.1 Test setup

To prepare a computer for testing the sensor network it is necessary to setup extra network interfaces. In Ubuntu this is done with the following command:

~# ipconfig eth0:x <new_ip> netmask <netmask>

In this situation the ethernet port `eth0` is used for the testing and only because it is not used for anything else. The `x` is the desired number for the new pseudo interface. `new_ip` is the desired new IP address and the `netmask` is the netmask used. In the test one interface was set up using the following command:

~# ipconfig eth0:1 10.10.10.1 netmask 255.255.255.0

This command is executed as many times as desired to form the basis for the test.

A.2 Test execution

After the preparations is done it is now possible to run a number of instances of the `sensor` program. Each instance is tied to one interface, hence the number of interfaces is the maximum number of possible instances. The `sensor` is executed with the following command:

~# ./sensor v4 <ip>

Where `ip` is any of the available IP adresses set up earlier and `v4` indicates that an IPv4 address will be provided.

The following will describe a number of test cases to show the functionality of the system.

A.2.1 Start up

Whith the sensor instances running one `user` is needed to promote one sensor node to admin. This is done with the following:

1. Start the user

   ~# ./user

2. Enter `2` for Connect to node.

3. Enter IP address of a sensor node.

4. Enter `3` for Promote node.

After this test a number of sensor nodes is running of which one has been promoted to admin node and the user program is running.

A.2.2 Adding a sensor node to the network

This test assumes completion of section A.2.1 and a running sensor network with the admin node functioning.

The only step in this test is to start up a new instance of the sensor program and see that it finds the admin node and starts transmitting temperature data.

A.2.3 Getting a new admin node

This test assumes completion of section A.2.1.

1. Stop the admin node instance

2. With the user promote another sensor to admin node as in section A.2.1.

All working sensors will now receive the broadcast message from the new admin node and start transmitting data to the new admin.

A.2.4 Fetching the average temperature of all nodes

This test assumes completion of section A.2.1. All interaction in this test is done with the user program.

1. Enter `2` for Connect to (admin) node.
2. Enter IP address on the admin node.
3. Enter `4` for Get average T from node.

The user should now see a temperature value in the terminal.

A.3 Test output

—- put test output here —-