Multi-Functional Platform for Indoor and Outdoor Monitoring

Frederico Gonçalves
frederico.goncalves@ist.utl.pt

Instituto Superior Técnico
Av. Prof. Dr. Aníbal Cavaco Silva
Lisbon, Portugal

Abstract. Wireless sensor networks have revolutionized the industry and science. In particular, these networks play a crucial role in monitoring indoor and outdoor events. While there are several solutions that solve multiple problems encountered in the field of monitoring with wireless sensor networks, many prove to be too specific. This makes it difficult to adapt applications for various types of monitoring. This thesis proposes a unique platform for sensor networks for monitoring events, allowing easy implementation of applications for this area and which is extensible and reconfigurable as much as possible. It also proposes a synchronization and aggregation protocol which can be used with the platform.

Keywords: Monitoring systems, wireless sensor networks, synchronization, in-network aggregation.

1 Introduction

In the last few decades, technological evolution has provided us with smaller equipments with higher computational resources and a broad range of wireless interfaces. Wireless Sensor Network (WSN) have become more powerful providing means to develop a broad range of applications. Indoor and outdoor monitoring systems are one of the many examples of applications using WSNs [1,2,3,4,5]. Typically each case gives origin to a different monitoring system, which contributes to increase the developing time and the overall cost. Also, it is difficult to adapt already developed applications to new scenarios. A paradigm that enables buildings with sensing capacities is pervasive computing. In this paradigm, spaces need to be filled with sensor networks which continuously monitor users in order to act accordingly with their needs [6]. Outdoor monitoring has been in the WSNs application domain for several years and many solutions have been developed. However, each solution is too specific and difficult to port to other scenarios.

Therefore, the need for a common platform capable of supporting multiple applications and requirements for monitoring systems can be satisfied with modern hardware and software. Such platform must be capable of supporting both indoor and outdoor monitoring requirements and ease the developing of new applications for such scenarios.

2 State of The Art

This section provides a study in the state of the art in synchronization protocols, in-network aggregation techniques and data transport techniques for WSNs, which are the focus of this thesis and should be provided in monitoring systems for WSNs.

Synchronization Protocols

Time synchronization has been studied and applied in distributed systems for several years. Solutions such as Network Time Protocol (NTP) [7] are well adapted to such systems. However, they are not suitable for WSNs [8].

Several solutions have been developed which address this issue. Solutions which rely on broadcast synchronization beacons include *Post-facto* synchronization [9], Reference Broadcast Synchronization (RBS) [10] and Flooding Time Synchronization Protocol (FTSP) [11]. There are also solutions which synchronize nodes using two way synchronization, that is, with every message being acknowledged in order to estimate Round Trip Time (RTT). An example of this is a protocol specially designed for mobile ad-hoc networks [12]. Finally, some solutions synchronize nodes using only one way synchronization [13].

Data Aggregation Techniques

Aggregation emerges as one of the most studied techniques to reduce energy consumption in sensor networks [14]. Data aggregation at the network level is known as in-network aggregation.

Several solutions have been developed which explore in-network aggregation. Solutions which are specific for each application and based in query languages include TAG [15], Cougar [16], TiNA [17] and Data Aggregation and Dilution by Modulus Addressing (DADMA) [18]. These solutions are too specific and some work as been done to implement generic aggregation techniques, such as AIDA [19].

Transport in Wireless Sensor Networks

In WSNs, primary traffic flows in the upstream direction from the nodes to the sink [20]. Applications normally collect data and send it at a predefined rate. Some applications do not need reliability when sending this data. However, some do require such property. The problem in WSNs is that the percentage of packet loss is much higher than in cabled networks, thus making reliability more difficult to implement. Adding to this fact, near the sink node congestion might become an issue if applications send data at high rates [21].

The typical method for loss detection is the usage of sequence numbers. This scheme is a receiver-based loss detection mechanism. It is also possible to have loss detection implemented as a sender-based mechanism. In this scheme, packet loss is typically detected through timeouts or overhearing mechanisms.

Loss detection can further be classified as end-to-end or hop-by-hop. Loss recovery can also happen on end-to-end basis or hop-by-hop basis. Essential three schemes exist of receiver-based loss recovery. These are Acknowledge (ACK), Not Acknowledge (NACK) and Implicit Acknowledge (IACK) schemes. ACK schemes are worse than NACK schemes in WSNs, because the rate of success delivery is greater than the rate of packet loss [22]. In contrast, IACK schemes would be the best choice as they do not create extra control messages.

3 System Architecture

The communication platform incorporates three different modules. A configuration module, which configures the system and the modules. A generic handler factory, which is responsible for creating instances of handlers with specific transport protocols. A traffic manipulation module, which provides means to inspect and alter traffic.

Figure 1 shows a detailed overview of the proposed platform. It highlights how modules interact with each other and how handler instances communicate with the platform.

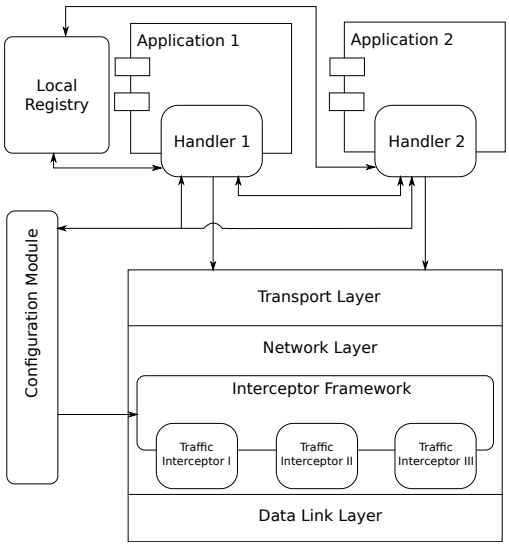

Fig. 1. Architecture overview of the proposed platform.

Handler

Handlers are the main structure of the proposed platform. The idea behind these structures is to abstract applications from transport details and traffic manipulation as shown in figure 2. Applications instantiate handlers and configure them based on the kind of transport they require. An handler may have more than one communication interface. One of them performs local communications with other modules and another performs network communication. Therefore, handlers have more than one address. In order to overcome such problem we propose that handlers have a unique identifier, known to each application that can be mapped into handlers addresses. Handler address resolution is perform by communicating with the local registry present in each node.

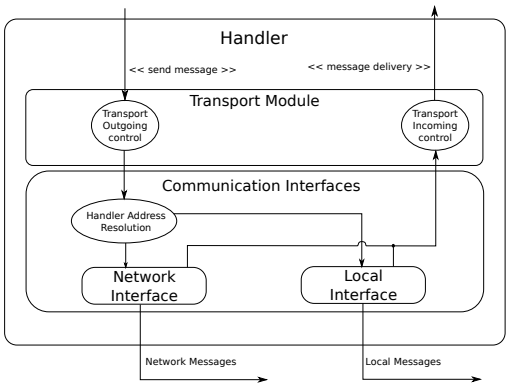

Fig. 2. Handler architecture.

Traffic Interceptors

Traffic interceptors are responsible for traffic manipulation. We propose an architecture based in a plug-in scheme, as shown in figure 3. Each interceptor is treated as a plug-in that can be registered with the interceptor framework and perform traffic manipulation.

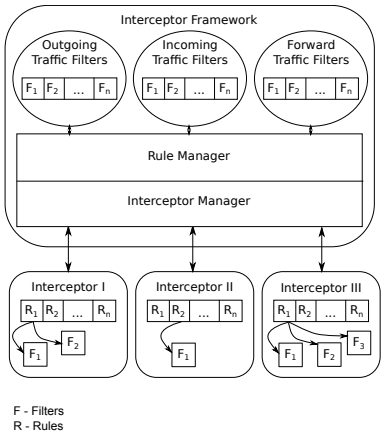

Fig. 3. Interceptor framework architecture.

The interceptor framework is composed by an interceptor manager, a rule manager, several rules and several filters. The interceptor manager is responsible for registering and unregistering interceptors with the interceptor framework. This manager communicates with the rule manager, requesting it to register rules for a given interceptor. These rules describe which packets should be intercepted and when. The rule manager is responsible for registering and unregistering rules. Rules are composed by several filters, which will filter packets at different interception points - incoming packets, outgoing packets and forwarded packets. In other words, filters describe which packets should be intercepted and rules combine several filters and describe at which interception point each filter should act.

Handler Registry

Handlers may have more than one communication interface. These structures must resolve other handlers' identifiers into addresses. This information is stored in a local registry in each node. Each registry entry should map a handler identifier into an address. Also, the handler registry provides means for managing its current translation table. Address operations may involve creating, deleting and obtaining handler addresses.

System Configuration

The proposed architecture is required to work in different scenarios. Therefore, the architecture must be configurable to such scenarios. For this purpose A configuration module is proposed. This module is responsible for configuring the interceptors and handler parameters in a node. This should not be confused with the configuration an application does when instantiating an handler. The configuration module only stores information about parameters each handler should use, such as how many messages the transport module can buffer and how many should it keep buffered in case of reliable transmission. The configuration module will also configure and load each interceptor according to the given configuration.

3.1 Functional Specification

Handler Functionalities

Prior to sending messages, handlers add a Protocol Data Unit (PDU) header to application payload. Most of its fields are used in the developed transport protocols. In section 2 we described two main types of transport protocols used in wireless sensor networks - reliable and unreliable transport. The proposed unreliable protocol is very simple as it sends each message without applying any kind of control. Likewise, upon receiving a message there is no procedure done by the unreliable transport control. Messages are delivered to applications as soon as they are received.

As for reliable transport, the control protocol is based in a NACK scheme. Messages are marked sequentially in order to control message flow and guarantee reliability. It is also necessary to provide error checking for each message received. This can be achieved through checksums performed in the data sent by each handler. The reliable transport module functionality can be separated in three components. One that is responsible for sending application data, another which is responsible for data reception and another which delivers messages to applications and sends the necessary NACK messages. To avoid flooding the network with NACK messages, we proposed an alternative NACK scheme, where a NACK message identifies gaps of missing messages by sending the smallest and largest sequence numbers in a gap of missing messages. This will keep the NACK message size small and will avoid multiple NACK messages.

Traffic Interceptors Functionalities

Synchronization Interceptor Functionality

An algorithm for delay estimation is proposed to synchronize data sent by applications. In this solution, each node gives an estimation of the time it contributed to delaying a certain packet and sends it to the next node. The sum of these delays will give a fairly good approach to the time passed since the packet was created until it arrived at its destination. Therefore, the proposed solution is to create an interceptor which implements the synchronization protocol.

When a PDU reaches its destination, it will contain the total time elapsed since it was created. Considering a path with N nodes, a PDU's creation time can be computed as:

$$T_{creation} = T_{reference} - \sum_{n=1}^{n-1}(Tn_n) - \sum_{n=1}^{n-1}(Tt_n) \quad (1)$$

Where $T_{reference}$ is the time taken by any node, which needs to know the creation time of the PDU. Tn_n is the time the PDU spent inside node n and Tt_n is the time node n calculated for the PDU's transmission. Propagation time between two nodes is simply the distance between them divided by the speed of light ($3 \times 10^8 m/s$). In most applications of WSNs the distance between two nodes is sufficiently small to ignore propagation time. As for transmission time in general, it is possible to estimate it based on the size of the frame being transmitted and its transmission rate. This yields the formula:

$$T_{trans} = \frac{\text{Frame Size}}{\text{Transmission Rate}} \quad (2)$$

Aggregation Interceptor Functionality

Aggregation is done at interceptor level. Packets are aggregated by destination address. This ensures that aggregated packets are all destined to the same node, which will ease the computation procedure for the desegregation protocol. Aggregation can be done at two distinct levels - Application and network level. Application level aggregation aggregates PDUs, while network level aggregation aggregates Internet Protocol (IP) packets. In order to support synchronization, each time a PDU or an IP packet is stored in the aggregation buffer, the protocol will compute the difference between the packet being aggregated and the one at the head of the buffer. For this reason, packets are aggregated in such way that the last packet in the buffer is the one with the smallest sequence number.

Desegregation Interceptor Functionality

Desegregation functionality is separated from aggregation to provide extra flexibility. This way a node can desegregate traffic without needing to load the aggregation interceptor and vice versa. The desegregation interceptor will separate packets inside an aggregated IP packet and inject then in the node's network stack.

4 Implementation

The prototype was developed using the programming language C. Part of it is implemented as a dynamic link library for application development and the rest is implemented as Linux Kernel modules. Figure 4 shows an overview of its architecture.

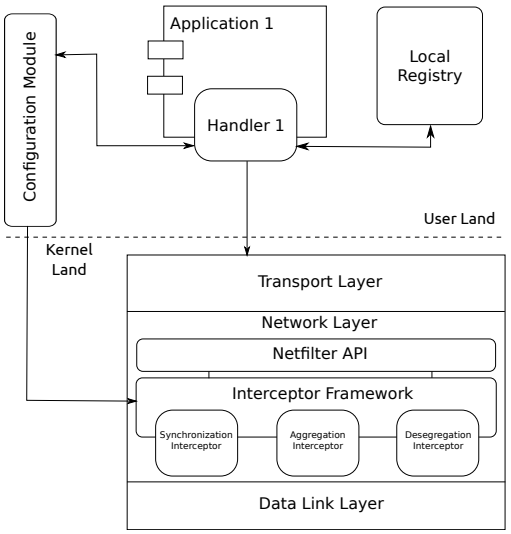

Fig. 4. Overview of the implemented architecture.

Each time an application creates an handler it must specify its configuration. Handlers are part of the developed shared library, thus they run in the same process as the application that is using them. The registry module is a separate process running in the background accepting requests in a Unix socket.

Three interceptors were implemented. As implied by their names, one is responsible for temporal synchronizing messages and the other two are responsible for message aggregation and desegregation. The

interceptor framework and each interceptor are implemented as separated Linux Kernel modules. As shown in figure 4 the Netfilters (NF) API is used by the interceptor framework, which makes network traffic interception much more flexible.

Finally, the configuration module was also implemented as a separated process running in the background, which can be queried through a Unix socket.

Handler Implementation

The handler structure was implemented in two different parts in order to conform to the proposed architecture. Messages sent by applications are stored in a message buffer inside the transport module. The transport module control will decide when messages are dequeued from the buffer and sent to the socket interface. The number of messages an application can send to the transport module is limited and if such limit is reached the application will block. The PDU header is prefixed to application data at the moment it is stored in the message buffer.

The prototype configuration module holds the implemented system configuration. This configuration specifies parameters necessary for configuring the transport module, such as how many messages an application can send to the transport module without blocking. Applications may configure an handler by specifying a path to a file in the filesystem or pass a string with the configuration directly in code. Nevertheless, the format for such configuration must be Extensible Markup Language (XML).

Reliable and unreliable transport were implemented, in order to provide applications with different methods for transferring their data. In the case of unreliable transport, the transport module does not perform any kind of control.

As for reliable transport, the implemented prototype supports the end-to-end protocol described in section 3, with two particularities. First, upon receiving a message, the transport receiver component will try to buffer it. However, since receiving buffers have a maximum size it is possible that there is no space in the buffer for the received message. In such case, the implemented protocol chooses to drop said message in the sense that it will send a NACK message for it later on. Second, due to the nature of protocols based in a NACK scheme, there is no way for the sending handler to be sure when to remove messages from the sending buffer. As a result of this fact and given that the sending buffer has a limited size, it is possible that such handler receives NACK messages for messages it no longer has. In such situation the handler chooses to skip those messages and notifies the receiving handler about this fact, using a control message called Oldest Sequence Number (OSN).

Interceptor Framework Implementation

Interceptors can be loaded and unloaded at runtime. Essentially, the framework provides two functions for interceptors. One is to register an interceptor and the other to unregister it.

The proposed interceptor framework architecture suggested that it should be possible to manipulate incoming, outgoing and forwarded traffic. However, to take advantage of the NF API, not only does the prototype support those three kinds of traffic, but it also supports pre-routed traffic and post-routed traffic.

Synchronization Interceptor Implementation

The synchronization protocol is implemented via an interceptor and called Cross Layer One Way Delay Estimation (CLOWDE). Each rule for the synchronization interceptor has two filters. One will be applied at pre-routing and the other at post-routing.

The experimental boards where the prototype was tested run a version of Linux as their Operating System (OS) and data is transmitted using its implementation of the Open Systems Interconnection (OSI) model. One important characteristic of this implementation is that each layer buffers data before passing it to the next layer. Therefore, it is necessary to measure the transmission time when a PDU leaves or enters the last layer, in order to get an accurate reading. However, since the synchronization interceptor is implemented at the IP layer it is not able to get an accurate estimation of the frames' transmission time. Therefore, it relies on a modified Linux driver, which is capable of doing this estimation for it.

The overall algorithm does not cover the entire PDU's path in each node. It fails to consider time spent in the data link layer and Universal Serial Bus (USB) transmission delays (when frames are transmitted to the WiFi card). In order to minimize the error, the modified driver includes yet another estimation accounting for these times. The following equation computes the estimation for USB transmission time:

$$\epsilon = \frac{\text{Frame Size}}{\text{USB rate}} \cdot 2 + \sigma \tag{3}$$

Where the USB rate is 12 Mb/s for USB 1.1 and σ is a constant value. The equation includes a factor of 2 to include the receivers USB transmission delay. The value for σ was determined experimentally and

represents an upper bound to minimize the delay estimation error. The procedure used to determine this constant is explained in section 5.

Aggregation Interceptor Implementation

The developed aggregation protocol supports aggregation at two different levels. Because the developed prototype relies on a transport protocol such as User Datagram Protocol (UDP), buffers must be identified not only by packets' destination IP address but also by their destination port. This information is required to build an aggregated packet correctly.

Desegregation Interceptor Implementation

The implementation of the desegregation protocol as an independent interceptor was necessary, because nodes aggregating packets may not need to desegregate them and vice-versa. Each rule in this interceptor will register only one filter at the local in NF hook. This ensures that aggregated packets are destined to the local node and need to be desegregated.

Upon receiving an aggregated packet, this interceptor will check its control byte, apply the appropriate procedure and inject the desegregated packets into the network. This guarantees that other NF hooks will run on them and therefore registered interceptors. There is no risk of flooding the network with injected packets. Linux Kernel routing facility will detect that such packets are to be delivered locally and will automatically call the IP local delivery functions.

5 Tests

5.1 Synchronization Protocol

To validate the performance of our synchronization algorithm (CLOWDE) we installed a Global Positioning System (GPS) device in the sender and receiver nodes and compared the results achieved with our algorithm with the results of the GPS device. Thus, as there are only two GPS devices available a very limited set of tests were possible. Given this limitation, we assessed CLOWDE using two simple, but yet effective, tests. The first one uses the most simple network, comprising only one sender and one receiver. In this case, the results obtained with CLOWDE should be very precise, as most of the delays are effectively taken into account. In a second case, there is an intermediate node, which introduces additional delay to process and that may cause collisions in the wireless medium. Hence, one might expect less precise results with our algorithm.

Obtaining the GPS Time

The used GPS receptor is the Trimble Lassen iQ, which provides a serial port for communication. Time accuracy is within a tenth of millisecond. This feature was used only to learn the current time up to the second. However, the designed protocol requires a higher precision, thus a different approach was developed to get an accuracy of microseconds. The GPS device provides a Pulse per Second (PPS) signal, with an accuracy of 50 nanoseconds. The PPS was used to set the seconds on the Technologic Systems (TS) boards. An interrupt line was connected from the PPS to the board's processor and an interrupt handler was written to accurately set the board's clock. At boot time, a user level program reads the time from the GPS and sets a variable in the Kernel, which holds the GPS seconds. The interrupt handler will increment this variable by one each time it is triggered. The node's clock is used to learn the microseconds.

Delay Error Minimization

In order to minimize the delay error several tests were ran to determine a value for σ in equation 3. In these tests only two nodes were used. Packets were sent from one to another at regular intervals with 25Hz frequency. In these tests the formula presented in equation 3 was not used. The experiment was repeated using different payload sizes (20, 500 and 1000 Bytes) and with different transmission rates (1, 5.5 and 11 Mb/s). Each run consisted in sending 100 packets.

We then calculated a σ value of 645 microseconds for use with Equation 3 in order to minimize the delay estimation error as follows. First we computed the mean of the delay error, which is 1406.98×10^{-6} seconds. This is the value for ϵ in equation 3. We can compute σ as:

$$1406.98 \times 10^{-6} = \frac{\text{frame size} \cdot 8}{12 \times 10^{6}} \cdot 2 + \sigma$$

Performance tests

Scenario One

In each test, the sender node sends 100 packets to the receiver, using a Constant Bit Rate (CBR) traffic generation. Both GPS and synchronization protocol delays are measured to be compared. Different payload sizes have been used (20, 500 and 1000 Bytes), with different sampling frequencies (25, 50 and 100

Hz). At the 802.11 network card, the transmission rate has been adjusted to 1 Mb/s, 5.5 Mb/s and 11 Mb/s. The parameter σ has been set to the value previously mentioned.

Table 1 presents the average estimated delay and the average error for the delay estimated by the synchronization protocol. We can see that the protocol underestimates in some scenarios and overestimates in others. However, the delay error is very small, usually in the order of the tenths of microseconds and never crossing above 0.7 milliseconds.

Table 1. Synchronization protocol average delay estimation error with 2 nodes

Transmission rate (Mb/s)	Payload size (B)	Mean delay (μ) / Delay estimation error (μ/s) Sample rate=25Hz		Sample rate=50Hz		Sample rate=100Hz	
1	20	3292.44	-441.78	2519.90	-42.58	2515.15	45.73
	500	12711.70	73.62	7749.24	-85.89	8558.54	-82.53
	1000	17173.80	-63.06	8476.54	-196.48	9145.22	-118.43
5.5	20	2650.31	48.74	2583.94	49.85	2719.54	1.67
	500	8284.61	-147.29	9027.87	-74.39	8776.41	-63.99
	1000	9474.41	-213.66	10234.80	-204.08	10195.60	-200.28
11	20	3428.54	-44.48	2625.93	41.27	2494.13	39.80
	500	9536.90	-132.41	8363.14	-132.04	8261.68	690.46
	1000	9219.82	-198.25	9459.00	-236.85	9405.00	-212.20

Figure 5 compares the delay estimation provided by the synchronization protocol against the time measured using the GPS for two nodes at 1Mb/s transmission rate, 1000 Bytes of payload and 100Hz sampling frequency. Notice that the difference varies little as the synchronization protocol and the GPS show the same behavior.

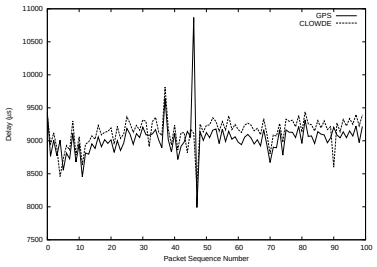

Fig. 5. Synchronization protocol delay estimation versus GPS delay using 100Hz, 1000B and 1Mb/s.

Figure 6 shows the Cumulative Distribution Function (CDF) of the error present in the synchronization protocol estimated using packets produced at 100Hz, under the various packet sizes and one hop.

Scenario Two

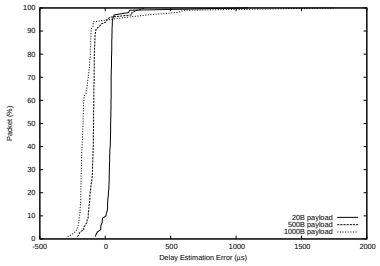

Fig. 6. Synchronization protocol error CDF using 1Mb/s rate and 100Hz.

As in the first scenario, the sender node sends 100 packets to the receiver, using a CBR traffic. Both GPS and synchronization protocol delays are measured to be compared. Transmission rates of 1 Mb/s, 5.5 Mb/s and 11 Mb/s were tested as well as different payload sizes (20, 500 and 1000 Bytes) and sampling frequencies (25, 50 and 100 Hz). The parameter σ has been set to the value previously mentioned.

The introduction of the third node caused the average delay to vary more widely. This variation was largely confirmed by the GPS and may be explained by the greater medium access competition.

Figure 7 shows the CDF of the error present in the synchronization protocol estimated using packets produced at 100Hz, under the various packet sizes and two hops.

In this scenario, the synchronization protocol provided less accurate delay estimations, however in most situations the average error remained below 10% or 1.5 milliseconds.

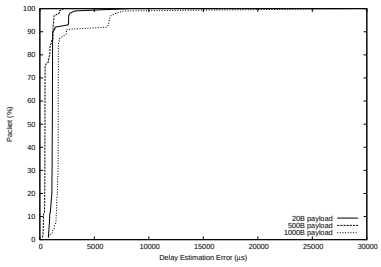

Fig. 7. Synchronization protocol error CDF using 1Mb/s rate and 100Hz.

Results Discussion

As a first analysis of the graphic shown in figure 5 it is possible to conclude that the synchronization

protocol is able to capture delay variation. Each peak registered by the GPS devices is also registered by the synchronization protocol. Table 1 provides information about the error of the synchronization protocol delay estimation. With one hop the error is usually in the order of tenth of microseconds never crossing above 0.7 milliseconds. With two hops the delay varies more widely, which can be explained by the greater medium access competition.

From the CDFs represented in figures 6 and 7 one can observe that a large percentage of packets have similar delay errors. It is also possible to notice that a small percentage of the packets suffers from larger delay errors. These can be filtered by application in scenarios where the delay is bounded.

5.2 Aggregation Protocol

Performance tests

To study the benefits and hindrances of using aggregation, two different test scenarios have been prepared. The first one analyzes the overhead reduction of each type of aggregation (application versus network level), while the second one analyzes how aggregation can reduce network congestion. Packet delay was also measured.

Scenario One

The aim of this test is to assess the overhead introduced by the different aggregation strategies used, using different parameterization and test conditions. Since the interest is to measure aggregation overhead reduction, the tests were executed using a cabled network. This prevents most of packet loss and delays related with wireless protocols enabling more precise results.

The scenario one's network topology is shown in figure 8, comprising four nodes: two senders (Node A and Node B), receiver (Node D) and intermediate node (Node C). Node A and Node B send 100 packets to node D, at a rate of 25 Hz.

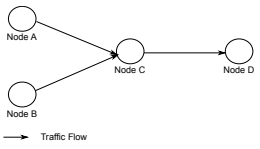

Fig. 8. Network topology scenario one.

Three different PDU sizes have been tested - 32, 64 and 128 Bytes. For each PDU size, six different aggregation strategies were tested:

No_Aggreg - without any aggregation, which will serve as a control test.
Apl_Aggreg_640 - with application layer aggregation, with an aggregated packet size of 640 bytes.
Apl_Aggreg_1280 - with application layer aggregation, with an aggregated packet size of 1280 bytes.
Net_Aggreg_320 - with network layer aggregation, with an aggregated packet size of 320 bytes.
Net_Aggreg_680 - with network layer aggregation, with an aggregated packet size of 680 bytes.
Net_Aggreg_1280 - with network layer aggregation, with an aggregated packet size of 1289 bytes.

Application aggregation is always performed in the nodes A and B, while network aggregation is always performed by node C (figure 8).

Figures 9 and 10 demonstrate the obtained delays with no aggregation (**No_Aggreg**) and application aggregation (**Apl_Aggreg_1280**), respectively. Only 100 packets are shown since the behavior is the same for the rest of the packets and with 100 packets it is possibly to get a better view of the delay behavior.

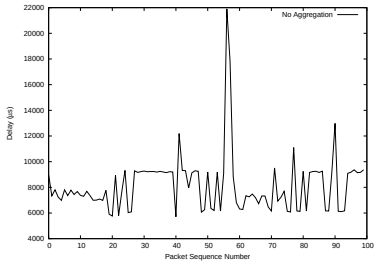

Fig. 9. Delay without aggregation.

Table 2 shows the total amount of bytes received at the data link layer by the receiver node and the percentage of overhead reduction for better comparison. Overhead reduction is always computed relative to the test where no aggregation was performed.

Scenario Two

Test scenario two evaluates if the aggregation protocol can reduce congestion in the network. Figure 11 shows the network topology used in this scenario. Node A and C send data to node B.

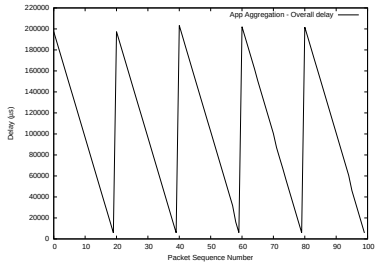

Fig. 10. Delay with application aggregation.

Table 2. Number of bytes that effectively arrived at the sink node measured in the MAC layer.

Packet Size (Bytes)	Aggregation Type									
	No Aggregation		Application (640 Bytes)		Application (1280 Bytes)		Network (640 Bytes)		Network (1280 Bytes)		Application and Network (320 and 1280 Bytes)	
	Size	Reduction	Size	Reduction	Size	Reduction	Size	Reduction	Size	Reduction	Size	Reduction
32	15000 B	-	6830 B	54.47 %	6615 B	55.90 %	13660 B	8.93 %	13230 B	11.80 %	6615 B	55.90 %
64	21400 B	-	13660 B	36.17 %	13230 B	38.18 %	23222 B	*8.51 %	21168 B	1.08 %	13230 B	38.18 %
128	34200 B	-	27320 B	20.12 %	26460 B	22.63 %	34150 B	0.14 %	33075 B	3.29 %	33075 B	3.29 %

* This value represents an increase in overhead instead of a reduction.

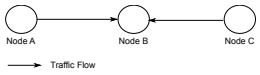

Fig. 11. Network topology scenario two.

In test scenario two only 64 Byte sized PDUs were used. To induce congestion, both senders send 1000 PDUs as quick as possible. The receiver node will log each received PDU so packet loss can be calculated after the test ended. Measuring packet loss is a good indication of congestion in the wireless medium.

Three different tests were conducted. First no aggregation (**No_Aggreg**) was used. Then application aggregation was turned on (**Apl_Aggreg_1280**). The final test was done with only network aggregated packets (**Net_Aggreg_1280**). To obtain accurate results each test was repeated several times. Sixty tests were performed in total, twenty for each case. Figure 12 shows the CDF of the packet loss percentage.

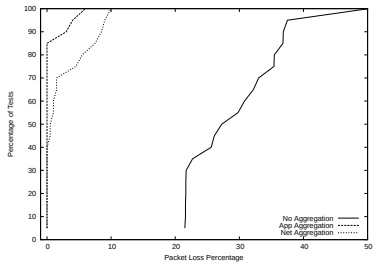

Fig. 12. CDF of the packet loss percentage.

Results Discussion

As it is clear from the test results in scenario one, the number of packets which arrive at the destination is significantly smaller when compared with the no aggregation scenario. Turning our attention to table 2 it is possible to better understand the reason for such. Although in most cases the header overhead reduction is smaller in percentage, the amount of bytes transmitted is significantly smaller. However, there are cases where the aggregation protocol performed poorly. As an example, consider the test where 64 Byte sized PDUs and 640 Byte sized network level aggregation packets were used. In this case the total amount of transmitted bytes is bigger than the case where no aggregation was used. This can be explained by the fact that the aggregated packet size is not a multiple of packet's size. This results in the last aggregated packet not being complete and some bytes are wasted.

What the protocol lacks in overhead reduction it compensates for congestion reduction as it is clear from scenario two results. When aggregation was not used, about 30% of the transmitted packets were lost. However, when aggregation was turned on there were almost no losses.

As expected, with aggregation turned on the obtained delay increases significantly. Moreover, graphic 10 shows what is expected from the aggregation protocol. Due to software limitations, the used TS 7500 boards cannot handle a sampling frequency higher than 100 Hz. Thus, it is not possible to conclude on the limitations of the aggregation protocol, that is, at which sampling frequency would the receiver node start dropping packets.

The obtained results show that most packet loss can be avoided by using the developed protocol, thus it is safe to conclude that fewer retransmissions occur. This leads to the fact that nodes do not waste energy in retransmitting packets and therefore the overall energy waste is reduced.

6 Conclusion

This thesis main goal was to develop a multifunctional platform for indoor and outdoor monitoring, which could be easily extended and configurable to different scenarios. In particular, this thesis focused in WSNs which perform this kind of monitoring and should be power-aware and need to synchronize their data. Therefore, synchronization and in-network aggregation protocols were developed, which had this in consideration.

Tests to the prototype revealed satisfactory results. The synchronization protocol revealed to be able to synchronize packets in the order of few milliseconds. As for the aggregation protocol, tests demonstrate that although it provides little reduction in the overhead of transmitted packets it is capable of minimizing congestion in the wireless medium. Therefore, less collisions occur, preventing nodes from retransmitting corrupted frames and thus not wasting energy associated with these retransmissions.

As for future work, the developed architecture provides a design which can easily be extended. New interceptors are easy to develop and integrate with the platform, without having to change previous work. Some features present in some WSN can be implemented by means of interceptors. As for the developed protocols, the synchronization algorithm may be moved to lower layers in the OSI stack to account for more delays related with the layers' buffering mechanisms. Also the USB delay estimation should also be improved to account for multiple transfers. The aggregation protocol may be improved to reduce more the overhead introduced by the network and transport headers.

References

1. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., et al.: Place lab: Device positioning using radio beacons in the wild. Pervasive Computing (2005) 301–306
2. Want, R., Hopper, A., Falcão, V., Gibbons, J.: The active badge location system. ACM Transactions on Information Systems (TOIS) **10**(1) (1992) 91–102
3. Bahl, P., Padmanabhan, V.: Radar: An in-building rf-based user location and tracking system. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 2., Ieee (2002) 775–784
4. Abdelzaher, T., Blum, B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., et al.: Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In: Distributed Computing Systems, 2004. Proceedings. 24th International Conference on, IEEE (2004) 582–589
5. Burrell, J., Brooke, T., Beckwith, R.: Vineyard computing: sensor networks in agricultural production. IEEE Pervasive computing (2004) 38–45
6. Xie, W., Shi, Y., Xu, G., Mao, Y.: Smart platform-a software infrastructure for smart space (siss). In: Multimodal Interfaces, 2002. Proceedings. Fourth IEEE International Conference on, IEEE (2002) 429–434
7. Mills, D.: Internet time synchronization: The network time protocol. Communications, IEEE Transactions on **39** (2002) 1482–1493
8. Elson, J., Römer, K.: Wireless sensor networks: A new regime for time synchronization. ACM SIGCOMM Computer Communication Review **33**(1) (2003) 149–154
9. Elson, J., Estrin, D.: Time synchronization for wireless sensor networks. (2001)
10. Elson, J., Girod, L., Estrin, D.: Fine-grained network time synchronization using reference broadcasts. ACM SIGOPS Operating Systems Review **36**(SI) (2002) 147–163
11. Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: The flooding time synchronization protocol. In: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM (2004) 39–49
12. Römer, K.: Time synchronization in ad hoc networks. In: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing, ACM (2001) 173–182
13. Xu, N., Rangwala, S., Chintalapudi, K., Ganesan, D., Broad, A., Govindan, R., Estrin, D.: A wireless sensor network for structural monitoring. In: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM (2004) 13–24
14. Fasolo, E., Rossi, M., Widmer, J., Zorzi, M.: In-network aggregation techniques for wireless sensor networks: a survey. Wireless Communications, IEEE **14**(2) (2007) 70–87
15. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review **36**(SI) (2002) 131–146
16. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. SIGMOD Rec. **31**(3) (2002) 9–18
17. Sharaf, M., Beaver, J., Labrinidis, A., Chrysanthis, P.: Balancing energy efficiency and quality of aggregate data in sensor networks. The VLDB journal **13**(4) (2004) 384–403
18. Cayirci, E.: Data aggregation and dilution by modulus addressing in wireless sensor networks. Communications Letters, IEEE **7**(8) (2003) 355–357
19. He, T., Blum, B., Stankovic, J., Abdelzaher, T.: Aida: Adaptive application-independent data aggregation in wireless sensor networks. ACM Transactions on Embedded Computing Systems (TECS) **3**(2) (2004) 426–457
20. Wang, C., Sohraby, K., Li, B., Daneshmand, M., Hu, Y.: A survey of transport protocols for wireless sensor networks. Network, IEEE **20**(3) (2006) 34–40
21. Hull, B., Jamieson, K., Balakrishnan, H.: Mitigating congestion in wireless sensor networks. In: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM (2004) 134–147
22. Santos, L.: Sos net: Rede de emergência para gestão de risco ambiental. Master's thesis, Instituto Superior Técnico, Portugal (2008)