Designing and Implementing a Middleware for Data Dissemination in Wireless Sensor Networks

Ronald Beaubrun
Department of Computer Science and Software Engineering
Université Laval
Quebec, Qc, Canada
e-mail: ronald.beaubrun@ift.ulaval.ca

Jhon-Fredy Llano-Ruiz, Alejandro Quintero
Department of Computer and Software Engineering
École Polytechnique de Montréal
Montreal, Qc, Canada
e-mail: {jhon-fredy.llano-ruiz, alejandro.quintero}@polymtl.ca

***Abstract*— In this paper, we propose an approach for designing and implementing a middleware for data dissemination in Wireless Sensor Networks (WSNs). The designing aspect considers three perspectives: device, network and application. Each application layer is implemented as an independent Component Object Model (COM) Project which offers portability, security, reusability and domain expertise encapsulation. For result analysis, the percentage of success is used as performance parameter. Such analysis reveals that the middleware enables to greatly increase the percentage of success of the messages disseminated in a WSN.**

***Keywords-** Data dissemination, design, implementation, middleware, wireless sensor network.*

I. Introduction

The main goal of a Wireless Sensor Network (WSN) is to gather environmental information in a specific region and make it available to users. For this purpose, it uses a set of sensor nodes, *i.e.*, a set of devices that sense and measure environmental variables, such as light, temperature, humidity and barometric pressure [1]. Another important component of a WSN is the Base Station (BS). Since sensors are normally battery-constrained and equipped with low system capabilities, they need to transfer their collected data to a long-life device. Laptops, Personal Computers (PCs), handhelds and access points to a fixed infrastructure are examples of physical devices used as BSs. To make the communication possible between SNs and BSs, a gateway (GW) is set in between, acting as a bridge. Figure 1 shows an example of a WSN.

In a WSN, the exchange of information between the SNs, the GW and the BS is done through a data-dissemination technique where the information is transported towards different destinations [2-5]. For this purpose, a middleware is required between the network and the applications to offer tracking capabilities of the disseminated information. Using a data dissemination protocol, this middleware can take on-time decisions when a maximum end-to-end delay constraint is exceeded. This paper proposes an approach for designing and implementing a middleware for data dissemination in WSNs.

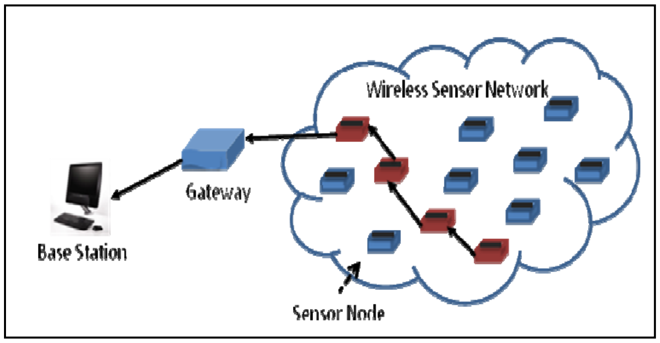


Figure 1. Example of a Wireless Sensor Network.

The rest of the paper is organized as follows. Section II presents the designing aspects of the middleware. Section III focuses on the software components that lead to the middleware deliverables. Section IV describes the implementation of each software component. Section V presents some results and analysis, whereas Section VI gives some concluding remarks.

II. Designing aspects

A. *Reference architecture*

Figure 2 illustrates the general architecture considered for this research from the infrastructure point of view. It integrates a WSN with two other networks: the source of information is the sensor network, whereas the destination can be Internet or a Cellular Network. This architecture considers two roles: the *Message Originator* (MO), which is responsible for initializing the notification process, and the *Message Terminator* (MT), which receives the information and sends back a response. MT is a role played by any person or device in the system. In case of a person, it can be either a *Security Group* (SG) member, or a *User Group* (UG). The MO represents each single sensor node that is deployed. It collects information that could be disseminated. If an event is detected, the sensor node starts the dissemination process towards the gateway (GW), using the forwarders in between. The GW is responsible for receiving the information sent by any node in the WSN, and conveys it to the base station (BS). Once the BS receives the information, it will make a decision depending on its own configuration, *e.g.*, Send information to UG and SG through different protocols, such as *Short Message Services* (SMS), *email* or *twitter*.

B. Model roadmap

Figure 3 presents a general overview of the middleware. Therein, the middleware is initially related to *delay-constrained applications*, as it aims to produce support for such type of applications. In order to guarantee this support, it requires a direct communication with *data dissemination protocols* at any moment. Therefore, the top layer represents *delay-constrained applications* that use the middleware which, in turn, is the intermediate layer. In the meantime, it imposes some requirements (*e.g.*, end-to-end delay) to the underlying *data dissemination protocols* located in the bottom layer.

The middleware deals with different data dissemination protocols, and it requires to be executed on different types of devices (*i.e.*, SNs, GWs, BSs), which forces each environment to control different configurations and specificities. For such a purpose, the approach from [6] is adopted. It considers three points of view: *device*, *network* and *application*. Firstly, *device perspective* focuses on each device and its components, considering five features: *type of devices*, *operating systems*, *radio technology*, *development technologies* and *storage*. *Type of devices* represents different machines where the middleware is intended to be executed (*i.e.*, SNs, GWs, BSs). *Operating systems* represent different operational platforms running on the types of devices (*i.e.*, *TinyOS*, *Linux* and *Windows*). *Radio technology* is used to establish communication with other nodes of the architecture (*i.e.*, 802.11). Additionally, *development technologies* features need to be taken into consideration. For the suitability of these technologies and their widely acceptance in the academia, *nesC*, *Java* and *C++* have been chosen. Finally, *storage* takes care of the persistence of the information when needed (*i.e.*, databases, XML files).

Secondly, *network perspective* represents the dissemination of information among the network. It takes into account several network characteristics, *e.g.*, end-to-end delay, confirmation mechanisms and energy optimization. In other words, the *network perspective* takes into account three network services in order to achieve the requirements: *delivery manager*, *message sender manager* and *service manager*. *Delivery Manager* (DM) is responsible for managing the delivery process. It tracks messages sent along the process while considering delay constraints. It includes: reporting, receiving and analyzing capabilities. *Message Sender Manager* (MSM) is in charge of the message sending process. It is made up of three main processes: listening, analyzing and sending. *Service Manager* (SM) is a service that allows managing the protocols the system works with and the resources associated to each of them.

Finally, the *application perspective* represents the applications using the middleware services. It is divided into two main categories: *delay-constrained applications* and *user applications*. On one hand, delay-constrained applications have strict Quality of Service (QoS) constraints, and are used to warn people in emergency events. They require a continuous feedback from the middleware. On the other hand, user applications tolerate lower QoS constraints due to their specific goals. Failures or delays are not as critical as they are for the former category. As a result, confirmation mechanisms could be avoided or delayed. Despite of that, user applications can use the middleware as well.

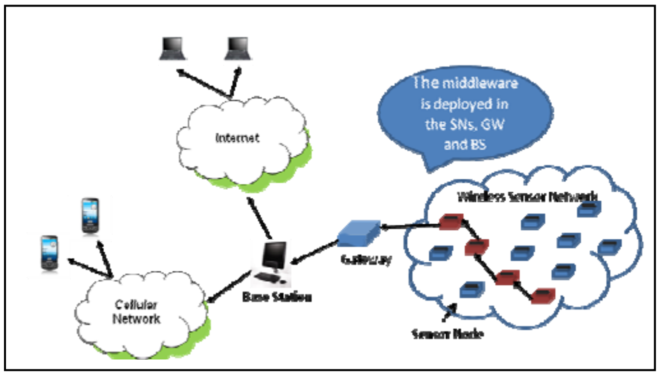


Figure 2. Global Architecture.

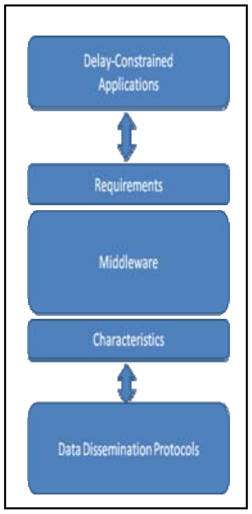


Figure 3. General overview of the middleware.

The integration of such perspectives constitutes the roadmap of this proposition, guarantying a holistic view of the system. This amalgamation is intended to show that all perspectives are present at any time in the system and the intersection of all of them produces the middleware. Figure 4 depicts such integration. The 3D-view offers the possibility to analyze the system from different perspectives while preserving the unity and respecting the requirements and constraints.

III. SOFTWARE COMPONENTS

This section focuses on presenting the software components that lead to the middleware deliverables. Firstly, the class diagram that shows the class interactions within the whole system is presented. Later on, the sequence diagrams that show the interaction of the architecture components are explained.

A. *Class diagram*

Figure 5 presents the class diagram of the middleware, giving a static view of the system. It is divided into four logical layers which depict the main components presented in the reference architecture. The first three layers refer to five main components: *Interfaces*, *Message Sender Manager*, *Delivery Reporter Manager*, *Data Access Manager* and *Service Manager*, whereas the bottom layer represents the data dissemination protocols to be used. On top of the diagram, a set of *Interfaces* classes offers a unique way for consumers to use the middleware services. It is made up of three classes that interact with the second layer components. In the second layer, the *Message Sender Manager* is responsible for managing the sending process while considering three main classes: *Listener*, *Analyzer* and *Sender*. *Listener* senses new messages that arrive to the middleware. *Analyzer* consists of four classes, which means that all classes need to participate in the process when the analyzer is executing. Finally, *Sender* is in charge of sending the analyzed message. Then, the *Delivery Report Manager* classes track the message status. Similarly to the previous component, it also considers three classes: *Reporter*, *Analyzer* and *Receiver*.

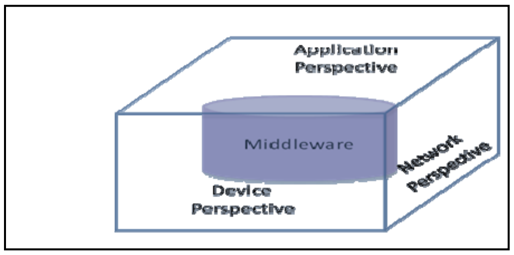


Figure 4. Middleware roadmap.

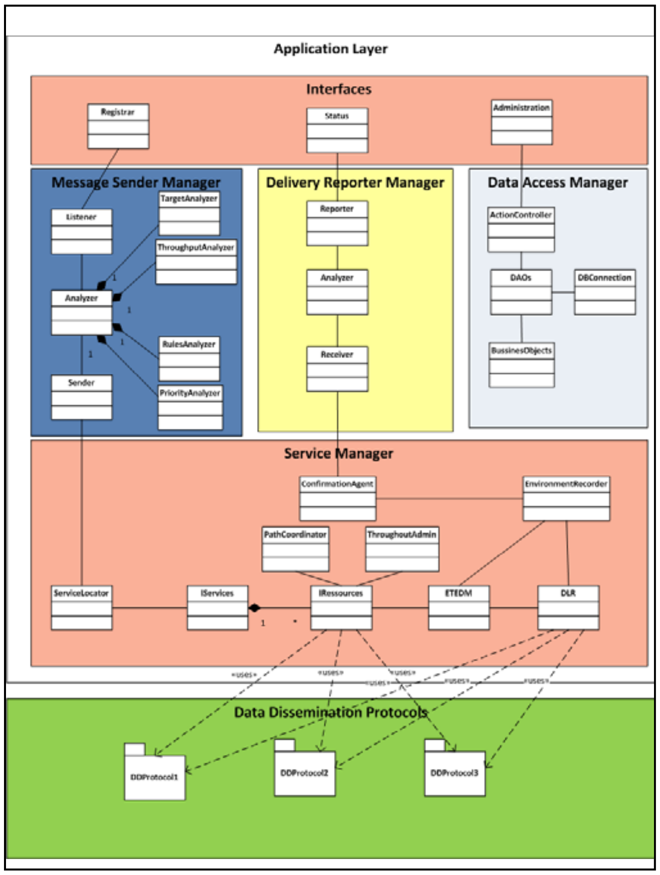


Figure 5. Class diagram of the reference architecture.

Furthermore, *Data Access Manager* is responsible for providing and modifying the data models (*i.e.*, databases and configuration files). It uses an *ActionController* which is responsible for receiving an action to be executed and identifying which component in the system will realize it. Normally, this action is assigned to a Data Access Object (DAO), which in turn, affects the information relying on any Business Objects (BOs). Finally, the *Service Manager* is responsible for interacting with the protocols and the network to complete either the message sending process, or the delivery report process. It consists of a set of classes that offer system characteristics, such as end-to-end delay (ETEDM), delivery report (DLR), environment events (*EnvironmentRecorder*), Confirmation features (*ConfirmationAgent*) and sending of messages (*IServices* and *IRessources*). *Service Manager* relies on a *ServiceLocator* which identifies the most appropriate services and protocols according to the application requirements.

As previously discussed, the middleware is located in the application layer. In order to perform its tasks, it should have access to specialized protocols that are normally located in lower layers in the communication stack. For such reason, the bottom layer shows the available protocols to be used and their interactions with the middleware. It is important to notice that this proposal is protocol independent, which means that any protocol could be used as long as it supports the application requirements. Therefore, it is up to the implementers to choose the right dissemination protocol.

B. Sequence diagrams

The sequence diagrams present a dynamic view of the system. Two main processes are described: the message sending process which shows how the components participate in order to offer end-to-end delay and confirmation support to the messages sent, and the delivery report process which enables to have knowledge about messages states at any moment.

1) Message sending process: As depicted in Figure 6, this process is initiated by a sensor, a gateway or a base station when a new message arrives. Any of them may register a new message using the *Registrar* interface. This interface puts the message into a Queue waiting for *Listener* to be in charge of it. *Listener* is a daemon process responsible for the surveillance of new messages that arrive. For this purpose, it executes asynchronous calls to *MessageQueue*. Once it discovers a message standing there, it takes the message and passes it to a new phase to be analyzed. This process is broken up into 4 stages: analysis of destinations, priorities, rules and throughput. These stages heavily depend on the environment where the middleware is deployed. Once the whole analysis is completed, a *Sender* class is called to send the message. Later, the *ServiceLocator* class receives the *Sender* request in order to locate the service and the resource that will be responsible for disseminating the information towards the destinations. For such a purpose, this class takes into consideration basic information, such as priorities and rules. Once the resource is identified (*i.e.*, dissemination protocol with its parameters), *IRessource* begins to interact with the protocol, which finally is responsible to convey the information to the destinations, considering the application requirements.

At the same time, ETEDM, which is the process to offer timeliness support, is activated. It controls delay-constraints for each message sent while verifying the acknowledgements (ACKs) or negative acknowledgements (NACKs) sent by the protocol. If no response is received by the end of this time period, it asks the *ServiceLocator* to look for another service and resource to disseminate the information, *i.e.*, the lookup process. This cycle is repeated based on the middleware configuration.

2) Delivery report process: Any device (*i.e.*, a sensor node, a gateway, a base station) or internal component in the architecture (*e.g.*, a sender) may want to know the status of a message sent at any time. The sequence shown in Figure 7 details how this process is executed. Once a device or a component interrogates the *Status* interface, this request is transferred to the system, then analyzed further to identity the message that is going to be tracked. Once this identification is performed, *ConfirmationAgent* is interrogated. It reads and analyzes the information presented by *EnvironmentRecorder*, which tracks all the events that happen with the message, such as end-to-end delay information, DLR and network failures. Based on this analysis, *ConfirmationAgent* presents a response to the system, which is sent back to the *Status* interface, then to the user or component interested in this information.

IV. IMPLEMENTATION

The application is divided into three main logic components: *Interfaces*, *Business Rules* and *Data Services*. Each layer is implemented as an independent Component Object Model (COM) Project which offers portability, security, reusability and domain expertise encapsulation.

A. Interfaces Layer

The *Interfaces* layer exposes functionalities as services and variables. It provides a method called *registrar* for the applications to register the events. The definition of this method is presented as follows:

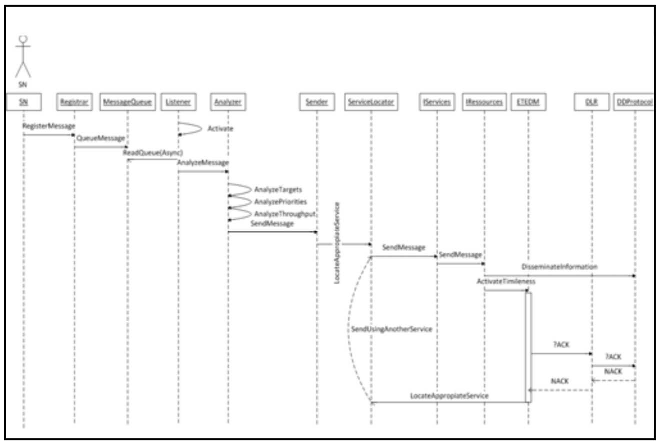


Figure 6. Message sending process.

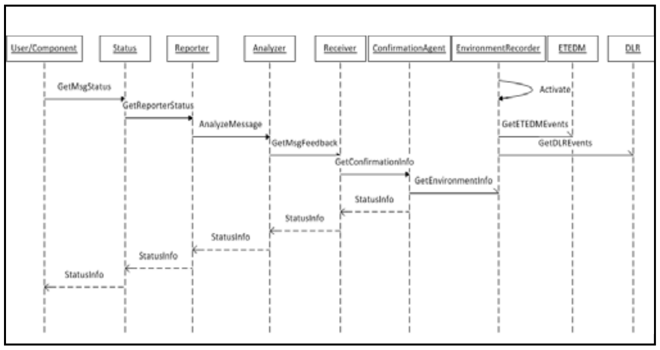


Figure 7. Delivery report process.

```
public static void registrar(int priority,
                             String shortDescription,
                             String description,
                             String source,
                             int type,
                             String comments);
```

It receives six mandatory parameters. First, *priority* is used to establish the priority of the message (*e.g.*, 100=emergency). Then, *shortDescription* contains a brief description about the event. The third parameter, *description*, contains a more detailed description of the incident (*e.g.*, sensor x registered a value of light intensity y in the z-building). Next, *source* indicates the origin of this information. Then, *type* refers to the type of the originator (*e.g.*, sensor node, gateway, base station). Finally, *comments* permits to include any additional information required to complement the message. This information can be presented in an XML format for a better portability. Using this service, the events that come from the WSN are initiated in the middleware.

B. *Business Rules Layer*

The *Business Rules* layer is the core of the system, since it implements the basic components: *Message Sender Manager*, *Service Manager* and *Delivery Report Manager*, enabling messages to be sent through different protocols. To set up these protocols, an XML file is generated. It might be noticed that each protocol is composed by one or multiple resources, supporting the definition made in Figure 5. Table I describes the tags composing the file. As described in this table, each resource might require several parameter values to be described and configured. Figure 8 presents a fragment of *resources.xml* file for the implemented prototype.

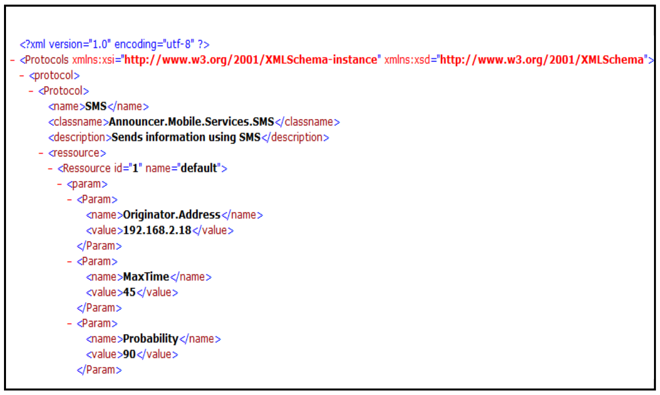
It can be noticed that the instance shows an SMS resource configuration. The *tag name* is used to identify the protocol used. The tag class describes the name of the class that implements the service. It is dynamically executed using on-the-fly *.Net* capabilities (also known as assemblies). This feature makes the environment execution more versatile, since it only requires setting up the XML. The information is sent in strict order according to its appearance in this file. The maximum set up time for each resource to complete its task is obtained from the XML file. This information is defined using a probabilistic approach based on studies done on the efficiency of these resources, as stated in [7]. The DLR interface is simulated using these probabilistic values to know whether the message was successfully received or not.

C. *Data Services Layer*

This layer is responsible for providing the interfaces the access to the information. This information is mainly stored in two locations: the database and the XML resources file. Figure 9 presents the Entity/Relation (E/R) diagram for the middleware. It can be seen that there is a table called *queued_message*, where the message is initially queued using the *registrar* service. Then, the middleware using the *listener* processes moves the record to the *message* table. Later on, after the analysis is done, the single message is multiplexed into multiple records. Each message is addressed to a single user, using a different protocol and device (*e.g.*, SMS-blackberry, Email-iPhone), as defined in the XML file and in the database configuration. This information is stored in the *sent_message* table. The DLR obtained from each service is recorded in the *status* attribute. By using this information, the middleware knows the state of each single message sent to any user in the system.

TABLE I. XML RESOURCES FILE DESCRIPTION

Tag Name	**Tag Description**
Protocols	Indicates the beginning and the end of the resources file
Protocol	Indicates the beginning or the end of a protocol
name (protocols)	Contains the name of a protocol
Classname	Describes the name of the class fully specified. *Package.ClassName.* This value is used by the middleware to dynamically execute the class using on-the-fly capabilities (*i.e.*, assemblies loaded and executed when needed). It allows the middleware to execute assemblies that might or might not be part of it.
description	Brief description of the protocol
Resource	Indicates the beginning or the end of a resource. For instance, a SMS could be sent using different SMS Gateways.
name (resource)	Contains the name of a resource.
param-name	Details all the parameters required to describe a resource. For instance, a SMS Gateway requires an IP address, a port, a user, a password and URL among others. It additionally describes maximum time to wait for a response and the probability of receiving an ACK.



```
  <?xml version="1.0" encoding="utf-8" ?>
- <Protocols xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <protocol>
    - <Protocol>
        <name>SMS</name>
        <classname>Announcer.Mobile.Services.SMS</classname>
        <description>Sends information using SMS</description>
      - <ressource>
        - <Ressource id="1" name="default">
          - <param>
            - <Param>
                <name>Originator.Address</name>
                <value>192.168.2.18</value>
              </Param>
            - <Param>
                <name>MaxTime</name>
                <value>45</value>
              </Param>
            - <Param>
                <name>Probability</name>
                <value>90</value>
              </Param>
```

Figure 8. Resource file.

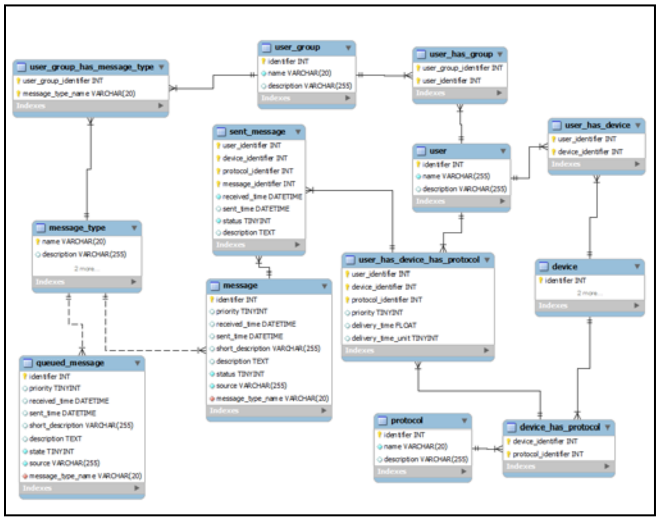


Figure 9. Entity/Relation Diagram.

V. Results and Analysis

For the experiments, we use the architecture shown in Figure 2, with 3 sensor nodes in the WSN. For each message sent through a resource (*i.e.*, SMS, *email* or *twitter*), the percentage of success, *i.e.*, the ratio between the number of messages sent and those successfully received by the destination, is registered. These values are then processed and analysed in MATLAB. Table II presents a fragment of the results obtained from the first experiment. The first column shows the corresponding statistical attributes analyzed, *i.e.*, the percentage of success, the number of received ACKs, the number of received NACKs or no responses (NRs). For the middleware, the percentage of success is 98.25%. Accordingly, 7 destinations are not successfully notified among 400 messages sent.

Now, we can analyze the percentage of success for each resource in 20 experiments. Figure 10 shows that the middleware outperforms the other resources taken individually. More specifically, the overall success of the middleware is close to 98%, which represents a great improvement when compared with the performance of the resources individually. For instance, SMS shows an average success of 78%. A slightly increment is seen in *email* with 79%. Finally, *twitter* offers the lowest success of the three individual resources (61%).

TABLE II. Results from the first experiment

	SMS	Email	Twitter	Middleware
Percentage of Success	79.50%	70.73%	70.83%	98.25%
ACK	318	58	17	393
NACK, NR	82	24	7	7

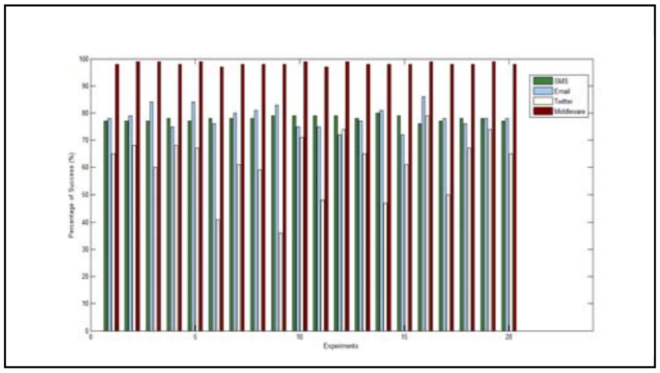


Figure 10. Comparison of percentage of success.

VI. Conclusion

In this paper, we proposed an approach for designing and implementing a middleware for data dissemination in WSNs. For the analysis, the percentage of success is used as performance parameter. Such analysis reveals a middleware success close to 98%, which is highly superior to the success of other individual resources.

Acknowledgment

This research has been supported by Prompt-Quebec and Imaginovation.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Networks, vol. 52, pp. 2292-2330, August 2008.

[2] H. M. Ammari and S. K. Das, "A trade-off between energy and delay in data dissemination for wireless sensor networks using transmission range slicing," Computer Communications, vol. 31, pp. 1687-1704, June 2008.

[3] D. Virmani and S. Jain, "Comparison of proposed data dissemination protocols for sensor networks using J-Sim," in 2009 IEEE International Advance Computing Conference. IACC 2009, Patiala, India, 2009, pp. 1179-1186.

[4] Y. Zhang and L. Wang, "A comparative performance analysis of data dissemination protocols in wireless sensor networks," in Proceedings of the 7th World Congress on Intelligent Control and Automation, Chongqing, China, 2008 pp. 6663-6668.

[5] S. Saha and M. Matsumoto, "A framework for disaster management system and WSN protocol for rescue operation," in TENCON 2007 - 2007 IEEE Region 10 Conference, Taipei, Taiwan, 2007, pp. 1315-1318.

[6] F. C. Delicato, L. Fuentes, N. Gamez, and P. F. Pires, "A middleware family for VANETs," in Ad-Hoc, Mobile and Wireless Networks. 8th International Conference, ADHOC-NOW 2009, Murcia, Spain, 2009, pp. 379-384.

[7] R. Pries, T. Hobfeld, and P. Tran-Gia, "On the suitability of the short message service for emergency warning systems," in VTC 2006-Spring. 2006 IEEE 63rd Vehicular Technology Conference, Melbourne, Vic., Australia, 2006, pp. 991-995.