Web Display of Real-time Wind Sensor Data

by

Victoria Pimentel and Bradford G. Nickerson

TR11-214 , December, 2011

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
http://www.cs.unb.ca

Contents

List of Figures

Abstract

This project explores different ways of accessing and delivering real-time environmental sensor data efficiently at a relatively high speed. We compared average latencies between the WebSocket protocol, HTTP long-polling and HTTP polling in delivering real-time wind sensor data at a rate of 4 Hz. On average, we observed that the WebSocket protocol provides an improvement in latency for real-time applications. We implemented two Jetty servlets in two different computers and a MySQL database. One servlet runs an application that reads data from the sensor and uses the WebSocket protocol to transmit real-time sensor data and to provide a full duplex communication channel with clients. This application also displays and stores real-time measurements in a database that is used by the second Jetty servlet. This second application provides a query channel to display subsets of sensor data. The web applications use the Canvas element from HTML5 to display sensor data. Graphical user interfaces are provided for display of real-time wind data and for displaying (and exporting) the results of a database query.

1 Introduction

This project investigates the real-time measurement and display of relatively high-speed (i.e. up to 4 Hz) environmental sensor data. This project develops a web-based software platform for use by graduate students and researchers working on sensor networks. The project involves developing software for two computers. One reads real-time sensor measurements from a serial port and a second one stores and serves up sensor data. The speed at which measurements are delivered to the user provides evidence of the application's ability to deliver real-time sensor data and to keep up with the sensor output rate.

We explored different ways of accessing data from a sensor connected to a serial port. This data needs to be accessed from a web application to make sensor measurements available to all connected clients. We explored the feasibility of current web-based protocols in delivering such data efficiently to web browsers from web servers. We considered the requirement of a full duplex communication channel between users and the wind sensor.

The project also investigated storage of real-time sensor data in a web-accessible database. Support for queries on date range and on range of wind direction and speed was implemented.

A graphical user interface was designed and implemented to provide both real-time and historical measurement data. This report documents this investigation and presents our findings.

2 Software Architecture

The project develops in two computers running CentOs. The observation base station 1 computer has a Jetty [3] server running with the WindComm web application. This application provides a near real-time channel for the clients that request new sensor data. The database base station 2 computer has a Jetty server running with the BaseComm web application.

This application provides a database query channel for clients that request less recent sensor data. The project architecture is shown in Figure 1 and 2.

2.1 The Gill WindSonic Wind Sensor

The Gill WindSonic is a robust, ultrasonic wind sensor with no moving parts. The sensor measures wind direction and speed. We connected the WindSonic option 3 with analog plus serial (RS232, RS422 and RS485) output and serial number 08320007, to the base station 1 through an RS232 output cable, which is connected to a USB serial port in the base station 1 computer via an adapter. We simulated wind with a rotating fan that provides continuously changing wind direction and speed.

The WindSonic wind sensor can operate in three modes; continuous, polled and configuration. Continuous and polled modes are measurement modes. In continuous mode the sensor sends wind measurements at a fixed constant rate. In polled mode the sensor sends data upon a request from the base station 1 computer. In configuration mode, settings for the wind sensor can be changed such as measurements units, change of measurement mode and output rate.

In the default format, a measurement message from the WindSonic wind sensor is 24 characters long. As explained in the WindSonic user manual [1], a message from the WindSonic looks like the one shown in Figure 3.

2.2 Observation Base Station 1

Base station 1 runs a Jetty server with the WindComm web application. The WindComm application implements a Jetty servlet that manages HTTP and WebSocket requests. The application reads data from the sensor and makes it accessible to clients through a web interface. The WindComm web application stores the data that is currently being received from the sensor in a remote database running on the base station 2 computer.

2.3 Database Base Station 2

Base station 2 runs a MySQL database and a Jetty server with the BaseComm web application. The database is used by base station 1 to store sensor readings and by the BaseComm web application. The BaseComm web application implements a Jetty servlet that manages HTTP requests to query the database with parameters especified by the user through an HTML form.

2.4 Client

The client connects to either WindComm or BaseComm web applications through a web browser. The web browser must support the Canvas element and the WebSocket protocol from HTML5. The client can connect to the WindComm web application to see real-time wind sensor data or to the BaseComm web application to query the database.

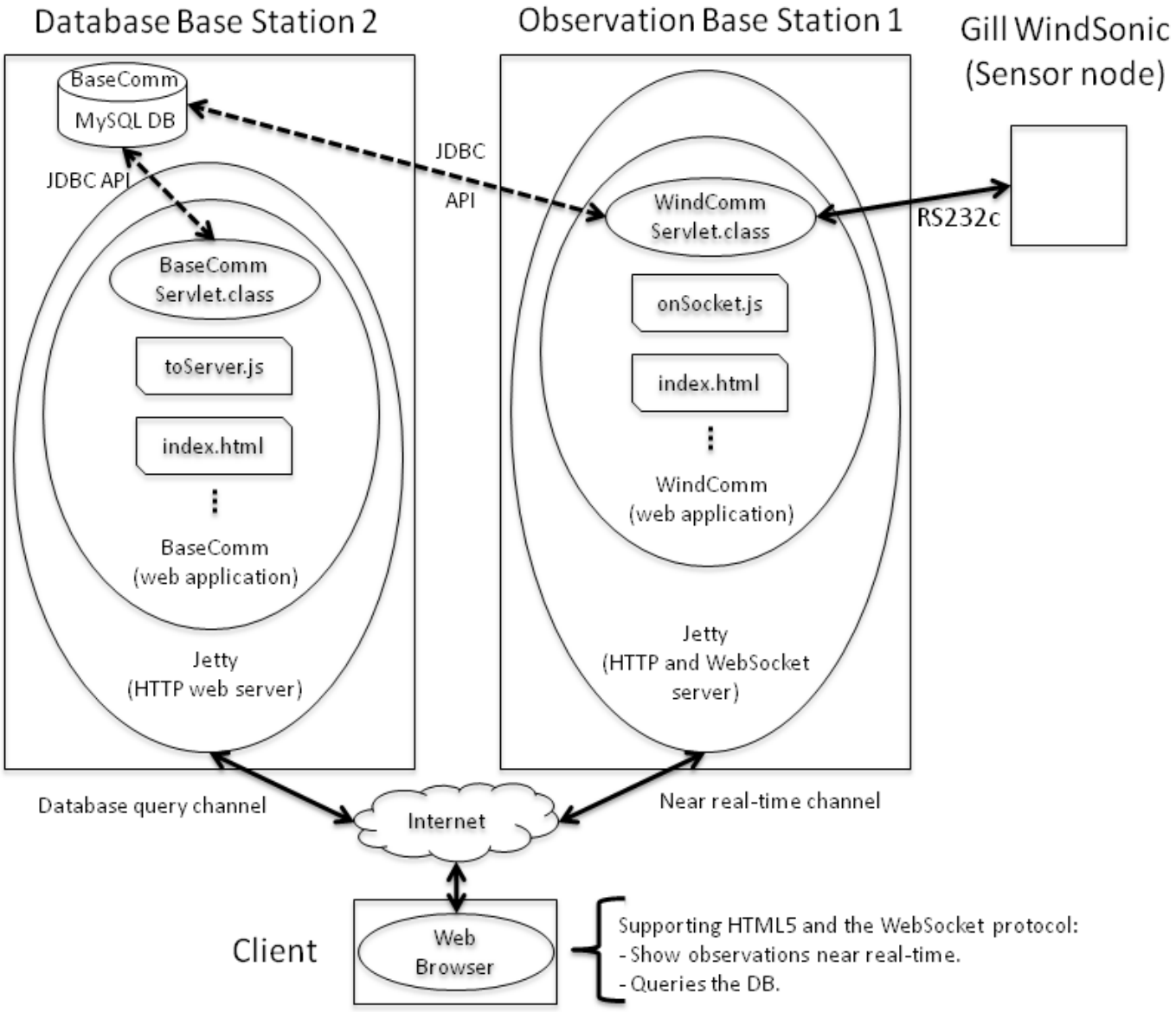


Figure 1: Architecture diagram showing the base station computers and their components.

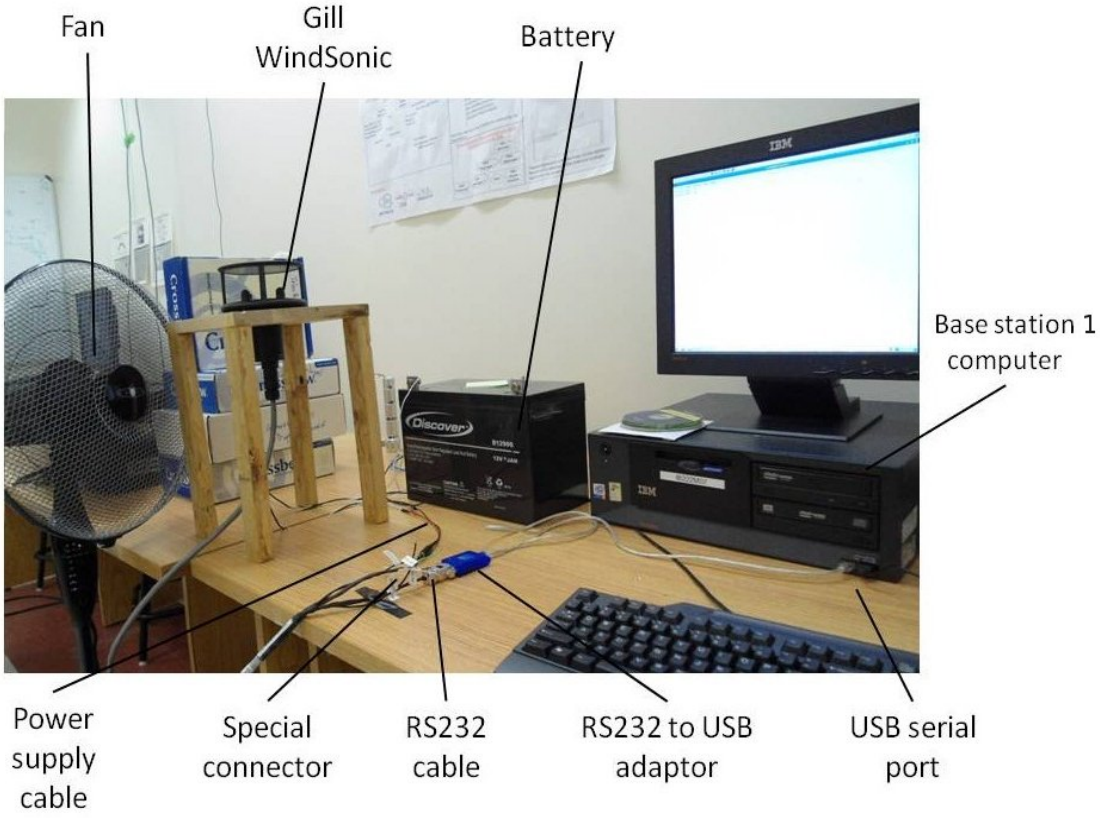


Figure 2: The Gill WindSonic wind sensor and the instruments and devices used in developing the applications.

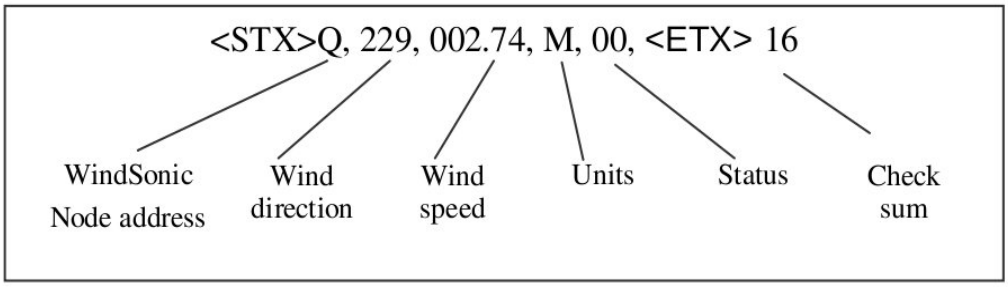


Figure 3: Default format for a wind measurement message of the WindSonic (extracted from [1]).

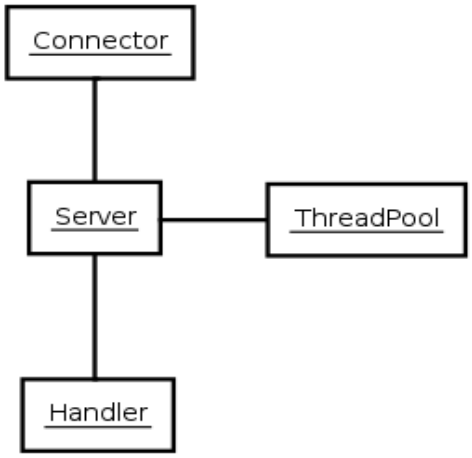


Figure 4: Jetty is the union between connectors, handlers and a thread pool (extracted from [2]).

3 Solution Design

There are many strategies for reading data from a sensor that is connected to a computer serial port. However, displaying this data in real time involves accessing the serial port and making the data accessible to any client efficiently. The database must keep up with the continuous execution of INSERT statements while processing any possible query by any client.

3.1 Wind Sensor Communication

WindComm uses the RXTX [4] Java library to access the computer serial port and communicate with the wind sensor. RXTX was chosen among three other techonologies [5] because the RXTX Java library does not involve file manipulation and can easily be accessed from a Java program. This project investigates web-based protocols to deliver high speed sensor data efficiently to web browsers from web servers. Therefore, efficient access to the serial port is very important and this can be achieved from a Java servlet using the RXTX library.

3.2 Jetty Web Server

Jetty is defined in Wikipedia as "a pure Java-based HTTP client/server, WebSocket client/server and servlet container (Application server) developed as a free and open source project as part of the Eclipse Foundation" [6]. Jetty is a server that implements the WebSocket API developed and being standardized by the W3C [7]. As shown in Figure 4, Jetty is the union between three components; connectors, handlers and a thread pool.

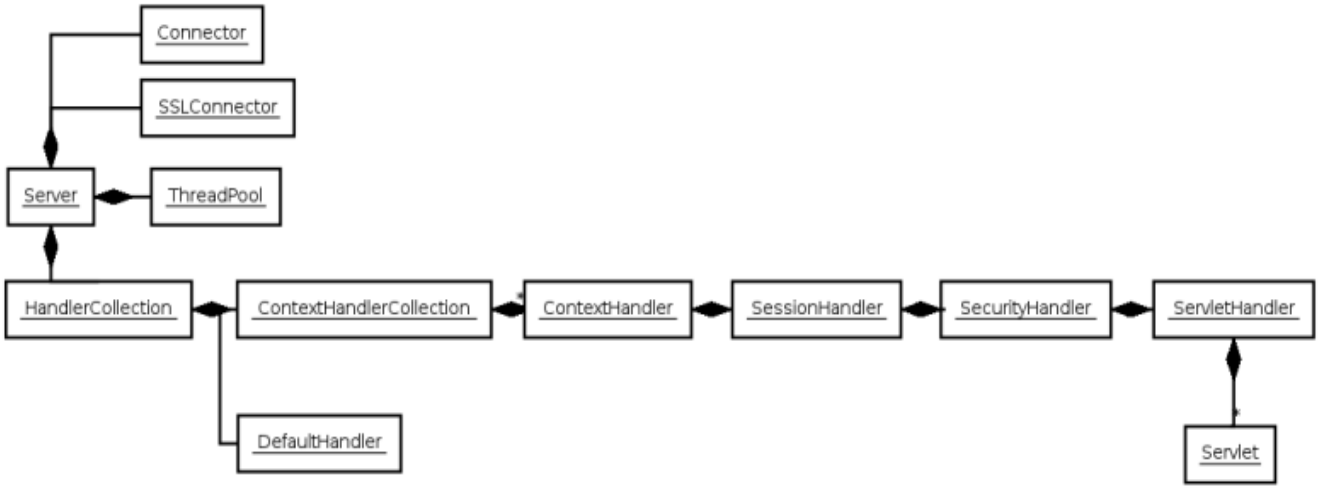


Figure 5: Jetty's detailed components and classes (extracted from [2]).

Connectors accept HTTP connections. Handlers service requests from the connectors and produce responses. The thread pool is simply the set of threads that do connectors and handlers work.

3.2.1 Web Applications

A web application can be configured in Jetty with sessions, security, listeners, filters, and servlets via a web.xml file used as a descriptor. This file is a WebbAppContext, which is a derivation of a ContextHandler. A ContextHandler is a handler that groups other handlers below a particular URL. A WebAppContext is a convenience class that assists the construction and configuration of other handlers to achieve a standard web application configuration. Figure 5 shows the relation between the classes involved.

3.2.2 Implementation of the WebSocket API

The WebSockets protocol has two parts. The first one is called the handshake. It consists in a handshake message from the client and the response handshake from the server. The second part is data transfer. To establish a WebSocket connection, a servlet must receive and respond to HTTP requests and must support upgrade requests from HTTP to WebSocket connections.

Jetty's implementation of the WebSocket API is fully integrated into Jetty HTTP server and servlet container. Thus, a Jetty servlet can process and accept a request to upgrade an HTTP connection to a WebSocket connection. Figure 6 shows a sequence diagram that describes the interaction between the server and the client to establish a WebSocket connection.

In a Jetty servlet the method init() gets called once upon the first client request to the

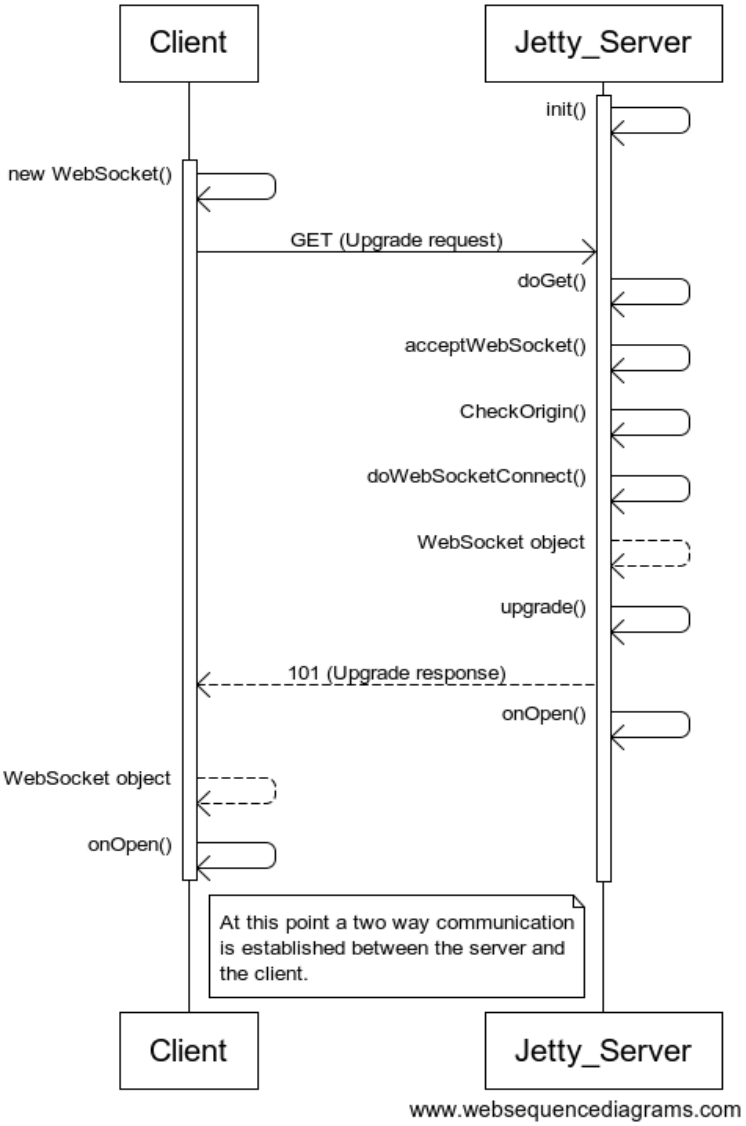


Figure 6: Sequence diagram that shows interaction between any WebSocket client and a Jetty servlet.

server. This method is to manage resources that are held for the life of the servlet and must start and configure the WebSocketFactory. When a server receives a GET request, the method doGet() is called. This method handles all HTTP GET requests. Both of this methods, init() and doGet(), belong to the HttpServlet Java class. Thus, the WindComm servlet class extends javax.servlet.http.HttpServlet class and implements the init() and doGet() methods.

Jetty manages the WebSocket implementation through a WebSocketFactory class. The implementation of the doGet() method should call the method acceptWebSocket() from the WebSocketFactory class. The acceptWebSocket() method belongs to an Acceptor interface and calls another two methods; checkOrigin() and doWebSocketConnect(). The method checkOrigin() must be implemented to check the origin of the incoming WebSocket handshake request. The method doWebSocketConnect() is called when an upgrade request is received in the server. If all the conditions are met, the doWebSocketConnect() method must be implemented to return a WebSocket object.

Once the WebSocket object is returned to the server, the method upgrade() gets called to upgrade the connection by sending the correct message to the client. This message corresponds to the server handshake and must have the 101 status code and a Sec-WebSocket-Accept key.

The WindComm servlet class contains a the WindCommWebSocket class that implements the class org.eclipse.jetty.websocket.WebSocket. Thus, the WindCommWebSocket class implements the methods that handle the socket connection such as onOpen(), onMessage(), onClose(), and onError(). Once the upgrade response has been sent to the client, the method onOpen() gets called.

On the other side, the client starts the WebSocket connection with the following call:

```
this._ws=new WebSocket(hostlocation, protocol);
```

The variable "hostlocation" is the name of the host and the location of the servlet. The variable "protocol" corresponds to the optional argument of a protocol for a particular web application.

This method parses components and returns a new object. It establishes a socket connection by sending an upgrade request to the server. Then, the client waits for the server response and validates it. If everything went fine, the client is ready to handle the socket methods such as onOpen(), onMessage(), onError() and onClose(). These methods are implemented in the client.

3.2.3 Testing Jetty Setup

Jetty distribution comes with a web application called "test". Once the Jetty server is locally installed, this test application can be accessed through http://localhost:8080 . The test.jar file comes with several demos for all of Jetty's features, including the WebSocket implementation. The chat demo application was used to test web browser support and to check the successful installation of Jetty web server. The chat implementation explained by

Aditya Yadav [8] provides a very good understanding on how to work with the WebSocket protocol using Jetty components with Java and JavaScript.

Jetty was chosen among other servers because the WebSocket protocol implementation is fully integrated into the Jetty servlet container. Thus, a servlet that receives HTTP requests can be implemented to upgrade to a WebSocket connection. Besides, Jetty is a pure Java based client/server and servlet container which supports the use of the RXTX Java library to access the serial port to communicate with the sensor.

3.3 Database Design

The database was designed to store all the information from the WindSonic messages and a timestamp taken at the time the measurement was received at the server. Considering that the WindSonic messages have the format presented in Figure 3, the database was designed according to the Entity Relationship (ER) model shown in Figure 7. The relational model corresponds to: Measurement(dateTime, ms, UTCoffset, nodeID, direction, speed, units, status).

'Measurement' is the name of the only entity in the database. This entity represents the sensor readings that are received by base station 1. The attribute 'nodeID' is the node address or identifier, the default value is 'Q'. The attribute 'direction' is the wind direction reported by the sensor, on the default format the units are degrees. The attribute 'speed' is the wind speed reported by the sensor, on the default format the units are m/s. The attribute 'units' is the units of measure for wind speed, on the default format the units are m/s represented with a letter 'M' in the units attribute. The attribute 'status' is an operation code that must be 00 to indicate correct operation. All these mentioned attributes are received from the sensor for each reading. The next two attributes; 'time' and 'UTCoffset', are calculated on the server before sending the information to the database.

The attribute 'dateTime' is a timestamp MySQL type that displays date and time values in a 'YYYY-MM-DD HH:MM:SS' format. Microseconds cannot be stored into a column of any temporal data type. However, milliseconds are important when the output rate of the sensor can be up to 4 times per second. Thus, the attribute 'ms' stores a three digit number that is the milliseconds part of the timestamp. The attributes 'dateTime' and 'ms' form the compound attribute 'time' which is the primary key of the database. Measurements are always taken at different times, so no two measurements can have the same 'time'.

The attribute 'UTCoffset' is the time offset from Coordinated Universal Time (UTC) at base station 1 when the measurement was received at the server. This attribute is important when calculating latency or comparing timestamps.

The database was implemented in base station 2 using MySQL relational database management system.

3.4 Experimental Design

To investigate the use of the WebSocket protocol to transmit real-time wind sensor data at a rate of 4 Hz, we ran tests to compare one way transmission latency of the WebSocket

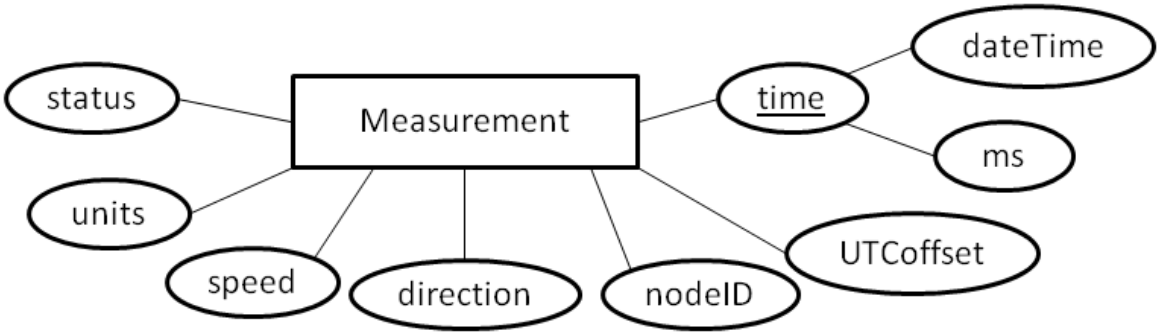


Figure 7: Database design to store WindSonic messages and a server timestamp.

protocol, long polling and the best case scenario for HTTP polling in a real-time application [9]. We experimentally validated latency behavior at a 4 Hz rate for low volume (around 100 bytes per second of sensor data) communication typical of real-time sensor networks.

We implemented three web applications called WindComm, LongPollingWindComm, and PollingWindComm that use the WebSocket protocol, HTTP long polling and HTTP polling, respectively. Our tests ran each of these applicationes one after another at three different times of day until 1,198 messages were succesfully delivered for each application. At an output rate of 4 Hz, the fastest test lasted five minutes. The first test was planned for around 8:00 (not busy), the second test around 13:00 (normal traffic), and the third test around 20:00 (busy). We chose these different times of day to vary the state of the network for all of the tests.

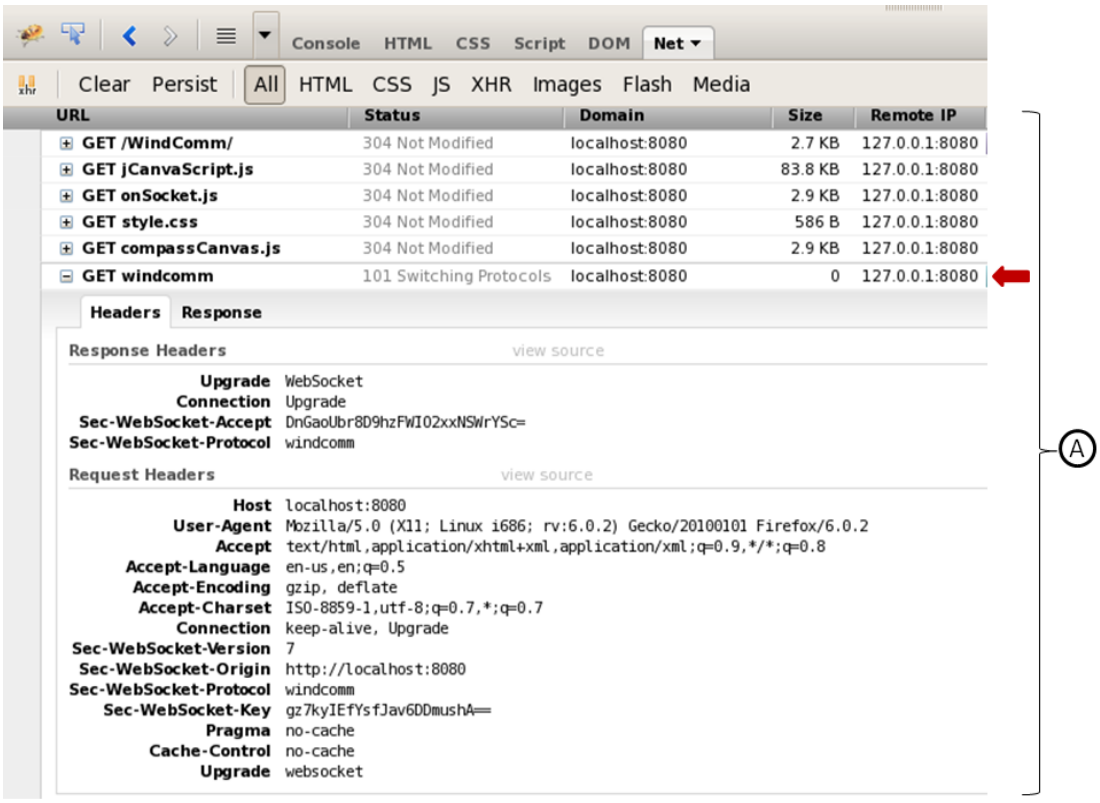
We ran the tests between our server located at the University of New Brunswick, Fredricton, N.B., Canada, with clients in Canada, Venezuela, Sweden and Japan.

4 WindSensor Web Application

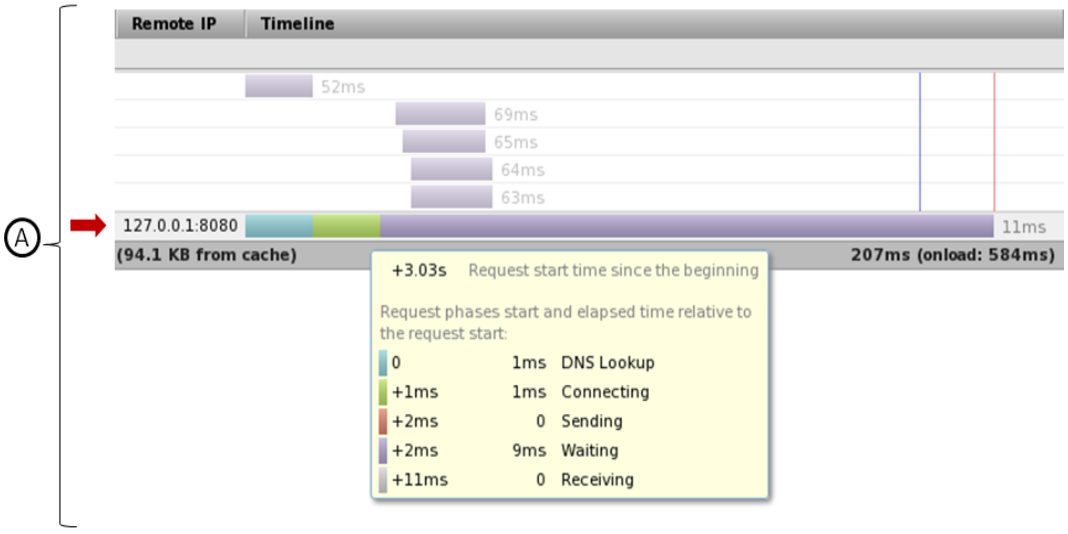
WindSensor is a very simple user friendly web application that shows the project homepage and has links to the WindComm web application in base station 1 and to the BaseComm web applicacion in base station 2.

5 WindComm Web Application

WindComm is the web application that runs in base station 1. WindComm is an application that uses the WebSocket protocol to communicate near real-time sensor data to all the clients connected. Figure 8 (a) shows HTTP headers from the WindComm web application obtained using Firebug [10] for Mozilla Firefox version 6.0.2. These headers show the upgrade request, from HTTP connection to WebSocket connection, and the server response. Figure 8 (b) shows the timeline provided by Firebug for the "GET windcomm" upgrade request shown in Figure 8 (a). These times may vary.



(a) Firebug shows the HTTP requests from the client. The red arrow points at the upgrade connection request from HTTP to a WebSocket connection. The HTTP headers displayed correspond to the upgrade connection request.



(b) Timeline from Firebug for the WindComm web application requests shown in part (a) of this Figure. The red arrow points at the "GET windcomm" upgrade request timeline.

Figure 8: Firebug Net panel screenshots. To get the complete view add part (b) to the right of part (a) of this Figure as indicated by the A letter.

The WebSocket protocol is a very helpful tool for this project. The output rate of the sensor can be up to 4 Hz. Thus, a high speed communication is required to deliver near real-time data to clients. The sensor has configuration settings that a client can change; measurement units, output rate, measurement mode, among others. Therefore, a full duplex communication between the sensor and the clients is necessary so a client can send configuration commands to the sensor and receive the response.

The WindComm web application was developed following Jetty specifications. Figure 9 shows Jetty's components class diagram and the WindComm web application file structure.

The Java class WindCommServlet is the servlet that handles HTTP and WebSocket requests. This class uses the BaseCommDB class to connect to the database in the base station 2, and the WindSonicSensor class to communicate with the sensor. onSocket.js implements the client socket methods and communicates with the server to receive sensor data.

The BaseCommDB class uses the Java DataBase Connectivity (JDBC) API [11] to communicate with the database running in the base station 2. JDBC provides access to remote and local databases from a Java program.

The WindCommServlet class communicates with the clients using the WebSocket protocol. The servlet creates a thread to start and handle communication with the sensor while the servlet serves requests from clients. The thread created by the servlet uses the RXTX Java library [4] to connect and communicate with the wind sensor. Then, the thread broadcasts the messages received from the sensor to all current connections. If the client has selected to store current data in the database, the thread executes an INSERT statement every time a sensor message is received using an instance of the BaseCommDB class.

As soon as a client is connected, the server sends information to announce if the sensor is in continuous, polled or configuration mode and if the sensor readings are being stored in the database. Once the client receives these messages, the information is displayed to the user and a starting code is sent to the server. This code can have three possible values; 0, 1 or 2. 0 indicates that the client started the real-time display. 1 indicates that the client started the database recording. 2 indicates that the client started both the real-time display and the database recording.

When a client opens the WindComm web application, the client is forced to start the real-time display before starting the database recording. This is done to establish the WebSocket connection first so the server can tell the client the current sensor mode and if the database recording has already been started or not. Then, the user can start the database recording. If so, the client sends to the server the starting code 1. When the server receives the starting code 1, the server starts the connection with the database. If successfull, the server will start storing sensor data in the database.

As soon as the server receives the starting code, a full duplex communication has been established. Thus, the client can send a message to the sensor through the servlet. The user interface has an input box, once the input text is entered, the servlet handles the message with the onMessage() method. This method assigns the input text to variable "forThread" that the thread checks periodically. When the thread finds that variable "forThread" is not

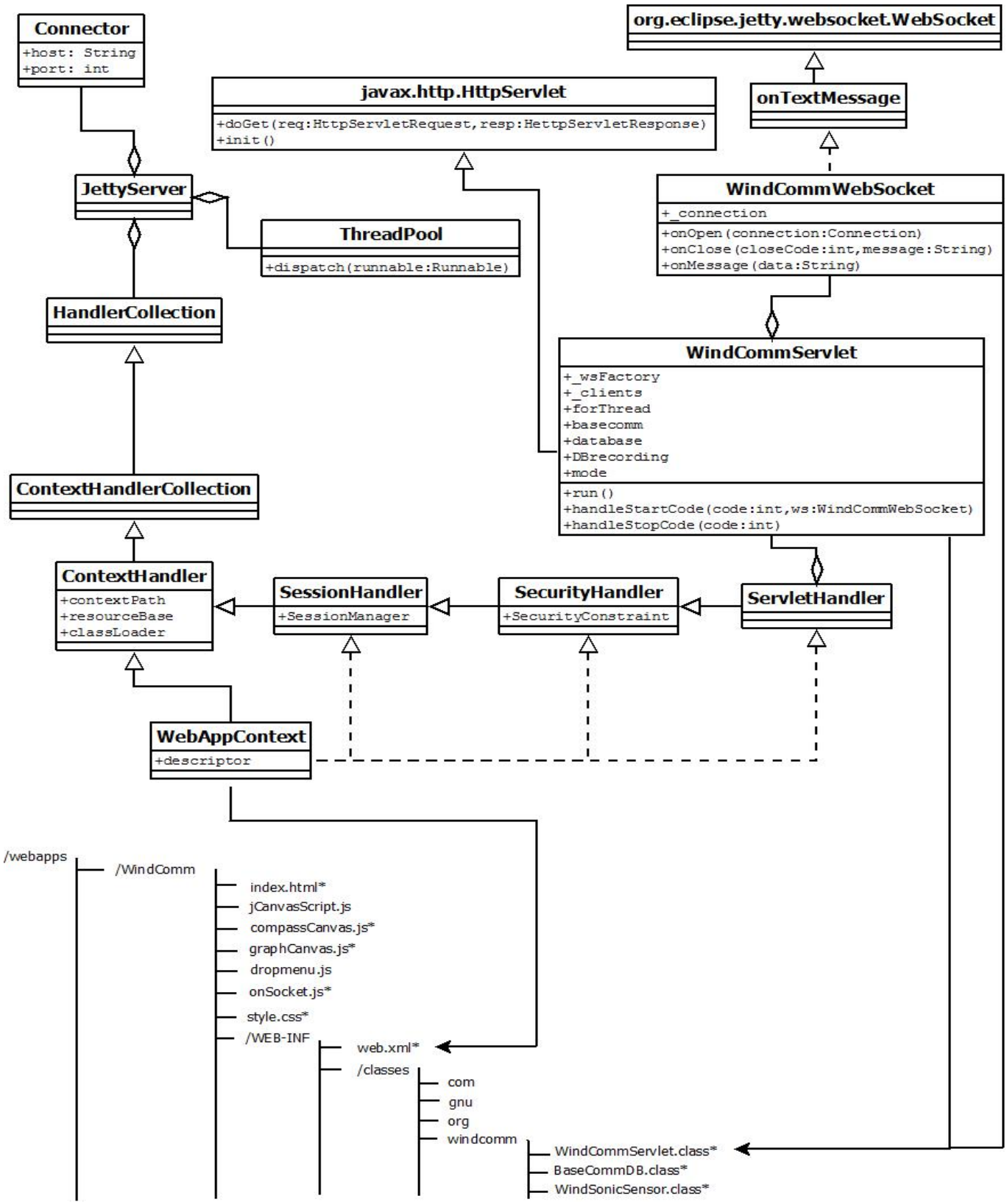


Figure 9: Jetty class diagram and WindComm web application file structure. Files followed by an * were created for this project.

null, it checks if "forThread" contains a valid command and sends it to the sensor. If the input text from the user is not a valid command, the thread does not send it to the sensor and a message is displayed to the user. The sensor responds to the command and the thread sends the response to all currently connected clients.

The sequence diagram shown in Figure 10 describes the methods and the messages sent between a client and the servlet to start a WebSocket communication with a starting code value equal to 0, and change the sensor from continuous to configuration mode.

The WindSonic does not report a value for wind direction when the speed value is below 0.05 m/s. To handle this, the WindComm web application saves the latest nonempty reported value for wind direction. This value is used when an empty direction value is reported. The most recent nonempty value for wind direction is stored in the database with a negative symbol to indicate a fake value given that the wind speed is below 0.05 m/s. If there has been no reported direction value since the application started (i.e. all wind speeds are less than 0.05 m/s since the application started), then WindComm uses the value zero for missing wind directions. These measurements are not real; they indicate missing wind direction values since the application started. In these cases, if the database recording is on, the database records the zero value for this measurements.

If the user wants to stop the real-time display or the database recording, the client sends a stop code to the server. This code can have three possible values; 0, 1 or 2. The value 0 indicates that the client stopped the real-time display; 1 indicates that the client stopped the database recording; and 2 indicates that the client stopped both the real-time display and the database recording.

5.1 WindComm User Interface

The interface implemented to display sensor data to clients consists in two graphics drawn using the Canvas element from HTML5 and JavaScript. These two graphics are continuously being updated at the output rate of the sensor.

5.1.1 Compass-like Graphic

The compass-like graphic follows the approach described in the technical report [5]. The graphic displays wind direction values of the measurements received from the server.

Every time a measurement is received it is parsed and the wind direction value is passed as an argument to a function that accesses the Canvas element of the file to display it. The compass-like interface shows the last 5 values received for wind direction as shown in Figure 11. The radius of the compass is divided into 5. Each segment represents a measurement reported. Time increases to the center of the compass, which means the most recent measurement is the one touching the compass circumference and the oldest is the one closest to the center of the compass circle. With the WindSonic in continuous mode, Δ can be 1, 0.5 or 0.25 seconds that correspond to the 1, 2 or 4 measuremets per second output rate. Δ could have other values in polled mode depending on the requests made by the user.

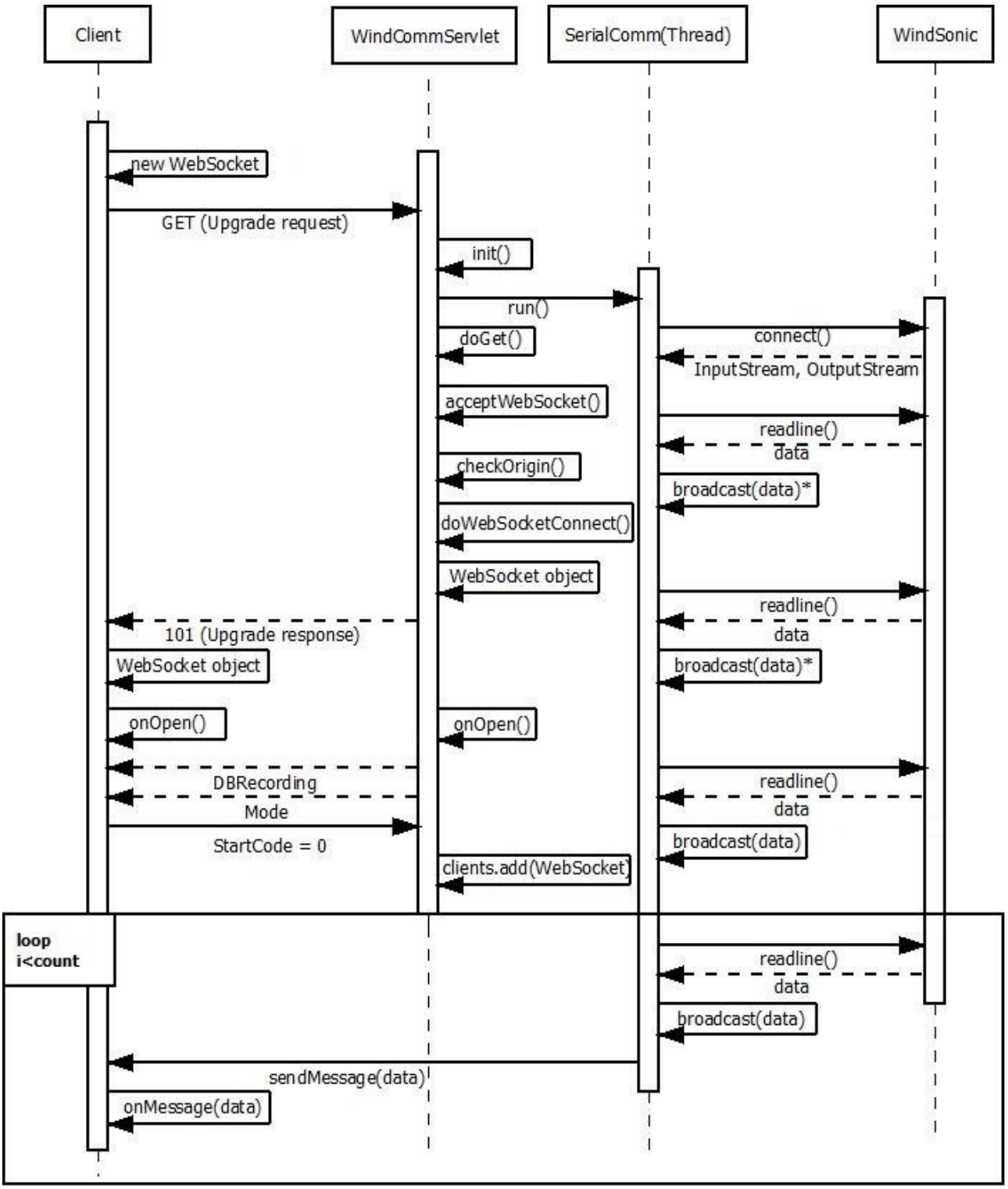


Figure 10: Sequence diagram that shows interaction between WindComm servlet and a client to change the sensor to configuration mode.

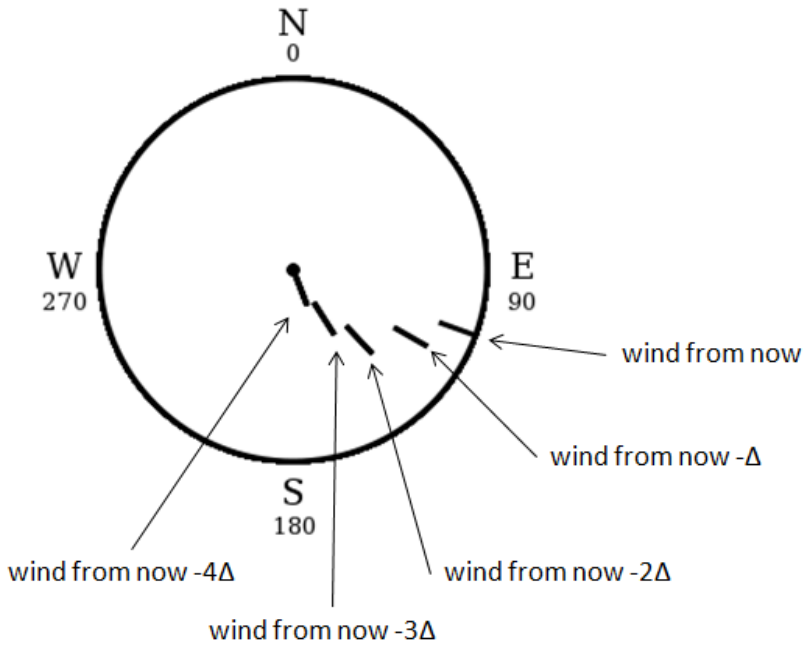


Figure 11: Compass-like graphic that shows 5 last measurements, time intervals are shown.

When the wind speed is below 0.05 m/s, the graph shows the most recent nonempty wind direction value colored in red.

5.1.2 Two-dimensional Graphic

The two-dimensional graphic uses two arrays with a maximun length of 200. One of these arrays contains wind speed values and the other one wind direction values. Both of the arrays are passed as arguments to a function that calculates the number of pixels per unit on both the x and y axes. The values contained in both arrays are multiplied by the number of pixels per unit. These new vales are interpreted as point coordinates and a line is drawn following each of these points.

As shown in Figure 12, the graphic has two y axes. The left y axis shows the scale for wind speed values reported in m/s. The right y axis shows the scale for wind direction values reported in degrees. The graph plots up to 200 wind speed and direction values. The most recent values received from the server are plotted at the right end of the graphic. Once the client has received 200 measurements, the graph behaves as a queue.

When the wind speed is below 0.05 m/s, the latest nonempty wind direction value is used in the graph.

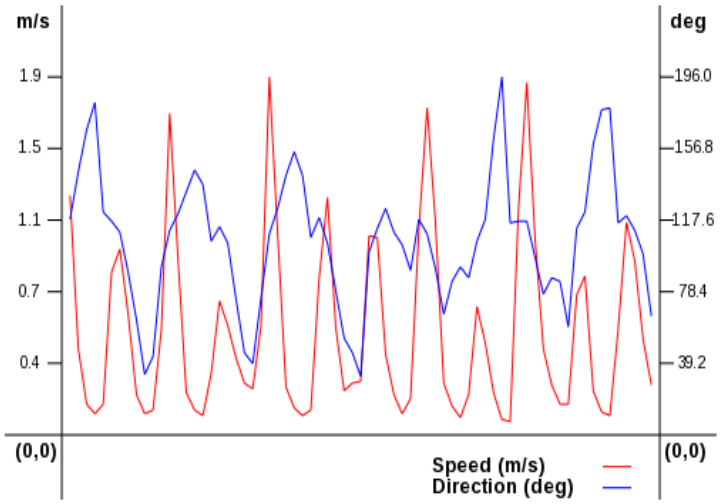


Figure 12: Two-dimensional graphic showing the last 71 wind speed and direction values received.

6 BaseComm Web Application

BaseComm is the web application that runs in base station 2. This application uses HTTP to receive requests from clients to query the database that runs locally. The BaseComm web application was developed following Jetty specifications. Figure 13 shows Jetty's components class diagram and the BaseComm web application file structure.

The Java class BaseCommServlet is the servlet that handles HTTP requests and responses. When the servlet is initialized, connection is established with the database. The user submits requests thorugh an HTML form, the servlet receives the user's input and builds a query that runs in the database. The client reads the response and displays the contents in a table.

6.1 BaseComm User Interface

A query result with negative values for wind direction means that the wind speed value is below 0.05 m/s and that the direction value displayed corresponds to the latest nonempty wind direction reported. To avoid confusing users, the negative symbol is removed from the wind direction value and another column labeled "<0.05 m/s" is displayed on the results table. This column can have two possible values: "Y" or "N". "Y" means the wind speed is below 0.05 m/s, and the wind direction value corresponds to the most recent nonempty value reported. "N" means the wind speed is greater or equal to 0.05 m/s.

Table 1 is an example of how the table presented to users looks like when the client displays a query response from the server.

Once the table is presented, the user can choose to 'Export' or 'Plot' the results presented in the table. To 'Export' means to present the table data in a plain text format with columns

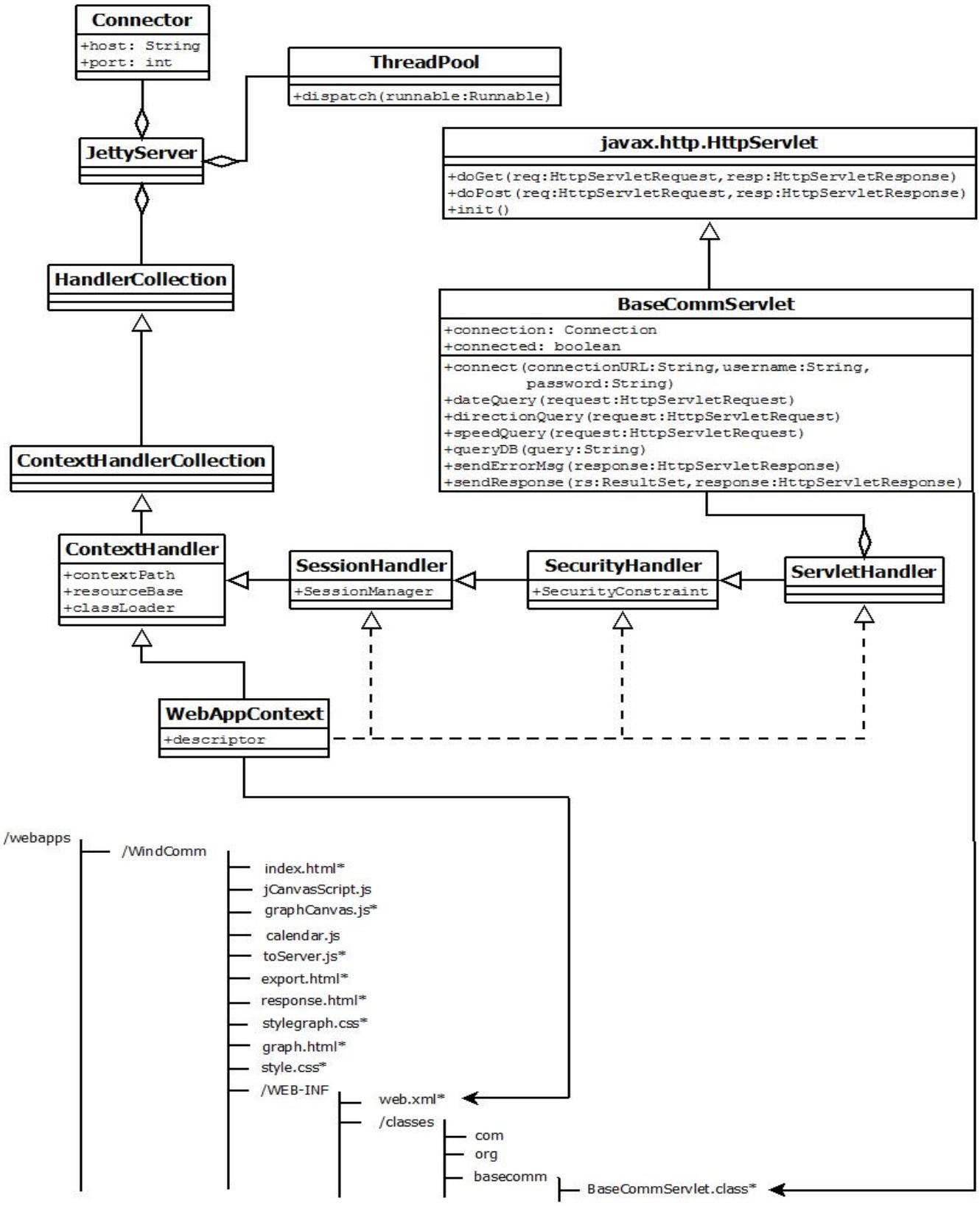


Figure 13: Jetty class diagram and BaseComm web application file structure. Files followed by an * were created for this project.

Node Address	Direction	Speed	Units	Status	Date and time	UTC offset	<0.05 m/s
Q	118	1.25	M	00	2011-12-0713:09:23.963	-4.0	N
Q	144	0.45	M	00	2011-12-0713:09:24.970	-4.0	N
Q	144	0.03	M	00	2011-12-0713:09:25.978	-4.0	Y

Table 1: Example of the table that is used to present query results to users.

separated by the tab character ('\t') and rows separated by the new line character ('\n'). The user can copy the text from the HTML table presented or use the exported plain text format as input to other applications.

To 'Plot' means that the data presented in the table as a result of a query is plotted in the two-dimensional graphic used in the WindComm web application. A new tab is opened with the graph in a transparent background that can be copied or saved as an image.

7 Results

We developed a database and two web-based softwares to display sensor data. The first web application is called WindComm and accesses the database to store sensor measurements. The second web application is called BaseComm and queries the database.

The WebSocket protocol was chosen to be used on the WindComm web application to deliver real-time sensor data. Tests results show that for the real-time, low volume continuous data used, HTTP polling average latency is signficantly higher (between 5 and 1,767 times higher) than either WebSocket or long polling. The WebSocket protocol can have a lower or greater average latency than HTTP long polling. Under heavy traffic conditions, or over longer distances (e.g. to Japan) the WebSocket protocol has significantly lower average latency than long polling. Detailed results of the testing done to measure latency are presented in [9].

WindComm displays real-time sensor data at a high speed efficiently. Sensor data is presented in graphics that are updated continuously at the output rate of the sensor. The WindComm web application also uses the WebSocket protocol to provide a full duplex channel between the user and the wind sensor through a user friendly interface. The user can change settings for the wind sensor and choose to save the data that is currently being displayed in a remote database.

BaseComm displays subsets of data that can also be converted into graphics and exported for future use. The user can query the database through a user friendly interface and obtain fast responses.

WindComm and BaseComm support multiple clients. However, WindComm only supports one WindSonic connected to the serial port of the base station 1 computer. The repercussions of the actions of one client (i.e. sending commands to the sensor, changing measurement mode, starting or stopping the database recording) will impact the rest of the clients connected and the application will display the changes to all the users currently

connected.

WindComm and BaseComm are complementary but they can run separately. WindComm can operate without BaseComm. However, if the MySQL daemon is not running in the base station 2, the user will not be able to save in the remote database the data that is being displayed. BaseComm can also operate without WindComm, but if there has been no data inserted in the database previously through WindComm, there will not be any data available.

8 Conclusions

We have investigated the use of the WebSocket protocol as an improvement on the latency of the HTTP polling technique for low volume real-time data communication on the internet. In high traffic and high network latency conditions, we observed that the WebSocket protocol was up to 534 times faster than the HTTP long polling protocol. The WebSocket protocol does provide better latency (on average) for real-time internet communication. More detailed conclusions on these tests can be found on [9].

We developed the WindComm web application that can keep up with a 4 Hz sensor output rate using the WebSocket protocol. Sensor measurements are delivered at a high speed and presented to the user in two graphics drawn using the Canvas element of HTML5. We worked with array sizes up to 200. The array contents were continuously changed. Our Canvas interface was erased and drawn again every time the WindComm application received a new measurement from the server; this happens depending on the output rate of the sensor at 1, 2 or 4 Hz. We observed that both Canvas graphics can keep up with continuous fast incoming data.

We used MySQL to create a database in which sensor data was continuously stored. If the database recording is on, every time the server receives a measurement from the sensor, an INSERT statement is executed using the JDBC API at the database that runs in the base station 2. As with the Canvas element, the INSERT statements were executed depending on the output rate of the sensor at 1, 2 or 4 Hz. Base station 1 is located in the same laboratory (ITB 214) as base station 2. Thus, they are part of the same network. This can influence the high speed of storing sensor measurements received at base station 1 in the database runnin on base station 2.

Jetty's implementation of the WebSocket API and protocol is very useful and provides a full duplex communication channel between the server and the clients. We observed latency values between 0 and 2 ms between the base stations 1 and 2. Between these two computers the network state barely affects latency and the traveling distance is very small. Jetty's implementation of the WebSocket is efficient and robust.

We developed a web-based Linux platform software for the WindSonic wind sensor that provides real-time sensor data and access to subsets of previous sensor readings. We used two base station computers to distribute the load and provide the user with high speed responses.

How this Document was Created

LaTeX [12] version pdfeTeX 3.141592-1.21a-2.2 (Web2C 7.5.4) with Texmaker [13] version 3.1 on CentOS version 5.7.
Figures 1 and 7 using PowerPoint 2007 on Windows 7.
Figure 6 using WebSequenceDiagrams.com [14] on CentOS version 5.7.
Class diagrams and the rest of sequence diagrams using Dia [15] version 0.96.1 on Windows 7.

References

[1] Gill Instruments Ltd. *WindSonic User Manual*, July 2011. Issue 20, Doc No. 1405-PS-0019 http://www.gill.co.uk/data/manuals/manuals.htm.

[2] The Eclipse Foundation. Jetty/reference/jetty architecture. Internet (accessed October, 2011). http://wiki.eclipse.org/Jetty/Reference/Jetty_Architecture.

[3] Jetty webserver. Internet. http://jetty.codehaus.org/jetty.

[4] Rxtx wiki. Internet. http://rxtx.qbang.org/wiki/index.php/Main_Page.

[5] Victoria Pimentel and Bradford G. Nickerson. Communication and control of a digital wind sensor. Technical report, University of New Brunswick, October 2011. www.cs.unb.ca/tech-reports/documents/TR11-211.pdf.

[6] Jetty (web server) - wikipedia. Internet (accessed October, 2011). http://en.wikipedia.org/wiki/Jetty_%28Web_server%29.

[7] Ian Hickson. The websocket api. Internet, September 2011. W3C Working Draft. http://www.w3.org/TR/websockets/.

[8] Aditya Yadav. *Deploying HTML5*. CreateSpace, 2010.

[9] Victoria Pimentel and Bradford G. Nickerson. Communication and display of real-time sensor data using the websocket protocol. IEEE, Submitted to Internet Computing Magazine, November 2011.

[10] Chandan Luthra and Deepak Mittal. *Firebug 1.5: Editing, Debugging, and Monitoring Web Pages*. Packt Publishing, April 2010.

[11] Maydene Fisher, Jon Ellis, and Jonathan Bruce. *JDBC API Tutorial and Reference*. Prentice Hall, third edition, 2003.

[12] Helmut Kopka and Patrick W. Daly. *Guide to Latex*. Addison Wesley, fourth edition, February 2004.

[13] Texmaker (free cross-platform latex editor). Internet (accessed October, 2011). http://www.xm1math.net/texmaker/.

[14] Websequencediagrams.com. Internet (accessed October, 2011). http://www.websequencediagrams.com/.

[15] Dia a drawing program. Internet (accessed October, 2011). http://projects.gnome.org/dia/.

A User Manual

This manual was written to guide the user under the installation of the software developed and presented in this project. The software has been developed and is intended to be installed on a Linux platform. Once installed, the software can be accessed from any web browser client supporting HTML5 regardless of the operating system. Web browsers suggested are the latest versions of Mozilla Firefox and Google Chrome.

A.1 Jetty Web Server Installation

These steps will guide you through the Jetty web server download and installation.

1. Go to `http://download.eclipse.org/jetty` and select the stable-8 version and click 'download'.
2. Select the file extension of your choice (recommended: tar.gz) and save the file in the local directory of your choice.
3. Go to the directory where you saved the Jetty downloaded file and extract it in the directory where you want to install Jetty. We will refer to this directory as $JETTY_HOME.
4. Open a terminal and cd to $JETTY_HOME.
5. Run the following command: java -jar start.jar. Jetty is running in this terminal. If you close or stop the terminal, Jetty will be stopped and the server will not be accessible.
6. Open a browser and go to: `http://localhost:8080`.
7. If you see a Jetty welcome page, Jetty was successfully installed. Click on the link 'WebSocket' and try the chat application to test browser support for the WebSocket protocol. If you get a message that says that WebSockets is not supported on your browser, update your browser to the latest version and try the previous step again.
8. To stop Jetty, press 'Ctrl+C' on the console where you ran step 5.

A.2 WindSensor Web Application

The following steps will guide you through the WindSensor web application installation on the Jetty web server. In the following steps it is asumed that Jetty is correctly installed on the selected computer under $JETTY_HOME.

A.2.1 Installing the WindSensor Web Application

1. Select a computer to be base station 1.
2. Open the project CD and go to `Code/JettyFolders/` and copy the folder WindSensor.
3. Make sure Jetty is not running and paste the folder into `$JETTY_HOME/webapps`.

A.2.2 Running the WindSensor Web Application

1. Start jetty on a terminal running: java -jar start.jar.
2. Open a web browser and go to: `http://localhost:8080/WindSensor` if accessing from localhost. To access from other computer, go to `http://ipaddress:8080/WindSensor` and replace 'ipaddress' with the ip address of the computer where you are running the Jetty web server.

A.3 WindComm Web Application

The following steps will guide you through the WindComm web application installation on the Jetty web server and the RXTX Java library download and setup. In the following steps it is asumed that Jetty and Minicom are correctly installed on the selected computer (most Linux distributions come with Minicom, type 'minicom' in a command line to check). Jetty is installed under $JETTY_HOME, and the Gill WindSonic wind sensor is available, powered up and the sensor RS232 output cable is connected to a USB adapter [5].

A.3.1 Installing the WindComm Web Application

1. Use the computer selected as the base station 1 and connect the USB adapter cable from the WindSonic RS232 output to the computer serial port.
2. Open the main project directory and go to `Code/JettyFolders` and copy the folder WindComm.
3. Make sure Jetty is not running and paste the folder into `$JETTY_HOME/webapps`.
4. Open the main project directory and go to `Code/NetbeansFolders/WindComm/src/windcomm`, open the file BaseCommDB.java and change the ip address part of the variable connectionURL in line 23 with the ip address where you will install BaseComm web application and database. Compile the new file and place the new BaseCommDB.class under `$JETTY_HOME/webapps/WindComm/WEB-INF/classes/windcomm` to replace the older BaseCommDB.class.
5. Go to `http://rxtx.qbang.org/wiki/index.php/Download`, download the binary release 'rxtx 2.1-7r2 (stable)' and save the file in the local directory of your choice. These are the files of the RXTX Java library that will be used by WindComm to communicate to the wind sensor through the computer serial port.

6. Go to the directory where you saved the RXTX downloaded file and extract it in the directory where you want to keep the RXTX files. We will refer to this directory as $RXTX_HOME.

7. If you have a Firewall enabled and want to allow access to the web application from other computers, open the ports 123, 8080, 843 and 8787. These ports will be used by Jetty and by the WebSocket protocol.

A.3.2 Running the WindComm Web Application

1. Check that the sensor is connected correctly to the computer through the serial port, open a terminal and run: minicom. Make sure Minicom is well configured; Press 'Ctrl+A' and then 'O'. Move to 'Serial port setup' and change the necessary fields until the configuration looks like:

```
Serial Port Setup:
    | A -    Serial Device      : /dev/ttyUSB0               |
    | B - Lockfile Location     : /var/lock                  |
    | C -   Callin Program      :                            |
    | D -  Callout Program      :                            |
    | E -    Bps/Par/Bits       : 9600 8N1                   |
    | F - Hardware Flow Control : No                         |
    | G - Software Flow Control : No                         |
```

   Press 'Esc' and move to 'Modem and Dialing' and change the the Init string value to empty as shown above:

```
Modem and Dialing:
    |A - Init string .........                               |
```

   Press 'Esc', move to 'Save setup as dfl', press 'Enter' and then move to 'Exit' and press 'Enter'. Restart Minicom to check that the sensor is in continuous mode. You should see data incoming. If not, it might be that Minicom is not well configured or that the sensor is not in continuous mode. Make sure that Minicom is well configured and change the sensor to continuous mode using the WindSonic configuration commands. Exit Minicom pressing 'Ctrl+A', then 'X' and 'Enter'.

2. Open a terminal and cd to $JETTY_HOME.

3. If your computer runs in a 32-bit processor, run the following command:

```
export LD_LIBRARY_PATH="/$RXTX_HOME/Linux/i686-unknown-linux-gnu/"
```

   If your computer runs in a 64-bit processor, try the following:

```
export LD_LIBRARY_PATH="/$RXTX_HOME/Linux/ia64-unknown-linux-gnu/"
```

or:

```
export LD_LIBRARY_PATH="/$RXTX_HOME/Linux/x86_64-unknown-linux-gnu/"
```

4. Start Jetty on the opened terminal (where you performed the previous step) by running:

   ```
   java -jar start.jar
   ```

5. Open a web browser and go to: `http://localhost:8080/WindComm` if accessing from localhost. To access from other computer, go to `http://ipaddress:8080/WindComm` and replace 'ipaddress' with the ip address of the computer where you are running the Jetty web server.

6. Instructions for using WindComm are found in the user interface presented at `http://ipaddress:8080/WindComm`. Please read the button legends and the supported commands for the WindSonic.

WindSensor and BaseComm run in the same Jetty server. After going through the sections A.2.1 and A.3.1, you can run the steps on section A.3.2 and have both WindSensor and WindComm web applications running in the server.

A.4 BaseComm Web Application

The following steps will guide you through the BaseComm web application installation on the Jetty web server and the BaseComm database setup. On the following steps it is asumed that: Jetty and MySQL are correctly installed on the selected computer (most Linux distributions come with MySQL, type 'mysql' in a command line to check), Jetty is installed under $JETTY_HOME, and you have root access to MySQL.

A.4.1 Installing the BaseComm Web Application

1. Select a computer to be base station 2.

2. Open the main project directory and go to `Code/JettyFolders` and copy the folder BaseComm.

3. Make sure Jetty is not running and paste the folder into `$JETTY_HOME/webapps`.

4. Open the main project directory and go to `Code/NetbeansFolders/BaseComm/src/basecomm`, open the file BaseCommServlet.java and change the ip address part of the variable connectionURL in line 153 with the ip address where you are installing BaseComm web application and database. Compile the new file and place the new BaseCommServlet.class under `$JETTY_HOME/webapps/WindComm/WEB-INF/classes/windcomm` to replace the older BaseCommServlet.class.

5. Open a terminal as root and start the mysql daemon service with the following command: service mysqld start

6. Log into MySQL with root privileges. Create a database named BaseComm and add the following table:

```
CREATE TABLE Measurement (
        nodeaddress CHAR(1) not null,
        direction SMALLINT not null,
        speed DECIMAL(5,2) not null,
        units CHAR(1) not null,
        status VARCHAR(2) not null,
        datetime DATETIME,
        ms SMALLINT,
        utcoffset DOUBLE(2,1),
PRIMARY KEY(datetime, ms)
);
```

7. Create the following users with the following passwords:

```
username: basecomm        password: basecommaccess
username: windcomm        password: windcommaccess
```

8. Make sure yo are using the BaseComm database created and grant all privileges and usage to the users created on the previous step with the following commands:

```
> grant all privileges on BaseComm.* to basecomm;
> flush privileges;
> grant usage on *.* to basecomm identified by 'basecommaccess';
> flush privileges;

> grant all privileges on BaseComm.* to windcomm;
> flush privileges;
> grant usage on *.* to windcomm identified by 'windcommaccess';
> flush privileges;
```

A.4.2 Running BaseComm Web Application

1. Open a terminal and cd to $JETTY_HOME.

2. Start jetty on the opened terminal by running:

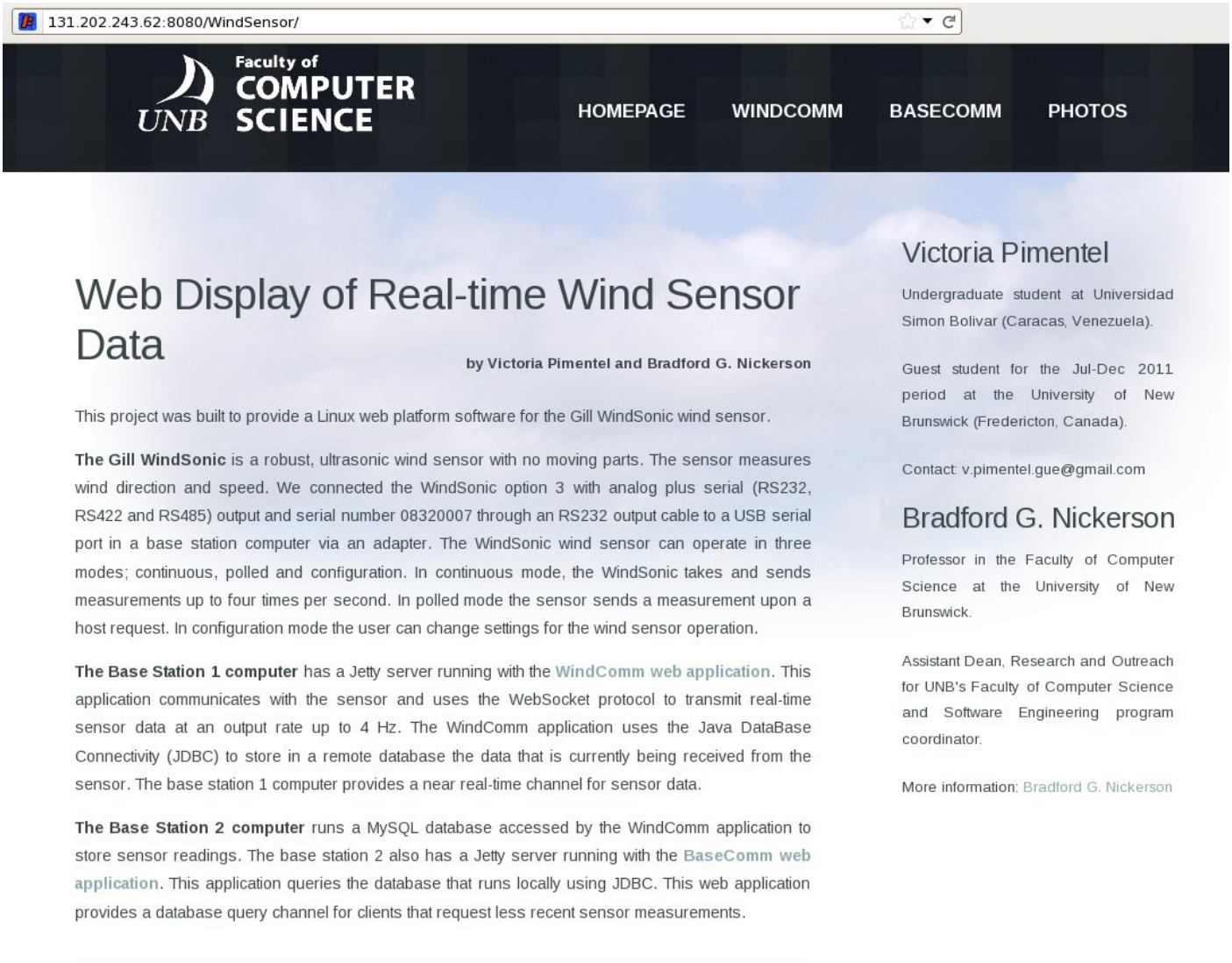
```
java -jar start.jar
```

3. Open a web browser and go to: `http://localhost:8080/BaseComm` if accessing from localhost. To access from other computer, go to `http://ipaddress:8080/BaseComm` and replace 'ipaddress' with the ip address of the computer where you are running the Jetty web server.

B Screenshots

B.1 WindSensor Web Application

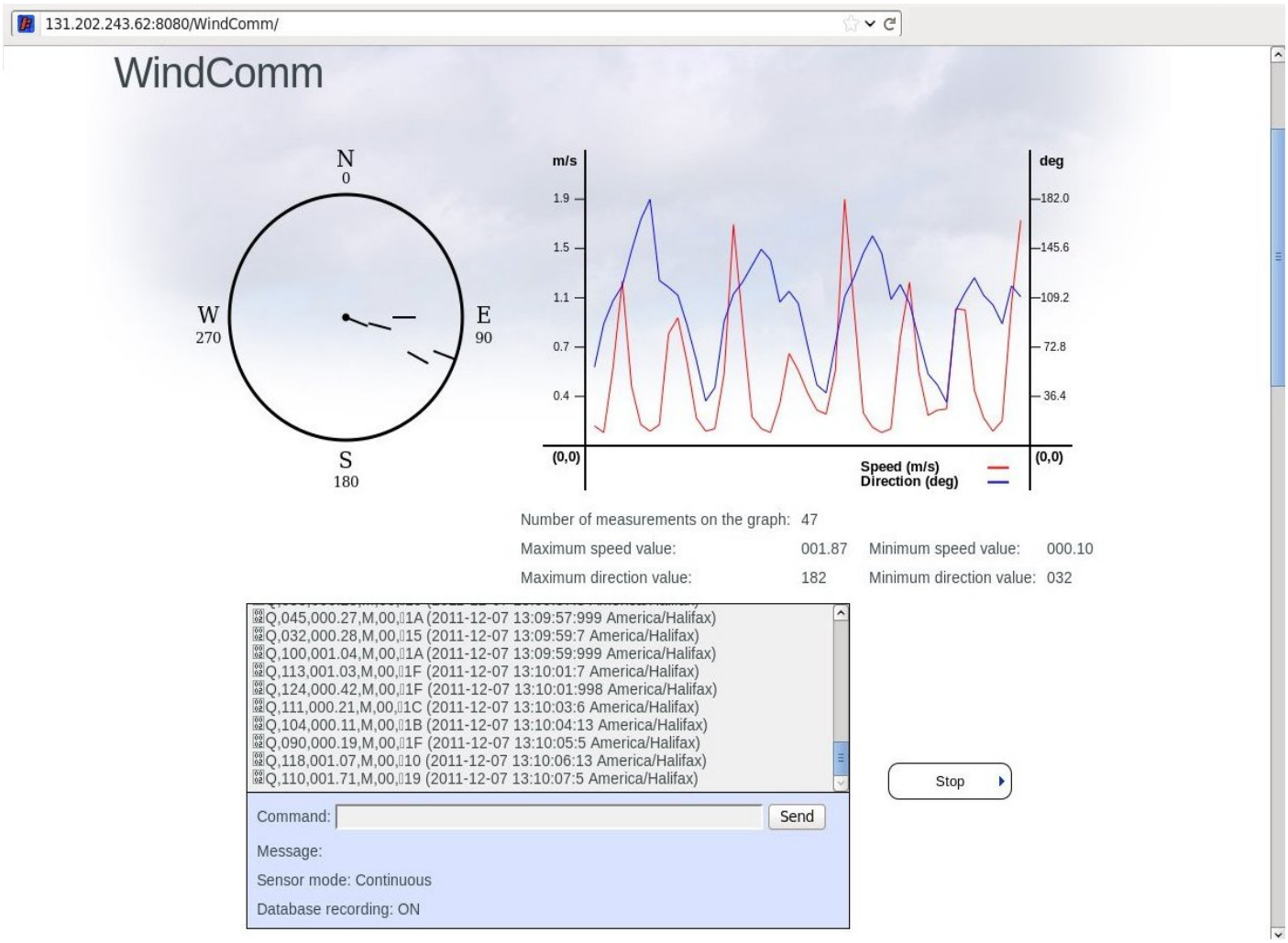
Figure shows a screenshot at a running WindSensor web application.



Screenshot at a browser window running the WindSensor web application.

B.2 WindComm Web Application

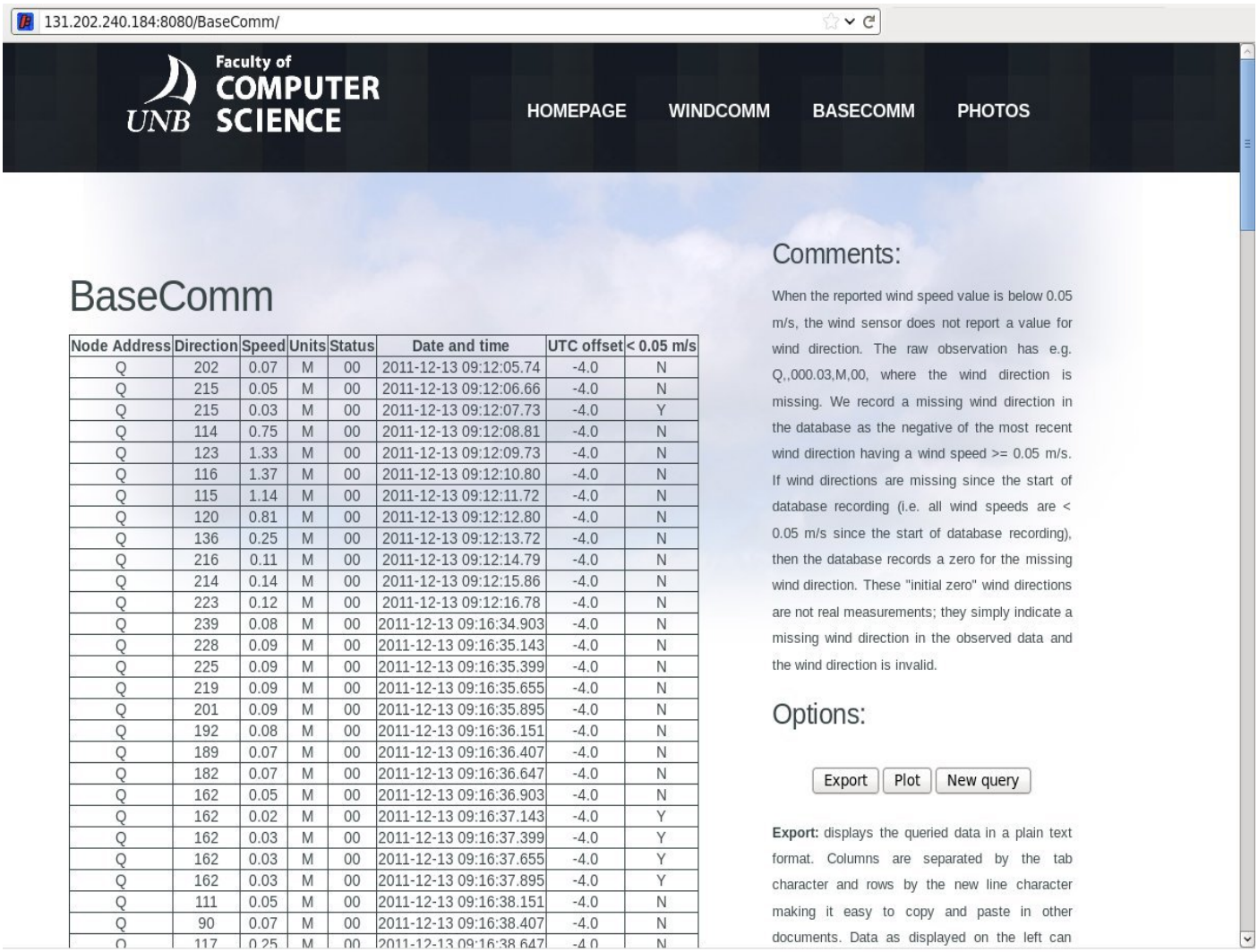
Figure shows a screenshot at a running WindComm web application.



Screenshot at a browser window running the WindComm web application with the sensor in continuous mode and the database recording ON.

B.3 BaseComm Web Application

Figure shows a screenshot at a BaseComm web application showing a query results set.



Screenshot at a browser window with the BaseComm web application showing a result table for a date query.