# Consulo: A 2-Tier SNMP Poll Manager

Vikhyat Korrapati (B11018)
Uday Mittal (B09074)
Dhruv Parmar (B09057)

May 14, 2012

## Contents

# 1 Features

- A convenient RESTful API to make it easier to write applications that use the Poll Manager.
- Automatic assignment of Network Elements to Low-level Data Collectors (LDCs) to simplify configuration.
- Supports adaptive polling, in which the rate of polling automatically adapts to how the variable changes.

# 2 Design

## 2.1 Description

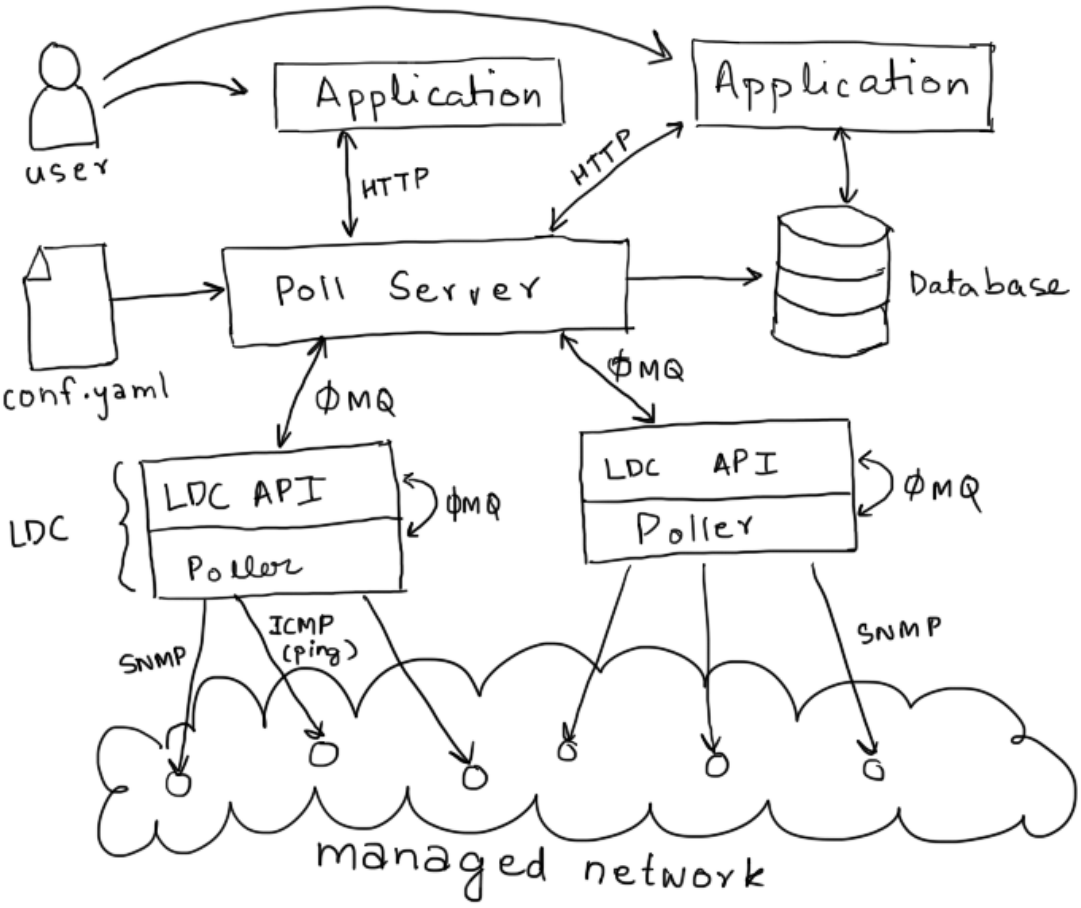
The Poll Manager consists of three components: a poll server, a LDC API layer and a poller. The Network Administrator will configure the poll server by giving it details of all of the LDCs and the set of network elements, OIDs and interval of the variables to be polled. When the server application is run, it reads all of the configuration variables and assigns network elements to LDCs automatically on the basis of the hopcount. This helps in reducing the amount of configuration work that the network administrator needs to do.

Each LDC consists of two major components: the LDC API and the Poller. The LDC API handles computing the hopcount, and forwards all other requests to the Poller. The Poller and LDC API communicate using IPC (Inter-process communication), and the LDC API and Poll Server communicate over TCP by default. The communication is protocol-agnostic, and can be changed to Inproc, IPC, TCP, PGM or EPGM (encapsulated PGM, in which PGM packets are wrapped into UDP packets to make it more administrator-friendly).

The Poll Server periodically reads data from the LDCs and saves it to a database. The database defaults to in-memory SQLite3, but database access is abstracted away and in order to switch to a different database backend one only needs to install the appropriate database drivers. Drivers for PostgreSQL and MySQL are provided for the database abstraction library we used in addition to the SQLite3 drivers.

In addition to saving values to a database, the Poll Server also provides a RESTful HTTP API to help simplify the implementation of some basic applications that don't need to perform advanced queries.

## 2.2 Block Diagram




## 2.3 Configuration, Communication and Serialization

All configuration is done using the YAML format. YAML is a straightforward machine parsable data serialization format designed for human readability and interaction with scripting languages. YAML was chosen over XML and JSON because it more easily read and written by humans.

For communication between the various components (with the exception of the HTTP API provided by the Poll Server), Consulo uses a messaging system, or "message-oriented middleware" called ZeroMQ. It is used in environments as diverse as financial services, game development, embedded systems, academic research and aerospace. ØMQ was originally conceived as an ultra-fast messaging system for stock trading and so the focus was on extreme optimization. Later on during the development, the focus shifted to providing a generic system for building distributed applications and supporting arbitrary messaging patterns, various transport mechanisms, arbitrary language bindings, etc.

ZeroMQ is a messaging system, and doesn't provide serialization and unserialization of data before and after transmission. For this purpose (across the application and even in the database) we use MessagePack, which is a binary-based efficient object serialization library that enables the exchange structured objects between many languages. All commands transmitted between the components are actually string arrays. The first (mandatory) element is the name of the command, and may be followed by parameters which are required for some of the commands.

## 2.4 Database

Database access by the Polling Server is completely abstracted away by the Datastore class, which in turn uses "rdbi" to connect to the server. A database table is used for the data collected each day, and this table is named in the format "data_for_%Y_%m_%d". Each of these tables have the same schema: the columns are the UNIX timestamp, network element (IP address), OID and value. Before saving the value is serialized using MessagePack so that it can be unserialized to the original value after reading it from the database.

By default an in-memory SQLite3 database is used because this is the most convenient for development and testing. In a production environment this should be changed to MySQL or PostgreSQL, or atleast an SQLite3 database which is saved to a file. This is fairly straightforward to do and only involves changing the database driver being used.

## 2.5 Design of the Components

The Poll Server primarily consists of two parts: the HTTP API and a worker. Internally the worker consists of 3 threads, two for regularly adding the tasks of reassigning NEs to LDCs and fetching values from the LDCs to a queue and one for actually executing the tasks. As it fetches the values from LDCs, the worker thread saves these to the Datastore. The HTTP API simply reads from the API. In order to add or remove LDCs or NEs/OIDs, a network administrator needs to edit the configuration file in the Poll Server.

The Poll Server may send the following commands to an LDC:

- ["PING"] – used for checking whether an LDC is up or not.
- ["HOPCOUNT", IP] – get the hopcount to the specified IP address.
- ["TRACK", NE, OID, INTERVAL] – start tracking the specified NE/OID using the given interval.
- ["CLEAR_TRACKS"] – stop tracking all previously assigned variables.
- ["TRACKED"] – return a list of tracked variables.
- ["FETCH"] – return all the new readings since the last time this command was sent.

Of these commands, all but the first two are directly forwarded to the Poller by the LDC API.

The Poller and LDC API use the reactor pattern without the Synchronous Event Demultiplexer. The Poller creates a new "Tracked" object for each of the variables it tracks. Each "Tracked" object is associated with a thread (This will create problems if there are a very large number of variables to be tracked. It would be wise to use a queue and worker threads-approach similar to the one used in the Poll Server.)

The Poller uses a load sensitive polling algorithm which performs the poll at the next scheduled time if the server ever gets overloaded, causing the time taken to perform a poll to exceed the polling interval.

# 3 Adaptive Polling

A single exponential smoothing based forecasting formula, known as EWMA (Exponentially Weighted Moving Average), has a basic recursive equation of the form,

$$S_{t+1} = \alpha y_t + (1 - \alpha) S_t$$

where $S_{t+1}$ is the next step estimate, $S_t$ is the running EWMA and $y_t$ is the latest reading.

The single exponential smoothing does not work well with datasets that exhibit trends. Hence, we consider EDS (Exponential Double Smoothing) which uses a second smoothing parameter $\beta$ for dealing with linear trends, which must be chosen in conjunction with $\alpha$.

$$S_{t+1} = \alpha y_{t+1} + (1 - \alpha)(S_t + b_t)$$

$$b_{t+1} = \beta(S_{t+1} - S_t) + (1 - \beta) b_t$$

The first equation adjusts the smoothed parameter from the trend $b_t$ and the estimate of the previous period $S_t$. The next equation then updates the trend as the difference between the last two estimates. This form of EDS is Holt's method.

Holt's method assumes regular data sampling intervals and so, in order to perform adaptive sampling, we need to adopt a modification of EDS using Wright's extension. However we found that in practice this is not necessary because Holt's method works well enough.

The basic algorithm used for adaptive sampling is as follows:

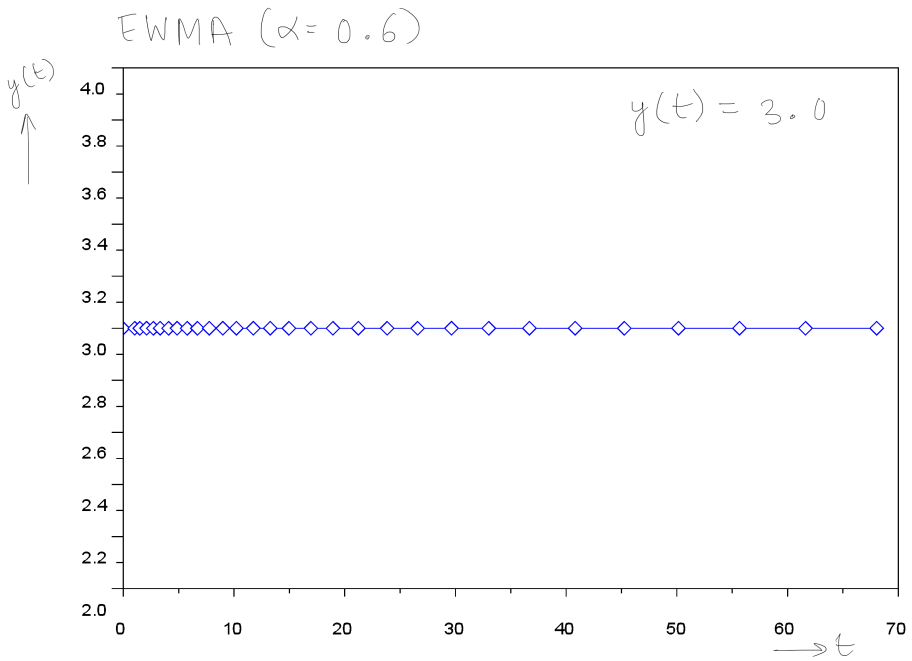
1. Read the next value.
2. Compare it with the forecasted value, if the difference is within a factor of $\delta$ increase the poll delay by a factor of 1.1, otherwise halve the poll delay.
3. Update the next forecast value using the value read in.
4. Wait for the amount of time specified by the new poll delay.

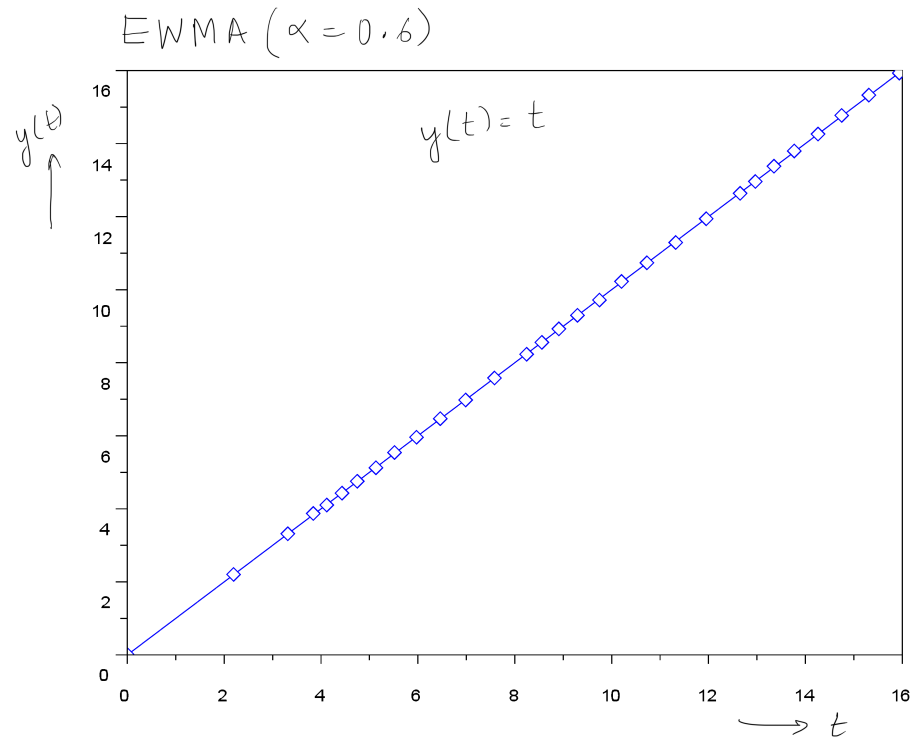
5. Go to step 1.

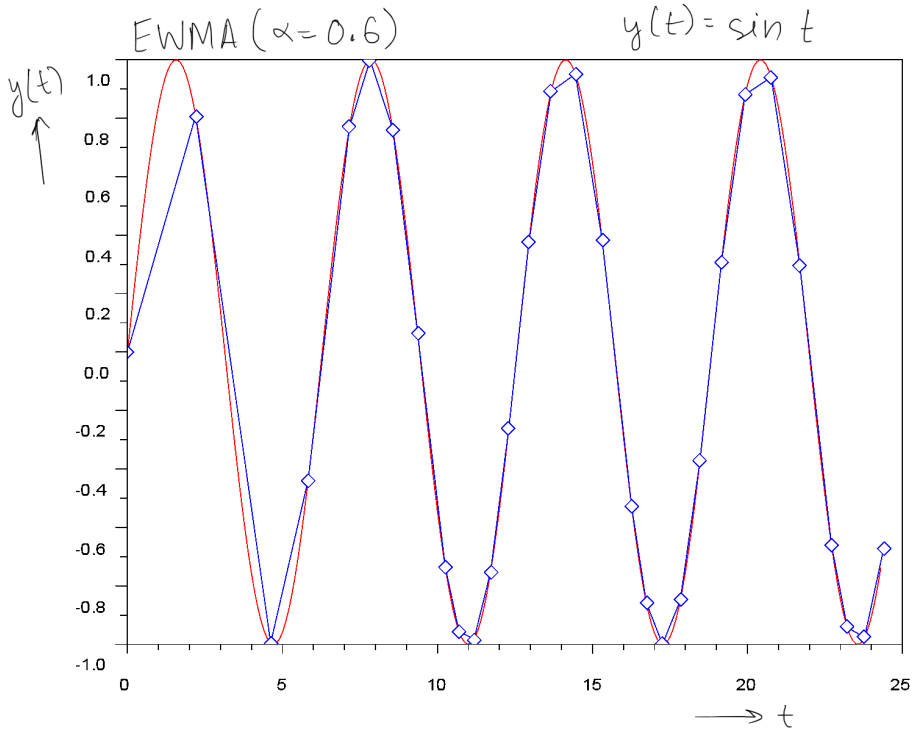
Below are some graphs that depict simulations of adaptive polling using both EWMA and Holt's method for different types of data and different parameters. The red line is the true value of the function, the blue diamonds are the samples and the blue lines are the result of linear interpolation between the samples.

## 3.1 EWMA

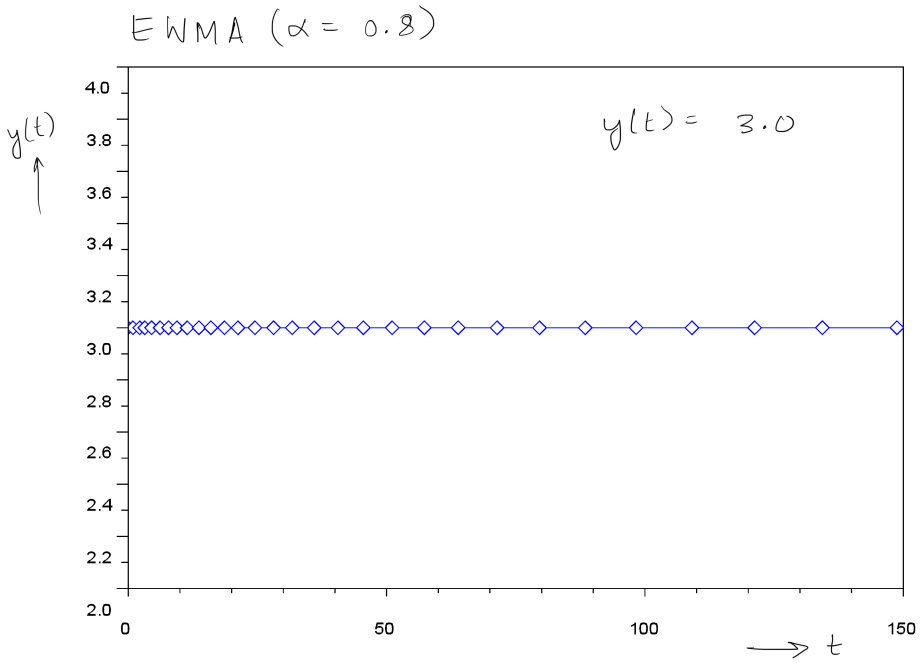
### 3.1.1 $\alpha = 0.6$, $\delta = 1$

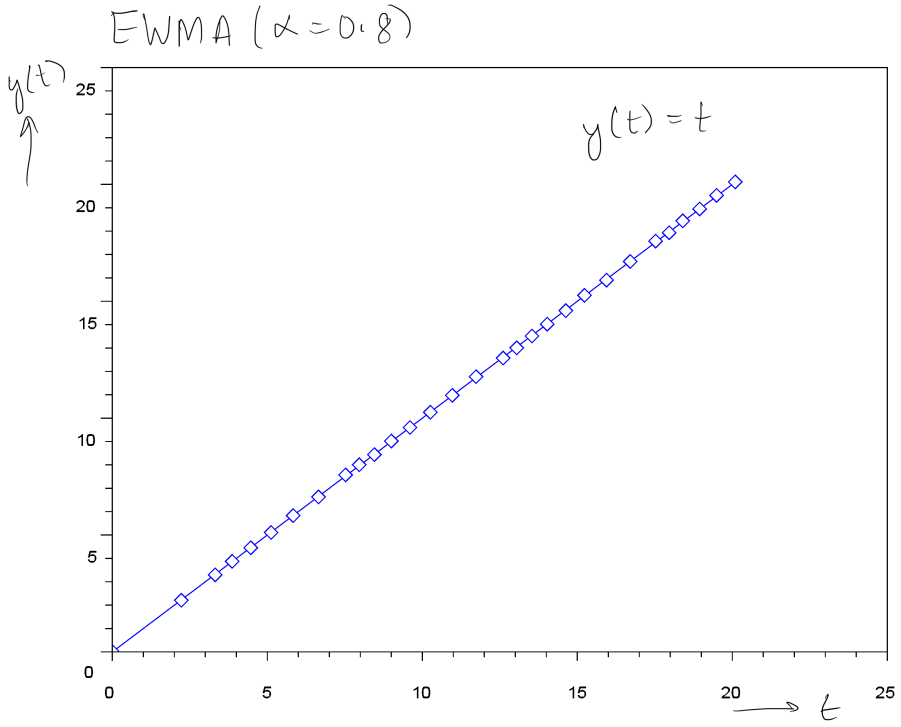


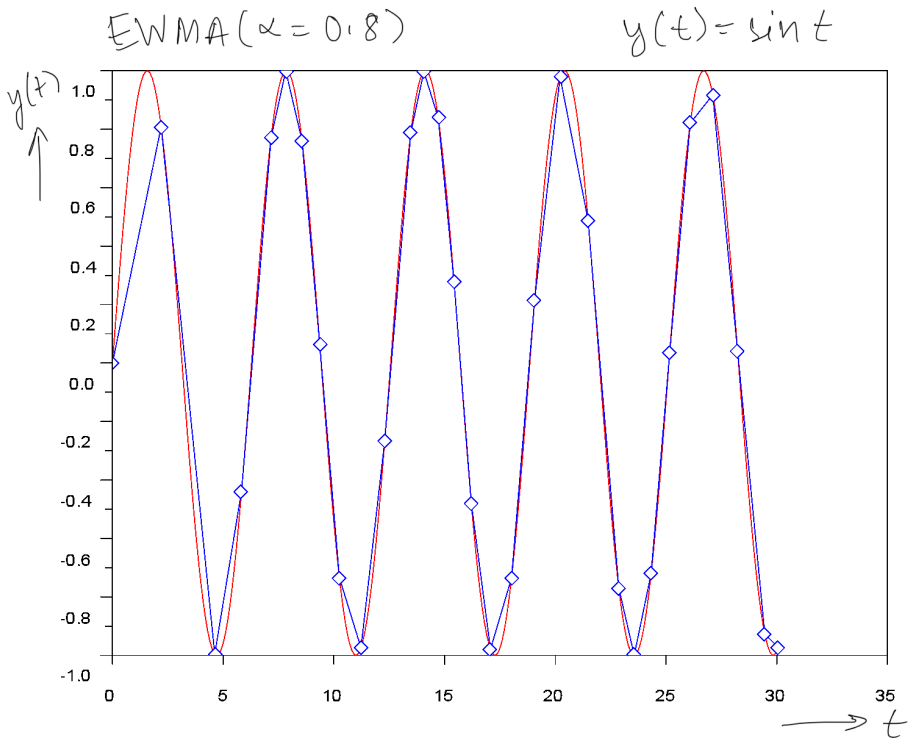

EWMA (α = 0.6)
y(t) = t
y(t)
t
0
2
4
6
8
10
12
14
16


EWMA (α = 0.6)
y(t) = sin t
y(t)
t
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
-0.8
-1.0
0
5
10
15
20
25

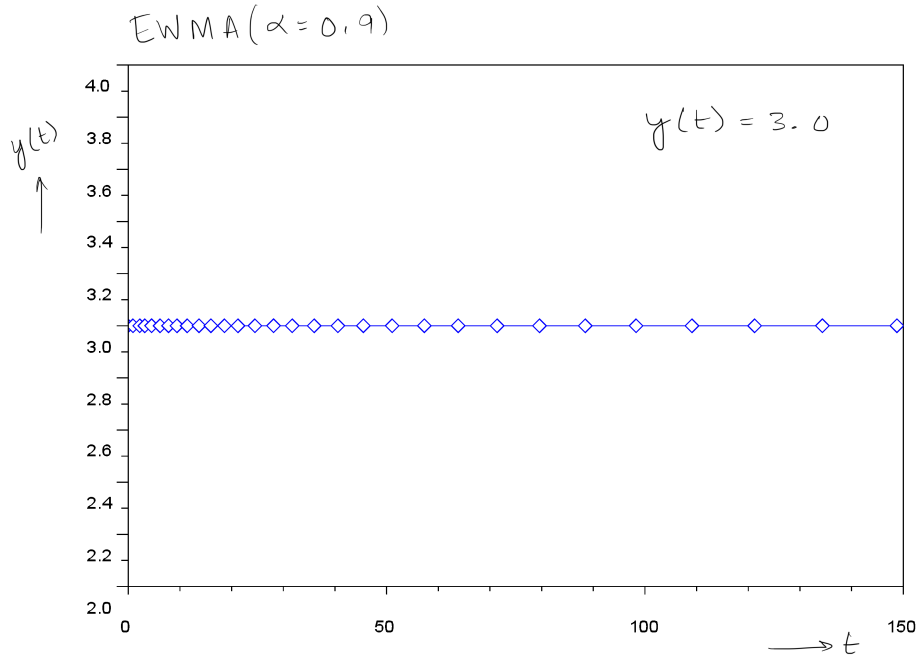
### 3.1.2 $\alpha = 0.8$, $\delta = 1$

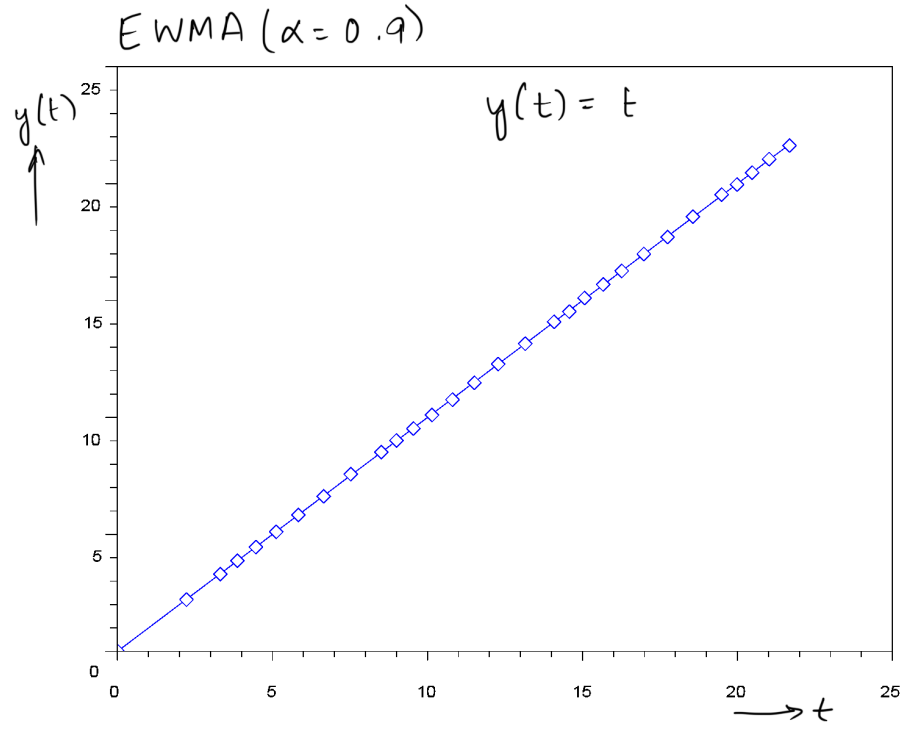


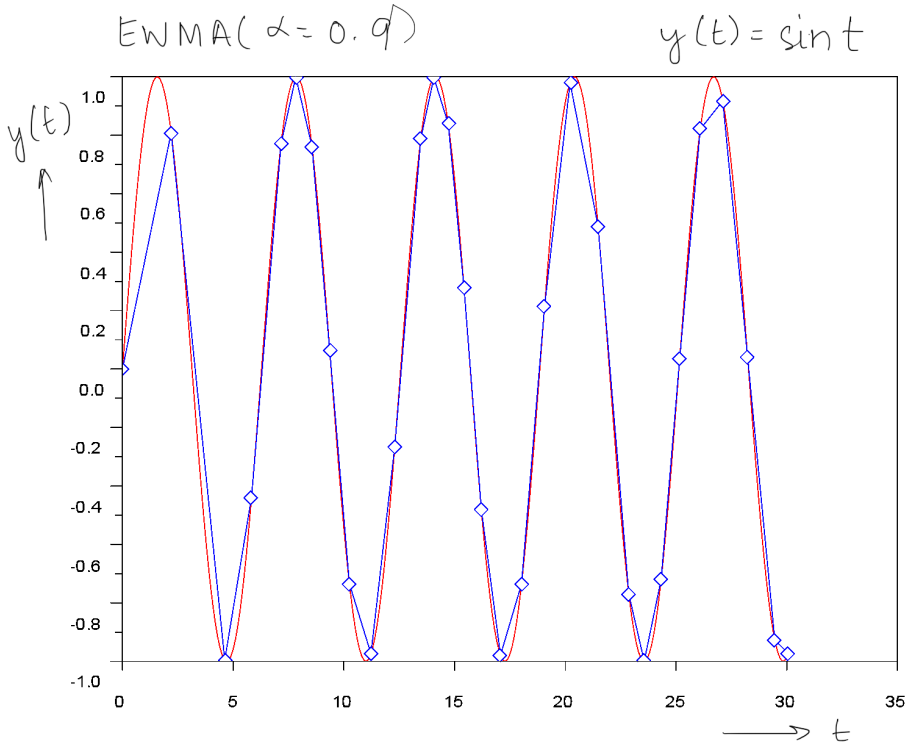






### 3.1.3 $\alpha = 0.9$, $\delta = 1$

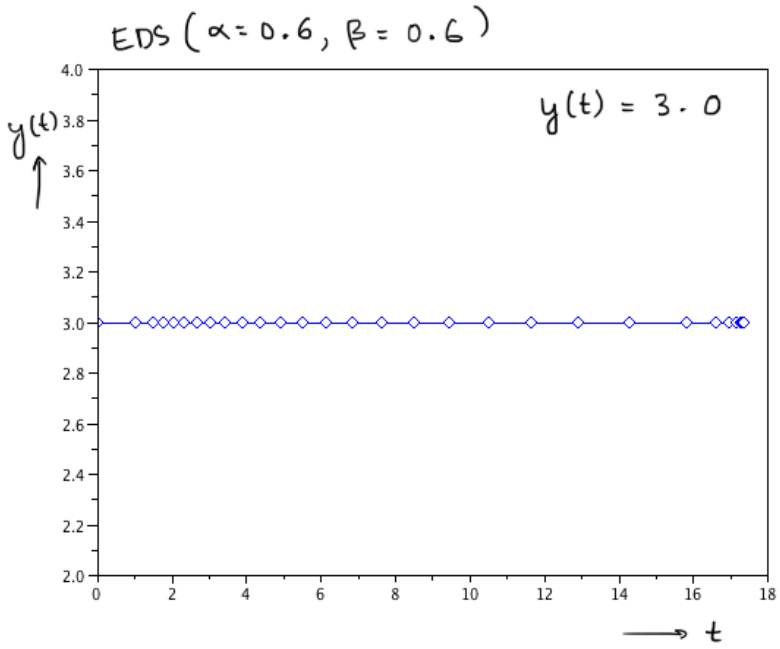


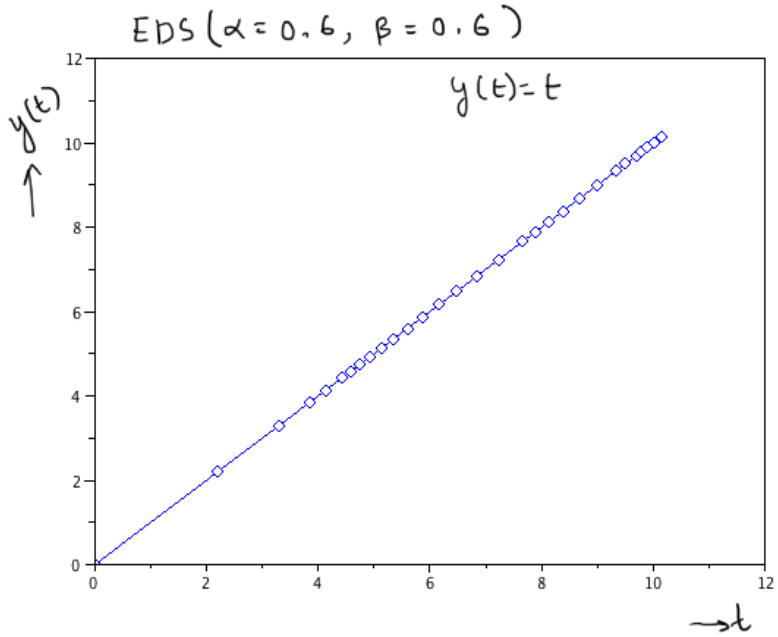

EWMA (α = 0.9)
y(t) = t
y(t)
t
0
5
10
15
20
25

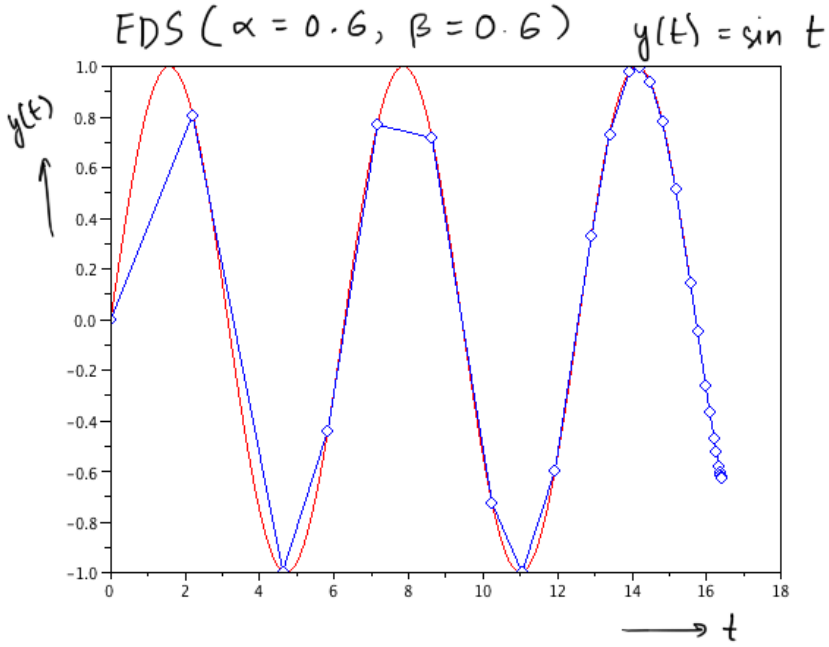

EWMA( α = 0.9)
y(t) = sin t
y(t)
t
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
-0.8
-1.0
0
5
10
15
20
25
30
35

## 3.2 Holt's Method

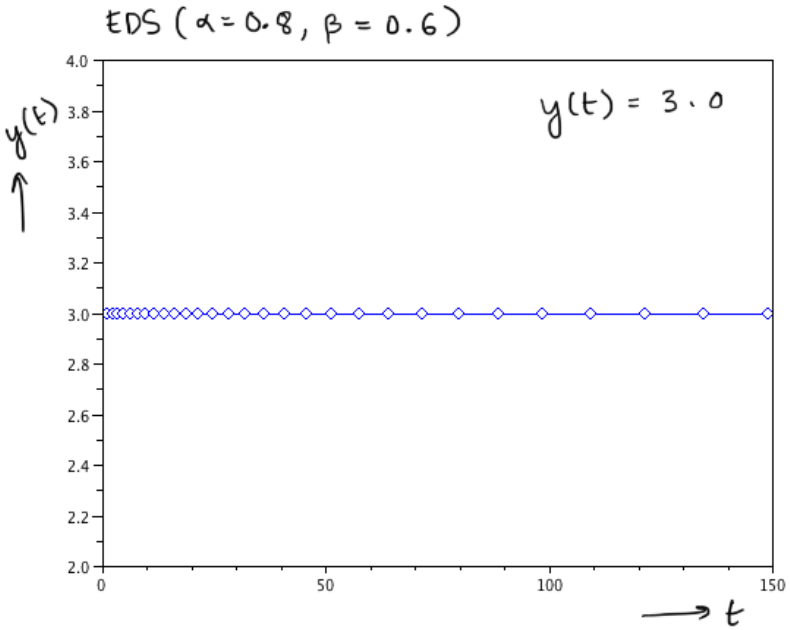
### 3.2.1 $\alpha = 0.6$, $\beta = 0.6$, $\delta = 1$

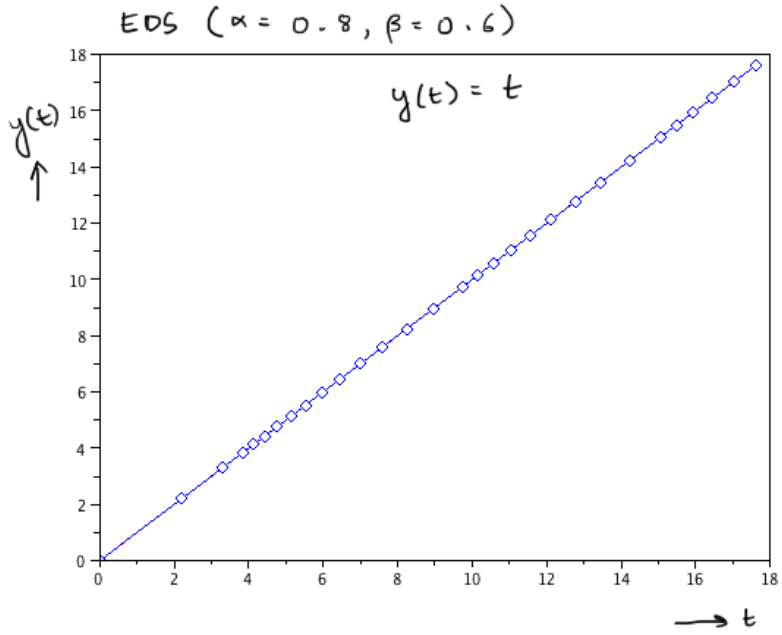


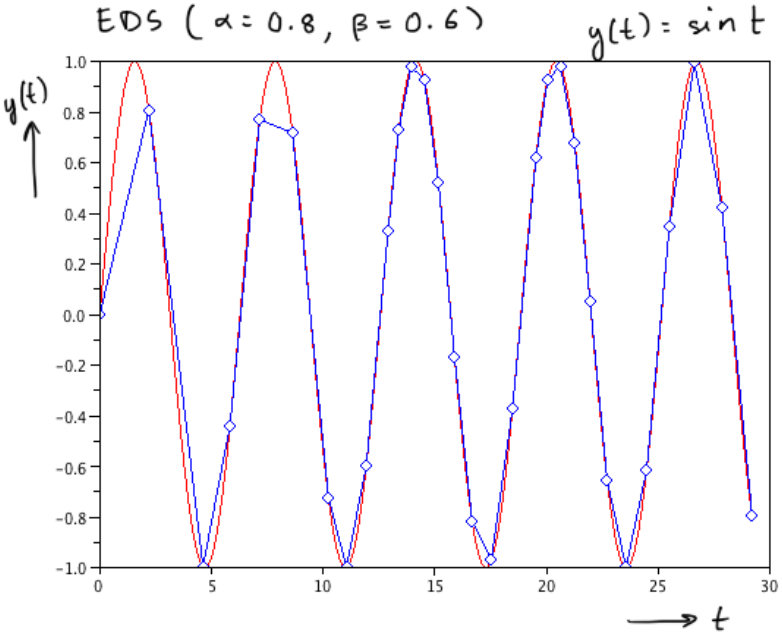





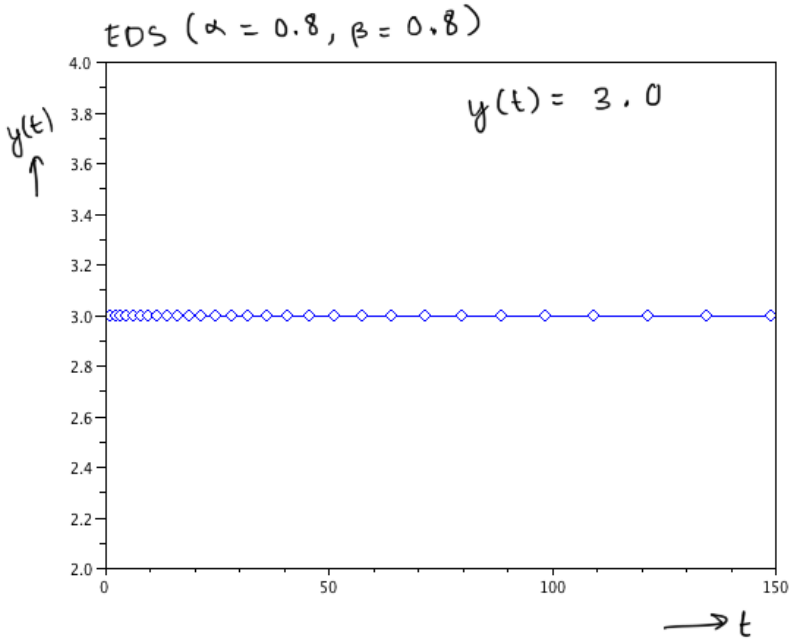

**3.2.2** $\alpha = 0.8$, $\beta = 0.6$, $\delta = 1$

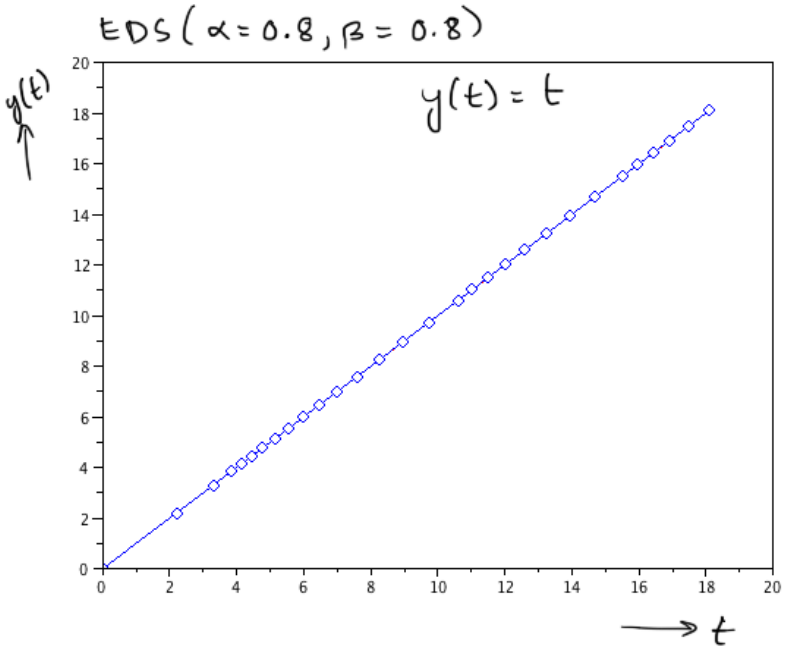


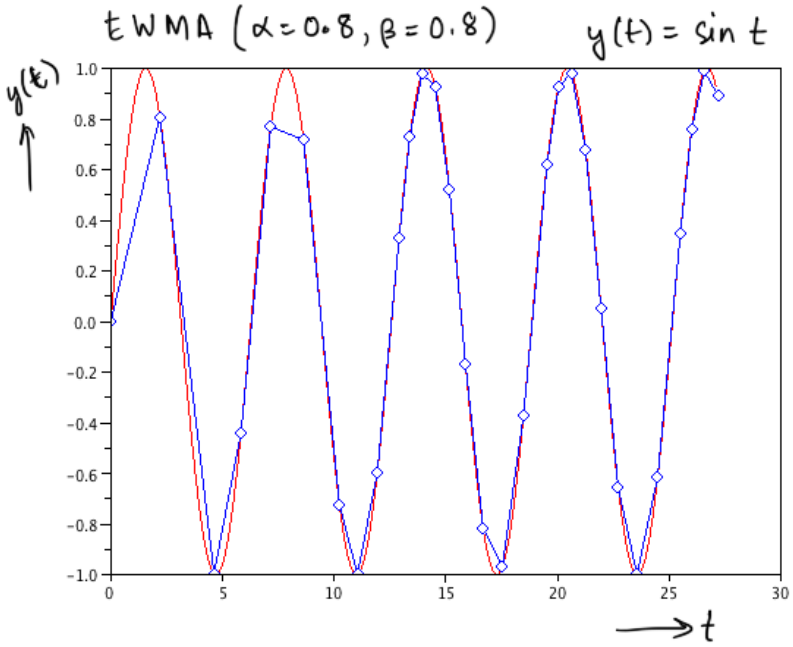

EDS (α = 0.8, β = 0.6)
y(t) = t
y(t)
t


EDS (α = 0.8, β = 0.6)
y(t) = sin t
y(t)
t

**3.2.3** $\alpha = 0.8,\ \beta = 0.8,\ \delta = 1$






## 4 Sample Application: Real Time Grapher

As an example of how to use the REST API provided by the Poll Server, we have included a simple sample application which graphs the values of the variables supplied in realtime.

The application is written using the "Sinatra" micro-framework, which is basically a DSL for quickly creating web applications in Ruby with minimal effort. The core logic is under 40 lines of code. The amount of code written for markup and styling is much larger than this.

Accessing the root webpage when the application is running will give a list of all of the network elements being tracked. Clicking on a network element will give a list of all of the OIDs tracked for that network element. Clicking on an OID will draw a real-time graph of that variable if it is a boolean or numeric quantity.

For drawing the realtime graph without reloading the page we use AJAX (Asynchronous Javascript and XML) to transmit JSON (Javascript Object Notation) encoded objects to the browser from the server.

## 5 Citations

1. Chapter 5 of Roy Fielding's doctoral dissertation, in which he introduces and defines the term "representational state transfer" or REST. (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

2. ZeroMQ Chapter by Martin Sústrik from "The Architecture of Open Source Ap-

plications Volume II: Structure, Scale and a Few More Fearless Hacks". (http://www.aosabook.org/en/zeromq.html)

3. The Official YAML Website. (http://yaml.org/)

4. The MessagePack Project Website. (http://msgpack.org/)

5. "Design and Evaluation of an Adaptive Sampling Strategy for a Wireless Air Pollution Sensor Network" by Manik Gupta, Lamling Venus Shum, Eliane Bodanese, Stephen Hailes.