13th Computer Control for Water Industry Conference, CCWI 2015

Modeling water supply system control system algorithms

Markus I. Sunela\textsuperscript{a,*}, Raido Puust\textsuperscript{b}

\textsuperscript{a}FCG Design and Engineering Ltd., Tampere 33200, Finland
\textsuperscript{b}Tallinn University of Technology, Tallinn 19086, Estonia

Abstract

This paper introduces a control system model framework built into EPANET simulator. The control system model extends the current possibilities offered by EPANET to model the exact, dynamic behavior of the water distribution system working under complex or high-level control system.

Besides the control system model, a novel method for modelling parallel pumping stations in EPANET was developed. The new methodology allows varying the control type (pressure, flow) and setting to dynamically change during the simulation. The new pump battery component was utilized in the control system modelling.

The new modelling tools were demonstrated in a case study.

Keywords: parallel pumping, control system, modeling, SCADA, PID, EPANET

1. Introduction

EPANET\cite{1} is the most widely used hydraulic simulator for pressurized networks and it provides tools to easily model on-off control of pumps or change variable-speed driven (VSD) pumps’ relative rotational speed, either based on time or on hydraulic state variables, like flow, tank level or pressure.

The both control rule features provided by EPANET, however, are very simple and limited, and it’s not possible to do any calculations, like summing volumes or flows, either in rule conditions or actions. For example, implementing a proper proportional-integral-derivative controller (PID) \cite{2} for variable-speed drive (VSD) controlled pumps, interpolation or any other higher level control based on status of multiple stations is practically impossible without creating a separate computer program to drive the simulation, or extending the EPANET simulator code. \cite{3}

Current implementations of parallel pumping or pumping with fixed head or flow have several limitations.\cite{4}

Using tools found in basic, freely available EPANET only, one must control the head or flow through series of pump(s) and either flow control valve (FCV) for fixed flow, pressure reducing valve (PRV) for fixed head, or first FCV and then PRV for fixed flow with pressure limitation. The problem with this approach is, that while it enables to model the dynamics and pump(s) capacity, it fails to give accurate idea of energy consumption or flows through single

* Corresponding author. Tel.: +358-40-768-5165 ; fax: +358-10-409-6730.
E-mail address: markus.sunela@fcg.fi

© 2015 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Scientific Committee of CCWI 2015

doi:10.1016/j.proeng.2015.08.927
pumps. Other possibility is to create a large number of EPANET control rules that change relative rotational speed to a pre-calculated value, that produces desired flow or head based on the simulated head or flow through the pumps for each pump battery.

If EPANET is extended via callback mechanism or otherwise, hooking some code to be run after ENstep function call, one can use software PID controller to control the individual pumps’ rotational speed through pump’s setting value. It is however quite difficult to properly tune the controller because of typically several minutes long time steps are used in hydraulic simulation, compared to sub-second time scale used in controllers. It’s also not possible to change the target setting using the standard EPANET control rules. This approach enables accurately model functioning of every controller, pump and energy usage, but requires the tools to extend EPANET’s behaviour per model basis or driving EPANET simulation from a third-party program, such as Matlab, and thus limiting the usefulness of the method.

The third option is to use proper constant flow [5] or head [6] solver for EPANET’s gradient algorithm[7]. While the solutions are easy to use after their implementation, they still lack the possibility to change the control type from flow to head and vice versa during simulation, and they only provide some hydraulically feasible solution in the parallel pumping case. The algorithms only allow the use of identical pumps and don’t choose the optimal combination of pumps and their relative rotational speeds nor does the solution algorithms implement any specific typical naive control algorithms for driving the pump battery. One additional limitation of these solutions are, that there are no freely available implementations.

This paper solves the limitations of aforementioned methods using a two-step method, where the first, hydraulic part implemented into EPANET hydraulic solver is very simple and only calculates the flow and head for the given settings and control type. The control type (constant flow, constant head, constant head difference), flow or head setting and possible limit can all be changed dynamically during the simulation. This makes it possible to easily model parallel pumping and control systems.

The second part of the battery component is implemented partly via callback mechanism to check whether the resulting flow and head combination is valid for the pump battery, and partly via post-processing which calculates the energy usage, each pump’s rotational speed and flow. The second part, both validity checks and post-processing analysis, relies fully on the lookup table generated by using methods presented in [8] and [9]. The second part is only necessary, if analysis of pump performance and energy is needed - the first part alone can solve the system state and hydraulic behaviour.

A control system modelling framework was developed. The EPANET simulator was extended to allow specifying callbacks written in Python[10], a general purpose, object oriented programming language. The callbacks are called before simulation, after every time step and after the simulation. The control system model Python source code module, if module exists, is automatically loaded and interpreted by the modified EPANET, and thus the proposed method for modelling control system does not require use of any extra tools, and works with any EPANET based modelling tool.

Both new developments, the pump battery component and control system modelling tools, are tested in a case study.

2. Materials and methods

2.1. Pump battery

This paper introduces a new pump battery component into the EPANET hydraulic solver. The component enables one to model a pump battery consisting of one or more, possible non-identical, pumps working in parallel. The battery can be either flow, pressure or head difference controlled, and the mode of operation and setting can be dynamically controlled using application programming interface (API) and EPANET control rules.

The component accepts a limit to the non-controlled parameter, for example, if the pump battery is flow controlled, maximum allowed downstream pressure can be limited to user supplied value, typically 80 or 100 meters of pressure head. Alternatively in constant pressure mode of operation the maximum allowed flow can be limited. In practice, especially when operating in constant flow controlled manner, the maximum allowed pressure is limited in order to avoid pipe breakage when flow falls below the setting.
To allow efficient and advanced pump battery analysis and optimization, the pump battery component in EPANET is mathematically very simple. The component only calculates the head and flow required to meet the given setting and limit in the active controlling mode.

Component uses an externally defined callback function to check that the pump is working in allowed regime, and the program running EPANET simulation or utilizing the hydraulic results calculates the internal pump configuration, each pump’s frequency and energy consumption, based on the simulated head and flow. Thus the more complex and time consuming tasks are delegated to external code.

The pump battery component is described in EPANET *.inp file by identifier, and start and end nodes. Optionally the initial mode of operation (constant pressure/flow), initial setting (flow or pressure), and pressure/flow limit can be specified. An example is shown in Listing 1.

Listing 1: Example of defining pump batteries in EPANET inp file. First battery has initial mode of operation set to constant flow at 10 l s \(^{-1}\) with pressure limit of 80 m and the second has no initial values and is initially closed.

\[
\begin{align*}
\text{[BATTERIES]} \\
\text{Battery1 Reservoir1 Junction1 TYPE FLOW SETTING 10 LIMIT 80} \\
\text{Battery2 Reservoir1 Junction2}
\end{align*}
\]

The changes required in EPANET source are minimal and localized. Besides introducing a new component type, the new link values and the code to read battery specifications from the *.inp file, a few new functions are added into the hydraul.c module: batterycoeffs(), which is called by newcoeffs(), and batterystatus(int index, char status, double h1, double h2), which is called by linkstatus(). The matrix coefficients in the global gradient algorithm\cite{7} are calculated by batterycoeffs() and batterystatus(...) only changes the battery status based on the hydraulic results, and calls the possible external callback function to check that the battery is working within allowed regime.

When the pump battery is in flow control mode or the flow limit is exceeded in constant pressure or pressure difference mode, the pump battery works analogically to flow control valve in EPANET, but the head loss over the link is allowed to be negative. The system matrix coefficients [1,6,7] are

\[
\begin{align*}
p_{ij} & = \frac{1}{10^8} \\
P_i & = F_i - Q_{set} \\
F_j & = F_j + Q_{set} \\
A_{ij} & = A_{ij} - p_{ij} \\
A_{jj} & = A_{jj} + p_{ij} \\
A_{ii} & = A_{ii} + p_{ij} \\
y_{ij} & = Q_{ij} - Q_{set} \\
\end{align*}
\]

where \(i\) is the index of start node, \(j\) is the index of end node, and \(Q_{set}\) is the flow setting.

When the pump battery is in pressure or pressure difference control mode or the pressure limit is exceeded in constant flow mode, the pump battery works analogically to pressure reducing valve in EPANET, but the head loss over the link is allowed to be negative. The matrix coefficients are

\[
\begin{align*}
p_{ij} & = 0 \\
F_j & = F_j + 10^8 \cdot H_{set} \\
A_{jj} & = A_{jj} + 10^8 \\
\end{align*}
\]

where \(H_{set}\) is the head setting.

The EPANET API was extended to allow changing the mode and limit value, and to allow setting the callback function, which can check, that the pump is working in allowed regime and limit the generated head and/or flow if necessary.
The added link values are named EN_MODE, accepting settings CONST_FLOW, CONST_PRESSURE and CONST_DIFF, and EN_LIMIT, accepting flow limit in model units when the battery is operated in constant pressure or constant pressure difference mode, and pressure limit in model units when operated in constant flow mode. The values can be queried and set using the standard EN_getlinkvalue and EN_setlinkvalue functions.

The output from EPANET for a given pump battery is just the time dependent working points \((Q_t, H_t)\), setting and mode of operation. Thus the simulation package must implement some means to show and analyse each pump’s properties, like frequency and power at different working points.

We took lookup table based approach described in [8] and [9]. The chosen method allows to model pump battery with non-identical pumps with different allowed frequency ranges and different parallel pump control strategies: equal frequencies for all running pumps (naive 1), only last pump’s frequency is controlled (naive 2) or globally optimal control strategy. In addition the methodology handles the frequency scaling problem[11] and can, optionally, model the pump’s motor and variable-speed drive efficiencies, and thus give very accurate approximation of the real energy usage.

An example of a simple model and its results is shown in Figure 1. The pump battery’s constant pressure setting changes to higher setting for 8 am to 9 pm time period using EPANET control rules. The left hand side of the figure shows the model and water demand at the far end node, and the right hand side shows the simulated head at pump battery discharge node and pump’s relative speed.

![Figure 1](image_url)

Figure 1: A small sample model of pump battery working with different pressure setting and varying flow. The changes in pump outlet head and relative pump speed are shown in the figure.

2.2. Control system modelling framework

In order to allow for modelling complex control system algorithms, a Python 2.7.x programming language [10] based framework was built. The EPANET simulator was extended in a binary API compatible way so that the modified library can function as a drop-in replacement for the standard simulator or as a proxy for customized, closed-source EPANET based simulators found in many commercial hydraulic modelling software.

In EN_open function, the Python framework is initialized, and a Python module is searched, identified by the same filename as the EPANET model but with *.py extension. If the module is found, it is loaded using the Python interpreter and function pointers to epanet_init, epanet_callback and epanet_close functions are retrieved. During the Python module load, the module can import and use other Python modules libraries, such as xlrd[12] for reading control system parameters from MS Excel spreadsheet files.

After the hydraulic simulation is initialized in EN_openH function, the loaded Python module’s epanet_init function is called. The function can then instate the EPANET link and node objects that are required for its functioning. Typically this phase finds the indices of the controlled pump batteries and valves, and components representing the measurements needed in the operation in EPANET simulator. The init function also sets initial settings for all controlled components.

After each simulation time step, in EN_step function, the Python module’s epanet_callback function is called. The function can query the system state and alter settings for different components. This callback is where the control
system model done in Python language is given full control over the simulated system and all the control algorithm calculations take place.

The framework provides mapping of the standard C language programming interface available in EPANET in a higher level object oriented Python API. The EPANET errors are mapped into Python exceptions and the node and link properties are accessed through an object oriented wrapper part of which is shown in Figure 2. Properties are either read-only or read-write, depending whether the parameter can be changed or not. A lower level 1:1 Python mapping to the C API is also available, but it’s usage is not recommended.

Using the API, network state can be queried, controlled and altered during the simulation. It is possible, for example, to query flow, head and pressure, and tank level and volume. The API allows to open and close pipes, change valve and pump settings, and control pump batteries. Demands and emitter coefficients can be changed too.

But in order to remain strictly control system model, only those components that can be controlled in real world, should be controlled.

Finally, when the hydraulic simulation is completed, the ENclose function calls the epanet.close function, which can, for example, store internal control system state results to a file for later analysis. After the call the Python interpreter is closed.

The simulation and calling the control system model is wholly controlled by EPANET simulator, and thus the use of control system model is transparent to any program using the simulator. While the control system model can query and set model parameters during the simulation, it cannot control the simulation in any other manner.

The Python code can be divided into multiple modules which can call each other and EPANET. All tools and programming techniques available in Python can be used freely. Typically it is reasonable, for example, to create classes to present various system components or to read control system parameters. Porting code from any programmable logic controller (PLC) or supervisory control and data acquisition (SCADA) system is straightforward, and designing common libraries for often used components is easy.

A very simple example of a control system model is shown in Listing 2 and Figure 3. Pumping into the network is flow controlled, and the flow is linearly interpolated between minimum and maximum flow values based on the water tower level, so that, when water tower is at the upper level, flow is minimal and vice versa. While the example simple, similar control systems are common in Finland, and the example demonstrates some of the potential of using a general purpose programming language as a control system modelling tool.

Listing 2: An example of control system model, that interpolates pump battery flow setting based on a water tower level

```python
import epanet

MIN_LEVEL = 2.0  # meters
MAX_LEVEL = 4.5
MIN_FLOW = 3.0  # l/s
MAX_FLOW = 13.0
```

Figure 2: Part of the Python language object oriented wrappers around the low level EPANET API
3. Case study

The new pump battery component and control system model were applied to EPANET network model of Tuusula region water utility (Finland) serving cities of Kerava and Järvenpää, and municipalities of Tuusula and Sipoo. In addition significant amount of water is delivered to two industrial establishments (Koff and Ingman), two hospitals (Sairaalat) and two neighbouring municipalities: Mäntsälä and Pornainen.

The network has altogether 39 controllable stations, of which 11 are water sources and the others are either measurement, valve or booster pumping stations. All produced water is ground water, and two of the biggest water sources in Laakso area use also artificial ground water produced from Päijänne tunnel water. There are seven areas with their own water towers. The general water supply system layout and the stations is shown in Figure 4.
The network model is a full scale model that includes all pipes in the system, except the consumer connections, and all water consumers. For each measurement area, its own water demand pattern is used. All pumping stations are modelled as pump batteries with correct pump characteristic and efficiency curves, and all valve stations are modelled as flow control valves (FCV). The model has 3480 nodes and 4104 links, and the total modelled network length is 840 km.

Controlling the water supply system is completely automatized. The control system is implemented on the SCADA level. The high-level control system inspects the system state every half an hour, processes the control algorithm and sends flow settings to each station in the system. Each station has its own operating logic that controls the single pumps, their frequencies and any water treatment process.

The system is divided into eight calculation areas based on the pressure zone and municipality borders. The areas and their processing sequence from 1 to 8 is shown in Figure 4. All incoming and outgoing connections to the areas are measured, thus a full water balance can be calculated for each area.

For each tower a target water level curve is given at half an hour intervals. The control system tries to keep each water tower’s level on the defined level.

The high-level control system algorithm works as follows. Every half an hour each area is calculated in the order indicated by the numbers in Figure 4. The smaller areas without numbering are calculated first, and their demand estimate is set as demand for their respective measurement stations.

For each area, water usage during the previous time step is calculated based on the water balance. A water demand estimate for next time step, \( t + 1 \), is calculate based on the measured water consumption \( D_t \) and statistically calculated weekday specific 30 min consumption pattern \( P_t \).

\[
D_{\text{estimate}} = \frac{D_t}{P_t} \cdot P_{t+1} .
\]  

(11)

For areas with water tower, the volume difference between the target level and current level is calculated and converted into flow

\[
Q_{\text{water tower}} = \frac{V_{\text{target}} - V_{\text{current}}}{dt} .
\]  

(12)
Next the water demanded by previously calculated neighbouring areas is calculated by summing the flow settings at the \( n \) border stations:

\[
Q_{\text{neighbours}} = \sum_{i} Q_i.
\]  

(13)

Based on the water demand estimate, tower imbalance and neighbour demand, the flow needed by the area can be calculated

\[
Q_{\text{pumping}} = D_{\text{estimate}} + Q_{\text{water tower}} + Q_{\text{neighbours}}.
\]  

(14)

When the required flow \( Q_{\text{pumping}} \) is known, an area specific function decides, how the flow requirement will be satisfied. Typically the flow is divided among the local water sources if they exist, and any remaining flow is demanded from neighbouring areas by setting a flow demand value at the border stations. Each station has specified minimum and maximum flow constraints, which cannot be violated, and a station flow is limited to those constraints.

The target water tower level curves, demand patterns, station specific minimum and maximum flow settings, and possible manual settings for stations are stored as control system parameters. For the control system model’s purposes, the parameters are stored in a text file.

From the description above, it becomes apparent, that it is impossible the model this kind of a control system using plain EPANET. Thus all the control system logic, was reprogrammed from ABB MicroSCADA Supervisory Control Implementation Language[13] (SCIL) source files in Python using the framework developed earlier in this paper.

The control system model was programmed in an object-oriented fashion and divided into four modules: main.py as an entry point and hooking all the code together, settings.py for reading and handling the control system parameters, stations.py defining object classes for different types of stations and declaring the available stations and linking those to EPANET simulator, and finally areas.py defining the object class for a calculation area, declaring the areas and their calculation order and linking those areas to stations defined in stations.py.

Most of the code consists of area specific logic code. An example of an area specific logic is shown in Listing 3.

Listing 3: An example of one area specific program logic in the control system model. All the stations and areas are object classes.

```python
def calculate_jarvenpaa(jarvenpaa):
    demand = jarvenpaa.getDemand()

    if vahanummi15.controllable():
        vahanummi15.on(vahanummi15.yieldValue / 24)
        demand -= vahanummi15.getFlow()

    if hiihtomaja28.controllable():
        if demand >= hiihtomaja28.minFlow:
            hiihtomaja28.on(demand)
            demand -= hiihtomaja28.getFlow()
        else:
            hiihtomaja28.off()

    if tuomala8.controllable():
        if demand >= tuomala8.minFlow:
            tuomala8.on(demand)
        else:
            tuomala8.off()

    if ristinummi7.controllable():
        ristinummi7.off()

    return demand
```

The combination of the control system model and the hydraulic was found to agree with the real system at the station level. An example of the simulated values, on average day, in one of the biggest areas, Kerava, is shown in Figure 5. The left-hand side shows simulated water tower level and the target water tower level, and the right-hand side shows simulated values for different control system parameters: \( D_{\text{estimate}} \), \( Q_{\text{neighbours}} \), \( Q_{\text{water tower}} \) and \( Q_{\text{pumping}} \).
The model developed in case was used, among other things, for developing and optimizing the control system and its parameters further. In [3] several modifications to the parameters or control logic were examined, and a saving potential of 6.5% was reported. Figure 6 shows, how some of the examined changes affect the simulated flow at the biggest and final water source, Jäniksenlinna, on average day.

4. Discussion and conclusions

This paper introduced a novel way to model variable-speed driven pumps and pump batteries that are controlled by possibly time-varying constant flow, constant pressure or constant pressure difference strategies. The method uses EPANET to simulate the required head and flow for the given setting and control strategy, and then a precomputed lookup table is used for looking up the pumps’ energy consumption and frequencies.

Using the introduced pump battery component, it is possible to develop a standardized and transparent control system modelling framework on top of EPANET. The framework was developed in Python and integrated into EPANET simulator. Using Python, a general purpose programming language, it is possible to model whatever complex and high-level control systems.

Python modules being ordinary text files and the language being interpreted, no tools besides text editor is needed in order to create and edit control system models. Python language is, arguably, one of the easiest programming languages to learn. Thus, while some knowledge in programming is required, the barrier of entry is lowered.

The developed methods were applied in a case study, in which the correctness of complex control system model was demonstrated. The some modifications to the control system parameters and control logic and their effect on the energy consumption of the system were shown.
Modelling higher level control system on top of the hydraulic model opens up whole new possibilities to analyse and optimize the control system behaviour and effects of different control system parameters to, for example, energy use and quality of service. The methodology has been in active use at FCG Design and Engineering Ltd. since 2010 in dozens of projects.

Acknowledgements

This work was supported by institutional research funding IUT (IUT19-17) of the Estonian Ministry of Education and Research.

References