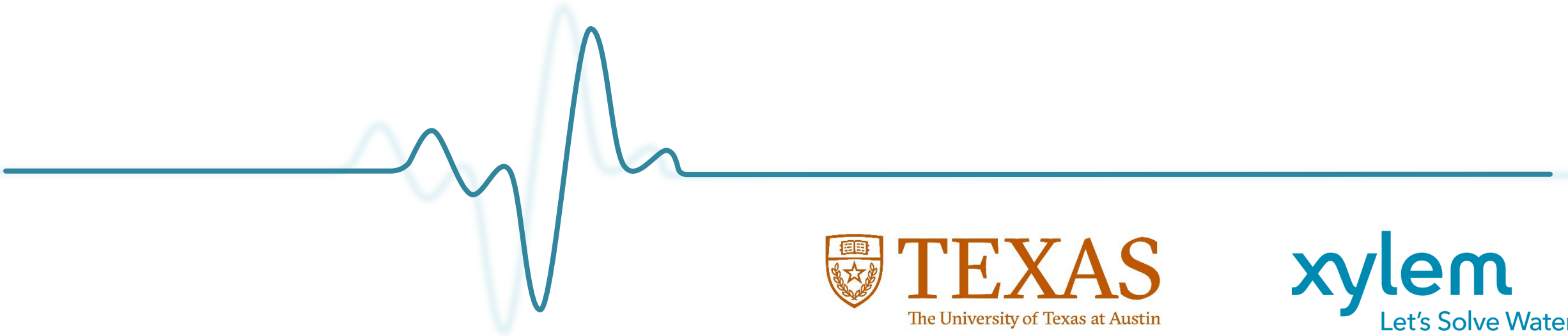
Transient Simulation in water Networks (TSNet)

Lina Sela and Lu Xing



TEXAS
The University of Texas at Austin

xylem
Let's Solve Water

About us

Lina Sela
Assistant Professor
The University of Texas at Austin

linasela@utexas.edu

Lu Xing
Data Scientists
Xylem

Lu.Xing@Xylem.com

Learning objectives

Agenda:

- ✓ Brief intro to transients and applications in WDSs
- ✓ Intro to TSNet governing equations
- ✓ Simulating with TSNet (hands-on exercises)
- ✓ Discussion and looking ahead

Quick survey

What describes **you** best from the following?

a) Student/Postdoc
b) Professor/Researcher
c) Practitioner
d) Other

Code 3950 7505

Are you familiar with pressure **transients**?

a) Have a rough understanding
b) Have a good understanding
c) Have hands on experience

Go to www.menti.com

Code 5174 3265

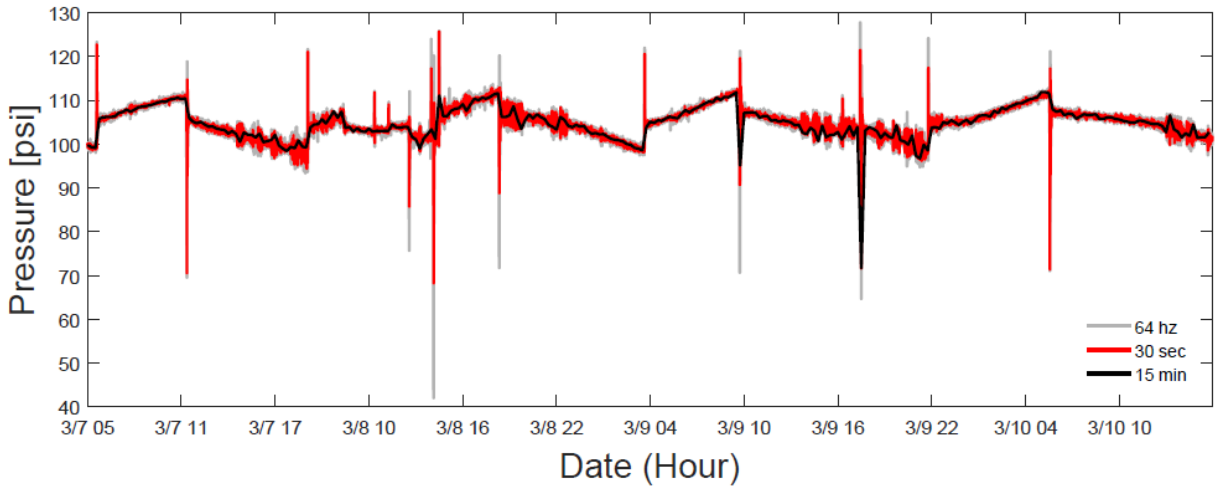
Are you familiar with coding and **Python**?

a) Don't have any coding experience
b) Know how to code but not with Python
c) Beginners level
d) Intermediate level
e) Advanced

Code 2190 7287

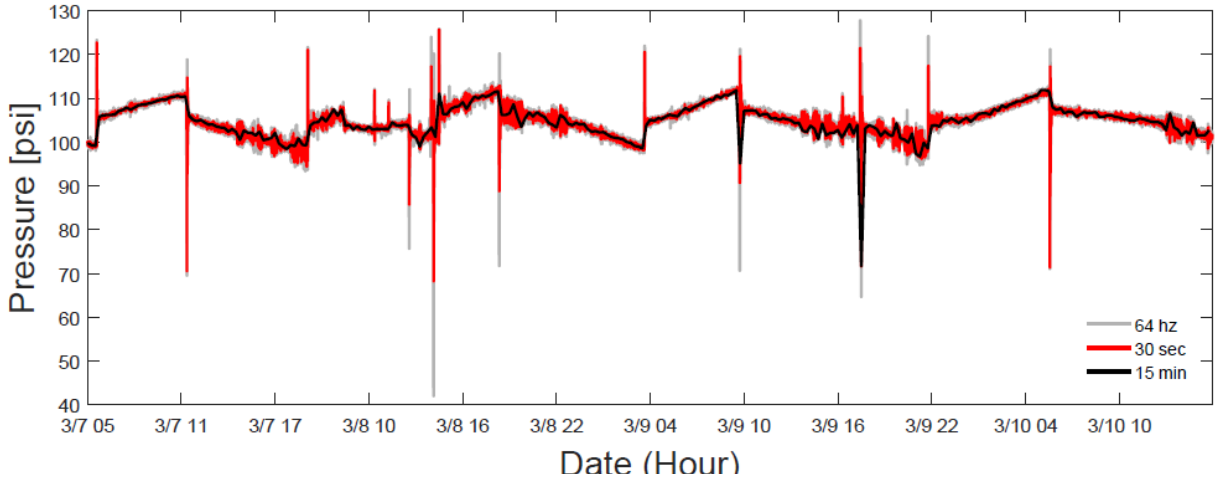
Pressure transients

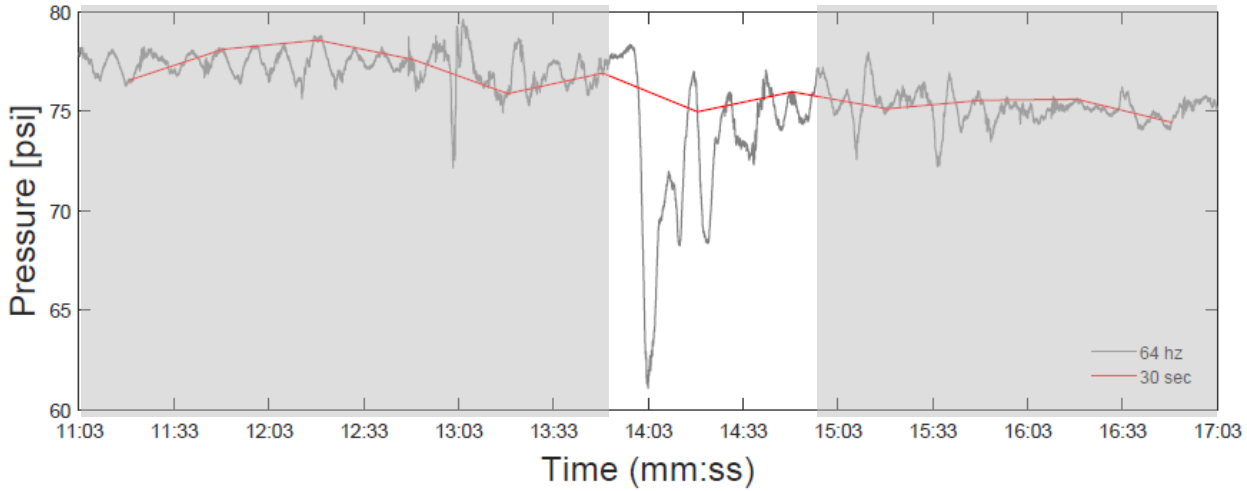
- Disturbance propagates as pressure waves
- High velocity ~ 900 – 1500 $\frac{m}{s}$
- Signal dissipates in the network



Pressure transients

- Disturbance propagates as pressure waves
- High velocity ~ 900 – 1500 $\frac{m}{s}$
- Signal dissipates in the network





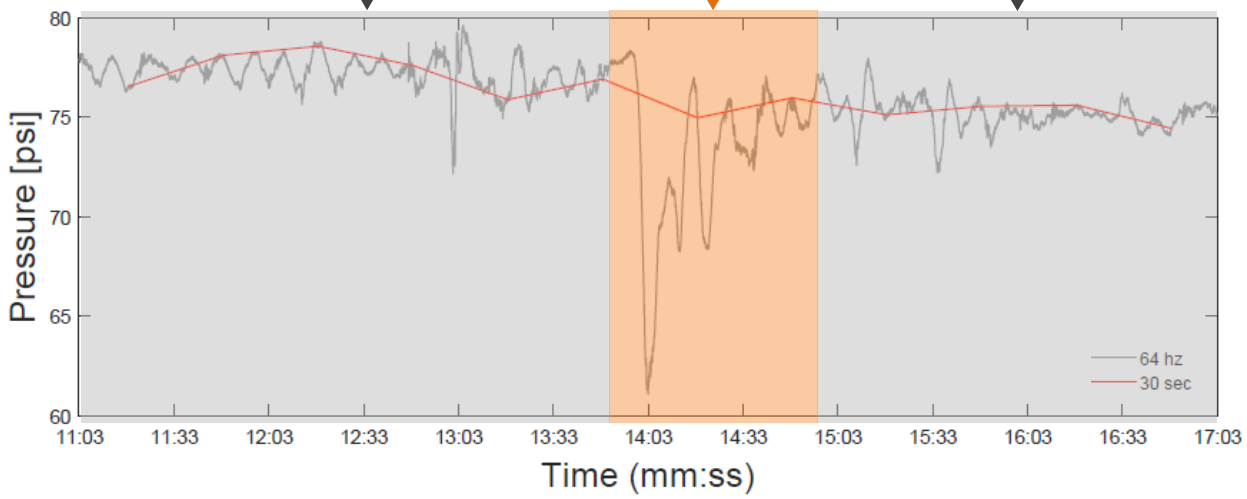
Pressure transients

TSNET EPANET WNTR Water Network Tool for Resilience

Simulation type

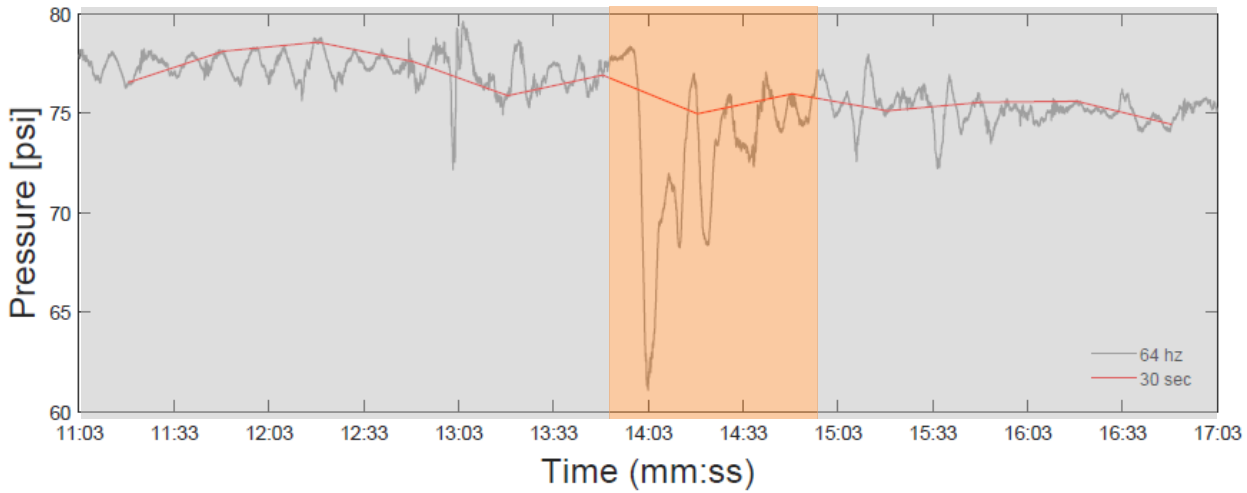
Transient

Steady state



Pressure transients

	TSNET	EPANET	WNTR
Simulation type	Transient	Steady state	
Time step	0.001~0.1s	15~60min	
Duration	10s – 5min	Several days	



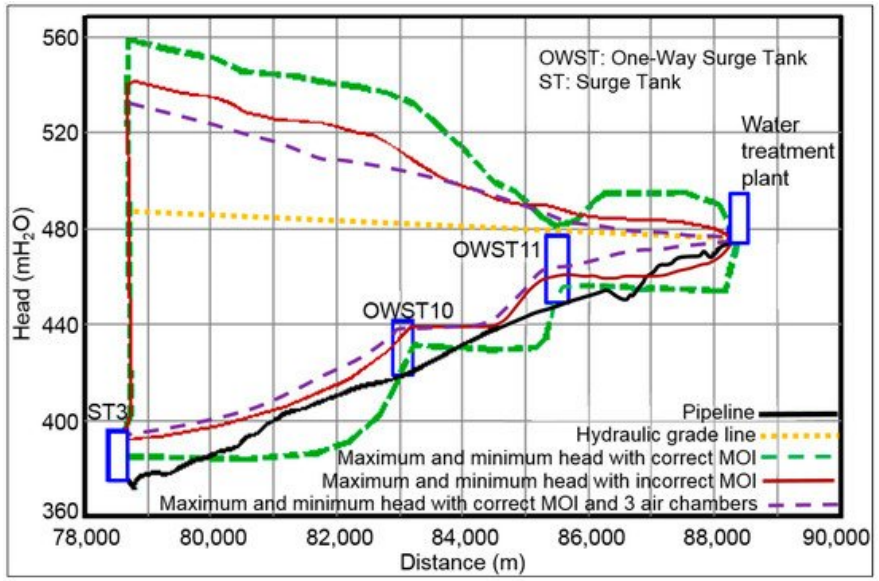
Applications of pressure transients in WDSs

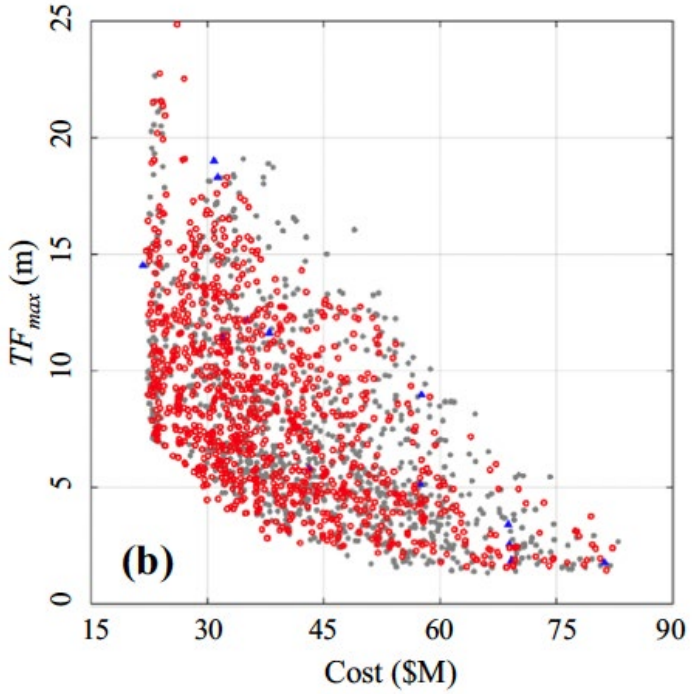
Examples:

- Pressure monitoring and transient detection and identification
- Burst detection and localization
- Sensor placement
- Stress estimation
- Pipe failure prediction
- Design and surge protection

Transients in WDSs

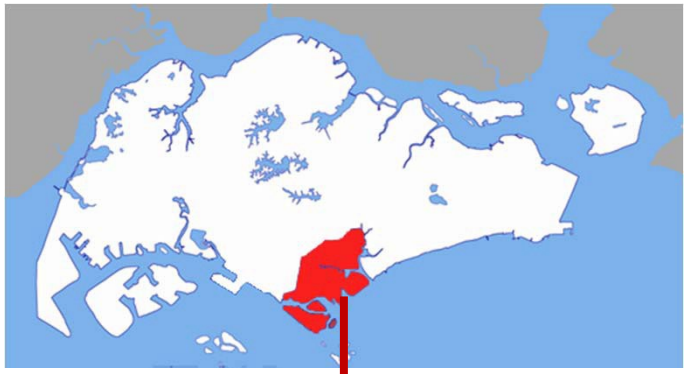
- Essential procedure for designing booster pump stations and choosing devices, i.e., by-pass check valves, and surge tanks, to mitigate transients[1]
- Multi-objective optimal design of WDSs accounting for transient impacts[2]





[1] Carmona-Paredes, R. B., Pozos-Estrada, O., Carmona-Paredes, L. G., Sánchez-Huerta, A., Rodal-Canales, E. A., & Carmona-Paredes, G. J. (2019). Protecting a pumping pipeline system from low pressure transients by using air pockets: a case study. *Water*, *11*(9), 1786.
[2] Huang, Y., Zheng, F., Duan, H. F., & Zhang, Q. (2020). Multi-objective optimal design of water distribution networks accounting for transient impacts. *Water Resources Management*, *34*(4), 1517-1534.

Transients in WDSs

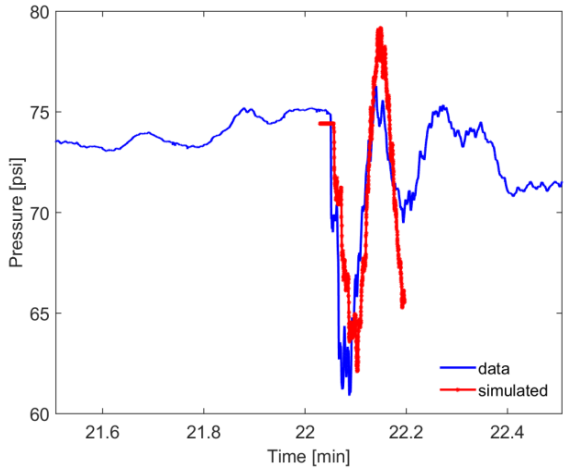


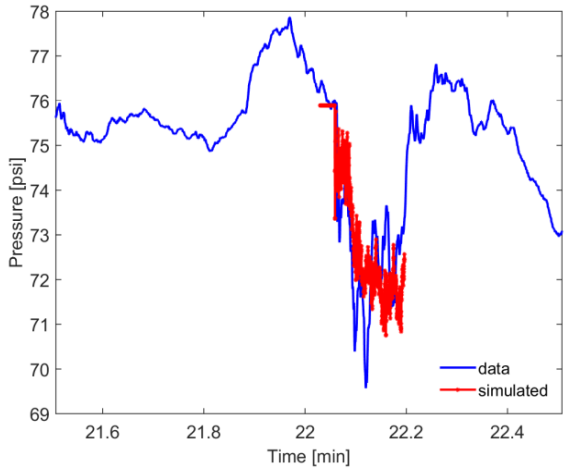
Fort Canning Pearl Hill

- 125 km pipe length
- Population ~600,000
- Daily supply ~30 MGD
- ~ 20,000 nodes; ~ 20,000 pipes

Distribution system

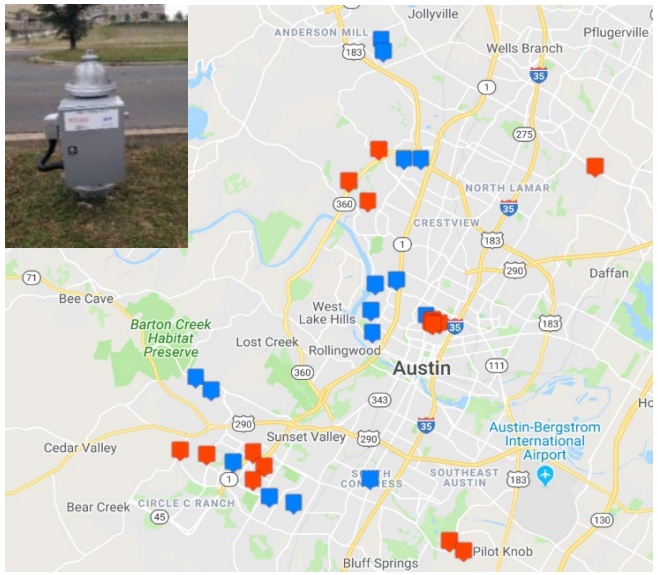
Burst event: measured vs. simulated



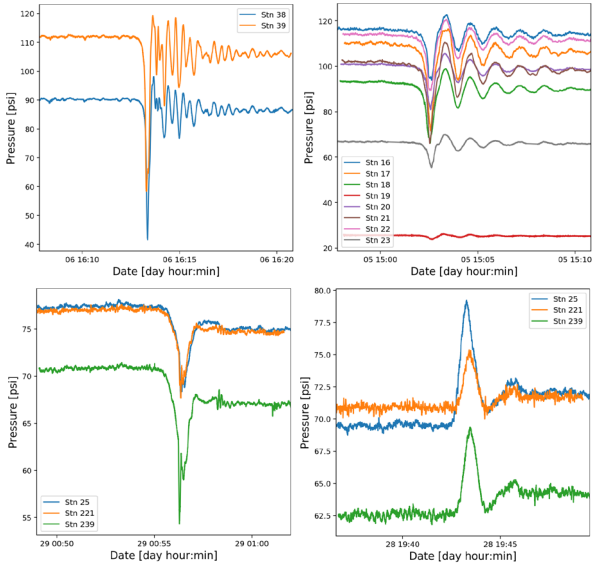


Transients in WDSs

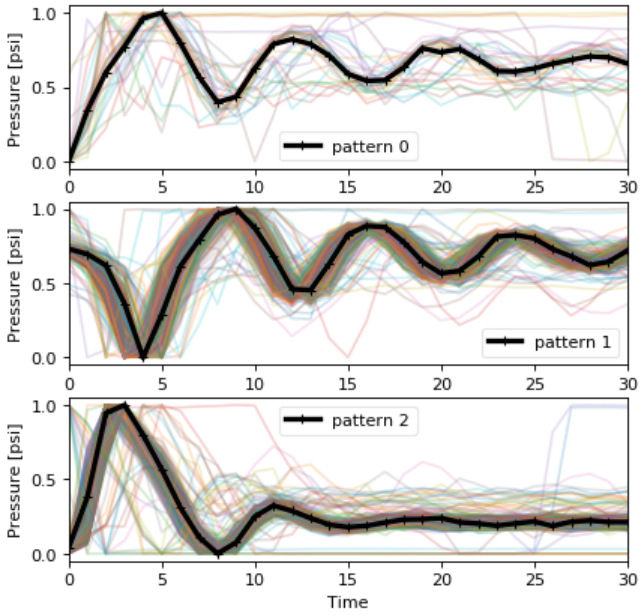
Distributed sensing



Pressure monitoring



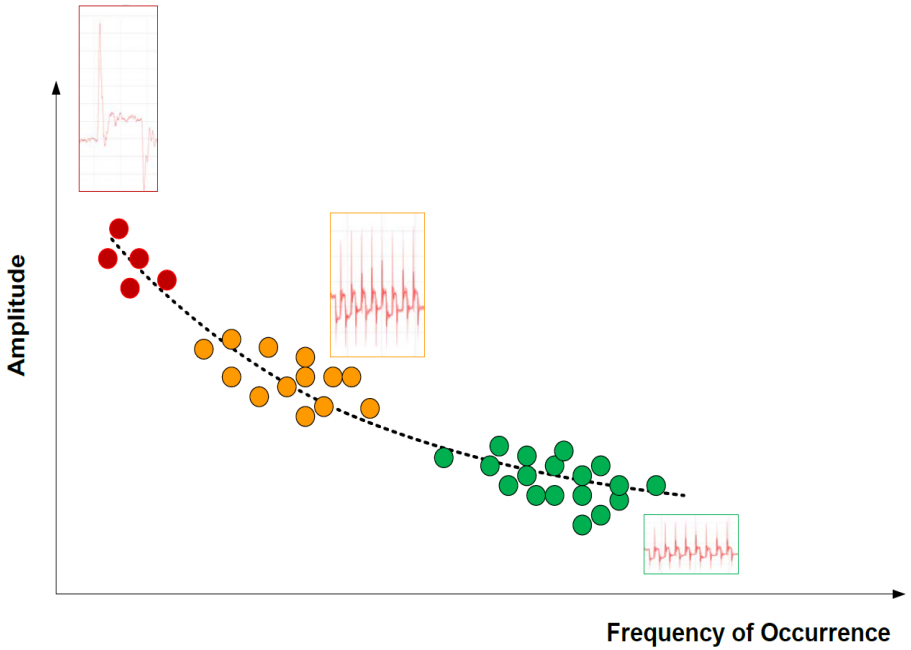
Identifying transients

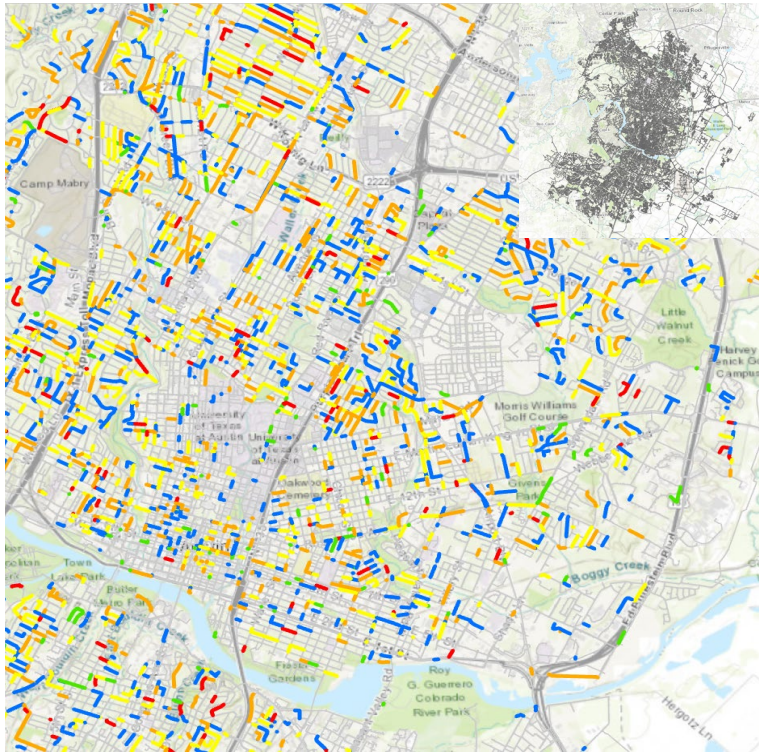


Xing, L., & Sela, L. (2019). Unsteady pressure patterns discovery from high-frequency sensing in water distribution systems. Water Research, 158, 291-300.

Transients in WDSs

- Integrate in condition assessment models through the interaction of the pressure transient waves with the pipelines
- Incorporate cumulative effects of long-term stress cycles in pipe failure prediction models





Inflowmatrix https://inflowmatix.com/

Transient software

Bentley KYPIPE InfoSurge TSNET

Commercial | **Open-source**

✓ Graphical user interface

✓ Extended modeling capabilities

✓ Integration with GIS

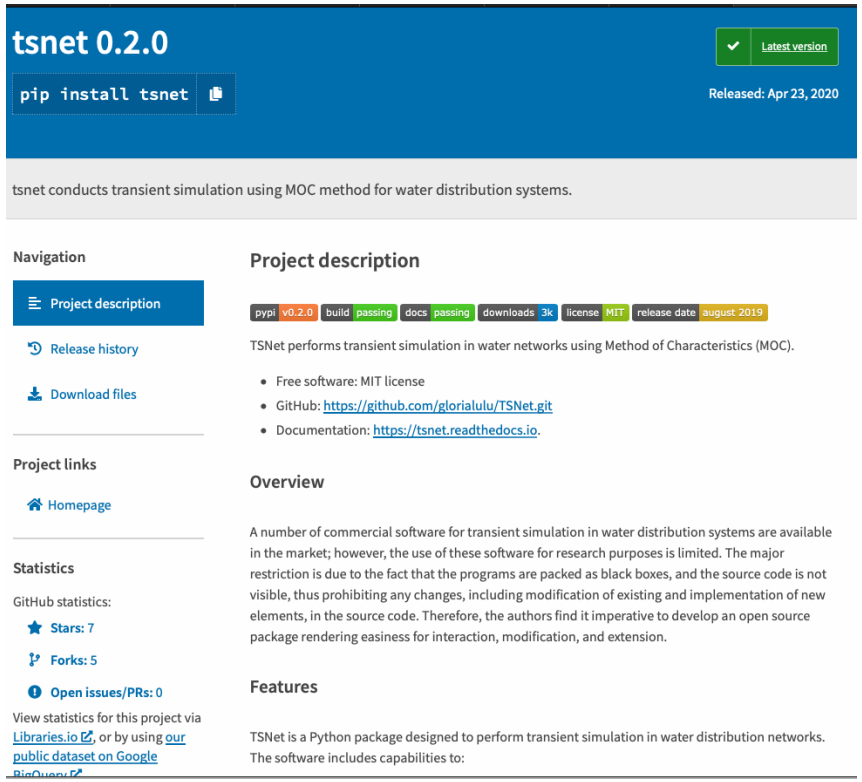
✗ Difficult to extract, analyze, and visualize results

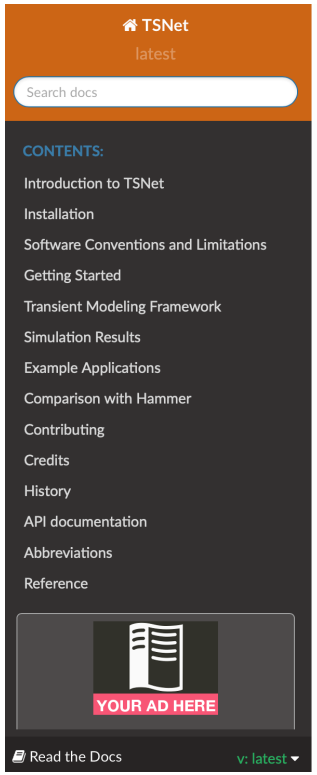
✗ Inability to run simulations systematically

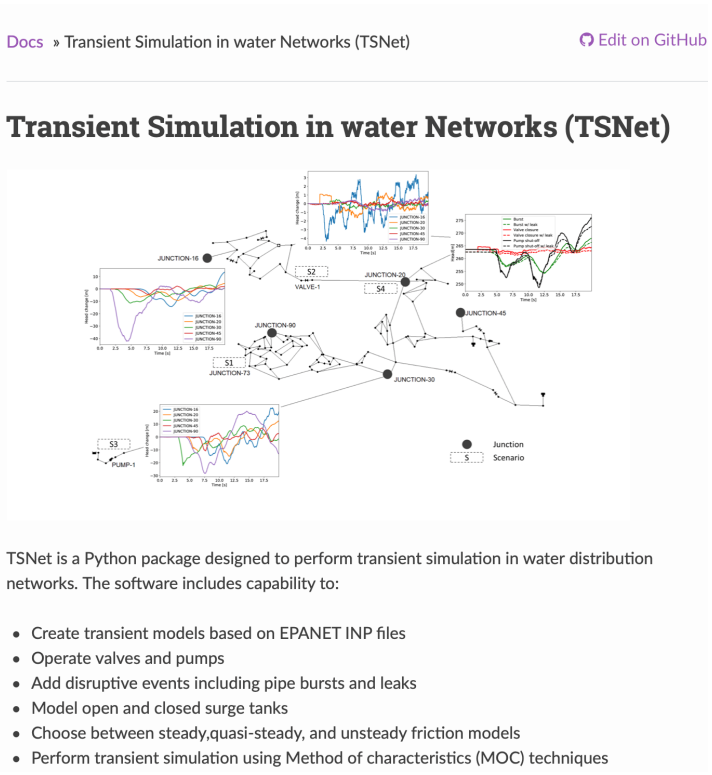
✗ Closed-source, users can't modify, add, and customize models

✗ Doesn't interact with programming software

Transient **S**imulation in water **Net**works (TSNet)







https://github.com/glorialulu/TSNet

https://tsnet.readthedocs.io.

References

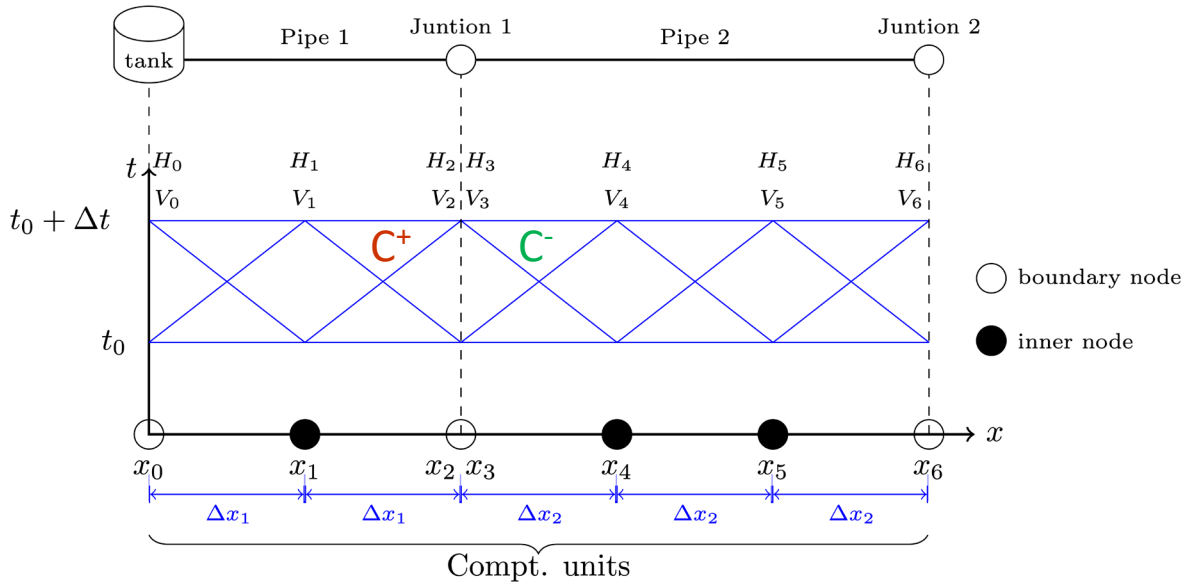
- Larock, B. E., Jeppson, R. W., & Watters, G. Z. (1999). Hydraulics of pipeline systems. CRC press.
- Wylie, E. B., Streeter, V. L., & Suo, L. (1993). Fluid transients in systems (Vol. 1, p. 464). Englewood Cliffs, NJ: Prentice Hall.

Governing equations

- Water hammer equations

$$\frac{\partial H}{\partial t} + \frac{a^2}{g}\frac{\partial V}{\partial x} - gV\sin\alpha = 0$$

$$\frac{1}{g}\frac{\partial V}{\partial t} + \frac{\partial H}{\partial x} + h_f = 0$$



- Method of characteristics

$$C+: \quad \frac{dV}{dt} + \frac{g}{a}\frac{dH}{dt} + gh_f - \frac{g}{a}V\sin\alpha = 0 \quad \text{along} \quad \frac{dx}{dt} = a$$

$$C-: \quad \frac{dV}{dt} - \frac{g}{a}\frac{dH}{dt} + gh_f - \frac{g}{a}V\sin\alpha = 0 \quad \text{along} \quad \frac{dx}{dt} = -a$$

Wave speed

- Wave speed characterizes the speed at which pressure wave propagates in the pipeline
- Estimated based on theoretical equations depending on pipe characteristics, such as modulus of elasticity and diameter

$$a = \frac{\sqrt{K/\rho}}{\sqrt{1 + \frac{K}{E}\frac{D}{e}(1-\mu^2)}}$$

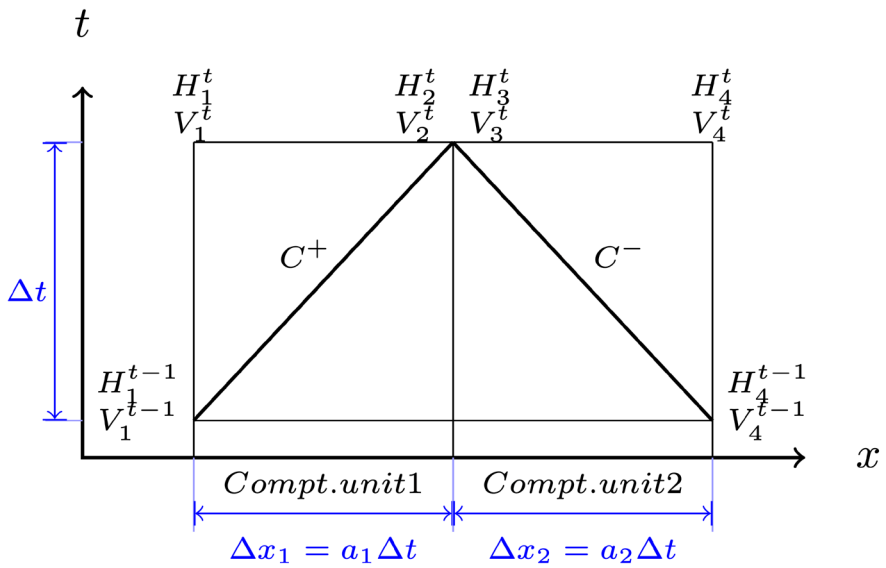
Material	Conditions	Size, in	Speed, ft/s	Speed, m/s
Steel	Schedule 80	12	4365	1331
Ductile iron	Class 52	12	3930	1198
Copper tubing	Type K	2	3905	1190
Cast iron	Class 24	12	3900	1189
PVC	SDR 18	12	1315	401
Polyethylene	Series 80	12	850	260
Concrete cylinder	C300 Class 150	24	3970	1210

https://blog.virtuosity.com/hydraulics-hydrology/wave-speed-for-transient-analysis

Friction models

- Quasi-steady friction model

$$h_f = f\frac{V^2}{2gD}$$



- Explicit MOC discretization

$$C+: \quad (V_i^t - V_{i-1}^{t-1}) + \frac{g}{a}(H_i^t - H_{i-1}^{t-1}) + \frac{f\Delta t}{2D}V_{i-1}^{t-1}|V_{i-1}^{t-1}| + \frac{g\Delta t}{a}V_{i-1}^{t-1}\sin\alpha = 0$$

$$C-: \quad (V_i^t - V_{i+1}^{t-1}) - \frac{g}{a}(H_i^t - H_{i+1}^{t-1}) - \frac{f\Delta t}{2D}V_{i+1}^{t-1}|V_{i+1}^{t-1}| - \frac{g\Delta t}{a}V_{i+1}^{t-1}\sin\alpha = 0$$

Friction models

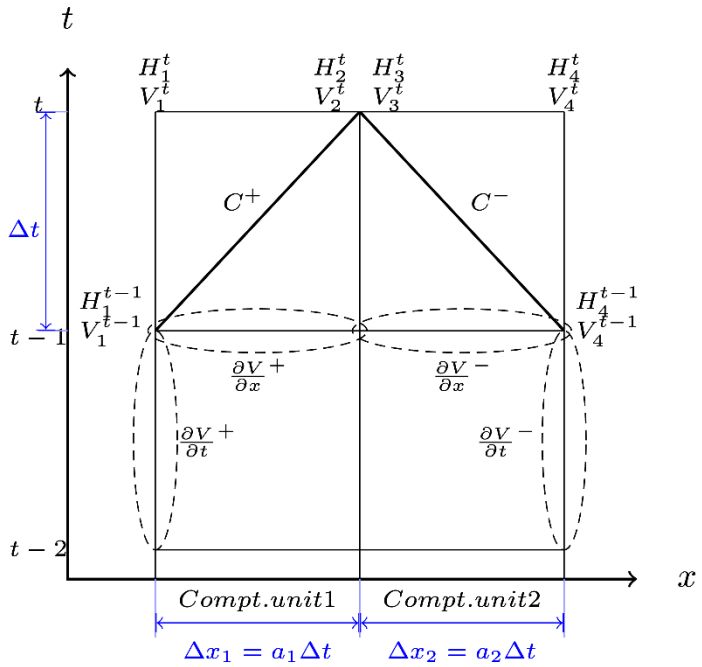
- Unsteady friction model

$$h_f = h_{f_s} + h_{f_u}$$

$$h_{f_u} = \frac{k_u}{2g}\left(\frac{\partial V}{\partial t} + a \cdot \mathrm{sign}(V)\left|\frac{\partial V}{\partial x}\right|\right)$$

$$k_u = \frac{C^*}{2}$$

$$C^* = \begin{cases} 0.00476 & \text{laminar flow } (Re \le 2000) \\ \dfrac{7.41}{Re^{\log\,(14.3/Re^{0.05})}} & \text{turbulent flow } (Re > 2000) \end{cases}$$



- Explicit MOC discretization

$$C+: \quad (V_i^t - V_{i-1}^{t-1}) + \frac{g}{a}(H_i^t - H_{i-1}^{t-1}) + \frac{g}{a}\Delta t V_{i-1}^{t-1}\sin\alpha + \frac{f\Delta x}{2D}V_{i-1}^{t-1}|V_{i-1}^{t-1}| + \frac{k_u}{2g}\left[(V_{i-1}^{t-1} - V_{i-1}^{t-2}) + \mathrm{sign}(V_{i-1}^{t-1})\left|V_i^{t-1} - V_{i-1}^{t-1}\right|\right] = 0$$

$$C-: \quad (V_i^t - V_{i+1}^{t-1}) - \frac{g}{a}(H_i^t - H_{i+1}^{t-1}) + \frac{g}{a}\Delta t V_{i+1}^{t-1}\sin\alpha - \frac{f\Delta x}{2D}V_{i+1}^{t-1}|V_{i+1}^{t-1}| - \frac{k_u}{2g}\left[(V_{i+1}^{t-1} - V_{i+1}^{t-2}) + \mathrm{sign}(V_{i+1}^{t-1})\left|V_{i+1}^{t-1} - V_i^{t-1}\right|\right] = 0$$

Pressure driven demand

Pressure-driven Demand

During the transient simulation in TSNet, the demands are treated as pressure-dependent discharge; thus, the actual demands will vary from the demands defined in the INP file. The actual demands (d_{actual}) are modeled based on the instantaneous pressure head at the node and the demand discharge coefficients, using the following equation:

$$d_{actual} = k\sqrt{H_p}$$

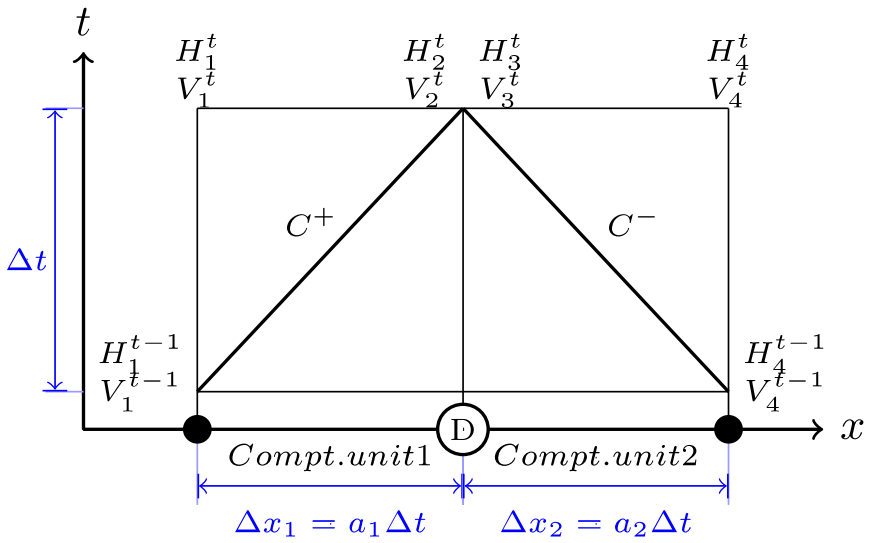
where H_p is the pressure head and k is the demand discharge coefficient, which is calculated from the initial demand (d_0) and pressure head (H_{p0}):

$$k = \frac{d_0}{\sqrt{H_{p0}}}$$

It should be noted that if the pressure head is negative, the demand flow will be treated zero, assuming that a backflow preventer is installed on each node.

Boundary conditions

- ✓ Pumps
- ✓ Valves
- ✓ Bursts
- ✓ Leaks
- ✓ Demand pulse
- ✓ Surge tanks



$$V_2^t A_1 - V_3^t A_2 = Q_T^t \qquad \text{continuity}$$

$$H_2^t = H_t^3 \qquad \text{energy conservation}$$

$$H_A^t = H2^t + H_b - z_t \qquad \text{energy conservation}$$

$$z^t = z^{t-1} + \frac{\Delta t}{a A_T}\left(Q_T^t + Q_T^{t-1}\right) \qquad \text{tank water level}$$

$$H_A^t \mathcal{V}_A^t = \text{constant} \qquad \text{perfect gas law}$$

$$\mathcal{V}_A^t = \mathcal{V}_A^{t-1} - A_T\left(z^t - z^{t-1}\right) \qquad \text{tank air volume}$$

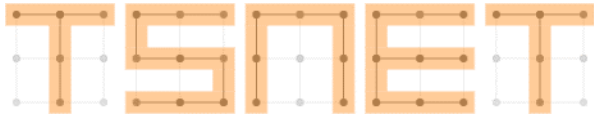
● inner node

Ⓓ device/boundary node

TSNet capabilities

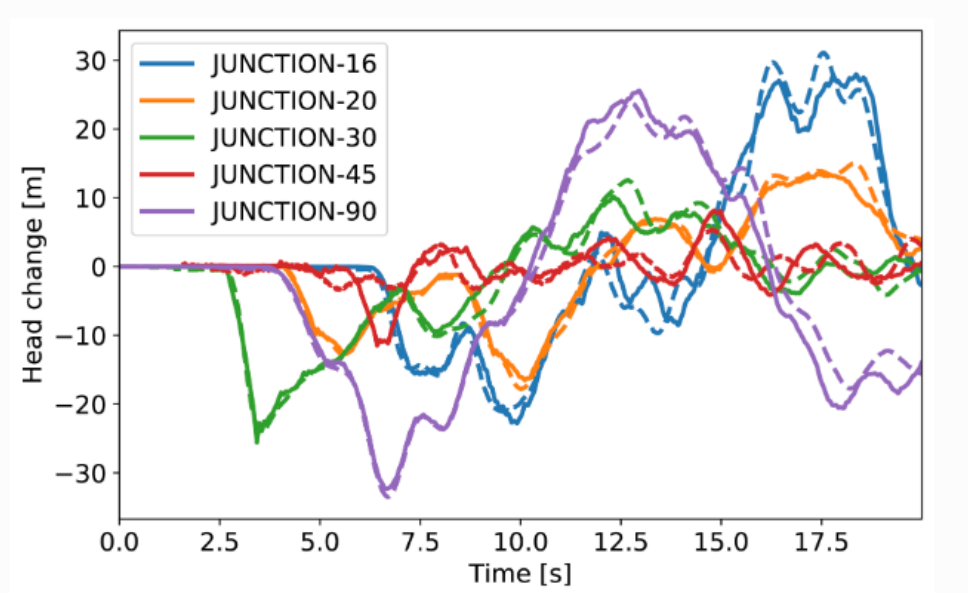
- Create water network based on EPANET .INP files
- Generate transient events by operating valves and pumps
- Add disruptive events, such as pipe bursts, leakage, and demand pulses
- Add surge protection devices, such as surge tanks and air chambers
- Choose between steady, quasi-steady, unsteady friction models
- Visualize and postprocess results

Validation

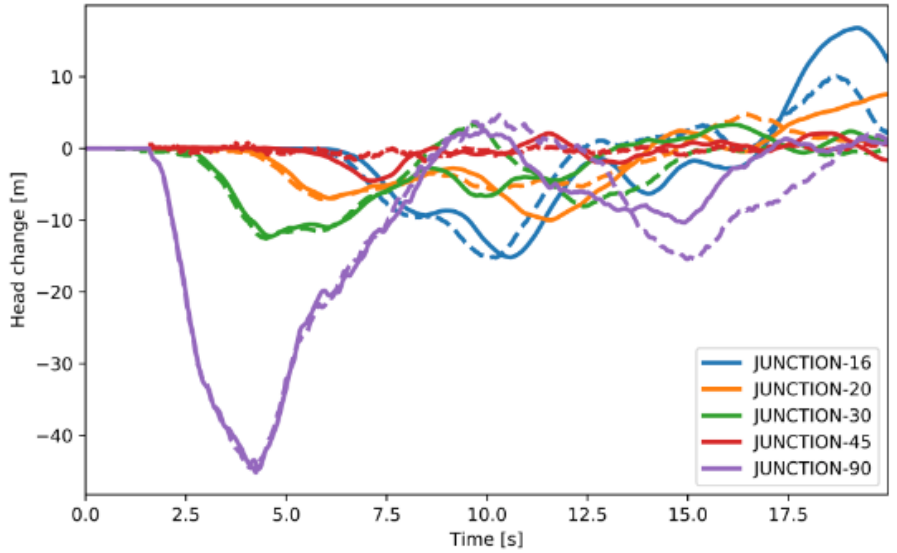


Comparison with Hammer

Pump shutoff



Burst

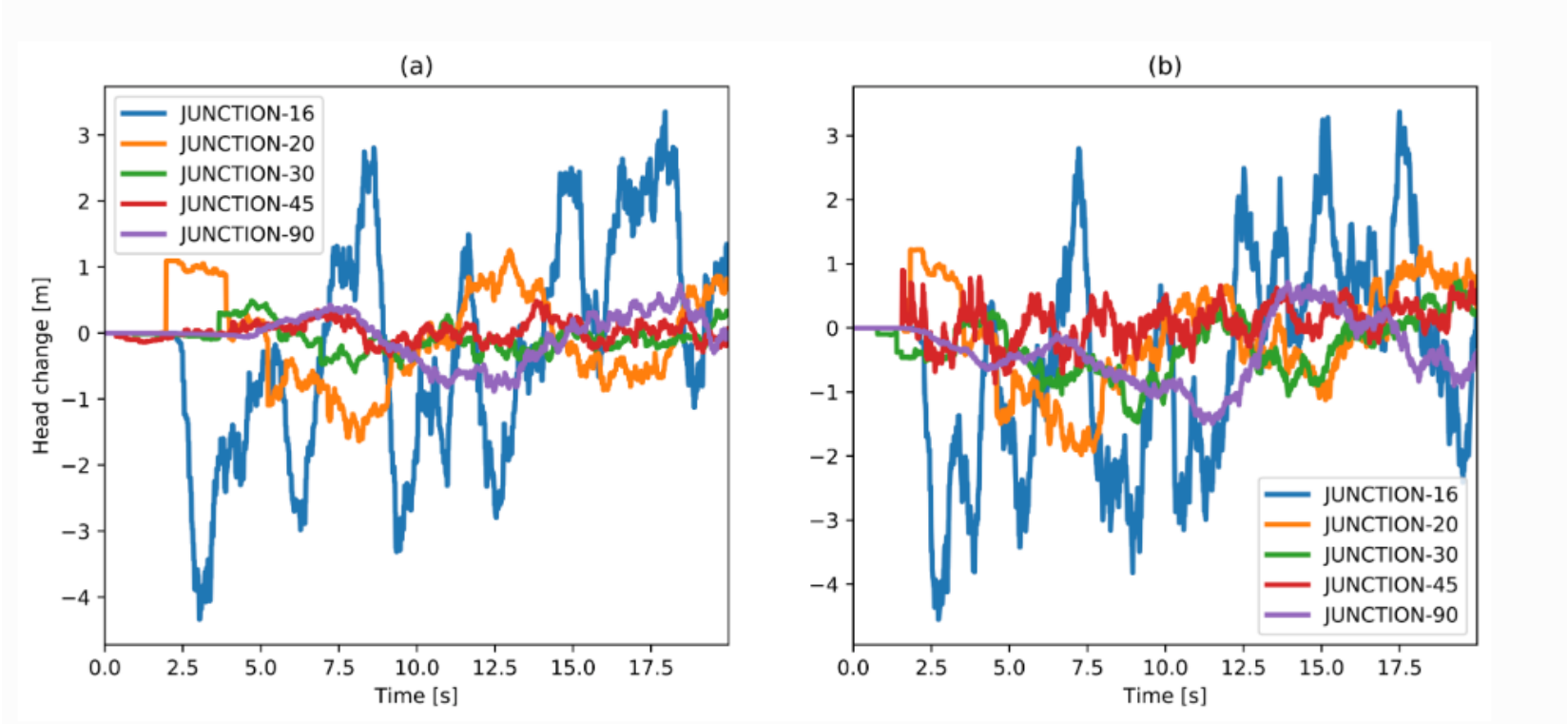


TSNet (solid lines); Hammer (dashed lines)

https://tsnet.readthedocs.io/en/latest/validation.html

Comparison with Hammer

Valve closure



TSNet (a); Hammer (b)

https://tsnet.readthedocs.io/en/latest/validation.html

Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Quick survey

Do you have tsnet installed?

a) Yes

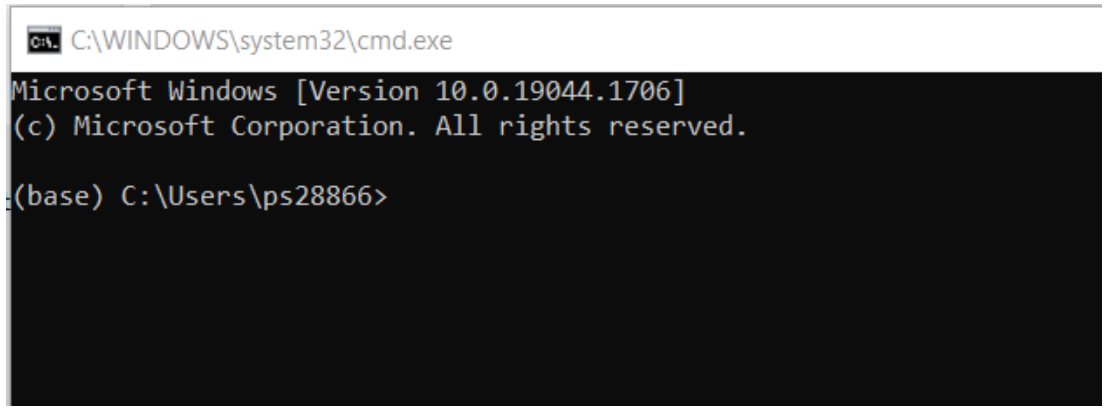
b) No

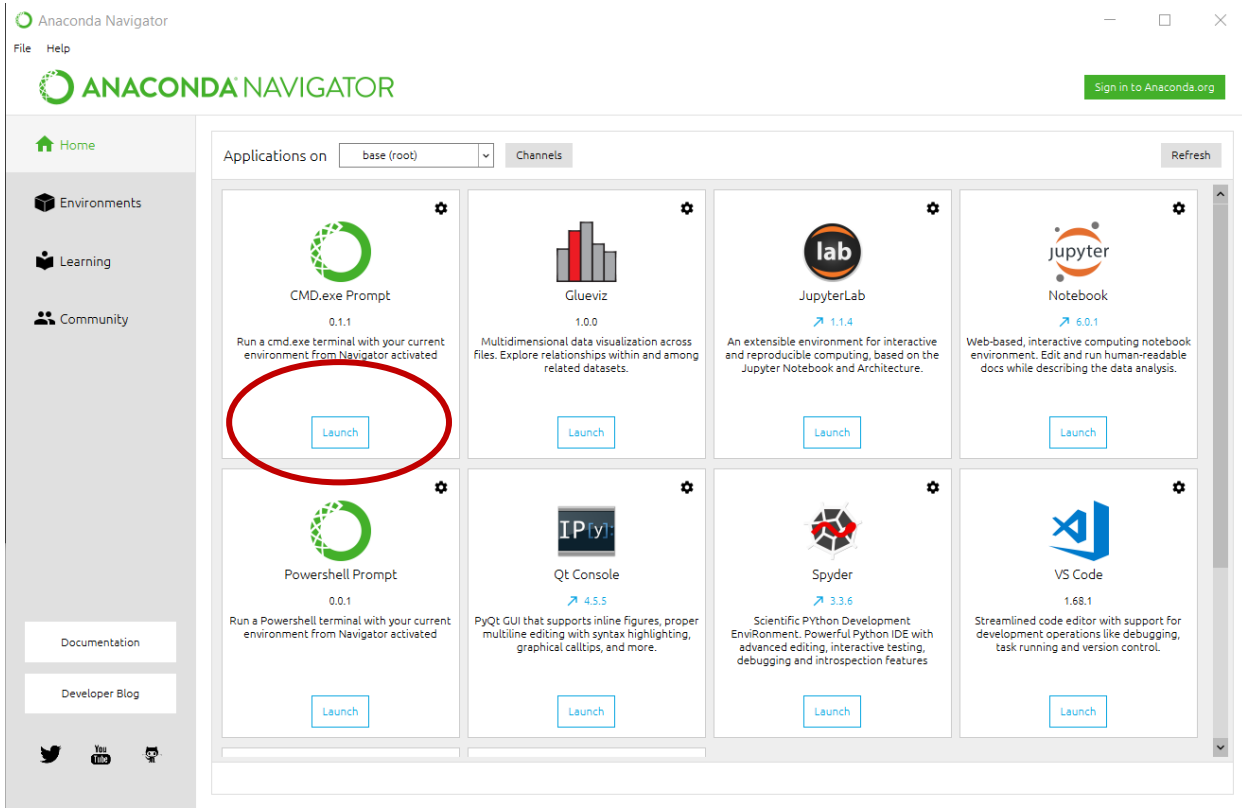
Go to www.menti.com

Code 5404 3693

Installing TSNet

1. Download Anaconda
2. Open your terminal (mac) or command prompt (windows)





Installing TSNet

1. Create conda environment (make sure to set **Python 3.7**):

   conda create --name tsnet python=3.7

2. Activate the newly created environment

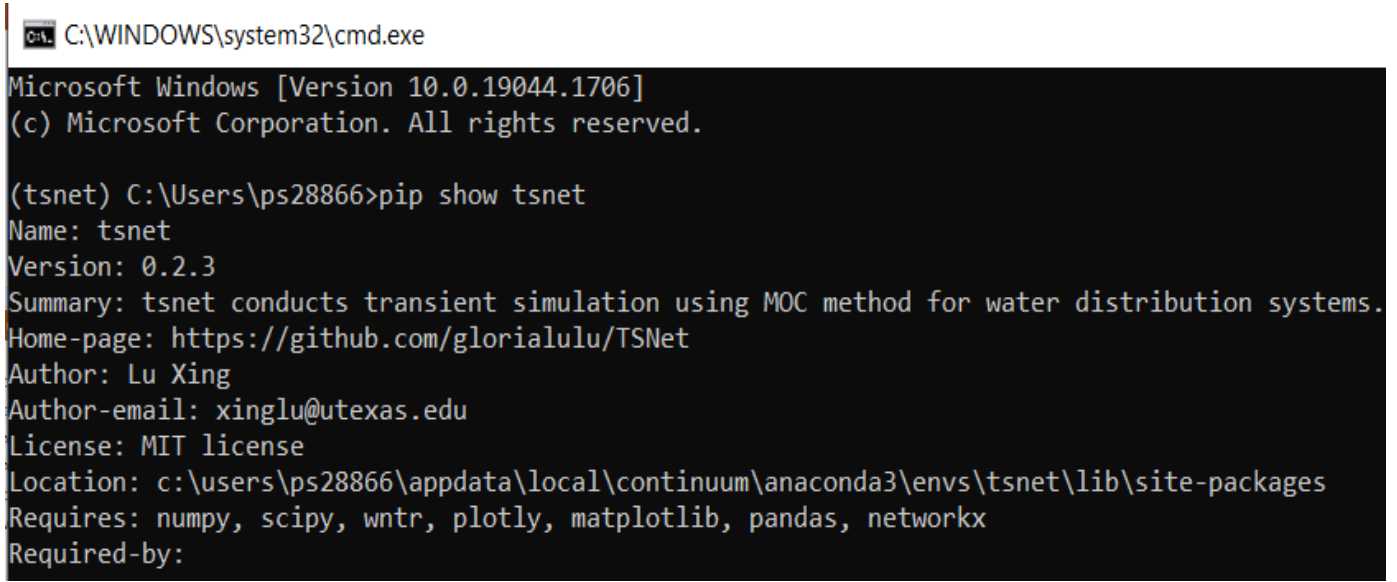
   conda activate tsnet

3. Install tsnet

   pip install tsnet

4. Check the installation.

   pip show tsnet



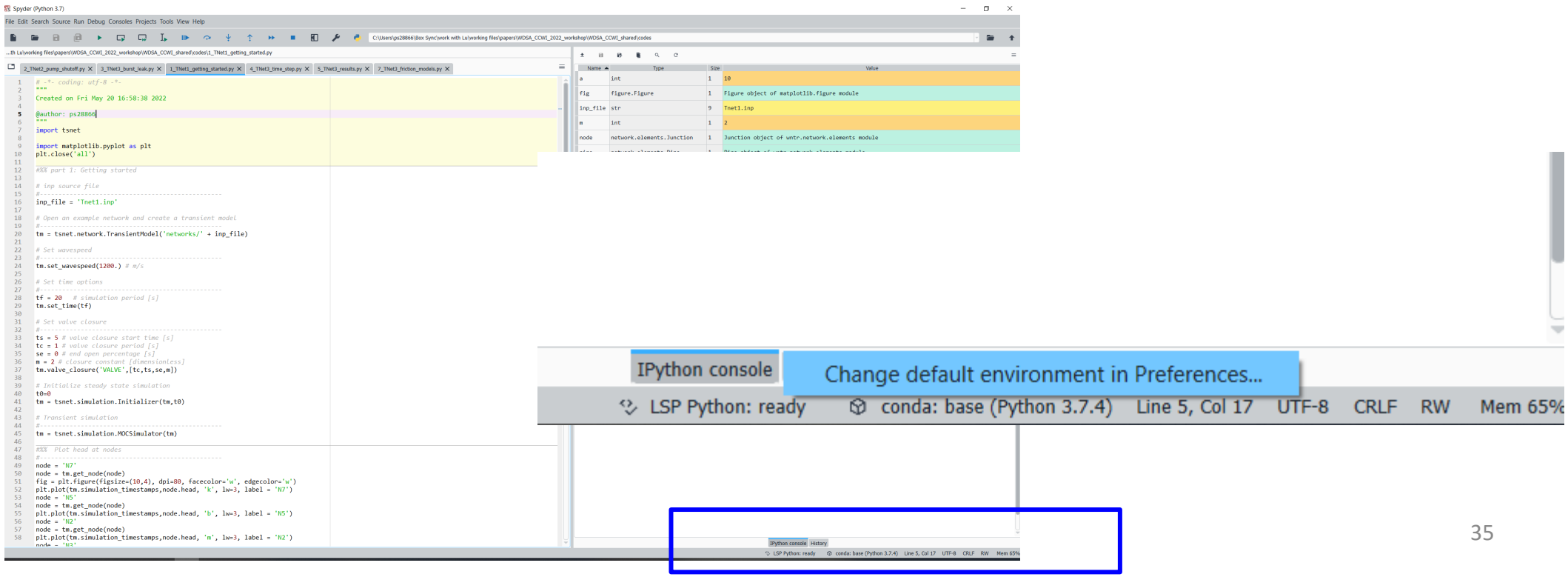
Installing TSNet

- After creating the new environment, you will need to get **Spyder** or **Jupyter** to work with the new environment.
- There are several way to do that, here are some suggestions:

Get Spyder to work with the new environment

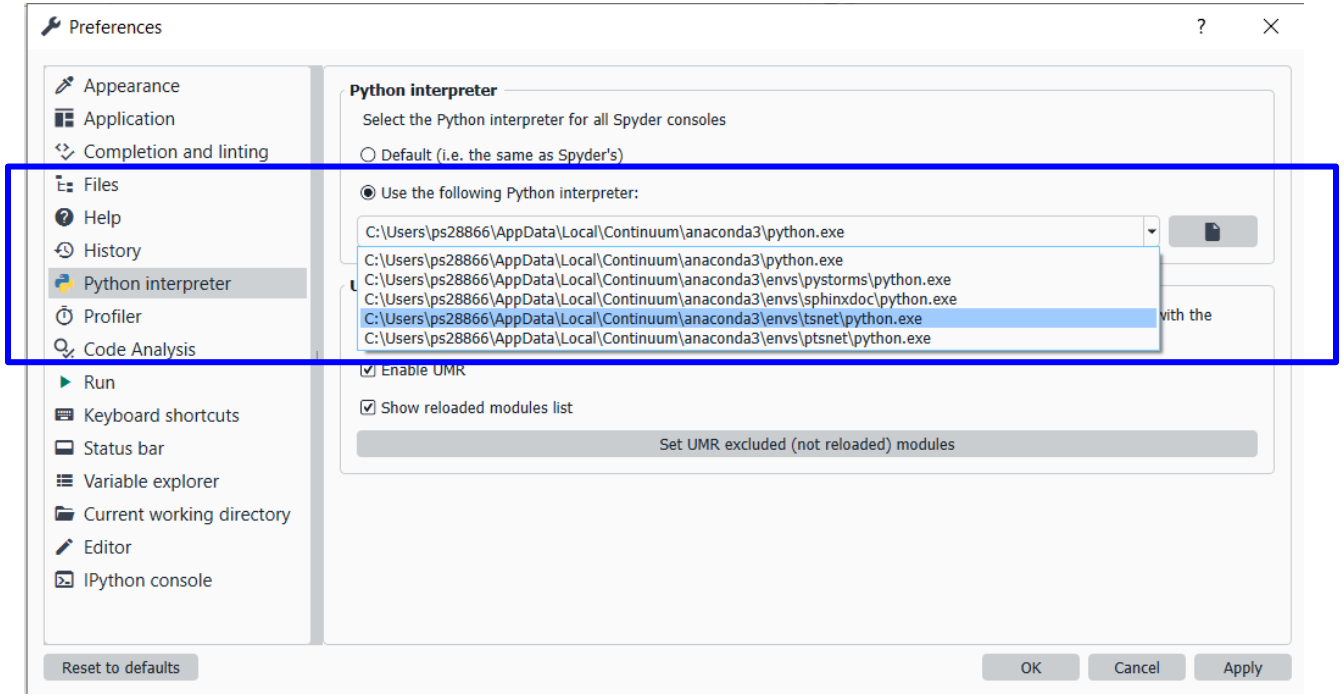
The “Easy” way

1. Click the name of the environment in the status bar
2. Click ‘*Change default environment in Preferences*’

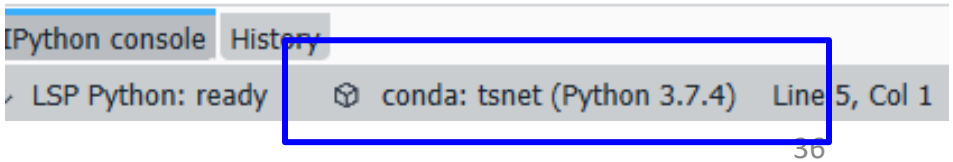


Get Spyder to work with the new environment

3. Select '*Use the following Python interpreter*'
4. Select your environment from the dropdown list
5. Click 'OK'



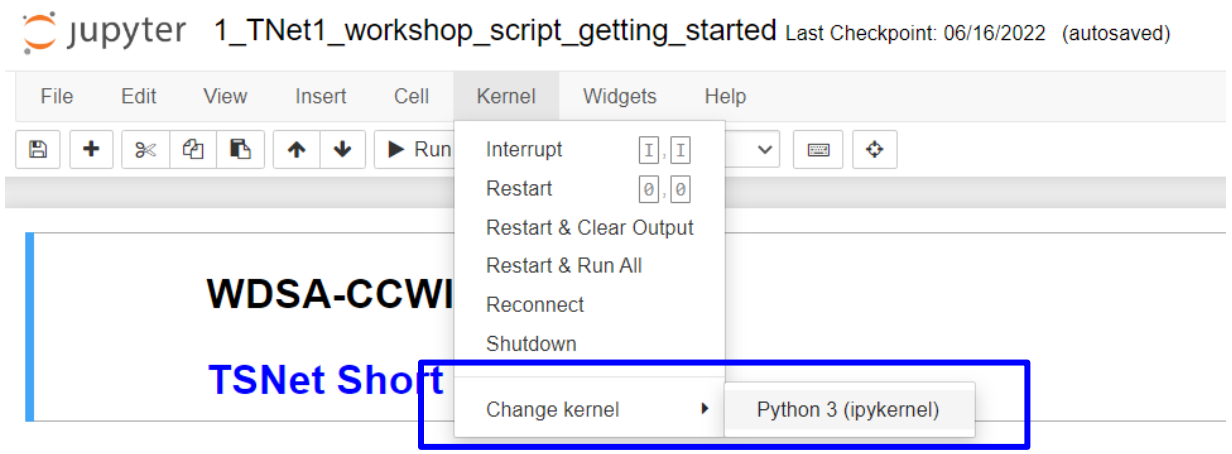
6. You should see tsnet environment in the status bar



Get Jupyter to work with the new environment

The “Easy” way

1. Open Jupyter notebooks from your base environment
2. Your kernel list will only show your current environment



Get Jupyter to work with the new environment

To get your other environment kernels to show:

1. Install `nb_conda_kernels` in your **base** environment

   conda install nb_conda_kernels

2. Activate the environment you want to use in your notebook

   conda activate tsnet

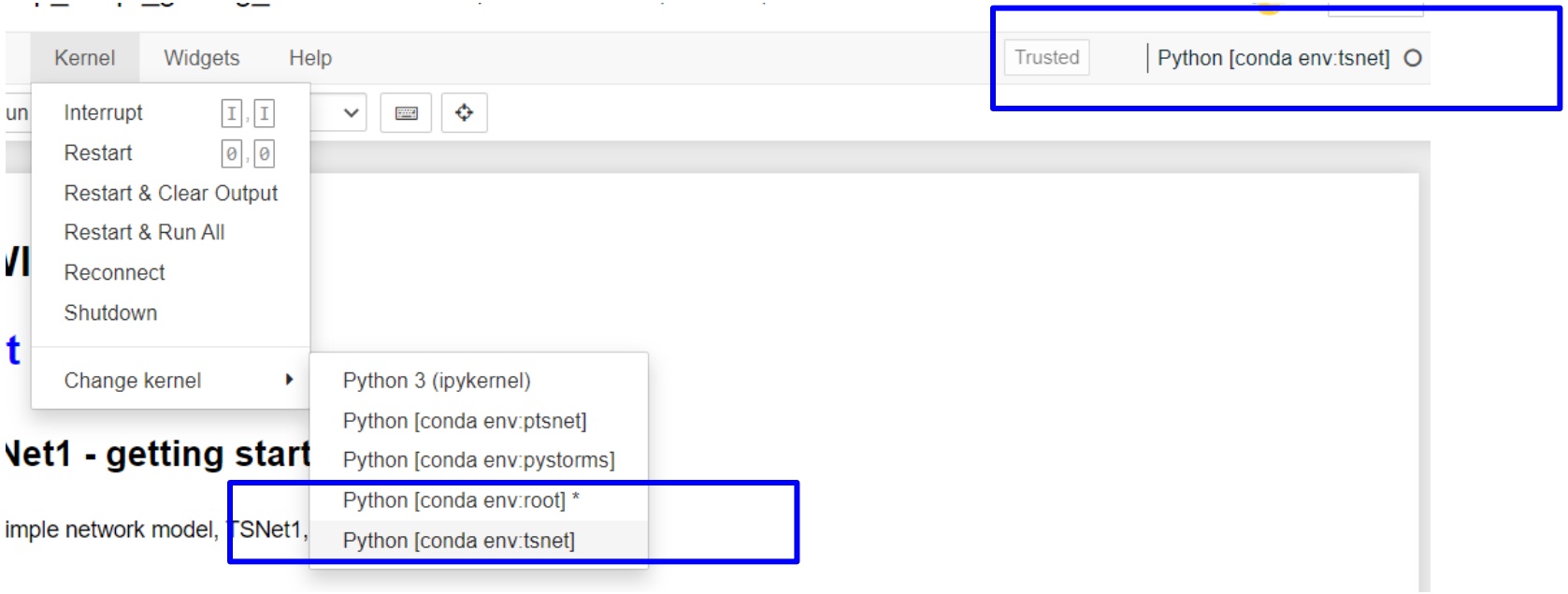
3. Install `iypkernel`

   conda install ipykernel

4. Restart Jupyter Notebooks from your base environment (you might need to restart Anaconda Navigator as well)

Get Jupyter to work with the new environment

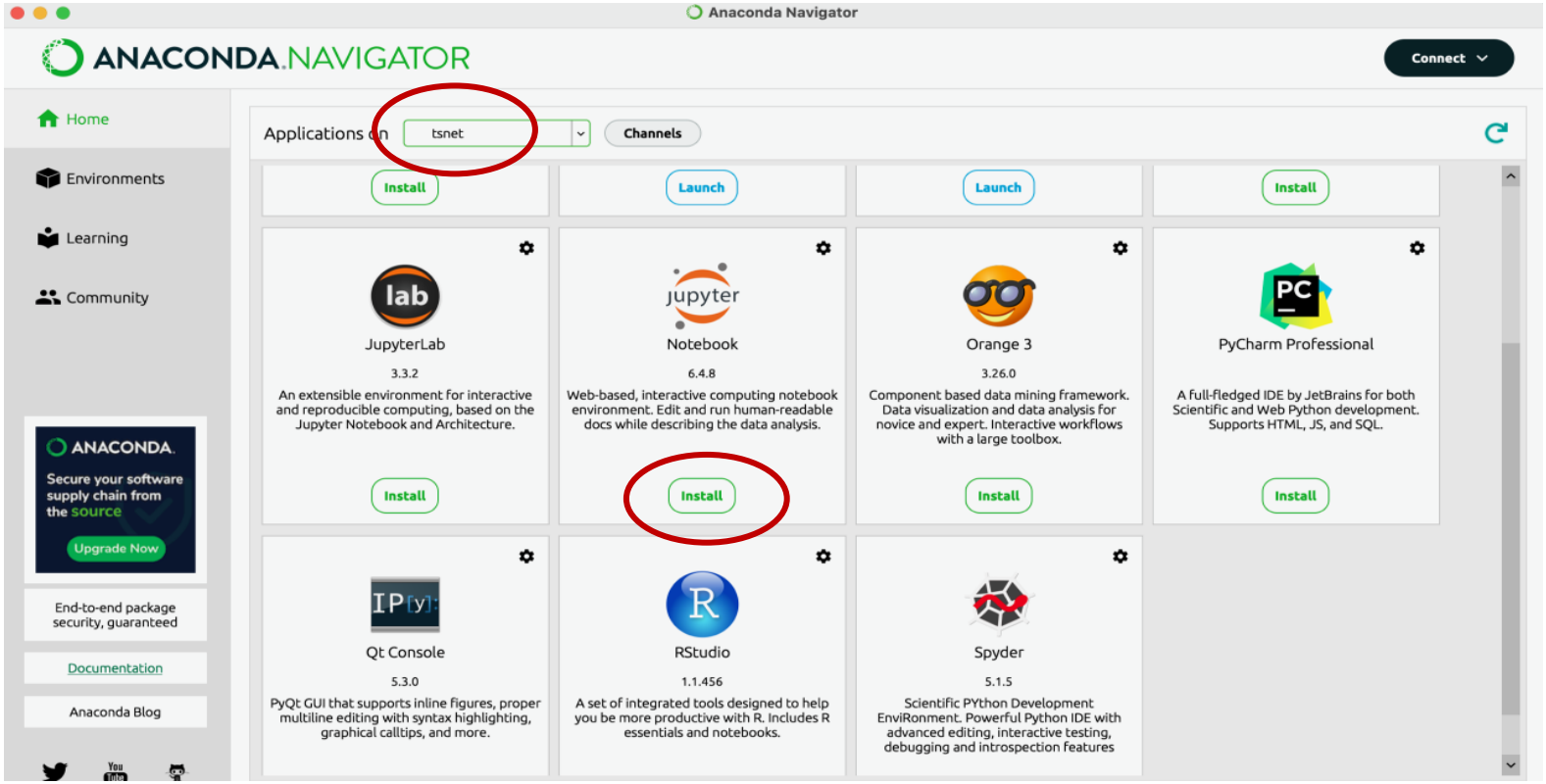
5. Your environments should appear now in your kernel list and you can switch between them



Get Spyder/Jupyter to work with the new environment

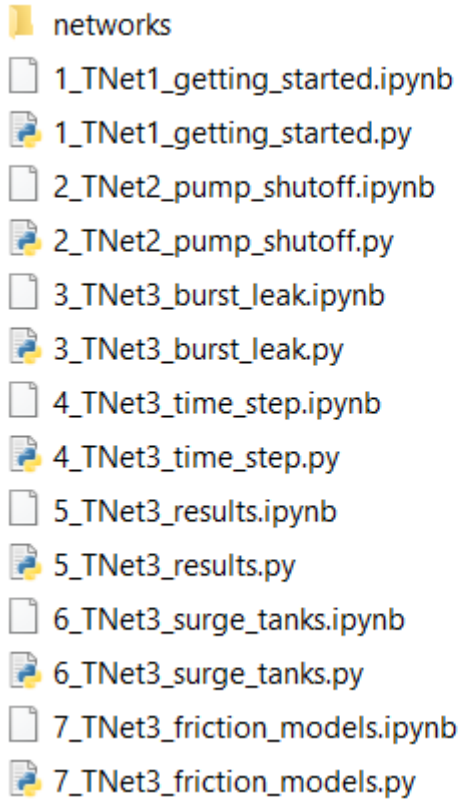
The "Quick and Dirty" way

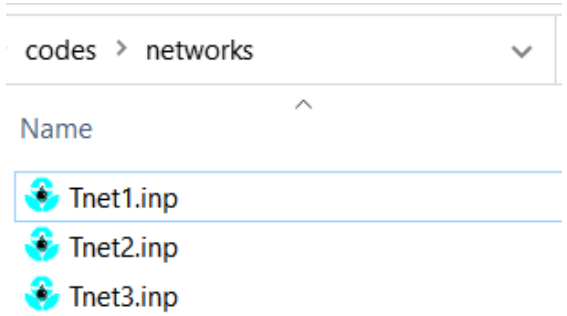
- Install via the **Anaconda Navigator** or **terminal** while in the `tsnet` environment.
- Note: this will take memory and not recommended if you have many environments



Download codes and networks

- Download the relevant codes
  https://utexas.box.com/s/io2azj1bci4gtdrouz3xrd629tb6k1wi
- **Spyder** and **Jupyter Notebooks** versions for each code
- Three example networks





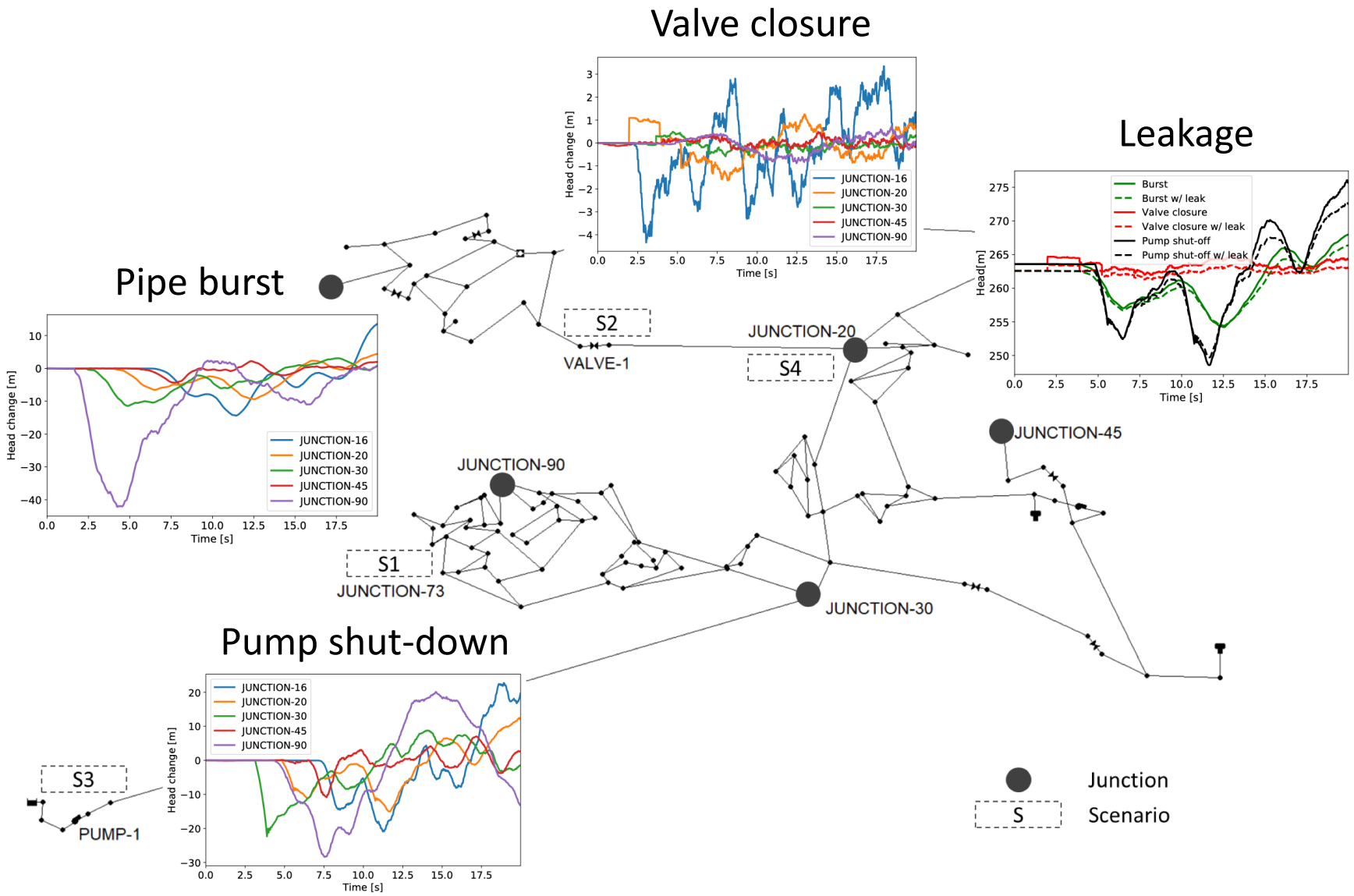
Installation & troubleshooting

Working with TSNet

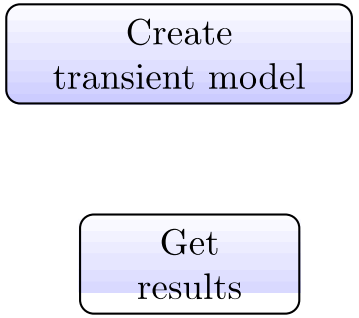
Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

TSNet: overview



Framework

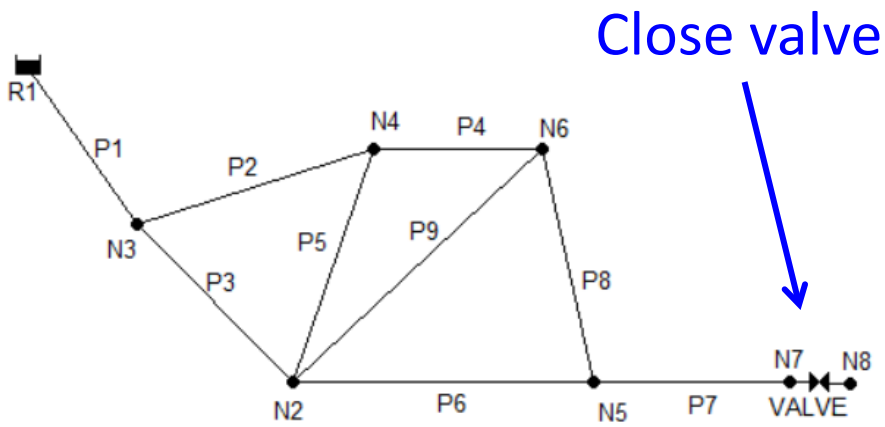


Xing, L., & Sela, L. (2020). Transient simulations in water distribution networks: TSNet python package. *Advances in Engineering Software*, *149*, 102884.

Scenario 1: Getting started

We will start with a simple network model, TSNet1, to demonstrate how to:

- Import tsnet
- Generate a transient model
- Set wave speed
- Set time step and simulation period
- Perform initial condition calculation
- Create transient event
- Run transient simulation
- Plot simulation results



Scenario 1: Getting started

- Open Spyder or Jupyter Notebooks: *1_TNet1_getting_started*

jupyter 1_TNet1_workshop_script_getting_started Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

Run Markdown

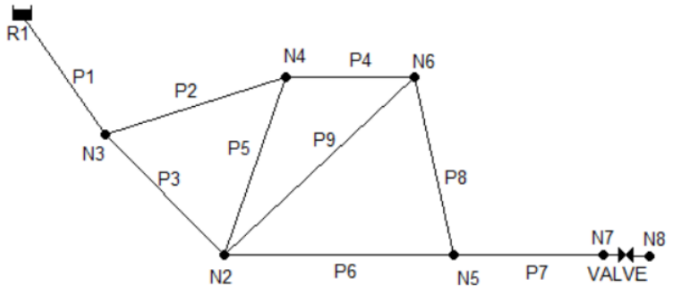
WDSA-CCWI 2022

TSNet Short Tutorial

Example 1: TNet1 - getting started

We start with using a simple network model, TSNet1, to demonstrate how to:

- Import tsnet
- Generate a transient model
- Set wave speed
- Set time step and simulation period
- Perform initial condition calculation
- Create transient event
- Run transient simulation and save results to .obj file
- Plot simulation results



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

...ith Lu\working files\papers\WDSA_CCWI_2022_workshop\codes\1_TNet1_workshop_script_getting_started.py

1_TNet1_workshop_script_getting_started.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 20 16:58:38 2022

@author: ps28866
"""
import tsnet

import matplotlib.pyplot as plt
plt.close('all')

#%% part 1: Getting started

# inp source file
#------------------------------------------------
inp_file = 'Tnet1.inp'

# Open an example network and create a transient model
#------------------------------------------------
tm = tsnet.network.TransientModel('networks/' + inp_file)

# Set wavespeed
#------------------------------------------------
tm.set_wavespeed(1200.) # m/s

# Set time options
#------------------------------------------------
tf = 20   # simulation period [s]
tm.set_time(tf)

# Set valve closure
#------------------------------------------------
ts = 5 # valve closure start time [s]
tc = 1 # valve closure period [s]
se = 0 # end open percentage [s]
m = 2 # closure constant [dimensionless]
tm.valve_closure('VALVE',[tc,ts,se,m])
```

Scenario 1: Getting started

```
import tsnet

import matplotlib.pyplot as plt
plt.close('all')

#%% part 1: Getting started

# inp source file
#------------------------------------------------
inp_file = 'Tnet1.inp'

# Open an example network and create a transient model
#------------------------------------------------
tm = tsnet.network.TransientModel('networks/' + inp_file)

# Set wavespeed
#------------------------------------------------
tm.set_wavespeed(1200.) # m/s

# Set time options
#------------------------------------------------
tf = 20   # simulation period [s]
tm.set_time(tf)

# Set valve closure
#------------------------------------------------
ts = 5 # valve closure start time [s]
tc = 1 # valve closure period [s]
se = 0 # end open percentage [s]
m = 2 # closure constant [dimensionless]
tm.valve_closure('VALVE',[tc,ts,se,m])

# Initialize steady state simulation
t0=0
tm = tsnet.simulation.Initializer(tm,t0)

# Transient simulation
#------------------------------------------------
tm = tsnet.simulation.MOCSimulator(tm)
```

Scenario 1: Getting started

```
7   import tsnet
8
9   import matplotlib.pyplot as plt
10  plt.close('all')
11
12  #%% part 1: Getting started
13
14  # inp source file
15  #-----------------------------------------------
16  inp_file = 'Tnet1.inp'
17
18  # Open an example network and create a transient model
19  #-----------------------------------------------
20  tm = tsnet.network.TransientModel('networks/' + inp_file)
21
22  # Set wavespeed
23  #-----------------------------------------------
24  tm.set_wavespeed(1200.) # m/s
25
26  # Set time options
27  #-----------------------------------------------
28  tf = 20    # simulation period [s]
29  tm.set_time(tf)
```

Import TSNet Package

Specify .inp file

Create tsnet model

Specify wave speeds

Define simulation duration

Scenario 1: Getting started

- Documentation:

valve_closure(*name*, *rule*, *curve=None*) [source]

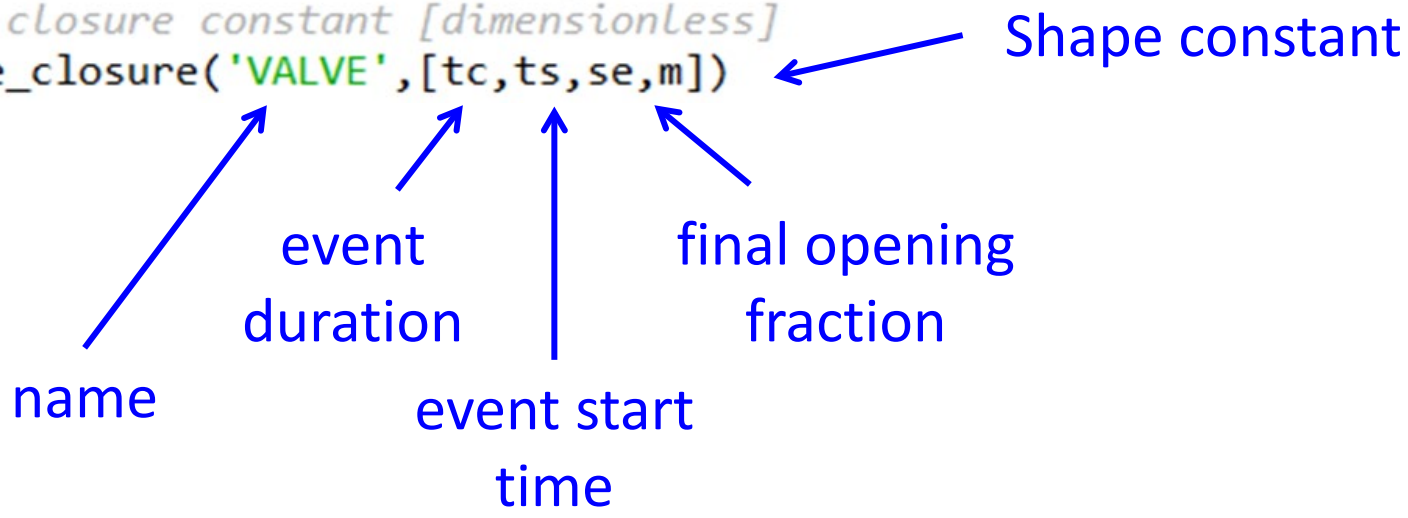
Set valve closure rule

Parameters:

- **name** (*str*) – The name of the valve to close
- **rule** (*list*) – Contains paramters to define valve operation rule rule = [tc,ts,se,m] tc : the duration takes to close the valve [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]
- **curve** (*list*) – [(open_percentage[i], kl[i]) for i] List of open percentage and the corresponding valve coefficient

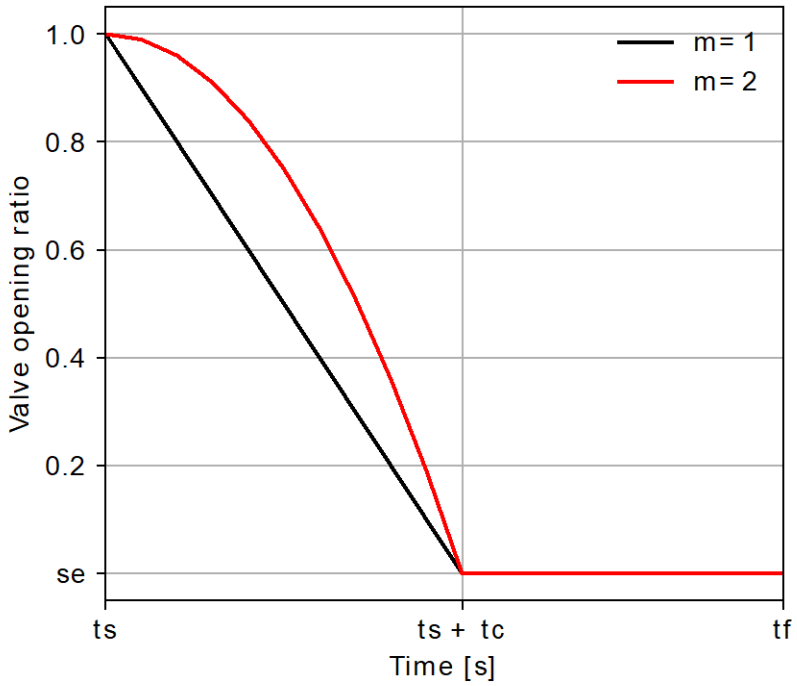
Scenario 1: Getting started

```
31  # Set valve closure
32  #----------------------------------------
33  ts = 5 # valve closure start time [s]
34  tc = 1 # valve closure period [s]
35  se = 0 # end open percentage [s]
36  m = 2 # closure constant [dimensionless]
37  tm.valve_closure('VALVE',[tc,ts,se,m])
```

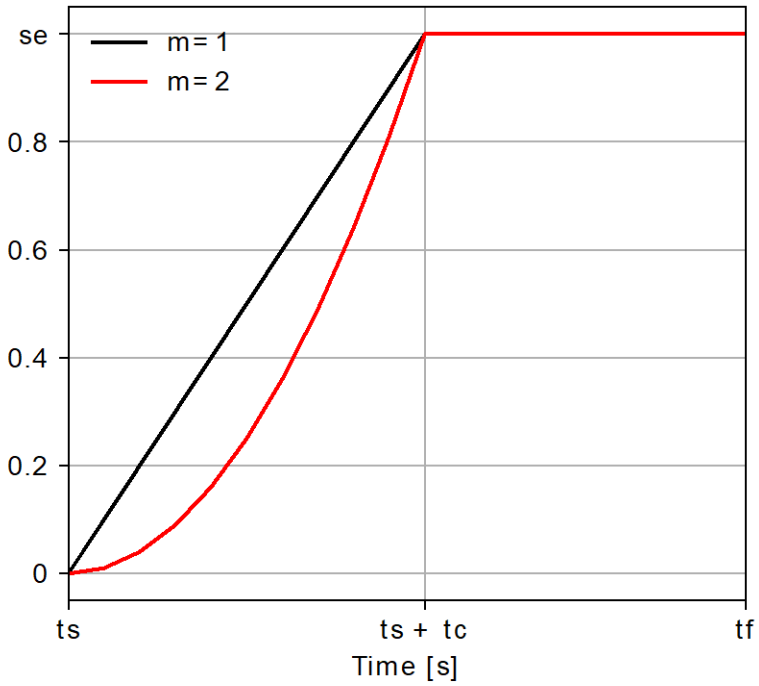


Scenario 1: Getting started

Valve closure



Valve opening curve



Scenario 1: Getting started

- Default valve characteristic curve: gate valve

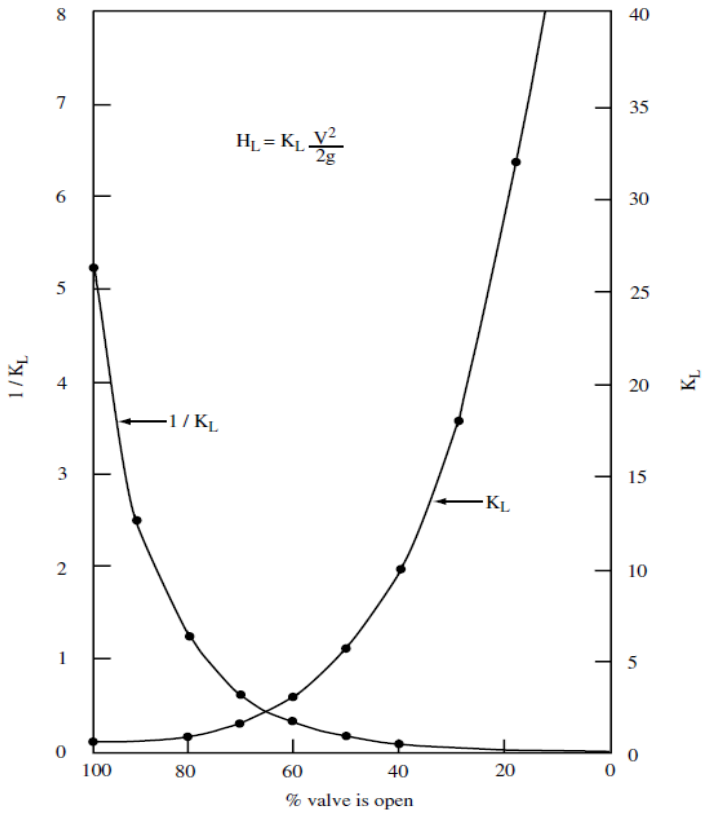


Figure 10.12 K_L and $1/K_L$ as functions of percent-open.

Larock, B. E., Jeppson, R. W., & Watters, G. Z. (1999). Hydraulics of pipeline systems. CRC press.

- User-defined valve characteristic curve:

```
20  valve_op = [tc,ts,se,m]
21  percent_open = np.linspace(100,0,11)
22  kl = [1/0.2, 2.50, 1.25, 0.625, 0.333, 0.17,
23      0.100, 0.0556, 0.0313, 0.0167, 0.0]
24  curve = [(percent_open[i], kl[i]) for i in range(len(kl))]
25  tm.valve_closure('VALVE', valve_op,curve)
```

Scenario 1: Getting started

```
39  # Initialize steady state simulation
40  t0=0
41  tm = tsnet.simulation.Initializer(tm,t0)
42
43  # Transient simulation
44  #----------------------------------------
45  tm = tsnet.simulation.MOCSimulator(tm)
```

Initialize simulation

Run transient simulation

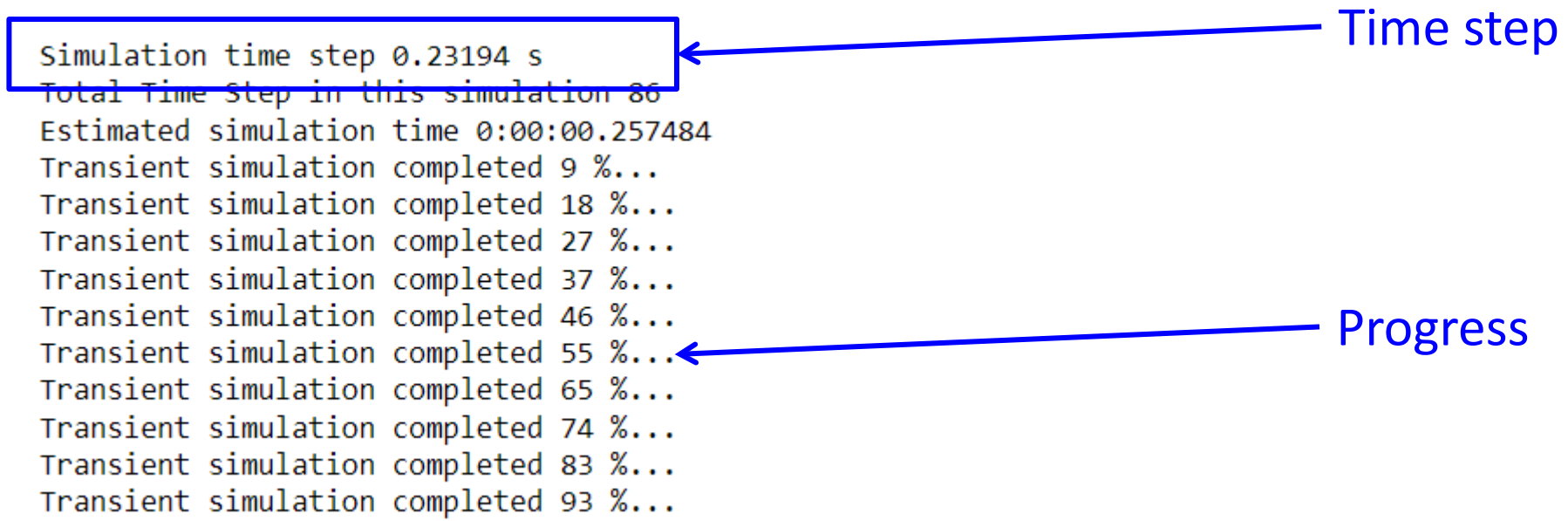
- Initial conditions for transient simulation are simulated using WNTR package
- User needs to provide the time step (`t0`) in the steady simulation to use as the initial conditions. This time step is w.r.t. the settings in the .inp file
- More details in next examples

Scenario 1: Getting started

```
import tsnet

import matplotlib.pyplot as plt
plt.close('all')

#%% part 1: Getting started

# inp source file
#------------------------------------------------
inp_file = 'Tnet1.inp'

# Open an example network and create a transient model
#------------------------------------------------
tm = tsnet.network.TransientModel('networks/' + inp_file)

# Set wavespeed
#------------------------------------------------
tm.set_wavespeed(1200.) # m/s

# Set time options
#------------------------------------------------
tf = 20   # simulation period [s]
tm.set_time(tf)

# Set valve closure
#------------------------------------------------
ts = 5 # valve closure start time [s]
tc = 1 # valve closure period [s]
se = 0 # end open percentage [s]
m = 2 # closure constant [dimensionless]
tm.valve_closure('VALVE',[tc,ts,se,m])

# Initialize steady state simulation
t0=0
tm = tsnet.simulation.Initializer(tm,t0)

# Transient simulation
#------------------------------------------------
tm = tsnet.simulation.MOCSimulator(tm)
```

Scenario 1: Getting started

- Simulation progress:



Scenario 1: Getting started

- Results Structure

Simulation results are returned and saved in the `tsnet.network.model.TransientModel` object for each node and link in the networks.

Node results include the following attributes:

- Head [m]
- Emitter discharge (including leaks and bursts) $[m^3/s]$
- Actual demand discharge $[m^3/s]$

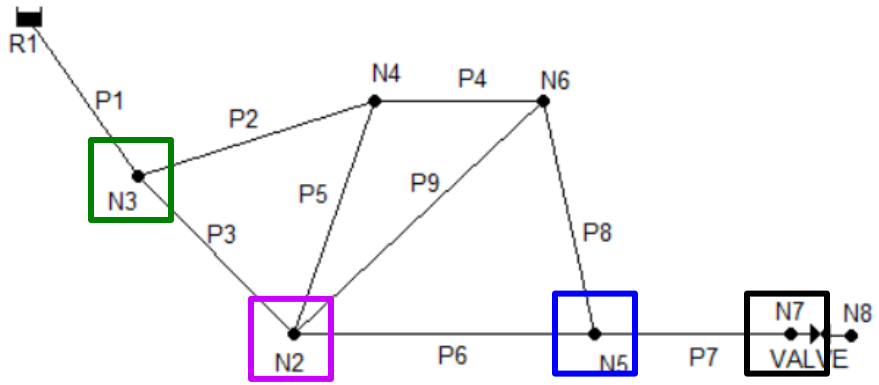
Link results include the following attributes:

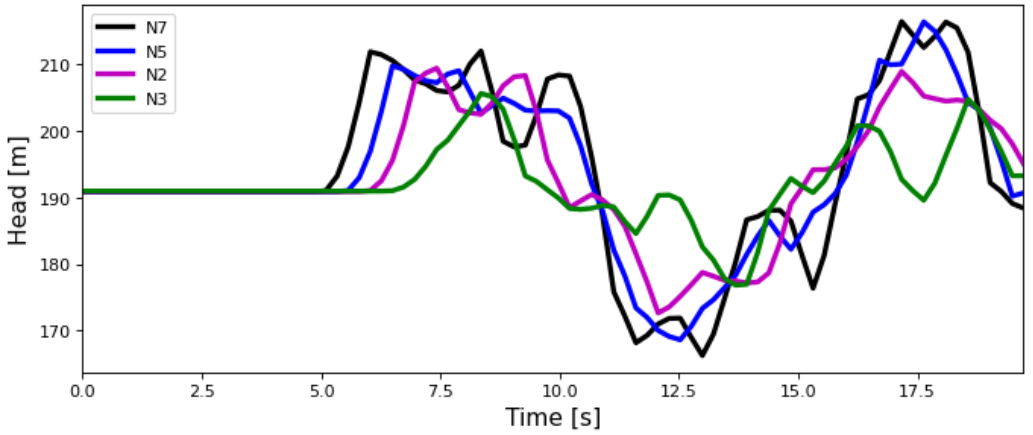
- Head at start node [m]
- Flow velocity at start node $[m^3/s]$
- Flow rate at start node $[m^3/s]$
- Head at end node [m]
- Flow velocity at end node $[m^3/s]$
- Flow rate at end node $[m^3/s]$

Scenario 1: Getting started

- Get results – heads

```
48  # Plot head at nodes
49  #-----------------------------------------------
50  node = 'N7'
51  node = tm.get_node(node)
52  fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='
53  plt.plot(tm.simulation_timestamps,node.head, 'k', lw=3, label = 'N7')
```

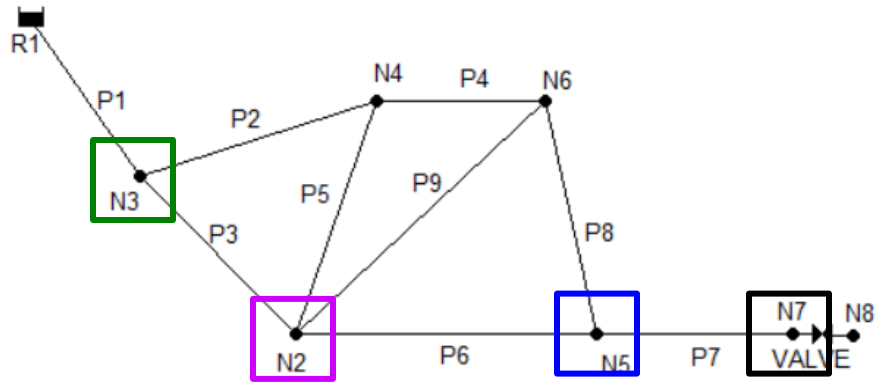


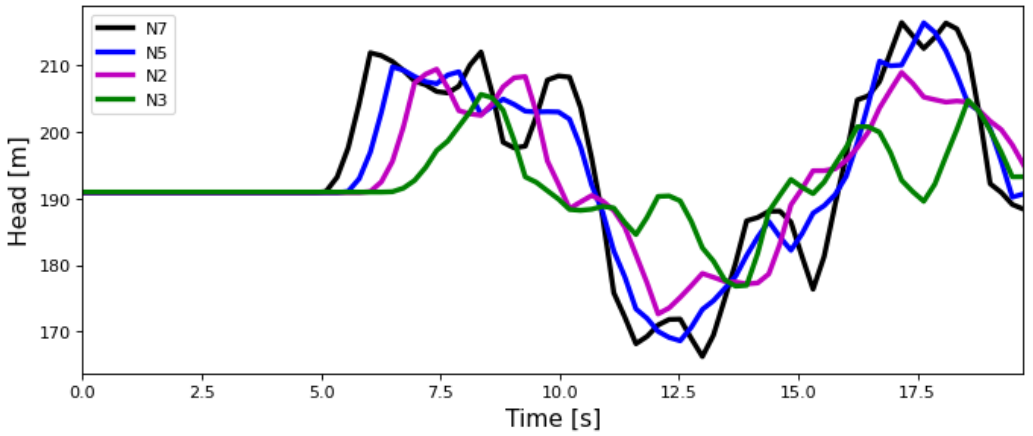


Scenario 1: Getting started

- Get results – heads

```
48  # Plot head at nodes
49  #-----------------------------------------------
50  node = 'N7'
51  node = tm.get_node(node)
52  fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
53  plt.plot(tm.simulation_timestamps,node.head, 'k', lw=3, label = 'N7')
54  node = 'N5'
55  node = tm.get_node(node)
56  plt.plot(tm.simulation_timestamps,node.head, 'b', lw=3, label = 'N5')
57  node = 'N2'
58  node = tm.get_node(node)
59  plt.plot(tm.simulation_timestamps,node.head, 'm', lw=3, label = 'N2')
60  node = 'N3'
61  node = tm.get_node(node)
62  plt.plot(tm.simulation_timestamps,node.head, 'g', lw=3, label = 'N3')
63  plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
64  #plt.title('Pressure Head at Node %s '%node)
65  plt.xlabel("Time [s]", fontsize=14)
66  plt.ylabel("Pressure Head [m]", fontsize=14)
67  plt.legend(loc='best')
```

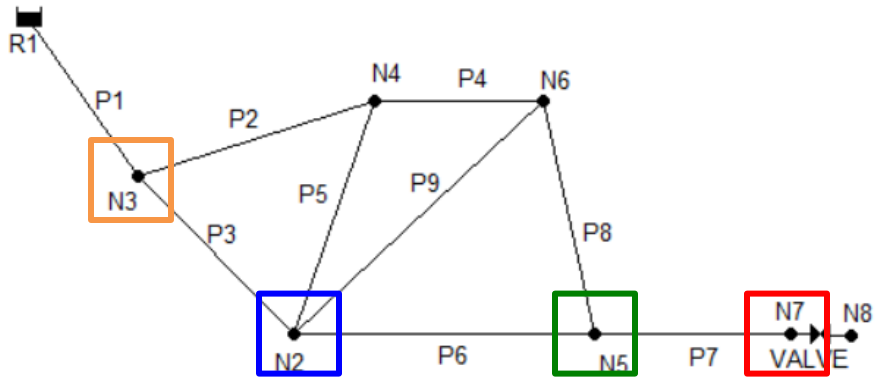


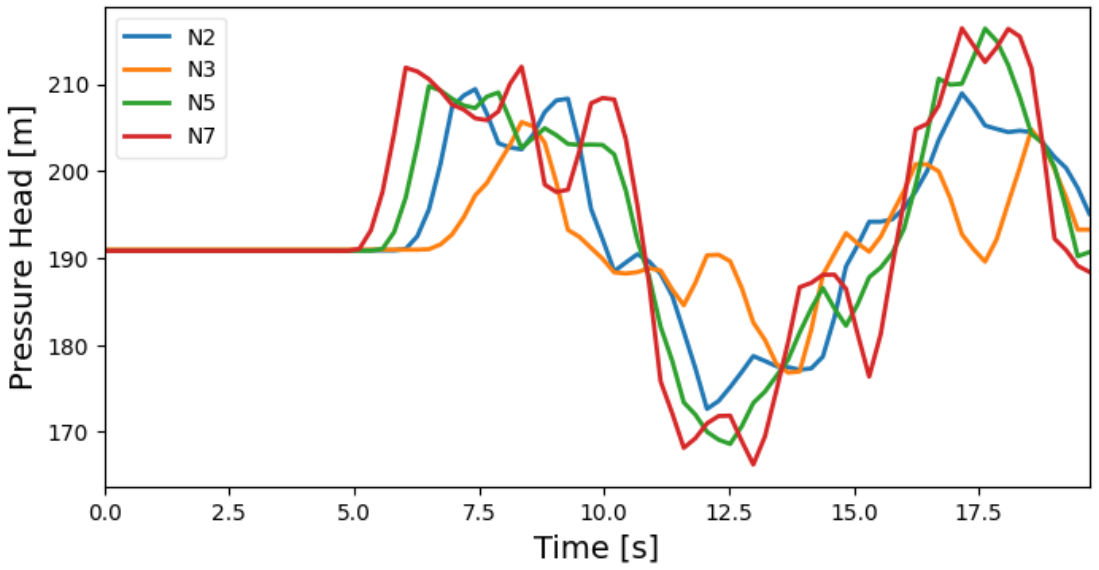


Scenario 1: Getting started

- Get results – heads

```
69  # build in function to plot heads
70  #-----------------------------------------------
71  tm.plot_node_head(['N2', 'N3', 'N5', 'N7'])
72
```

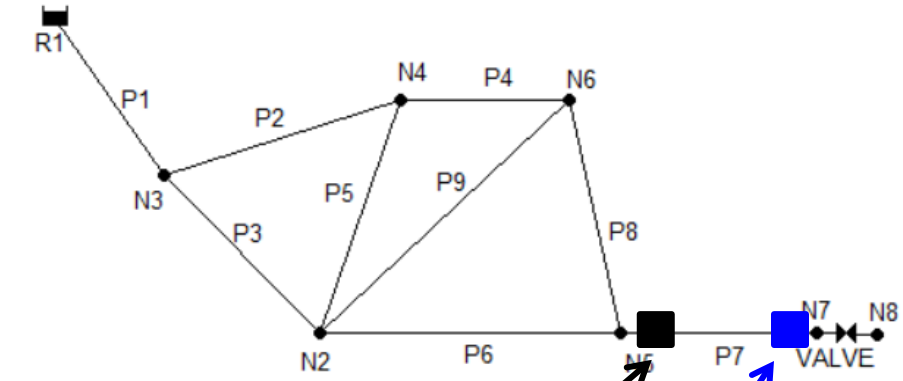


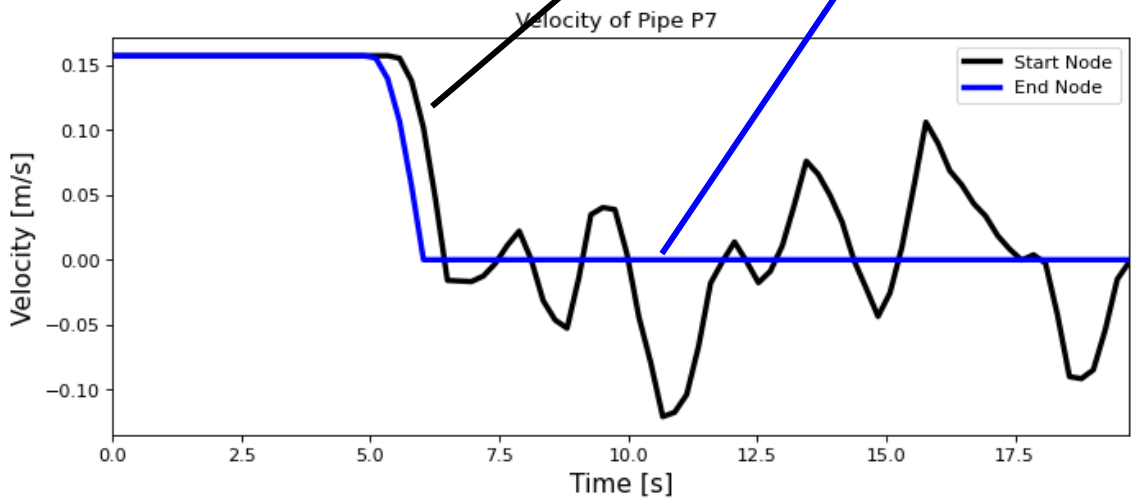


Scenario 1: Getting started

- Get results – velocities

```
73  # Plot velocity
74  #-----------------------------------------------------
75  pipe = 'P7'
76  pipe = tm.get_link(pipe)
77  fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
78  plt.plot(tm.simulation_timestamps,pipe.start_node_velocity,'k', lw=3, label='Start Node')
79  plt.plot(tm.simulation_timestamps,pipe.end_node_velocity, 'b', lw=3, label='End Node')
80  plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
81  plt.title('Velocity of Pipe %s '%pipe)
82  plt.xlabel("Time [s]", fontsize=14)
83  plt.ylabel("Velocity [m/s]", fontsize=14)
84  plt.legend(loc='best')
```

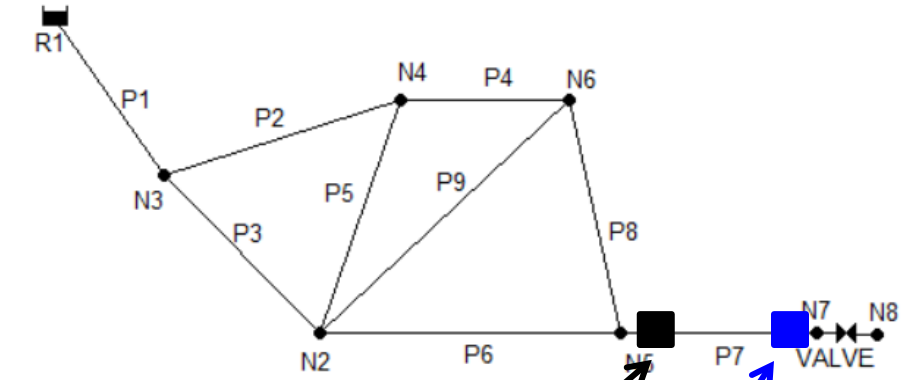


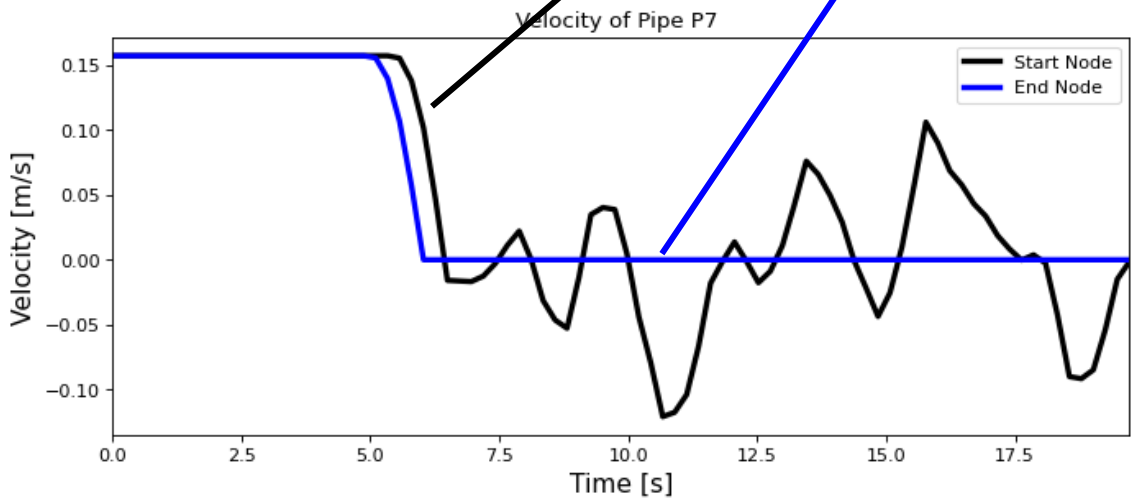


Scenario 1: Getting started

- Get results – velocities

```
73  # Plot velocity
74  #-----------------------------------------------
75  pipe = 'P7'
76  pipe = tm.get_link(pipe)
77  fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
78  plt.plot(tm.simulation_timestamps,pipe.start_node_velocity,'k', lw=3, label='Start Node')
79  plt.plot(tm.simulation_timestamps,pipe.end_node_velocity, 'b', lw=3, label='End Node')
80  plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
81  plt.title('Velocity of Pipe %s '%pipe)
82  plt.xlabel("Time [s]", fontsize=14)
83  plt.ylabel("Velocity [m/s]", fontsize=14)
84  plt.legend(loc='best')
```





Live Demo

Scenario 1: Getting started

- Open Spyder or Jupyter Notebooks: *1_TNet1_workshop_script_getting_started*

jupyter 1_TNet1_workshop_script_getting_started Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

Run Markdown

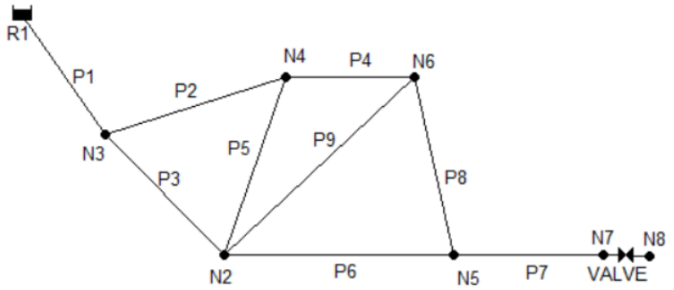
WDSA-CCWI 2022

TSNet Short Tutorial

Example 1: TNet1 - getting started

We start with using a simple network model, TSNet1, to demonstrate how to:

- Import tsnet
- Generate a transient model
- Set wave speed
- Set time step and simulation period
- Perform initial condition calculation
- Create transient event
- Run transient simulation and save results to .obj file
- Plot simulation results



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

...ith Lu\working files\papers\WDSA_CCWI_2022_workshop\codes\1_TNet1_workshop_script_getting_started.py

1_TNet1_workshop_script_getting_started.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri May 20 16:58:38 2022
4
5  @author: ps28866
6  """
7  import tsnet
8
9  import matplotlib.pyplot as plt
10 plt.close('all')
11
12 #%% part 1: Getting started
13
14 # inp source file
15 #------------------------------------------------
16 inp_file = 'Tnet1.inp'
17
18 # Open an example network and create a transient model
19 #------------------------------------------------
20 tm = tsnet.network.TransientModel('networks/' + inp_file)
21
22 # Set wavespeed
23 #------------------------------------------------
24 tm.set_wavespeed(1200.) # m/s
25
26 # Set time options
27 #------------------------------------------------
28 tf = 20   # simulation period [s]
29 tm.set_time(tf)
30
31 # Set valve closure
32 #------------------------------------------------
33 ts = 5 # valve closure start time [s]
34 tc = 1 # valve closure period [s]
35 se = 0 # end open percentage [s]
36 m = 2 # closure constant [dimensionless]
37 tm.valve_closure('VALVE',[tc,ts,se,m])
```

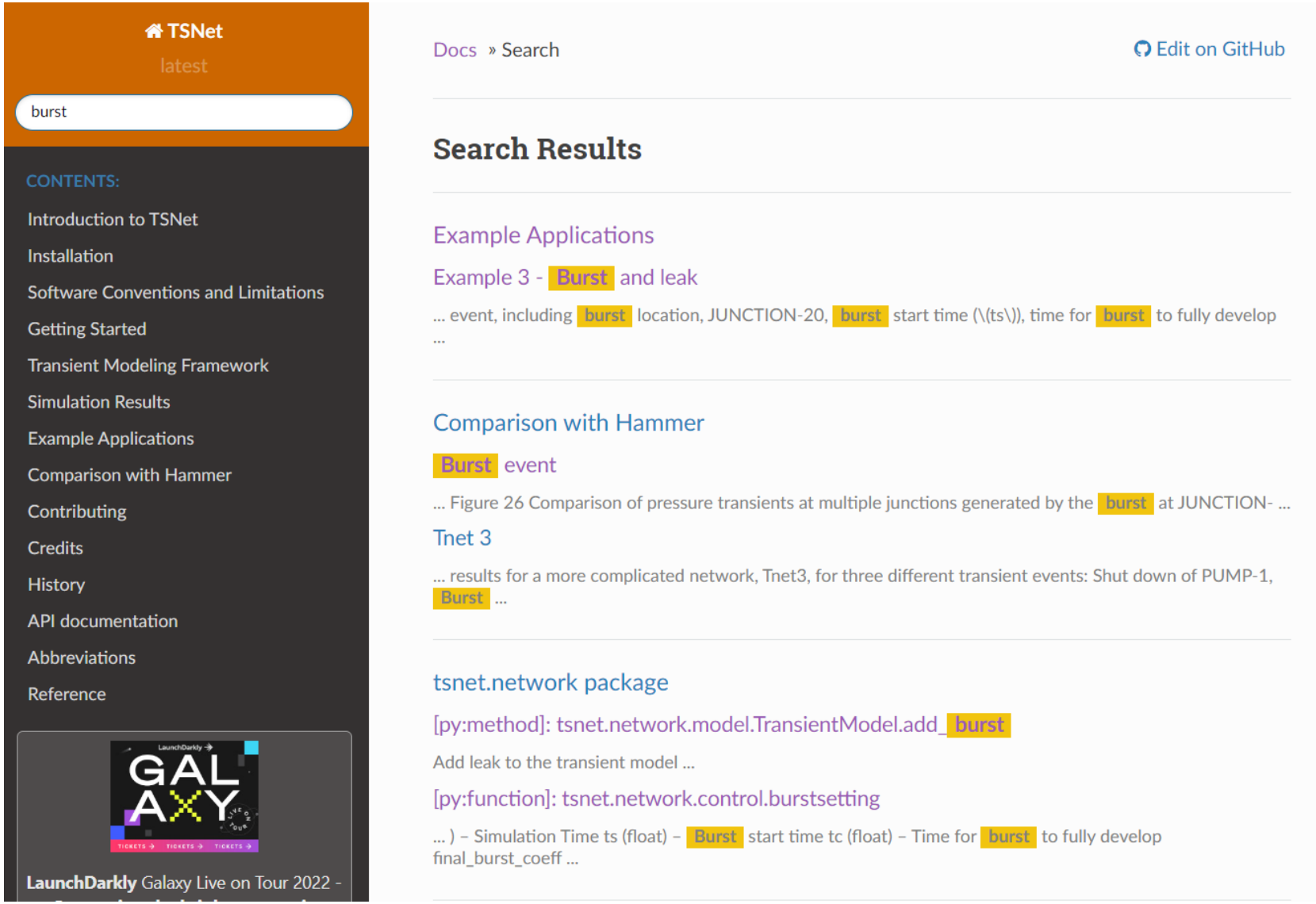
Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

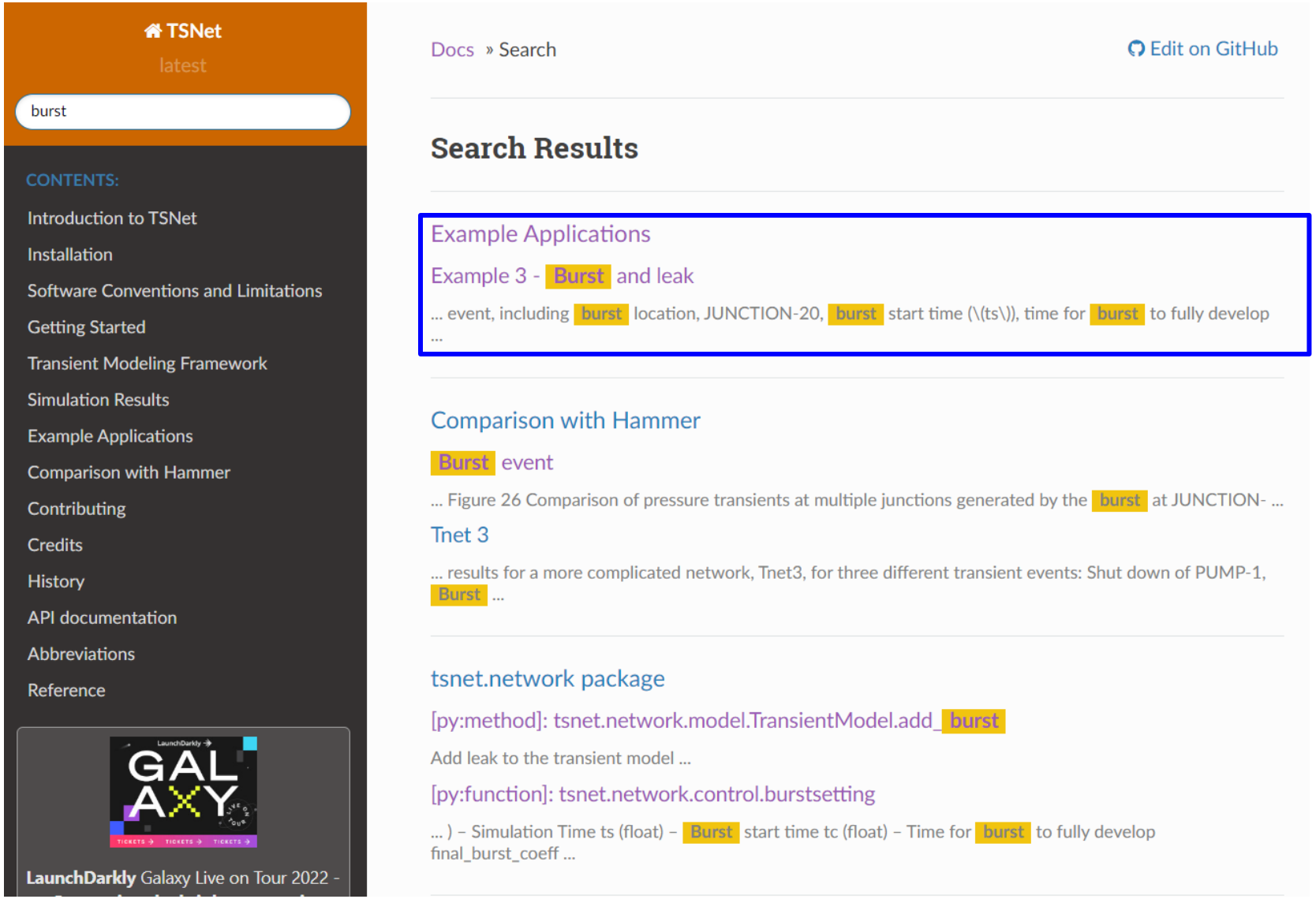
Online documentation

- Example – burst



Online documentation

- Example – burst



Online documentation

Example 3 - Burst and leak

This example reveals how TSNet simulates pipe **burst**s and leaks. This example network, adapted from [OSBH08], is shown below in Figure 18. Tnet3 comprises 168 pipes, 126 junctions, 8 valve, 2 pumps, one reservoir, and two tanks. The transient event is generated by a burst and a background leak. There are five steps that the user would need to take:

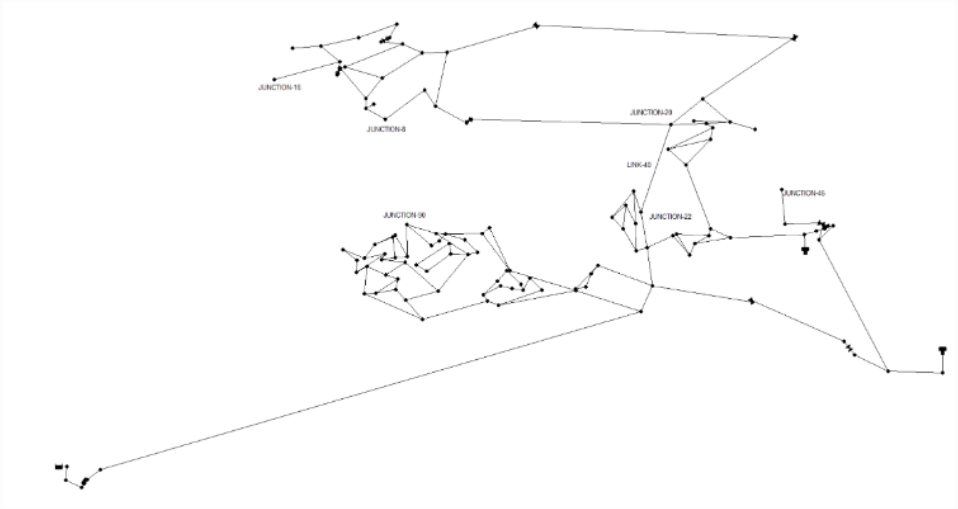
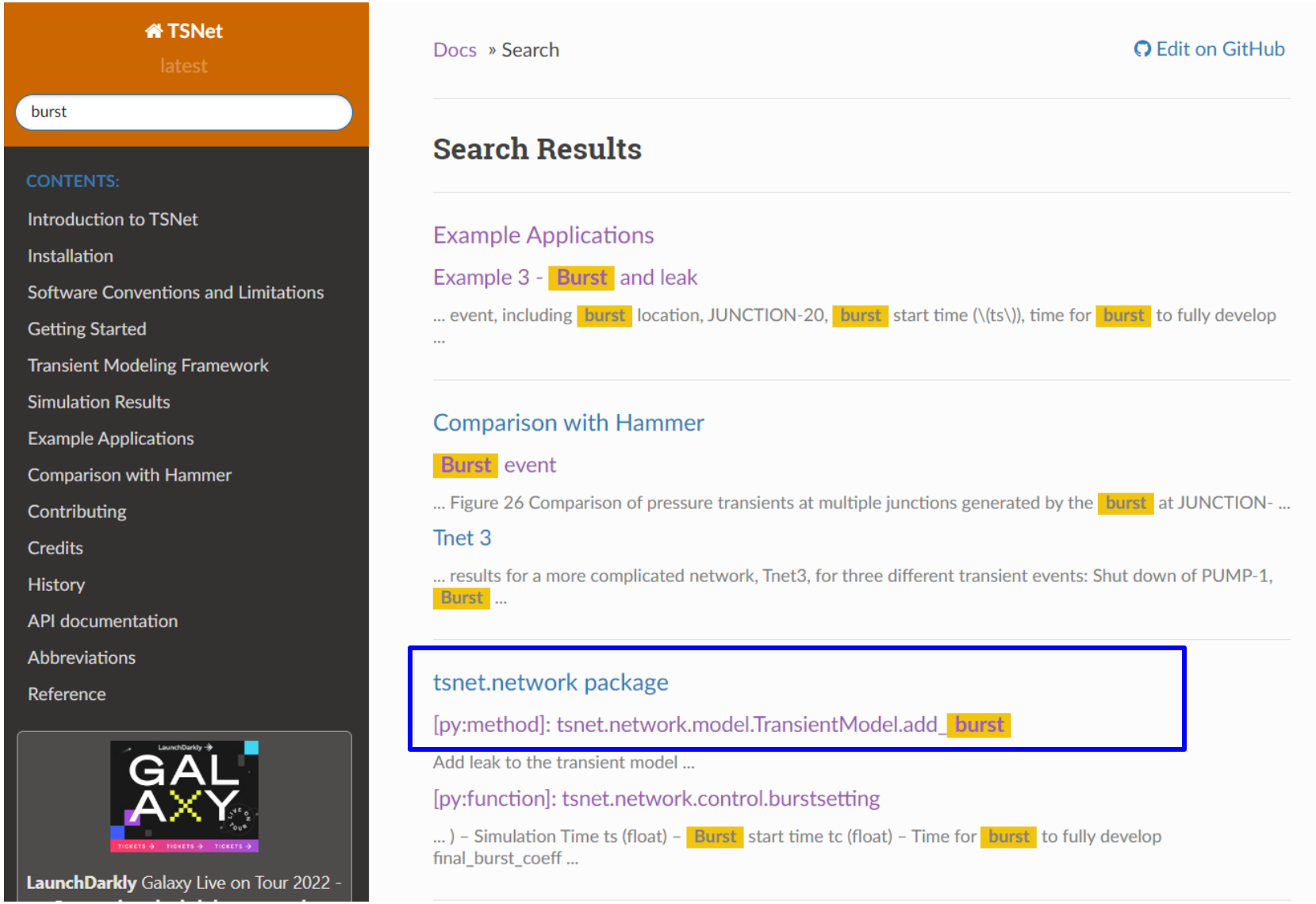


Figure 18 Tnet3 network graphics

1. Import TSNet package, read the EPANET INP file, and create transient model object.

```
import tsnet
# open an example network and create a transient model
inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)
```

Online documentation



Online documentation

- Example – burst

add_burst(*name*, *ts*, *tc*, *final_burst_coeff*) [source]

Add leak to the transient model

Parameters:

- **name** (*str*) – The name of the leak nodes, by default None
- **ts** (*float*) – **Burst** start time
- **tc** (*float*) – Time for **burst** to fully develop
- **final_burst_coeff** (*list or float*) – Final emitter coefficient at the **burst** nodes

Online documentation

TSNet Documentation

Release 0.2.2

Working with TSNet

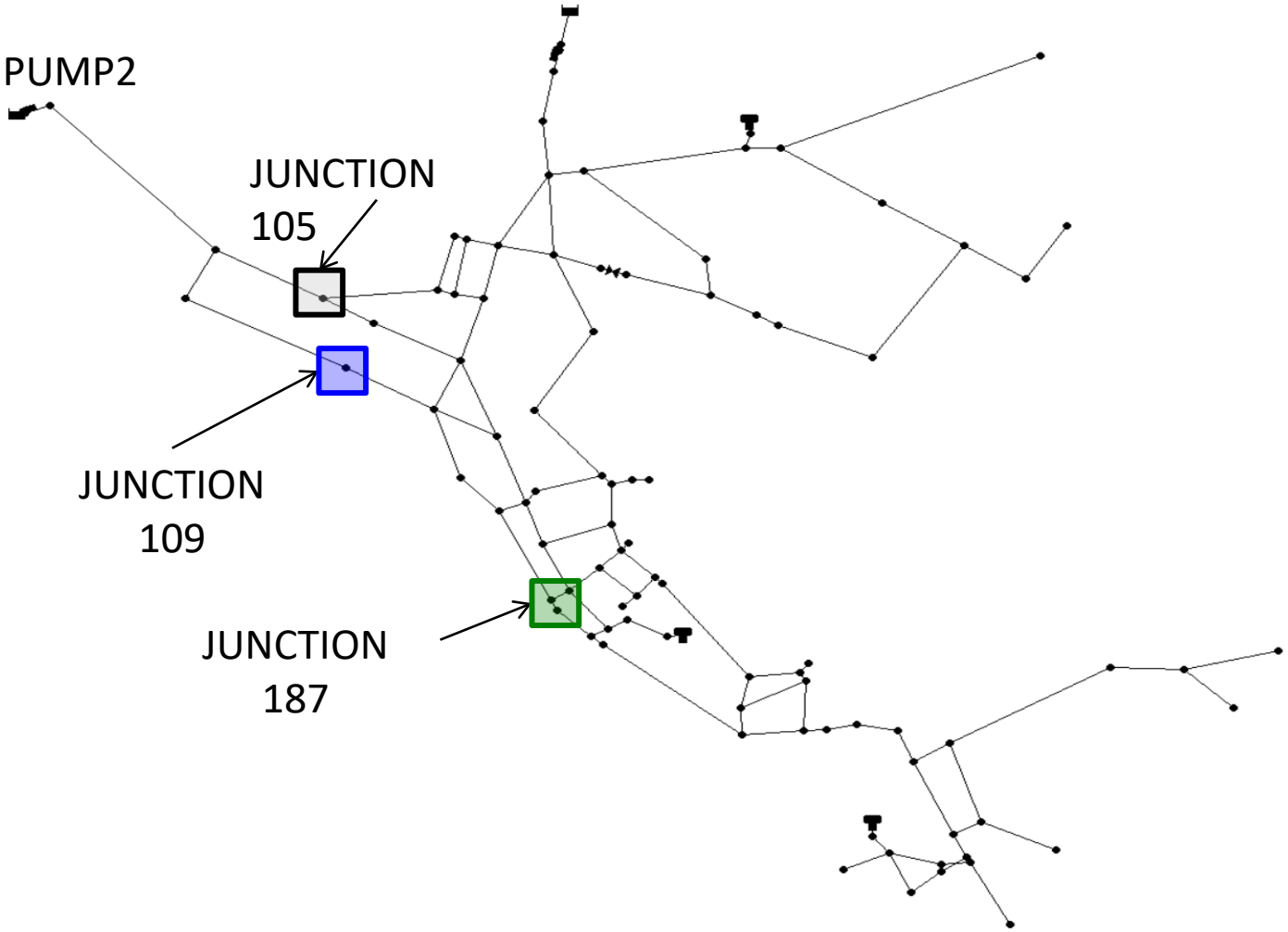
Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Scenario 2: Pump shutoff

In this example, we will use Tnet2 to demonstrate how to simulate pump shutoff:

- Close PUMP2
- Plot results



Demonstrate:

- Demand-driven vs. pressure driven
- Save results

Scenario 2: Pump shutoff

- Spyder or Jupyter Notebooks: *2_TNet2_pump_shutoff*

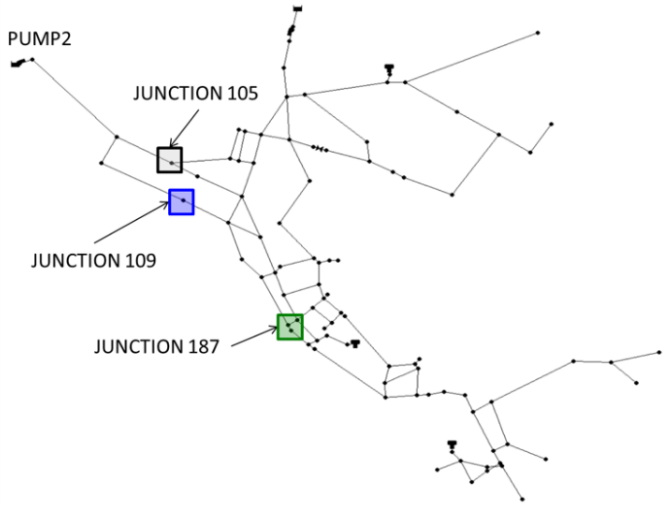
WDSA-CCWI 2022

TSNet Short Tutorial

Example 2: TNet2 - pump shutoff

In this example we will use Tnet2 to demonstrate how to simulate:

- Pump shutoff



Import packages

In [3]:
```
import tsnet
import matplotlib.pyplot as plt
```

Create the model, define settings, and execute simuation

We will generate a trasient event by closing PUMP2

In [4]:
```
inp_file = 'Tnet2.inp'

# Open an example network and create a transient model
#-------------------------------------------------
tm = tsnet.network.TransientModel('networks/' + inp_file)
```

C:\Users\ps28866

... with Lu\working files\papers\WDSA_CCWI_2022_workshop\WDSA_CCWI_shared\codes\2_TNet2_pump_shutoff.py

1_MAGNets_KY2_example.py | ATI.py | EXO.py | Shape_Analyze_V02.py | 2_TNet2_pump_shutoff.py

```
import tsnet

import matplotlib.pyplot as plt
plt.close('all')

#%% TNet2: Pump shotoff event
#-------------------------------------------------
inp_file = 'Tnet2.inp'

# Open an example network and create a transient model
#-------------------------------------------------
tm = tsnet.network.TransientModel('networks/' + inp_file)

# Set wavespeed
#-------------------------------------------------
tm.set_wavespeed(1200.) # [m/s]

# Set time
#-------------------------------------------------
tf = 20 # simulation duration[s]
tm.set_time(tf)

# Set pump shut off
#-------------------------------------------------
tc = 1 # pump closure period
ts = 1 # pump closure start time
se = 0 # end open percentage
m = 1 # closure constant
pump_op = [tc,ts,se,m]
tm.pump_shut_off('PUMP2', pump_op)

# Initialize steady state simulation
#-------------------------------------------------
t0 = 0. # initialize the simulation at 0s
engine = 'DD' # or PDD
tm = tsnet.simulation.Initializer(tm, t0, engine)

# Transient simulation
#-------------------------------------------------
results_obj = 'Tnet2_pump' # name of the object for saving simulation results.head

tm = tsnet.simulation.MOCSimulator(tm,results_obj)
```

Scenario 2: Pump shutoff

```
#%% Tnet2: Pump shutoff event
#-------------------------------------------------
inp_file = 'Tnet2.inp'

# Open an example network and create a transient model
#-------------------------------------------------
tm = tsnet.network.TransientModel('networks/' + inp_file)

# Set wavespeed
#-------------------------------------------------
tm.set_wavespeed(1200.) # [m/s]

# Set time
#-------------------------------------------------
tf = 20 # simulation duration[s]
tm.set_time(tf)

# Set pump shut off
#-------------------------------------------------
tc = 1 # pump closure period
ts = 1 # pump closure start time
se = 0 # end open percentage
m = 1 # closure constant
pump_op = [tc,ts,se,m]
tm.pump_shut_off('PUMP2', pump_op)

# Initialize steady state simulation
#-------------------------------------------------
t0 = 0. # initialize the simulation at 0s
engine = 'DD' # or PDD
tm = tsnet.simulation.Initializer(tm, t0, engine)

# Transient simulation
#-------------------------------------------------
results_obj = 'Tnet2_pump' # name of the object for saving simulation results.head

tm = tsnet.simulation.MOCSimulator(tm,results_obj)
```

- Import TSNet Package
- Specify .inp file
- Create tsnet model
- Specify wave speeds
- Define simulation duration

Define transient event: duration; start; end; shape

Initialize solver: DD or PDD

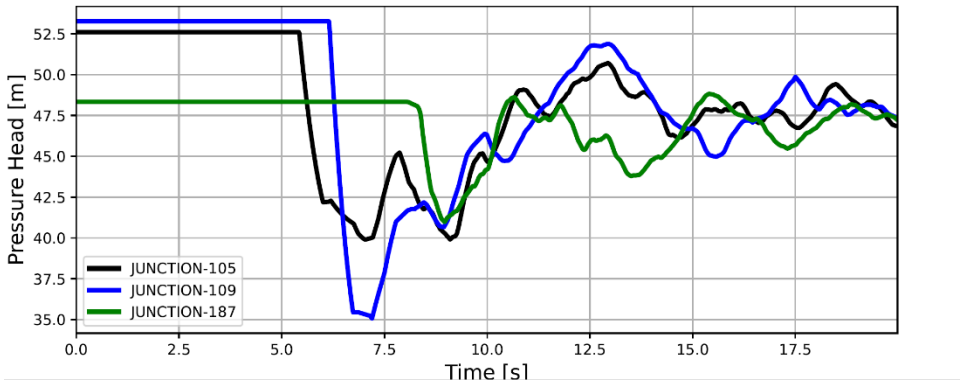
- Run simulation
- Save results

Scenario 2: Pump shutoff

- Calculate initial conditions based on:

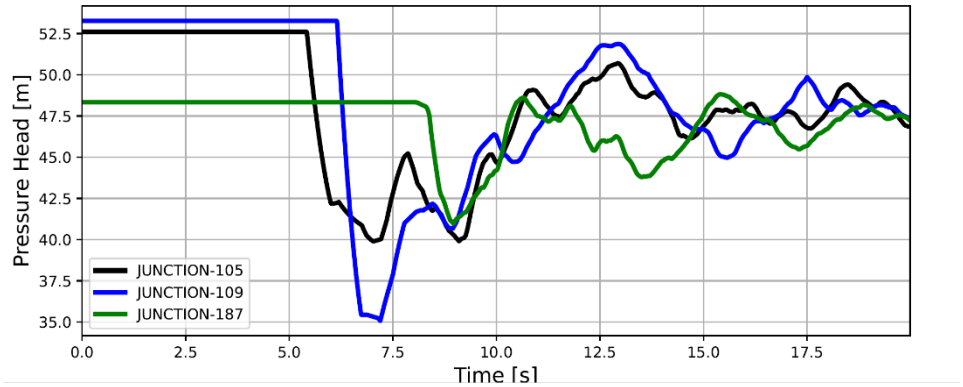
  - Pressure driven (WNTR solver)

```
31
32  # Initialize steady state simulation
33  #-------------------------------------------------
34  t0 = 0. # initialize the simulation at 0s
35  engine = 'PDD' # or PDD
36  tm = tsnet.simulation.Initializer(tm, t0, engine)
```



  - Demand driven (EPANET solver)

```
32  # Initialize steady state simulation
33  #-------------------------------------------------
34  t0 = 0. # initialize the simulation at 0s
35  engine = 'DD' # or PDD
36  tm = tsnet.simulation.Initializer(tm, t0, engine)
37
```

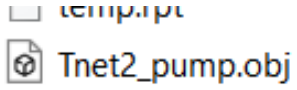


Scenario 2: Pump shutoff

- Store simulation results as object

```
38  # Transient simulation
39  #------------------------------------------------
40  results_obj = 'Tnet2_pump' # name of the object for saving simulation results.head
41
42  tm = tsnet.simulation.MOCSimulator(tm,results_obj)
```

- You should see a new object ‘Tnet2_pump.obj’ created in your working directory



- We will later see how to import these results

Scenario 2: Pump shutoff

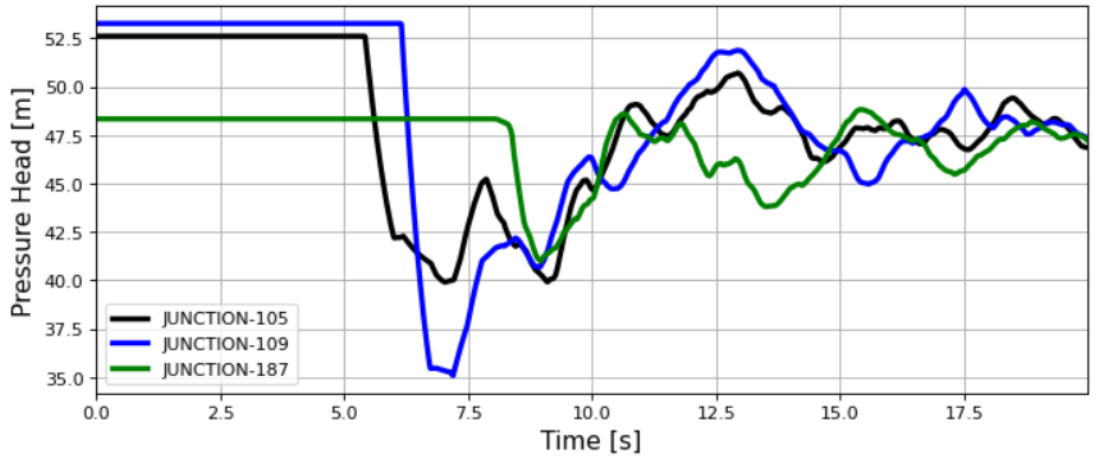
- Simulation progress:

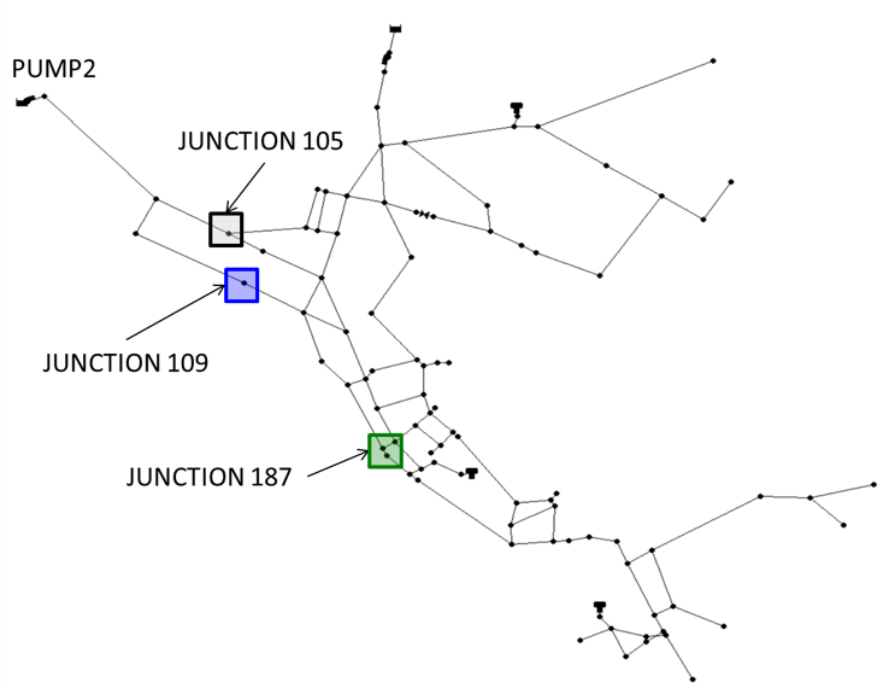
```
Simulation time step 0.01351 s
Total Time Step in this simulation 1480
Estimated simulation time 0:02:19.624680
Transient simulation completed 10 %...
Transient simulation completed 20 %...
Transient simulation completed 30 %...
Transient simulation completed 40 %...
Transient simulation completed 50 %...
Transient simulation completed 60 %...
Transient simulation completed 70 %...
Transient simulation completed 80 %...
Transient simulation completed 90 %...
```

Scenario 2: Pump shutoff

- Plot results: JUNCTIONS 105, 109, 187 and save as pdf

```
node = 'JUNCTION-105'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
plt.plot(tm.simulation_timestamps,node.head, 'k', lw=3,label = 'JUNCTION-105')
node = '109'
node = tm.get_node(node)
plt.plot(tm.simulation_timestamps,node.head, 'b', lw=3,label = 'JUNCTION-109')
node = '187'
node = tm.get_node(node)
plt.plot(tm.simulation_timestamps,node.head, 'g', lw=3,label = 'JUNCTION-187')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Pressure Head [m]", fontsize=14)
plt.legend(loc='best')
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet2_pump_shutoff_head.pdf', format='pdf',dpi=100)
```





Working with TSNet

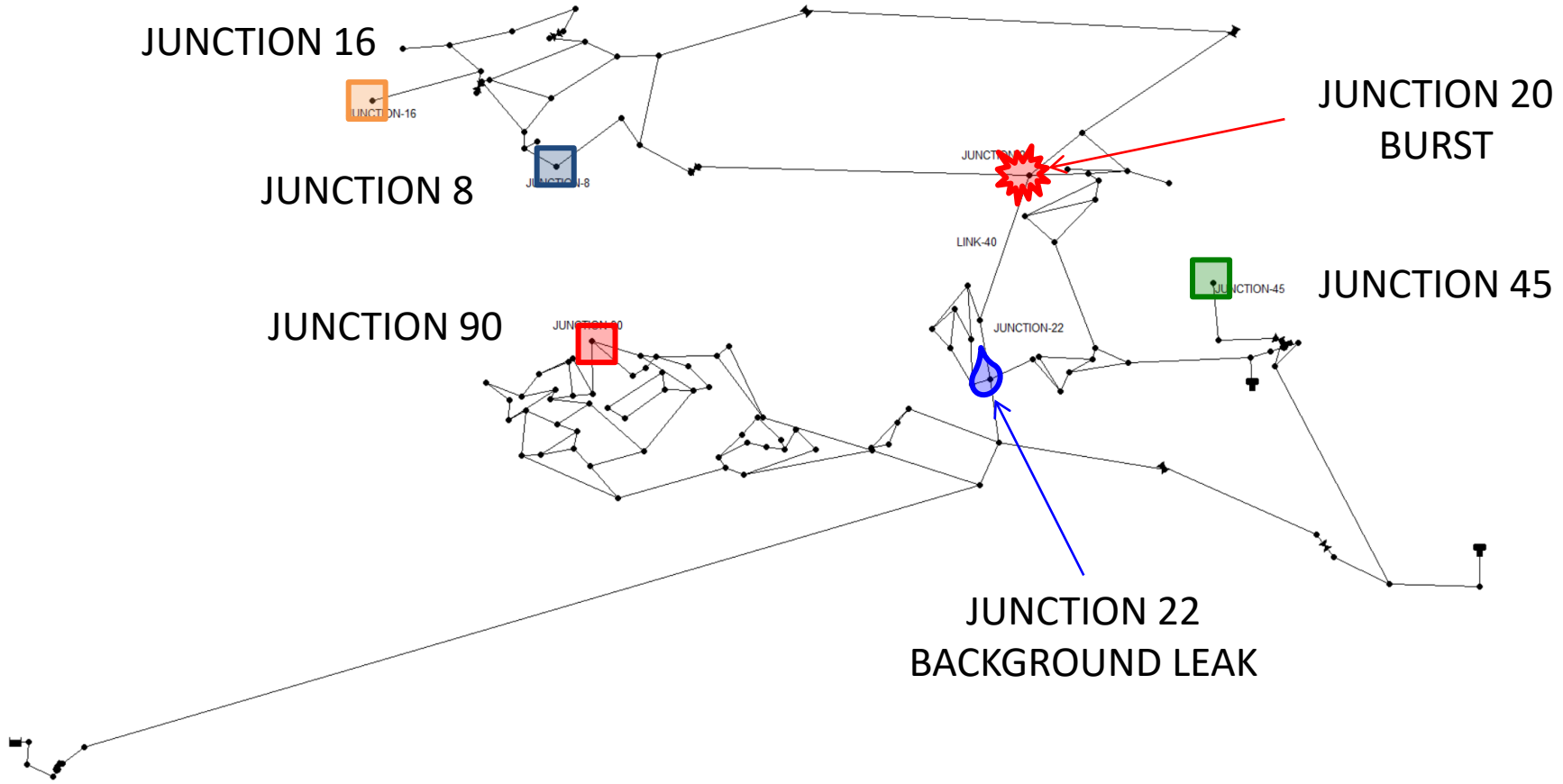
Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Scenario 3: Burst with background leak

In this example, we will use Tnet3 to demonstrate how to simulate burst event with a background leak:

- Generate a burst at Junction-20
- Existing leak at Junction-22



Scenario 3: Burst with background leak

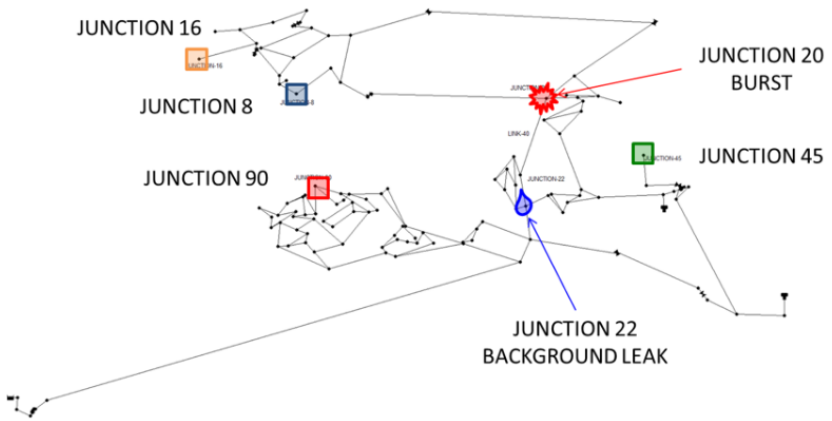
- Spyder or Jupyter Notebooks: *3_TNet3_burst_leak*

WDSA-CCWI 2022

TSNet Short Tutorial

Example 3: TNet3 - burst with background leak

In this example we will use Tnet3 to demonstrate how to simulate burst event with a background leak. We will generate a burst at Junction-20 with an existing background leak at Junction-22.



Import packages

```
import tsnet
import matplotlib.pyplot as plt
import numpy as np
```

Create the model, define settings, and execute simuation

```
#%% TNet3: Burst event with a background leak
#------------------------------------------------

inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#------------------------------------------------
wavespeed = np.random.normal(1200., 100., size=tm.num_pipes)
tm.set_wavespeed(wavespeed)
# Set time step
tf = 20 # simulation period [s]
```

2_TNet2_pump_shutoff.py | 3_TNet3_burst_leak.py | 1_TNet1_getting_started.py

```
# %%
"""
Created on Mon May 23 13:17:36 2022

@author: ps28866
"""
import tsnet
import matplotlib.pyplot as plt
import numpy as np

plt.close('all')

# %% TNet3: Burst event with a background leak
#------------------------------------------------
inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#------------------------------------------------
wavespeed = np.random.normal(1200., 100., size=tm.num_pipes)
tm.set_wavespeed(wavespeed)

# Set time
#------------------------------------------------
tf = 20 # simulation period [s]
tm.set_time(tf)

# Add leak
#------------------------------------------------
emitter_coeff = 0.01 # [ m^3/s/(m H20)^(1/2)]
tm.add_leak('JUNCTION-22', emitter_coeff)

# Add burst
#------------------------------------------------
ts = 1 # burst start time
tc = 1 # time for burst to fully develop
final_burst_coeff = 0.01 # final burst coeff [ m^3/s/(m H20)^(1/2)]
tm.add_burst('JUNCTION-20', ts, tc, final_burst_coeff)

# Initialize steady state simulation
#------------------------------------------------
t0 = 0. # initialize the simulation at 0s
engine = 'PDD'
tm = tsnet.simulation.Initializer(tm, t0, engine)
```

Scenario 3: Burst with background leak

```
14  # %% TNet3: Burst event with a background leak
15  #------------------------------------------------
16  inp_file = 'networks/Tnet3.inp'
17  tm = tsnet.network.TransientModel(inp_file)
18
19  # Set wavespeed
20  #------------------------------------------------
21  wavespeed = np.random.normal(1200., 100., size=tm.num_pipes)
22  tm.set_wavespeed(wavespeed)
23
24  # Set time
25  #------------------------------------------------
26  tf = 20 # simulation period [s]
27  tm.set_time(tf)
28
29  # Add leak
30  #------------------------------------------------
31  emitter_coeff = 0.01 # [ m^3/s/(m H20)^(1/2)]
32  tm.add_leak('JUNCTION-22', emitter_coeff)
33
34  # Add burst
35  #------------------------------------------------
36  ts = 1 # burst start time
37  tc = 1 # time for burst to fully develop
38  final_burst_coeff = 0.01 # final burst coeff [ m^3/s/(m H20)^(1/2)]
39  tm.add_burst('JUNCTION-20', ts, tc, final_burst_coeff)
40
41  # Initialize steady state simulation
42  #------------------------------------------------
43  t0 = 0. # initialize the simulation at 0s
44  engine = 'PDD'
45  tm = tsnet.simulation.Initializer(tm, t0, engine)
46
47  # Transient simulation
48  #------------------------------------------------
49  result_obj = 'Tnet3_burst_with_leak' # name of the object for saving simulation results
50  tm = tsnet.simulation.MOCSimulator(tm,result_obj)
```

Specify wave speeds:
Different for different pipes

Add leak:
location; emitter coefficient

Add burst:
location; start time; duration;
final burst coefficient

Initialize solver: PDD

Scenario 3: Burst with background leak

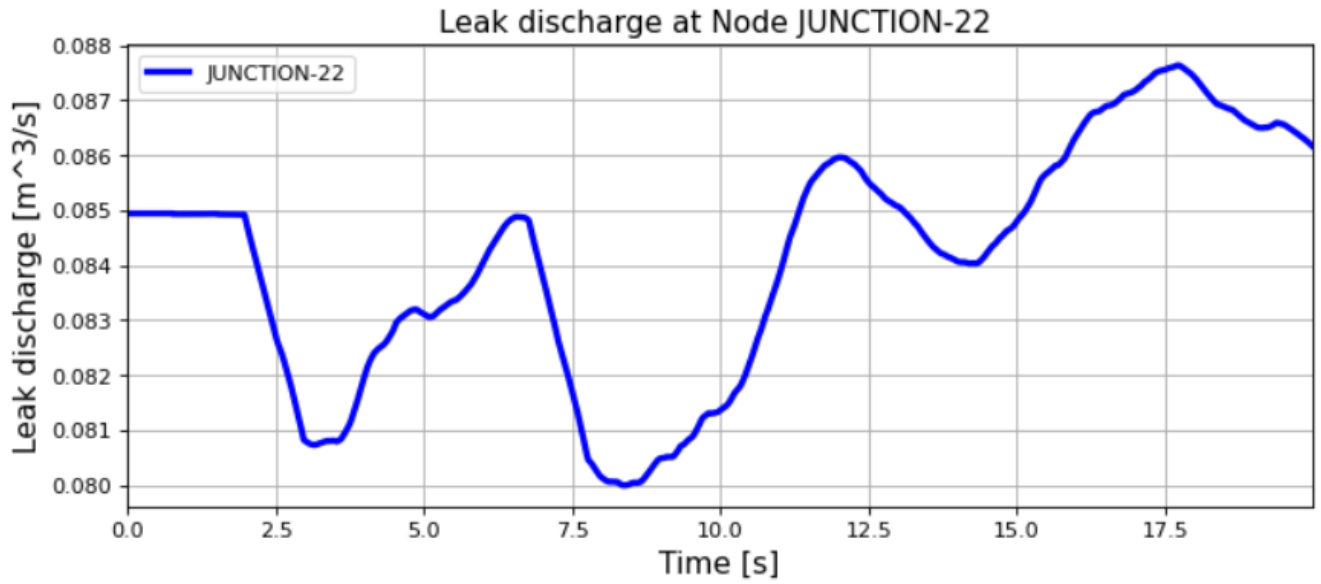
- Simulation progress:

```
Simulation time step 0.01322 s
Total Time Step in this simulation 1513
Estimated simulation time 0:01:23.886772
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
```

Scenario 3: Burst with background leak

- Leak discharge JUNCTIONS 22

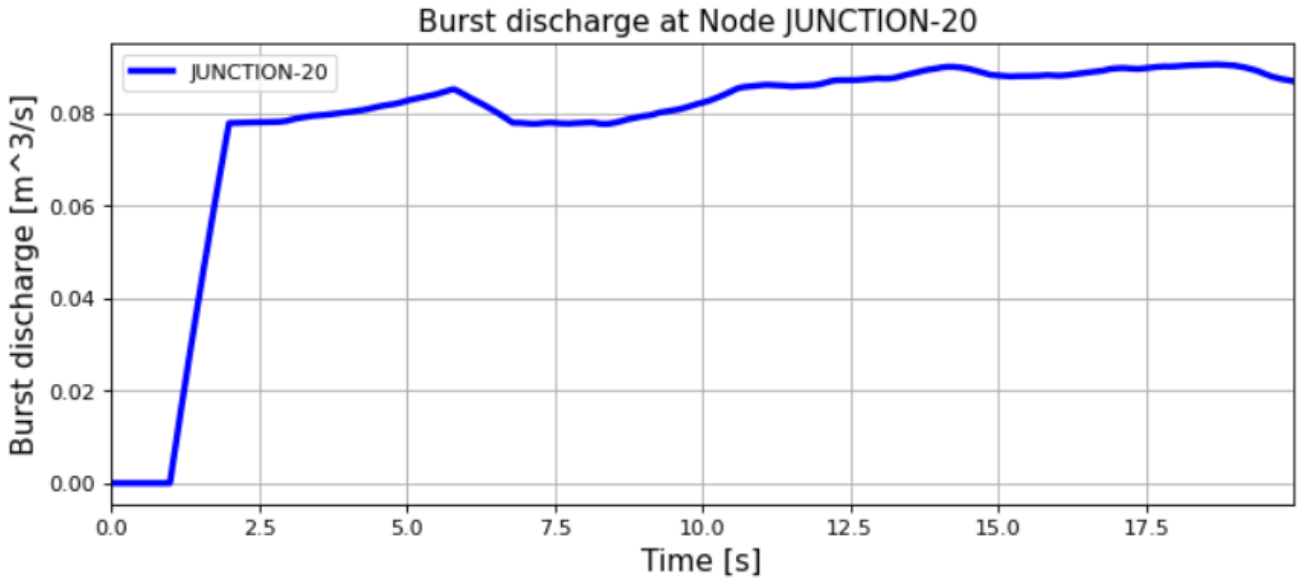
```
# Junction-22 leak discharge
#-------------------------------------------------
node = 'JUNCTION-22'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.emitter_discharge, 'b', lw=3,label = 'JUNCTION-22')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Leak discharge at Node %s '%node, fontsize=14)
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Leak discharge [m^3/s]", fontsize=14)
plt.legend(loc='best')
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_burst_w_leak_J22_leak_discharge.pdf', format='pdf',dpi=100)
```



Scenario 3: Burst with background leak

- Burst discharge JUNCTIONS 20

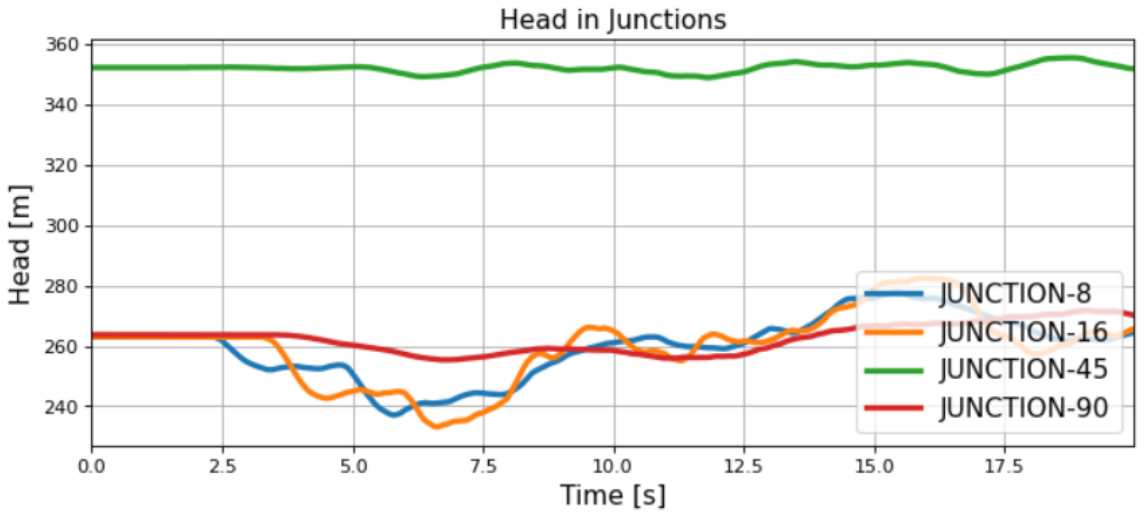
```
# Junction-20 burst discharge
#-------------------------------------------------
node = 'JUNCTION-20'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.emitter_discharge, 'b', lw=3, label = 'JUNCTION-20')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Burst discharge at Node %s '%node, fontsize=14)
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Burst discharge [m^3/s]", fontsize=14)
plt.legend(loc='best')
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_burst_w_leak_J20_burst_discharge.pdf', format='pdf',dpi=100)
```

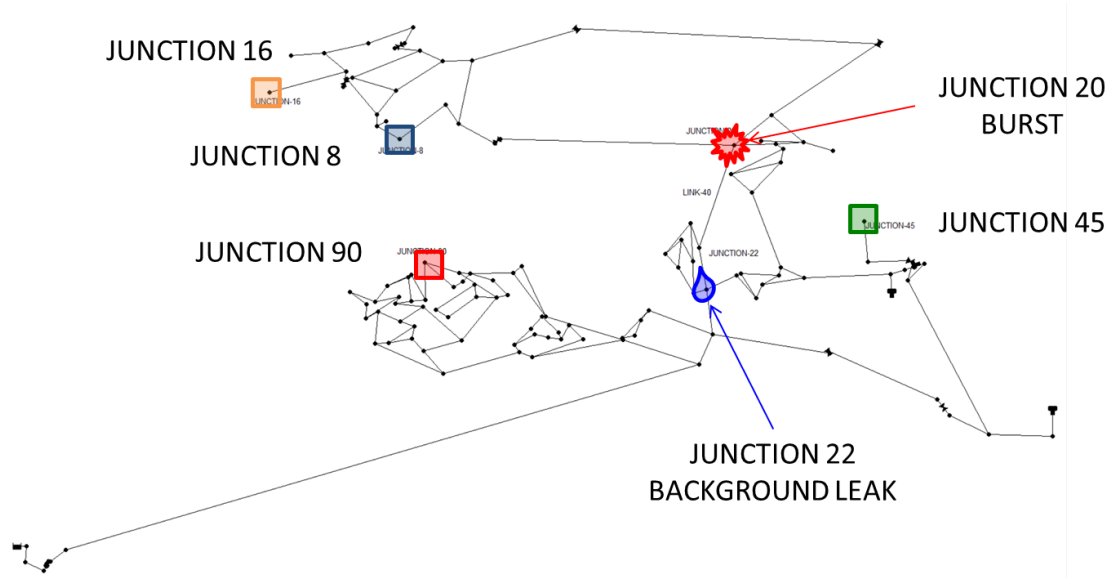


Scenario 3: Burst with background leak

- Head at different junctions in the network

```
# Head at different nodes in the network
#-------------------------------------------------
node1 = tm.get_node('JUNCTION-8')
node2 = tm.get_node('JUNCTION-16')
node3 = tm.get_node('JUNCTION-45')
node4 = tm.get_node('JUNCTION-90')
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node1.head, lw=3, label='JUNCTION-8')
plt.plot(tm.simulation_timestamps,node2.head, lw=3, label='JUNCTION-16')
plt.plot(tm.simulation_timestamps,node3.head, lw=3, label='JUNCTION-45')
plt.plot(tm.simulation_timestamps,node4.head, lw=3, label='JUNCTION-90')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Head in Junctions', fontsize=14)
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Head [m]", fontsize=14)
plt.legend(loc='lower right', fontsize=14)
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_burst_w_leak_head.pdf', format='pdf',dpi=100)
```



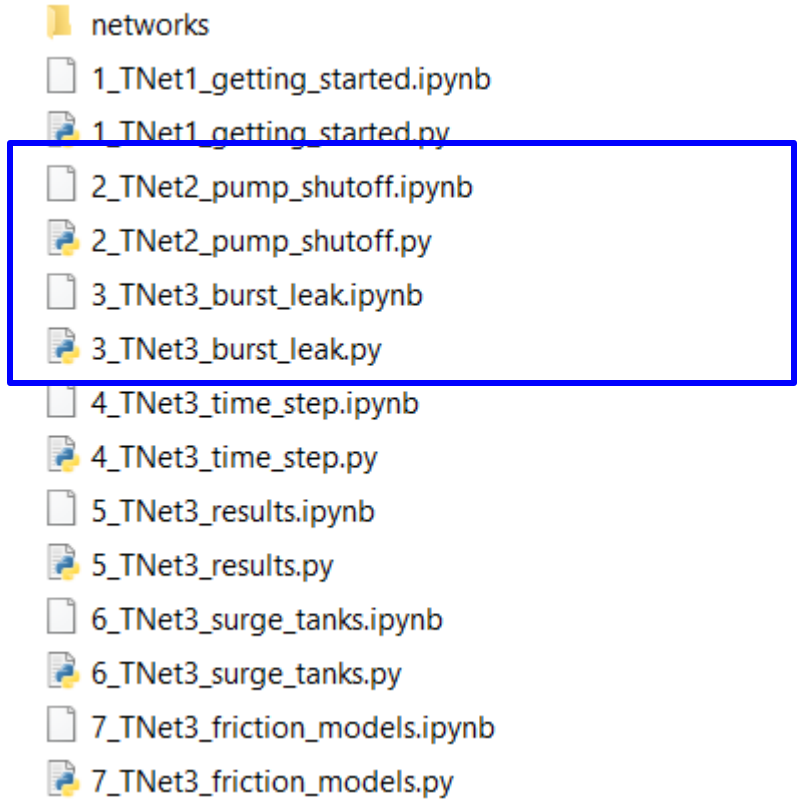


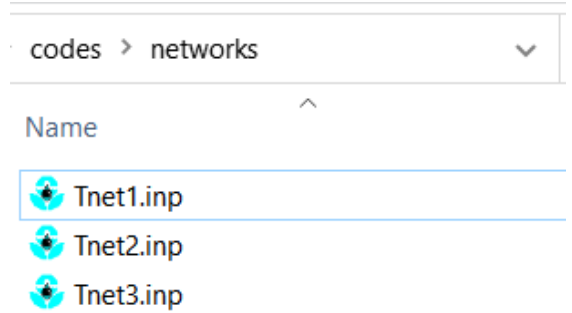
Live Demo

Scenario 3: Burst with background leak

Open Spyder or Jupyter Notebooks:

- *2_TNet2_pump_shutoff*
- *3_TNet3_burst_leak*





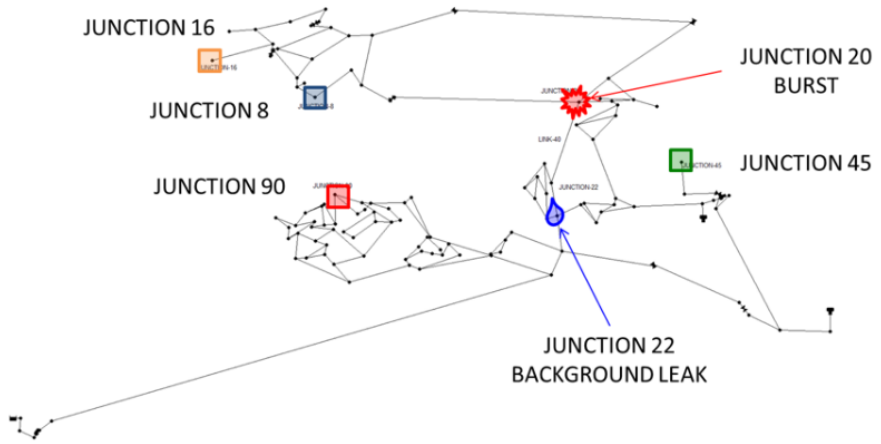
Scenario 3: Burst with background leak

WDSA-CCWI 2022

TSNet Short Tutorial

Example 3: TNet3 - burst with background leak

In this example we will use Tnet3 to demonstrate how to simulate burst event with a background leak. We will generate a burst at Junction-20 with an existing background leak at Junction-22.



Import packages

1]:
```
import tsnet
import matplotlib.pyplot as plt
import numpy as np
```

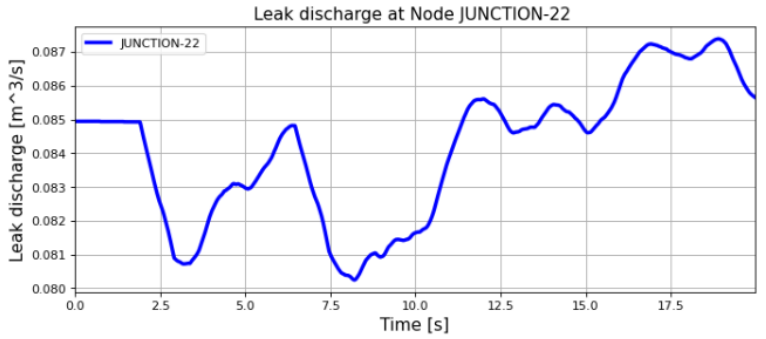
Create the model, define settings, and execute simuation

2]:
```
#%% TNet3: Burst event with a background leak
#------------------------------------------------

inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#------------------------------------------------
wavespeed = np.random.normal(1200., 100., size=tm.num_pipes)
tm.set_wavespeed(wavespeed)
# Set time step
tf = 20 # simulation period [s]
tm.set_time(tf)
```

Get results

- Junction-22 leak discharge

In [3]:
```
# Junction-22 leak discharge
#------------------------------------------------
node = 'JUNCTION-22'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.emitter_discharge, 'b', lw=3,label = 'JUNCTION-22')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Leak discharge at Node %s '%node, fontsize=14)
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Leak discharge [m^3/s]", fontsize=14)
plt.legend(loc='best')
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_burst_w_leak_J22_leak_discharge.pdf', format='pdf',dpi=100)
```



- Junction-20 burst discharge

In [4]:
```
# Junction-20 burst discharge
#------------------------------------------------
node = 'JUNCTION-20'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.emitter_discharge, 'b', lw=3, label = 'JUNCTION-20')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Burst discharge at Node %s '%node, fontsize=14)
plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Burst discharge [m^3/s]", fontsize=14)
plt.legend(loc='best')
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_burst_w_leak_J20_burst_discharge.pdf', format='pdf',dpi=100)
```

Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Selecting time step

Choice of Time Step

The determination of time step in MOC is not a trivial task. There are two constraints that have to be satisfied simultaneously:

1. The Courant's criterion has to be satisfied for each pipe, indicating the maximum time step allowed in the network transient analysis will be:

$$\Delta t \leqslant \min\left(\frac{L_i}{N_i a_i}\right), i = 1, 2, ..., n_p$$

2. The time step has to be the same for all the pipes in the network, therefore restricting the wave travel time $\frac{L_i}{N_i a_i}$ to be the same for any computational unit in the network. Nevertheless, this is not realistic in a real network, because different pipe lengths and wave speeds usually cause different wave travel times. Moreover, the number of sections in the i^{th} pipe (N_i) has to be an even integer due to the grid configuration in MOC; however, the combination of time step and pipe length is likely to produce non-integer value of N_i, which then requires further adjustment.

This package adopted the wave speed adjustment scheme [WYSS93] to make sure the two criterion stated above are satisfied.

Selecting time step

1. Get initial Δt_{max} based on the critical pipe with two segments
2. Get number of segments in each pipe, N_i
3. Find a global Δt that minimizes the sum of squares of wave speed adjustments

To begin with, the maximum allowed time step (Δt_{max}) is calculated, assuming that there are two computational units on the critical pipe (i.e., the pipe that results in the smallest travel time, which depends on the length and the wave speed for that pipe):

$$\Delta t_{max} = \min\left(\frac{L_i}{2a_i}\right), i = 1, 2, ..., n_p$$

After setting the initial time step, the following adjustments will be performed. Firstly, the i^{th} pipes (p_i) with length (L_i) and wave speed (a_i) will be discretized into (N_i) segments:

$$N_i = \text{round}\left(\frac{L_i}{a_i \Delta t_{max}}\right), i = 1, 2, \ldots, n_p$$

Furthermore, the discrepancies introduced by the rounding of N_i can be compensated by correcting the wave speed (a_i).

$$\Delta t = \text{argmin}_{\phi, \Delta t}\left\{\sum_{i=1}^{n_p} {\phi_i}^2 \mid \Delta t = \frac{L_i}{a_i(1 \pm \phi_i)N_i} \quad i = 1, 2, \ldots, n_p\right\}$$

Least squares approximation is then used to determine Δt such that the sum of squares of the wave speed adjustments ($\sum {\phi_i}^2$) is minimized [MISI08]. Ultimately, an adjusted Δt can be determined and then used in the transient simulation.

Selecting time step

Determination of time step:

- If the user does not provide initial guess for the time step

Δt_{max} *is used as initial guess*

- If the user provides initial guess Δt for the time step which is greater than calculated above

Δt *is used as initial guess*

- If the user provides initial guess for the time step which is less than calculated above

$TSNet$ *gives an error*

Selecting time step

Example

We use a small network, shown in Figure 2, to illustrate how time step is determined as well as the benefits and drawbacks of combining or removing small pipes. Figure 2 (a) shows a network of three pipes with length of 940m, 60m, and 2000m, respectively. The wave speed for all the pipes is equal to 1000m/s. The procedure for determine the time step is as follows:

- Calculate the maximum time step (Δt_{max}) allowed by Courant's criterion, assuming that there are two computational units on the critical pipe (i.e., the pipe that results in the smallest travel time, which depends on the length and the wave speed for that pipe), i.e., for pipe 2 $N_2 = 2$.}

$$\Delta t_{max} = \min\left(\frac{L_i}{2a_i}\right) = \left(\frac{L_2}{N_2 a_2}\right) = \frac{60}{2 \times 1000} = 0.03s$$

- Compute the required number of computational units for all other pipes, i.e, N_1 for pipe 1 and N_3 for pipe 3, using Δt_{max} as the time step. Since the number of computational units on each pipe has to be integer, the numbers are rounded to the closest integer, thus introducing discrepancies in the time step of different pipes.

$$N_1 = \text{round}\left(\frac{L_1}{a_1 \Delta t_{max}}\right) = \frac{940}{1000 \times 0.03} = 31$$
$$N_3 = \text{round}\left(\frac{L_3}{a_3 \Delta t_{max}}\right) = \frac{2000}{1000 \times 0.03} = 67$$

With these number of computational units, the time steps for each pipe become:

$$\Delta t_1 = \frac{L_1}{N_1 a_1} = 0.03032s$$
$$\Delta t_3 = \frac{L_3}{N_3 a_3} = 0.02985s$$

However, all the pipes have to have the same time step for marching forward; hence, we need to adjust the wave speed to match the time step for all pipes.

$$\Delta t = \frac{L_i}{a_i^{adj} N_i}$$

- Compensate the discrepancies introduced by rounding number of computation units through adjusting wave speed from a_i to $a_i^{adj} = a_i(1 + \phi_i)$. The sum of squared adjustments ($\sum {\phi_i}^2$) is minimized to obtain the minimal overall adjustment. In this example, the wave speeds of the three pipes are adjusted by $\phi_1 = 0.877\, \phi_2 = -0.196\%, \phi_3 = 0.693\%$, respectively.
- Finally, the time step can be calculated based on the number of computational units and the adjusted wave speed of each one of three pipes that now share the same time step:

$$\Delta t = \frac{L_i}{a_i(1 \pm \phi_i)N_i} = 0.03006s$$

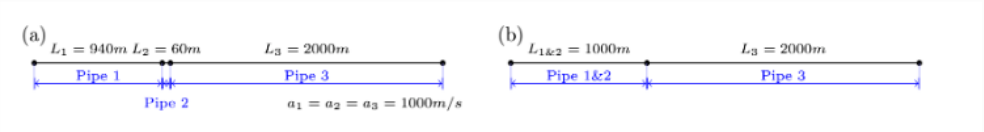
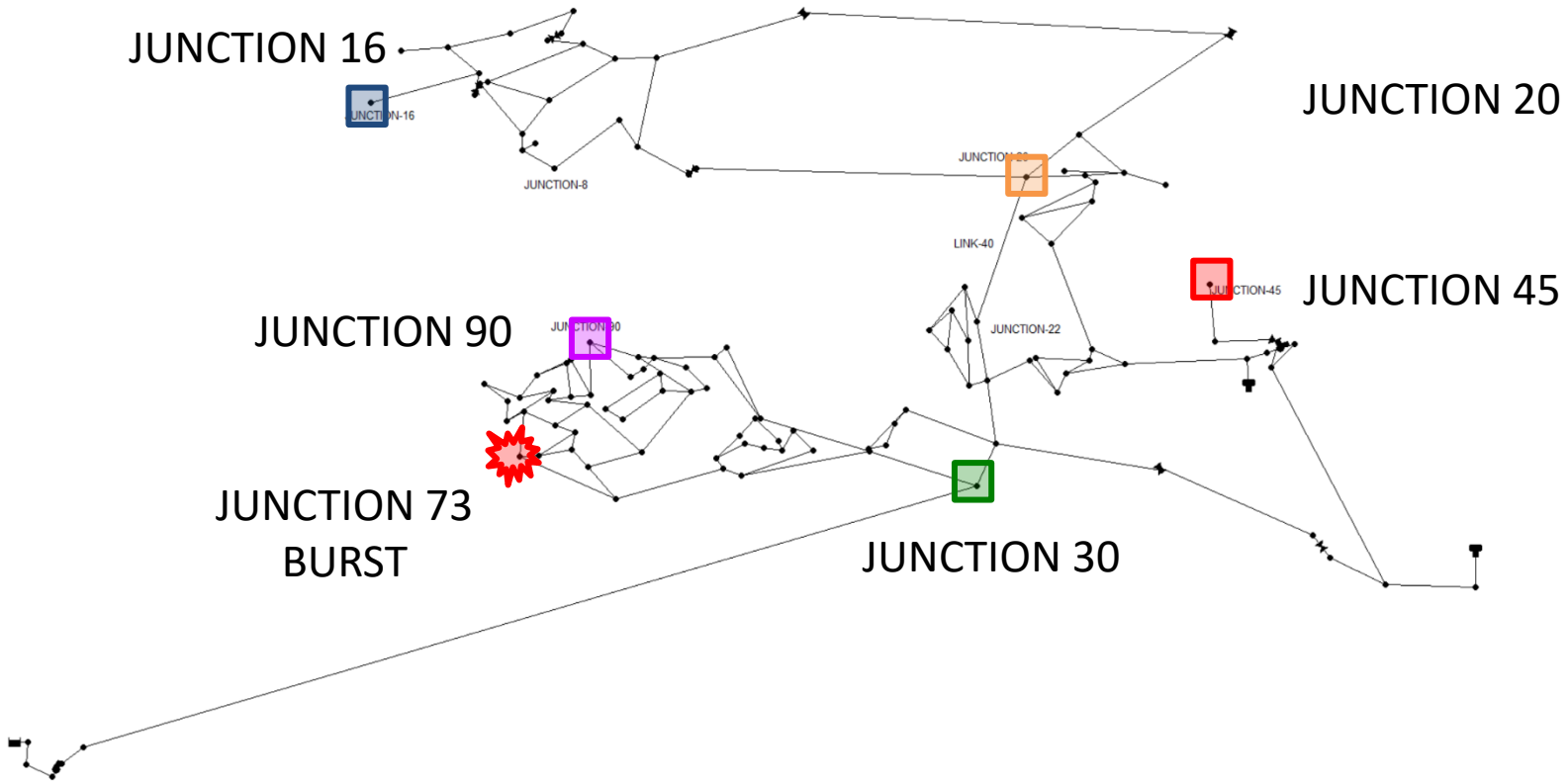


Figure 2 Example network for determining the time step: (a) before combing pipes; (b): after combing pipes.

Noticeably, the maximum allowed time step for this network is fairly small. Meanwhile, the total number of segments ($31 + 2 + 67 = 100$) is relatively large; thus, in order to conduct a transient simulation of $10s$, the overall number of computation nodes in x-t plane will be $10/0.03006 \times 100 = 33267$. The computation efforts can be significantly reduced by, for example, combing/removing the shorted pipe, i.e., pipe 2. Figure 2 (b) illustrates the network after combing pipe 1 and pipe 2. Following the same steps shown above, it can be determined that the maximum time step is $0.5s$, and the number of computation units for pipes 1 and 2 are 2 and 4, respectively, thus significantly reducing the total number of computation nodes in x-t plane required for $10s$ simulation to $10/0.5 \times (2 + 4) = 26$.

Scenario 4: Selecting time step

- Generate a burst at Junction-73
- Run simulation with:
  1. Default time step
  2. $\Delta t = 1s$
  3. $\Delta t = 0.005s$
  4. $\Delta t = 0.002s$
- Compare results



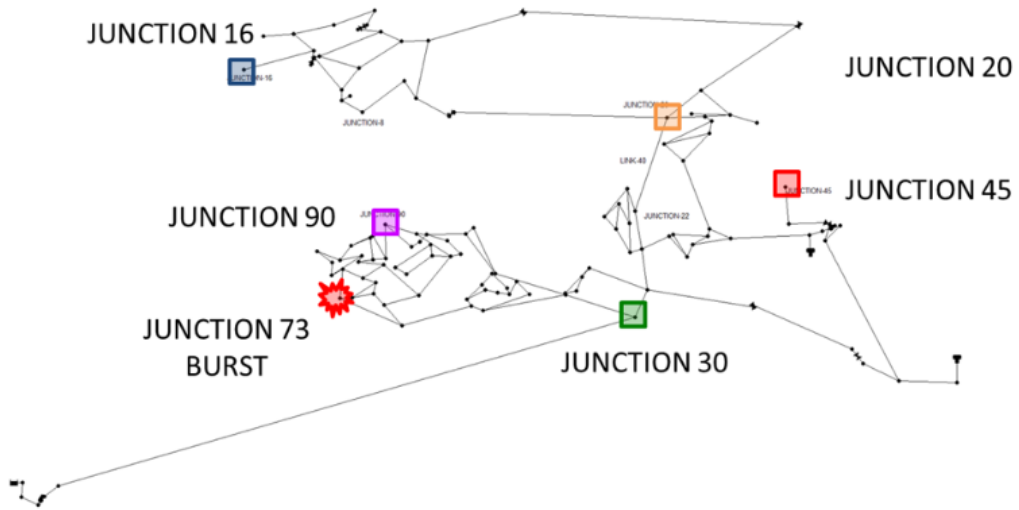
Scenario 4: Selecting time step

WDSA-CCWI 2022

TSNet Short Tutorial

Example 4: TNet3 - Selecting time step

In this example we will use Tnet3 to demonstrate the effect of time step on the simulation time and results.



Import packages

In [1]:
```
import tsnet
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
```

Use default timestep

Maximum allowed timestep assuming the critical pipe is descritized into two segments

In [2]:
```
start = datetime.now()
# open an example network and create a transient model
inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)
```

2_TNet2_pump_shutoff.py | 3_TNet3_burst_leak.py | 1_TNet1_getting_started.py | 4_TNet3_time_step.py

```
# -*- coding: utf-8 -*-
# %%
"""
Created on Mon May 23 13:17:36 2022

@author: ps28866
"""
import tsnet
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime

plt.close('all')

# %% TNet3: Use default time step
#------------------------------------------------
start = datetime.now()

# open an example network and create a transient model
inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#------------------------------------------------
wavespeed = 1200
tm.set_wavespeed(wavespeed)

# Set simulation duration
#------------------------------------------------
tf = 20 # simulation period [s]
tm.set_time(tf)

# Add burst
#------------------------------------------------
ts = 1 # burst start time
tc = 1 # time for burst to fully develop
final_burst_coeff = 0.01 # final burst coeff [ m^3/s/(m H20)^(1/2)]
tm.add_burst('JUNCTION-73', ts, tc, final_burst_coeff)

# Initialize steady state simulation
#------------------------------------------------
t0 = 0. # initialize the simulation at 0s
engine = 'DD' # or Epanet
tm = tsnet.simulation.Initializer(tm, t0, engine)

```

Scenario 4: Selecting time step

Use default timestep

Maximum allowed timestep assuming the critical pipe is descritized into two segments

```
start = datetime.now()
# open an example network and create a transient model
inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#----------------------------------------------
wavespeed = 1200
tm.set_wavespeed(wavespeed)

# Set simulation duration
#----------------------------------------------
tf = 20 # simulation period [s]
tm.set_time(tf)
```

User defined time step - Large timestep is not allowed

```
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#----------------------------------------------
wavespeed = 1200
tm.set_wavespeed(wavespeed)

# Set time step
#----------------------------------------------
tf = 20 # simulation period [s]
dt = 1
tm.set_time(tf, dt)
```

User defined time step - Specify time step as 0.005s

Note: this is not the actual time step used in the analysis. Further adjustment is taken.

```
start = datetime.now()
# open an example network and create a transient model
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#----------------------------------------------
wavespeed = 1200
tm.set_wavespeed(wavespeed)

# Set time step
#----------------------------------------------
tf = 20 # simulation period [s]
dt = 0.005
tm.set_time(tf, dt)
```

Specify even smaller time step (0.002s)

```
start = datetime.now()
# open an example network and create a transient model
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#----------------------------------------------
wavespeed = 1200
tm.set_wavespeed(wavespeed)

# Set time step
#----------------------------------------------
tf = 20 # simulation period [s]
dt = 0.002
tm.set_time(tf, dt)
```

Scenario 4: Selecting time step

Specified Δt [s]	Actual Δt [s]	Running times [mm:ss]
Default	0.01154	1:34

```
Simulation time step 0.01154 s
Total Time Step in this simulation 1732
Estimated simulation time 0:01:27.627076
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %
Actual simulation time for time step =0.01154 s is 0:01:34.509944
```

Scenario 4: Selecting time step

Specified Δt [s]	Actual Δt [s]	Running times [mm:ss]
1	NA	NA

```
# Set time step
#-------------------------------------------------
tf = 20 # simulation period [s]
dt = 1
tm.set_time(tf, dt)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4164/4081066764.py in <module>
     10 tf = 20 # simulation period [s]
     11 dt = 1
---> 12 tm.set_time(tf, dt)

~\AppData\Local\Continuum\anaconda3\envs\tsnet\lib\site-packages\tsnet\network\model.py in set_time(self, tf, dt)
    184             dt = max_time_step(self)
    185         self.simulation_period = tf
--> 186         self = discretization(self, dt)
    187         print('Simulation time step %.5f s' % self.time_step)
    188

~\AppData\Local\Continuum\anaconda3\envs\tsnet\lib\site-packages\tsnet\network\discretize.py in discretization(tm, dt)
     29     if dt > max_dt:
     30         raise ValueError("time step is too large. Please define \
---> 31                          a time step that is less than %.5f " %max_dt)
     32     else :
```

Scenario 4: Selecting time step

Specified Δt [s]	Actual Δt [s]	Running times [mm:ss]
0.005	0.00517	4:48
0.002	0.00202	21:42

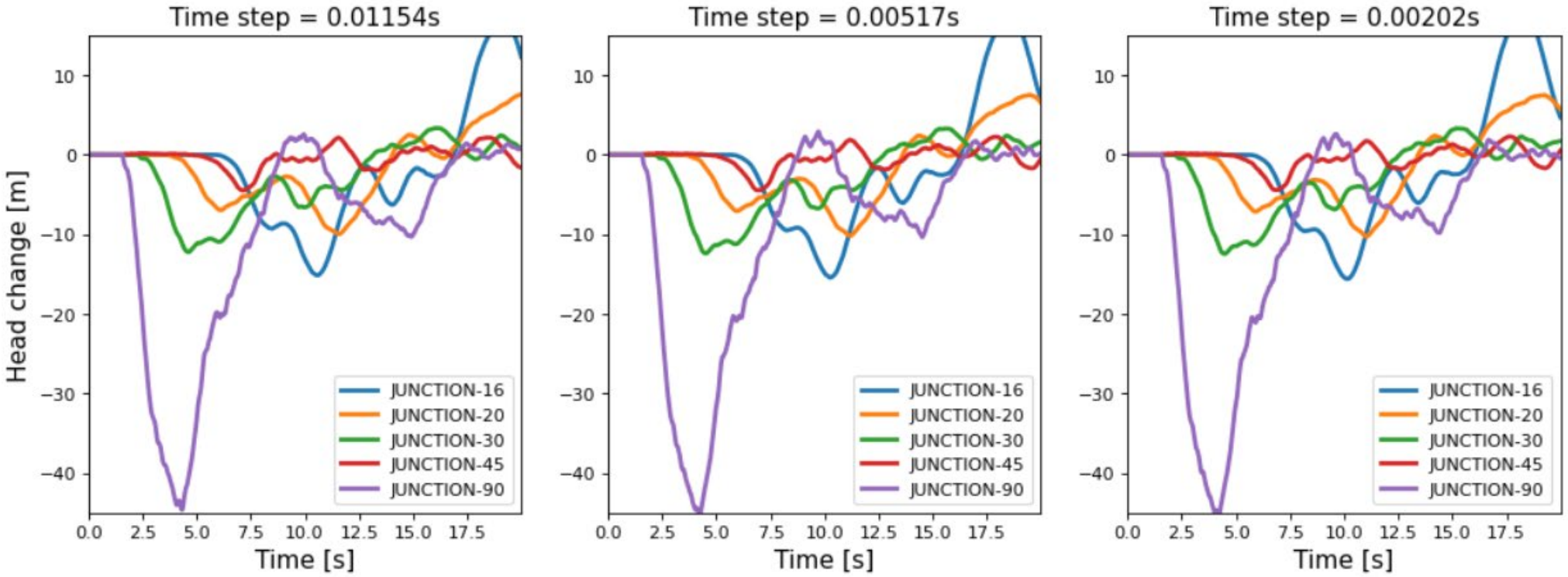
```
Simulation time step 0.00517 s
Total Time Step in this simulation 3868
Estimated simulation time 0:02:49.488024
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
Actual simulation time for time step =0.00517 s is 0:04:48.09
```

```
Simulation time step 0.00202 s
Total Time Step in this simulation 9884
Estimated simulation time 0:15:33.316468
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
Actual simulation time for time step =0.00202 s is 0:21:41.59
```

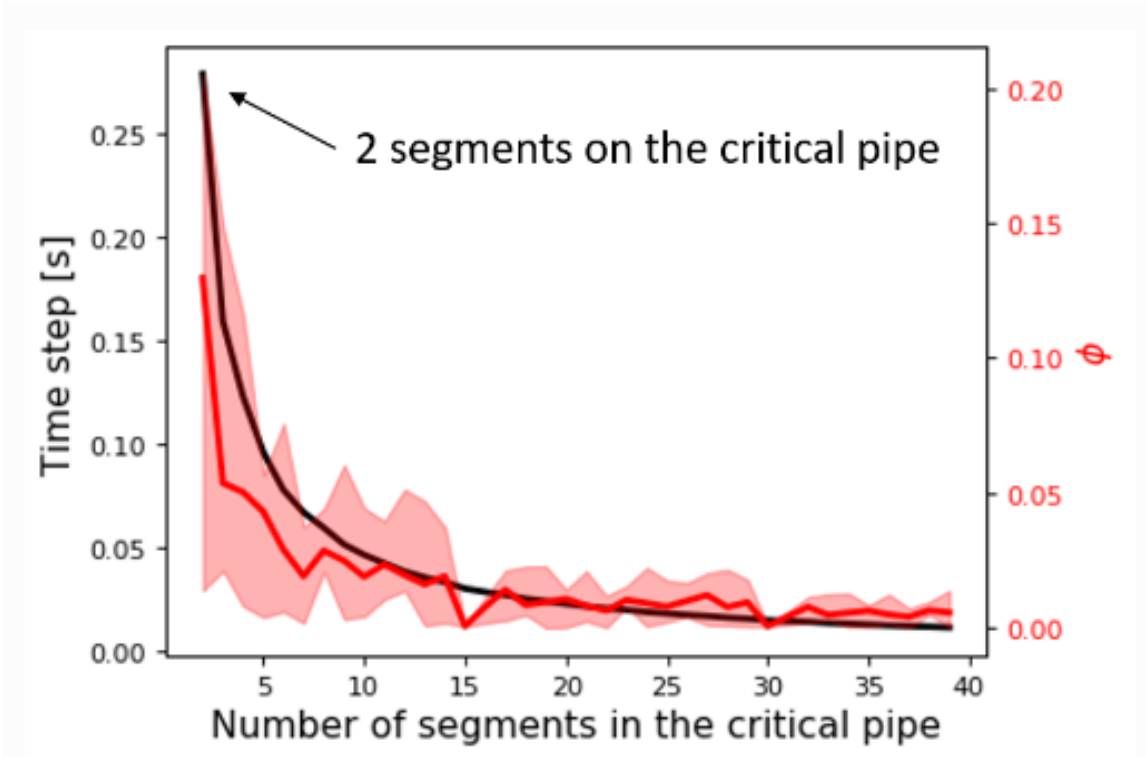
Scenario 4: Selecting time step

Specified Δt [s]	Actual Δt [s]	Running times [mm:ss]
Default	0.01154	1:51
1	NA	NA
0.005	0.00517	4:48
0.002	0.00202	21:42

Selecting time step



Selecting time step



- We recommend users start with default time step

Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Scenario 5: Working with results

Results Retrieval

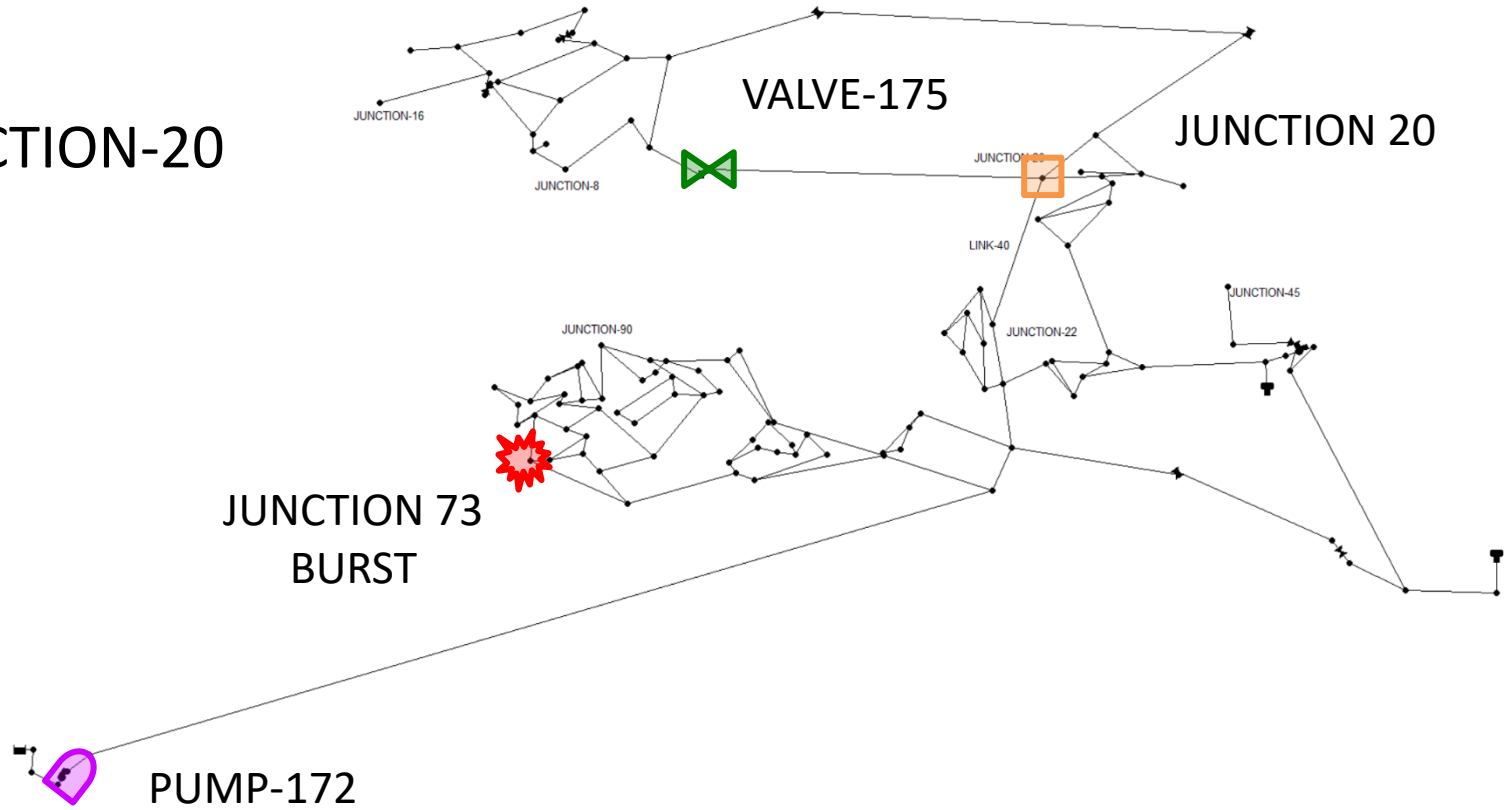
The `tsnet.network.model.TransientModel` object, including the information of the network, operation rules, and the simulated results, is saved in the file **results_obj.obj**, located in the current folder. The name of the results file is defined by the input parameter *result_obj*. If *result_obj* is not given, the default results file is *results.obj*.

To retrieve the results from a previously completed simulation, one can read the `tsnet.network.model.TransientModel` object from the **results_obj.obj** file and access results from the objet by:

```
import pickle
file = open('results.obj', 'rb')
tm = pickle.load(file)
```

Scenario 5: Working with results

- Generate transient events and save results:
  1. Burst at JUNCTION-73
  2. Valve closure VALVE-175
  3. Pump shutoff PUMP-172
- Load results
- Plot pressure response at JUNCTION-20



Scenario 5: Working with results

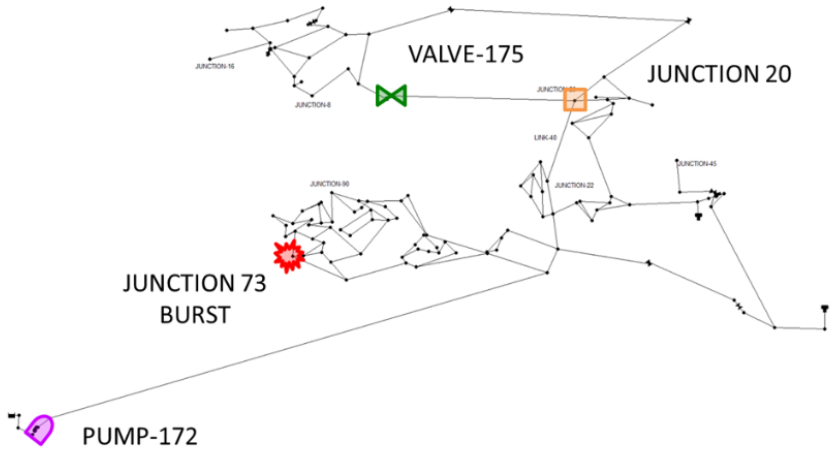
- Jupyter Notebooks: *5_TNet3_results*

WDSA-CCWI 2022

TSNet Short Tutorial

Example 5: TNet3 - Retrieving simulation results

In this example we will use Tnet3 to demonstrate how to retrieve results from previous simulations. We will load results from previous scenarios: (1) burst in JUNCTION-73, (2) closure VALVE-173, and (3) shutoff PUMP-172. Then, we will plot the pressure response in JUNCTION-20.



Import packages

```
l]: import tsnet
    import matplotlib.pyplot as plt
    import numpy as np
    import pickle
```

Load simulation results - TSNet3:

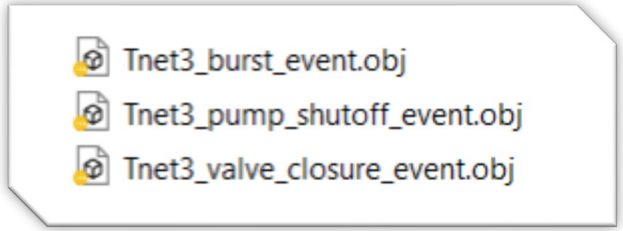
- Burst
- Valve closure
- Pump shutoff

```
114
115 #%% report results
116 import pickle
117
118 file = open('Tnet3_burst_event.obj', 'rb')
119 tm_burst = pickle.load(file)
120
121 file = open('Tnet3_valve_closure_event.obj', 'rb')
122 tm_valve = pickle.load(file)
123
124 file = open('Tnet3_pump_shutoff_event.obj', 'rb')
125 tm_pump = pickle.load(file)
126
127 #----------------------------------------------------
128 node = 'JUNCTION-20'
129 node = tm_burst.get_node(node)
130 fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
131 plt.plot(tm_burst.simulation_timestamps,node.head, 'k', lw=3,label = 'Burst')
132
133 #----------------------------------------------------
134 node = 'JUNCTION-20'
135 node = tm_valve.get_node(node)
136 plt.plot(tm_burst.simulation_timestamps,node.head, 'b', lw=3,label = 'Valve')
137
138 #----------------------------------------------------
139 node = 'JUNCTION-20'
140 node = tm_pump.get_node(node)
141 plt.plot(tm_pump.simulation_timestamps,node.head, 'r', lw=3,label = 'Pump')
142 plt.xlim([tm_burst.simulation_timestamps[0],tm_burst.simulation_timestamps[-1]])
143
144 plt.xlabel("Time [s]", fontsize=14)
145 plt.ylabel("Head [m]", fontsize=14)
146 plt.legend(loc='best')
147 plt.title('Transient response at %s '%node)
148 plt.grid(True)
149 plt.show()
150 fig.savefig('./networks/Tnet3_get_all_results.pdf', format='pdf',dpi=100)
151
```

Scenario 5: Working with results

```python
#%% report results
import pickle

file = open('Tnet3_burst_event.obj', 'rb')
tm_burst = pickle.load(file)

file = open('Tnet3_valve_closure_event.obj', 'rb')
tm_valve = pickle.load(file)

file = open('Tnet3_pump_shutoff_event.obj', 'rb')
tm_pump = pickle.load(file)

#-----------------------------------------------
node = 'JUNCTION-20'
node = tm_burst.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
plt.plot(tm_burst.simulation_timestamps,node.head, 'k', lw=3,label = 'Burst')

#-----------------------------------------------
node = 'JUNCTION-20'
node = tm_valve.get_node(node)
plt.plot(tm_burst.simulation_timestamps,node.head, 'b', lw=3,label = 'Valve')

#-----------------------------------------------
node = 'JUNCTION-20'
node = tm_pump.get_node(node)
plt.plot(tm_pump.simulation_timestamps,node.head, 'r', lw=3,label = 'Pump')
plt.xlim([tm_burst.simulation_timestamps[0],tm_burst.simulation_timestamps[-1]])

plt.xlabel("Time [s]", fontsize=14)
plt.ylabel("Head [m]", fontsize=14)
plt.legend(loc='best')
plt.title('Transient response at %s '%node)
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_get_all_results.pdf', format='pdf',dpi=100)
```

- Import simulation results

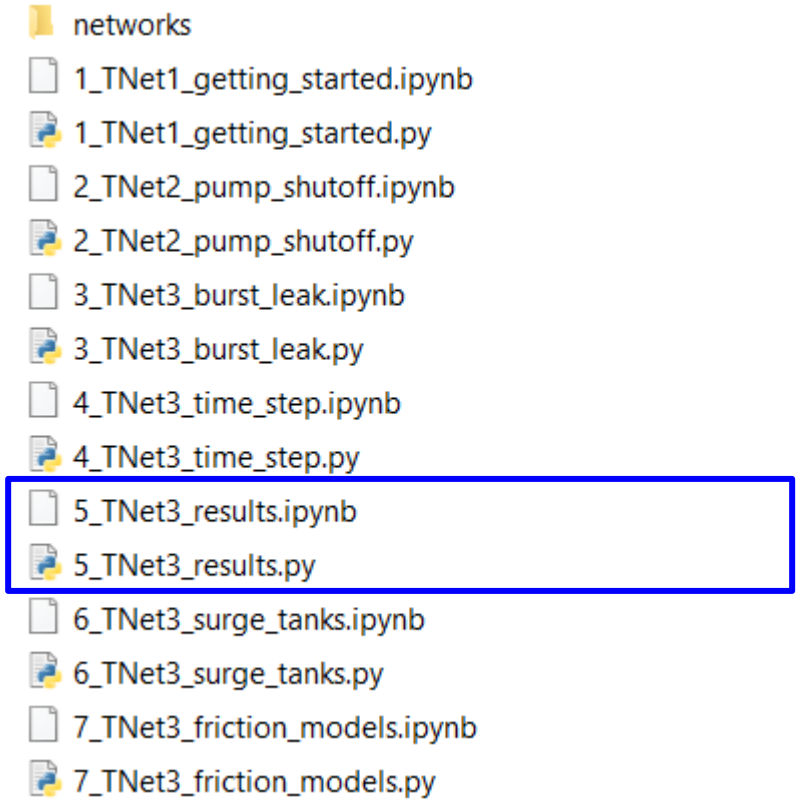


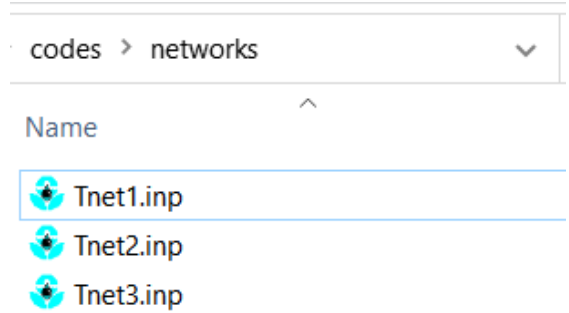
Live Demo

Scenario 5: Working with results

Jupyter Notebooks:

- *5_TNet3_results*





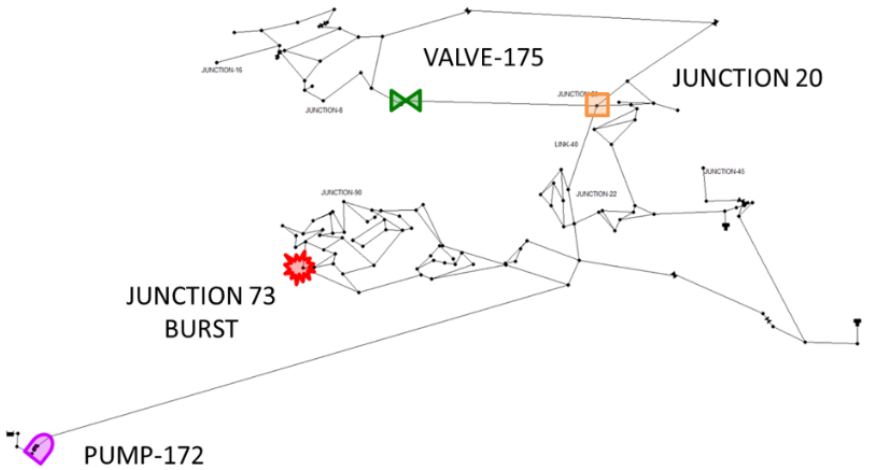
Scenario 5: Working with results

WDSA-CCWI 2022

TSNet Short Tutorial

Example 5: TNet3 - Retrieving simulation results

In this example we will use Tnet3 to demonstrate how to retrieve results from previous simulations. We will load results from previous scenarios: (1) burst in JUNCTION-73, (2) closure VALVE-173, and (3) shutoff PUMP-172. Then, we will plot the pressure response in JUNCTION-20.



Import packages

In [1]:
```
import tsnet
import matplotlib.pyplot as plt
import numpy as np
import pickle
```

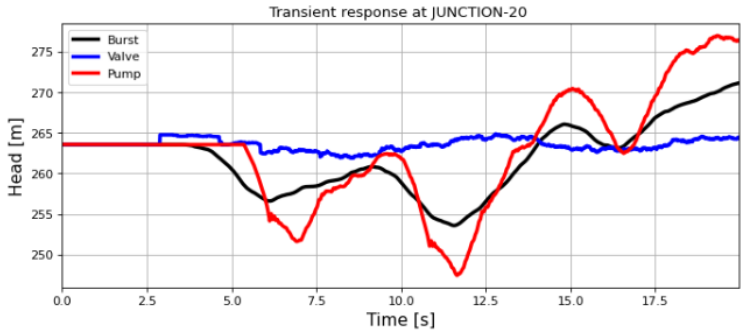
Load simulation results - TSNet3:

- Burst
- Valve closure
- Pump shutoff

In [2]:
```
file = open('Tnet3_burst_event.obj', 'rb')
tm_burst = pickle.load(file)

file = open('Tnet3_valve_closure_event.obj', 'rb')
tm_valve = pickle.load(file)

file = open('Tnet3_pump_shutoff_event.obj', 'rb')
tm_pump = pickle.load(file)
```

Plot transient response at JUNCTION 20

In [3]:
```
# burst
#-------------------------------------------------
node = 'JUNCTION-20'
node = tm_burst.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='w')
plt.plot(tm_burst.simulation_timestamps,node.head, 'k', lw = 3,label = 'Burst')

# valve closure
#-------------------------------------------------
node = 'JUNCTION-20'
node = tm_valve.get_node(node)
plt.plot(tm_burst.simulation_timestamps,node.head, 'b', lw = 3,label = 'Valve')

# pump shutoff
#-------------------------------------------------
node = 'JUNCTION-20'
node = tm_pump.get_node(node)
plt.plot(tm_pump.simulation_timestamps,node.head, 'r', lw = 3,label = 'Pump')
plt.xlim([tm_burst.simulation_timestamps[0],tm_burst.simulation_timestamps[-1]])

plt.xlabel("Time [s]", fontsize = 14)
plt.ylabel("Head [m]", fontsize = 14)
plt.legend(loc='best')
plt.title('Transient response at %s '%node)
plt.grid(True)
plt.show()
fig.savefig('./networks/Tnet3_get_all_results.pdf', format='pdf',dpi=100)
```



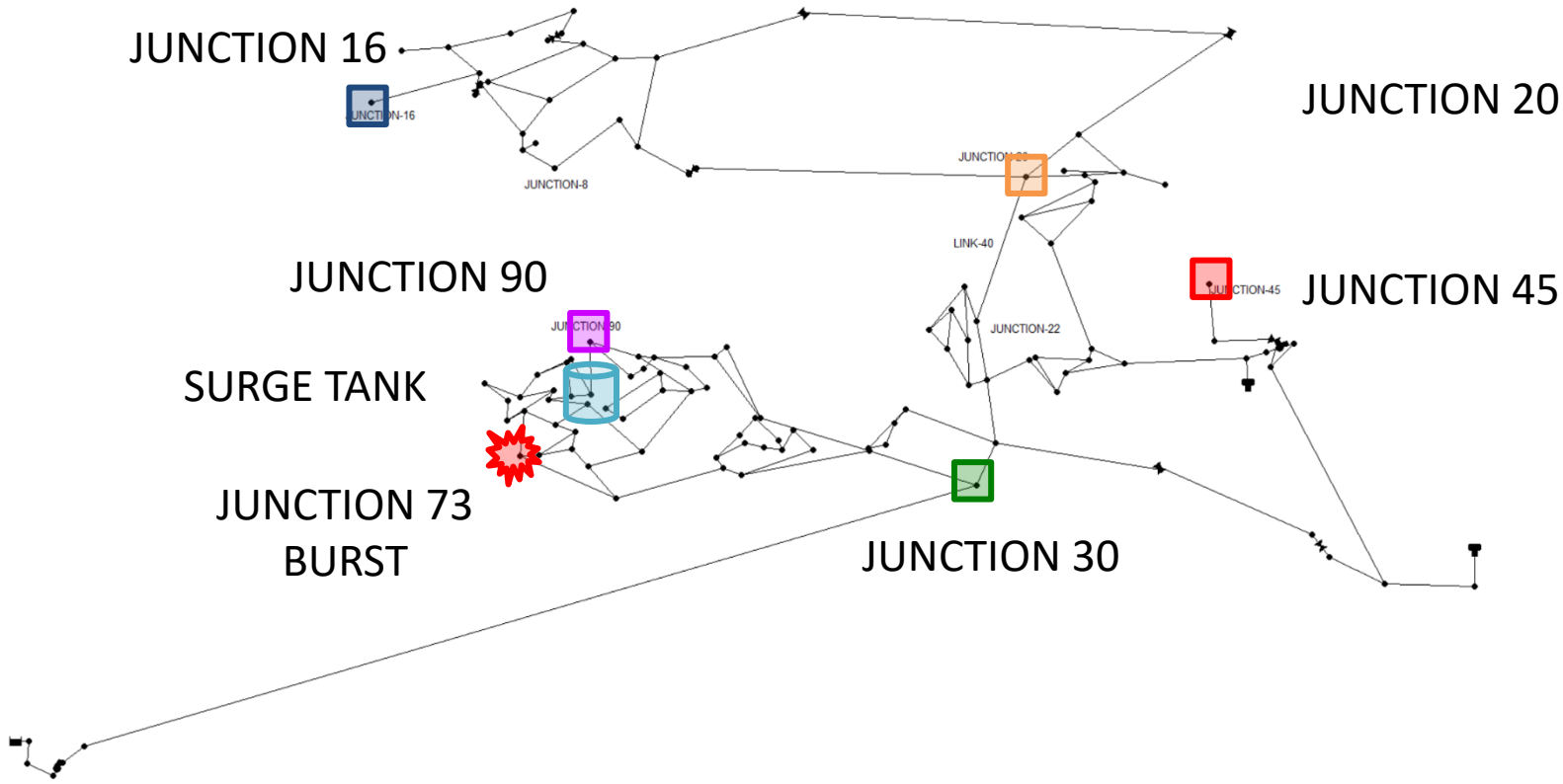
Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Scenario 6: Surge tanks

- Generate burst in JUNCTION-73
- Compare results at different junctions:
  1. Without a surge tank
  2. With closed surge tank
  3. With open surge tank



Scenario 6: Surge tanks

```python
# %% TNet3: Burst with and without surge tanks (open and closed)
#-------------------------------------------------

# open an example network and create a transient model
inp_file = 'networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
#-------------------------------------------------
wavespeed = 1200
tm.set_wavespeed(wavespeed)

# Set time step
#-------------------------------------------------
tf = 20 # simulation period [s]
tm.set_time(tf)

# Add burst
#-------------------------------------------------
ts = 1 # burst start time
tc = 1 # time for burst to fully develop
final_burst_coeff = 0.01 # final burst coeff [ m^3/s/(m H20)^(1/2)]
tm.add_burst('JUNCTION-73', ts, tc, final_burst_coeff)

# Initialize steady state simulation
#-------------------------------------------------
t0 = 0. # initialize the simulation at 0s
engine = 'DD' # or Epanet
tm = tsnet.simulation.Initializer(tm, t0, engine)

# Transient simulation
#-------------------------------------------------
result_obj = 'Tnet3_wo_surge_tank' # name of the object for saving simulat
tm_wo_st = tsnet.simulation.MOCSimulator(tm,result_obj)
```

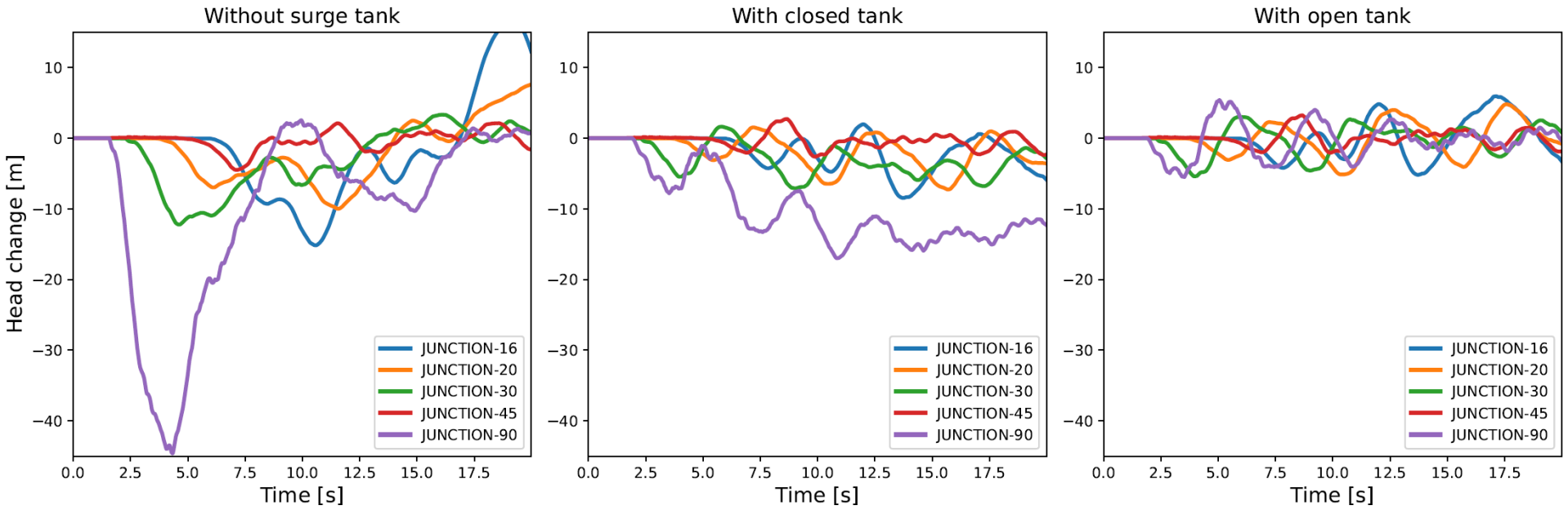
- Add open surge tank
- Define: location, area

```python
# add open surge tank
#-------------------------------------------------
tank_node = 'JUNCTION-89'
tank_area = 10   # tank cross sectional area [m^2]
tm.add_surge_tank(tank_node, [tank_area], 'open')

# Transient simulation
result_obj = 'Tnet3_w_o_surge_tank' # name of the object for saving simulation results
tm_w_ost = tsnet.simulation.MOCSimulator(tm,result_obj)
```

- Add closed surge tank
- Define: location, area, height, initial water level

```python
# add air chamber
#-------------------------------------------------
tank_node = 'JUNCTION-89'
tank_area = 10   # tank cross sectional area [m^2]
tank_height = 10  # tank height [m]
water_height = 9  # initial water level [m]
tm.add_surge_tank(tank_node, [tank_area,tank_height,water_height], 'closed')

# Transient simulation
#-------------------------------------------------
result_obj = 'Tnet3_w_c_surge_tank' # name of the object for saving simulation results
tm_w_cst = tsnet.simulation.MOCSimulator(tm,result_obj)
```

Scenario 6: Surge tanks



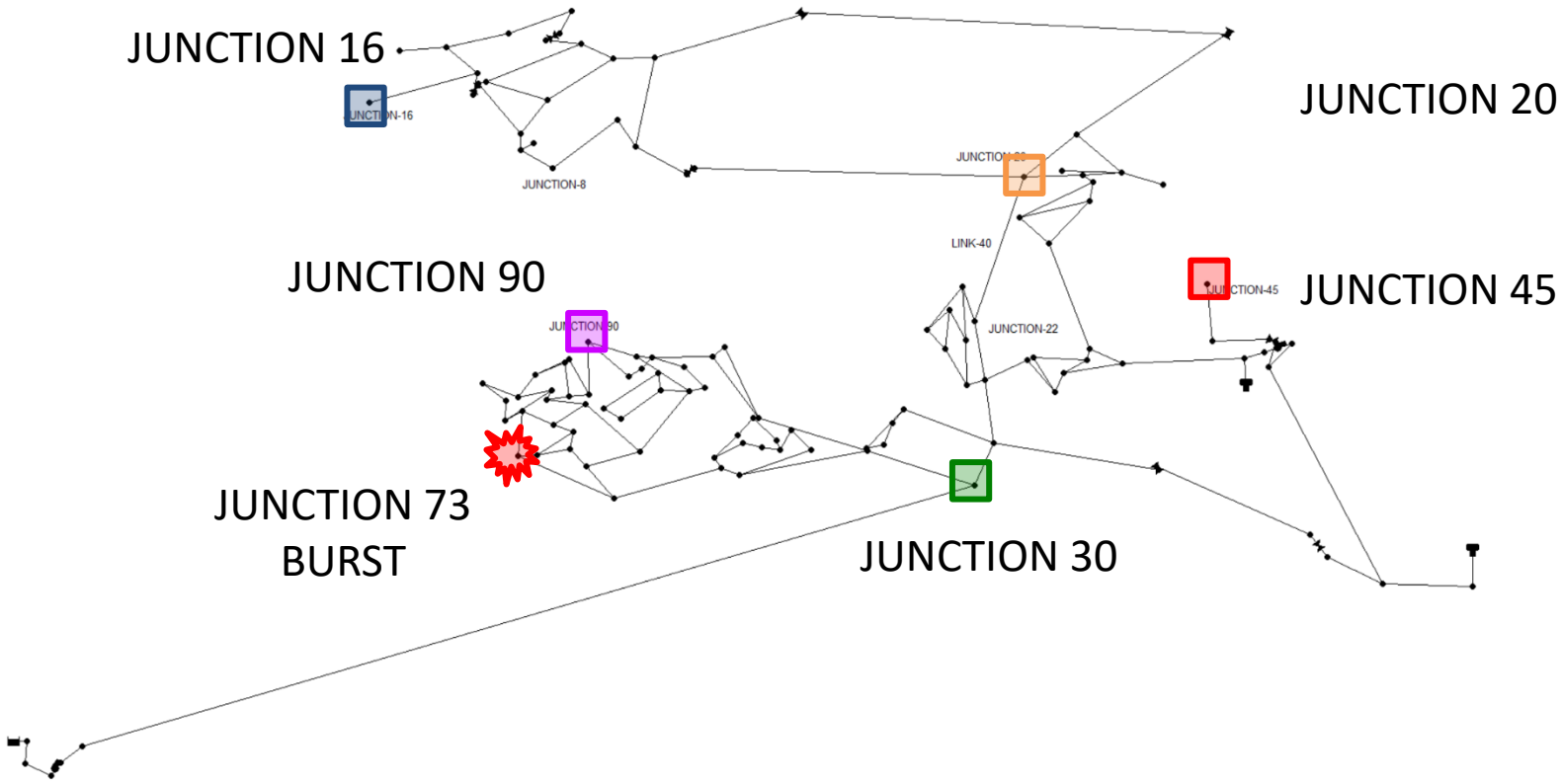
Working with TSNet

Topics

- ✓ Installation
- ✓ Getting started
- ✓ Working with the online documentation
- ✓ Pump shutoff scenario
- ✓ Burst + background leak scenario
- ✓ Choice of time step
- ✓ Working with results
- ✓ Adding surge tanks
- ✓ Testing friction models

Scenario 7: Friction models

- Generate burst in JUNCTION-73
- Compare results with different friction models:
  1. Steady
  2. Quasi-steady
  3. Unsteady



Scenario 7: Friction models

- Steady friction

```
49  # Transient simulation
50  result_obj = 'Tnet3_sf' # name of the object for saving simulation results
51  tm_sf = tsnet.simulation.MOCSimulator(tm,result_obj)
```

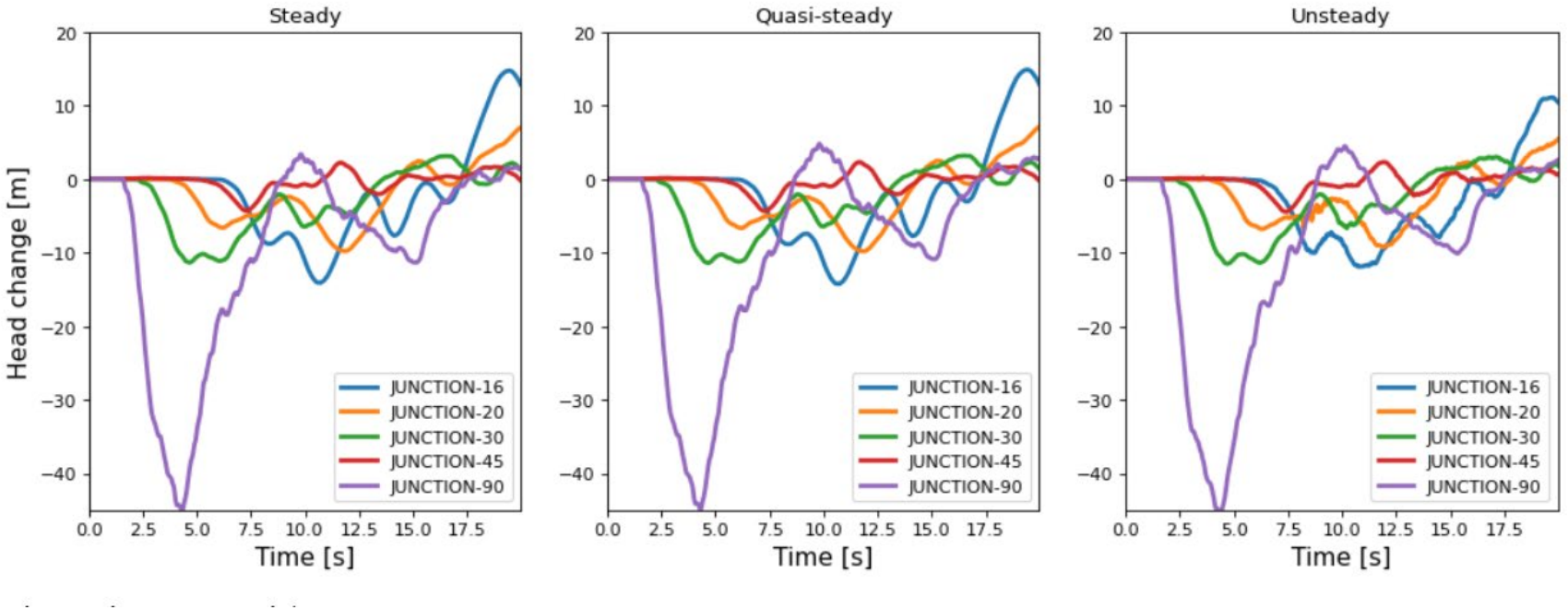
- Quasi-steady friction

```
84  # Transient simulation
85  #------------------------------------------------
86  result_obj = 'Tnet3_qsf' # name of the object for saving simulation results
87  friction ='quasi-steady'
88  tm_qsf = tsnet.simulation.MOCSimulator(tm,result_obj,friction)
```

- Unsteady friction

```
120 # Transient simulation
121 #------------------------------------------------
122 result_obj = 'Tnet3_usf' # name of the object for saving simulation results
123 friction ='unsteady'
124 tm_usf = tsnet.simulation.MOCSimulator(tm,result_obj,friction)
```

Scenario 7: Friction models



Scenario 7: Friction models

Friction model	Running times [mm:ss]
Steady	1:58
Quasi-steady	3:30
Unsteady	5:51

```
Simulation time step 0.01154 s
Total Time Step in this simulation 1732
Estimated simulation time 0:01:21.494064
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
Actual simulation time for steady friction 0:01:58.66
```

```
Simulation time step 0.01154 s
Total Time Step in this simulation 1732
Estimated simulation time 0:02:53.698816
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
Actual simulation time for quasi-steady friction 0:03:30
```

```
Simulation time step 0.01154 s
Total Time Step in this simulation 1732
Estimated simulation time 0:08:21.926672
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
Actual simulation time for unsteady friction 0:05:51.
```

Common warnings

Common warnings

- **Warning:** ”… time step is too large…”
- **Description:** user provided time step exceeds the maximum allowed
- **Action:** use default time step

```
ValueError: time step is too large. Please define

            a time step that is less than 0.01067
```

- **Warning:** “… not all curves were used…”
- **Description**: warning generated by wntr;
- **Action:** user can ignore

```
C:\Users\ps28866\AppData\Local\Continuum\anaconda3\envs\tsnet\lib\site-packages\wntr\epanet\io.py:2085: UserWarning: Not all cu
rves were used in "networks/Tnet2.inp"; added with type None, units conversion left to user
  warnings.warn('Not all curves were used in "{}"; added with type None, units conversion left to user'.format(self.wn.name))
```

Common warnings

- **Warning:** “Reverse flow stopped by check valve!”
- **Description:** reverse flow in the pump station is stopped, and the pump flow is treated as zero
- **Action:** it is recommended to take a closer look at the pump settings, or consider include surge protection devices

```
Reverse flow stopped by check valve!
  warnings.warn( "Reverse flow stopped by check valve!")
```

- **Warning:** “… the quadratic equation has no real solution…”
- **Description:** the equation related to the pump does not have real solution. An approximate solution is used, and the results may not be accurate
- **Action:** it is recommended to take a closer look at the pump settings

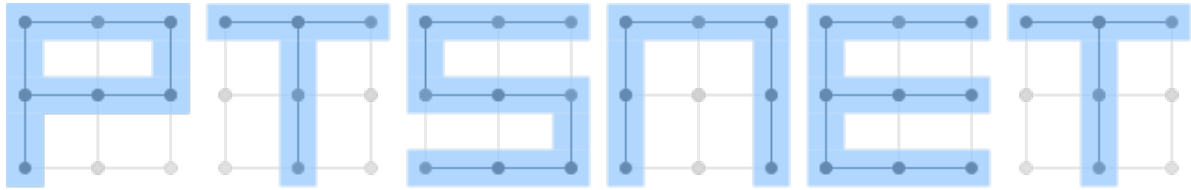
```
Error: The quadratic equation has no real solution (pump)
  warnings.warn('Error: The quadratic equation has no real solution (pump)')
```

Common warnings

- **Warning:** “Negative head gain activates by-pass!”
- **Description:** negative head gain is observed in the pump station. The head gain is treated as zero.
- **Action:** it is recommended to take a closer look at the pump settings

- **Warning:** "Negative pressure in NODE-XX. Backflow stopped by reverse flow preventer.”
- **Description:** negative pressure is observed in the reported node. Demands are pressure driven and zero demand is assigned.
- **Action:**. it is recommended to take a closer look at the settings and consider taking action to prevent negative pressure

Ongoing work

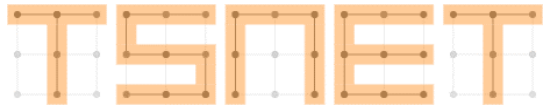


Parallel **T**ransient **S**imulation in water **NET**works

Public Domain Software:
https://github.com/gandresr/ptsnet

Current software limitations

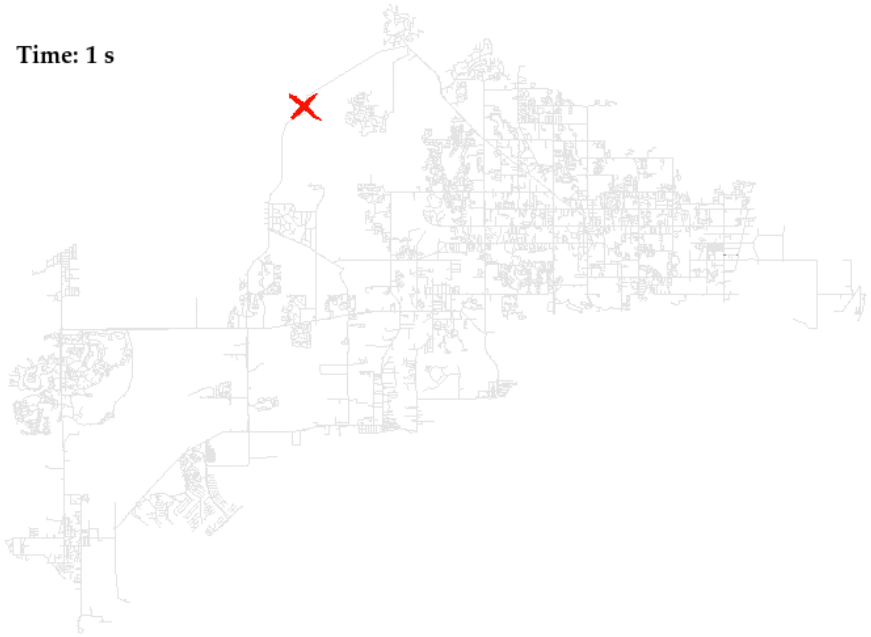
Commercial

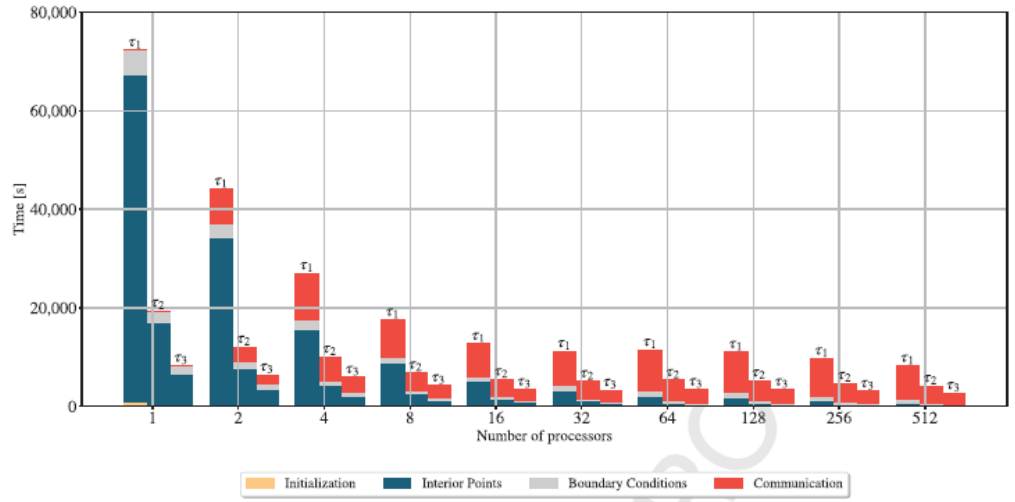


Open-source

- Accurate simulations for large-scale networks with thousands of pipes are intractable
- Lack of scalability and compatibility with high-performance computers
- Difficulty to extract, analyze, and visualize numerical results
- Inability to run transient simulations systematically

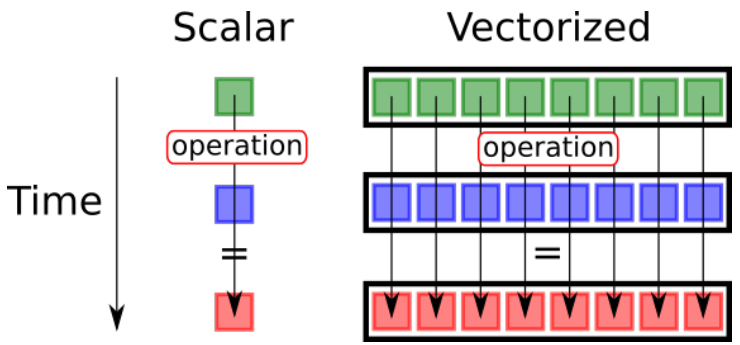
Ongoing work



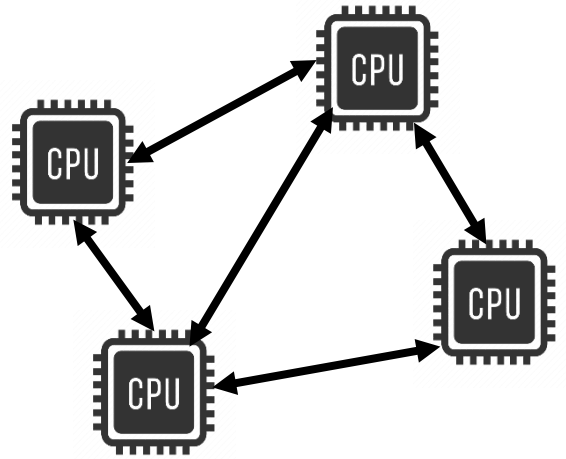


G. Riaño-Briceño, L. Sela, and B. R. Hodges (2021). "Distributed and vectorized method of characteristics for fast transient simulations in water distribution systems." Computer-Aided Civil and Infrastructure Engineering

Ongoing work



Vectorization



Distributed Parallel Computing

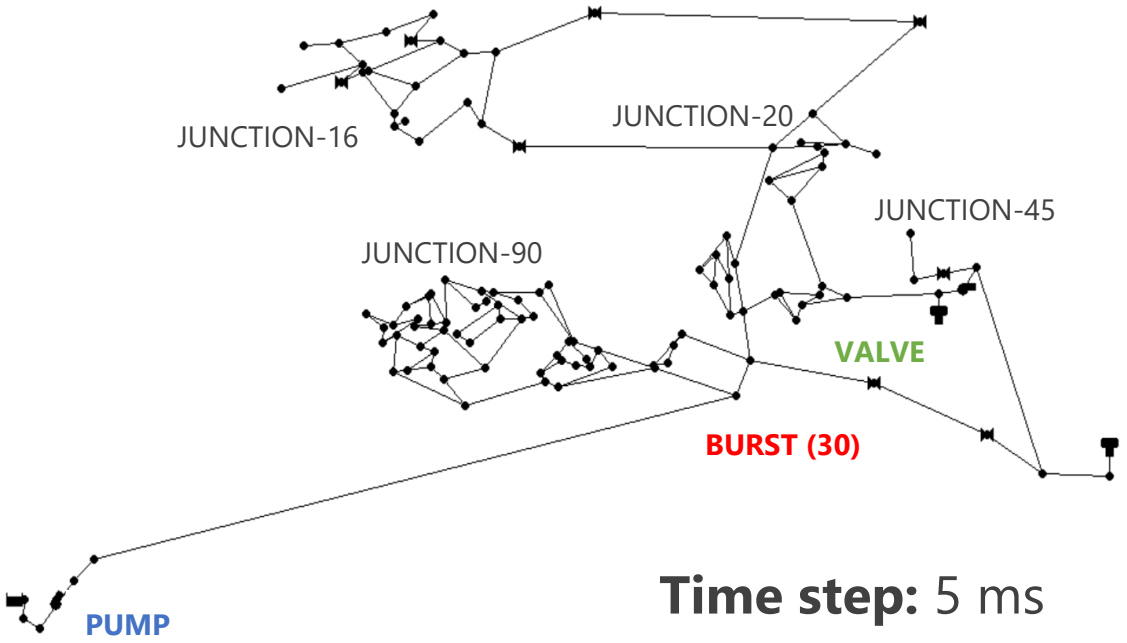
G. Riaño-Briceño, B. R. Hodges, and L. Sela (202X). "PTSNet: A Parallel Transient Simulator for Water Transport Networks Based on Vectorization and Distributed Computing." Submitted to Environmental Modelling & Software

Ongoing work

BWSN-I Network

Case	PTSNET [s]	TSNet [s]	Bentley [s]
Valve closure	6.01	729.50	53.85
Pump shut-off	6.09	709.512	40.89
Burst	6.23	711.80	41.66

- Single processor



Time step: 5 ms

G. Riaño-Briceño, B. R. Hodges, and L. Sela (202X). "PTSNet: A Parallel Transient Simulator for Water Transport Networks Based on Vectorization and Distributed Computing." Submitted to Environmental Modelling & Software

Looking ahead

Survey:

- Fill out a quick anonymous survey about this workshop

Discussion:

- Challenges with modeling transients in WDSs
- Barriers to adopting simulation models
- Success factors
- Open topics in transient research