Key Features

The main attributes relevant to modelling how a wave moves through a pipe network is the **speed of the wave** and the **length of the pipe**. This allows us to calculate how long the water takes to travel through each pipe, the **wave time**, which will be useful for actually demonstrating the propagation of the pressure wave later on.

Edges have a Boolean value ‘flowed’, which stores whether the water has passed through it yet.

Other attributes could be used to model or predict the likelihood a pipe will break, including:

- The year it was built - older pipes could be weaker
- Material - different materials may be stronger
- Wall thickness - thicker pipes are more resistant
- Poisson's ratio
- Elasticity

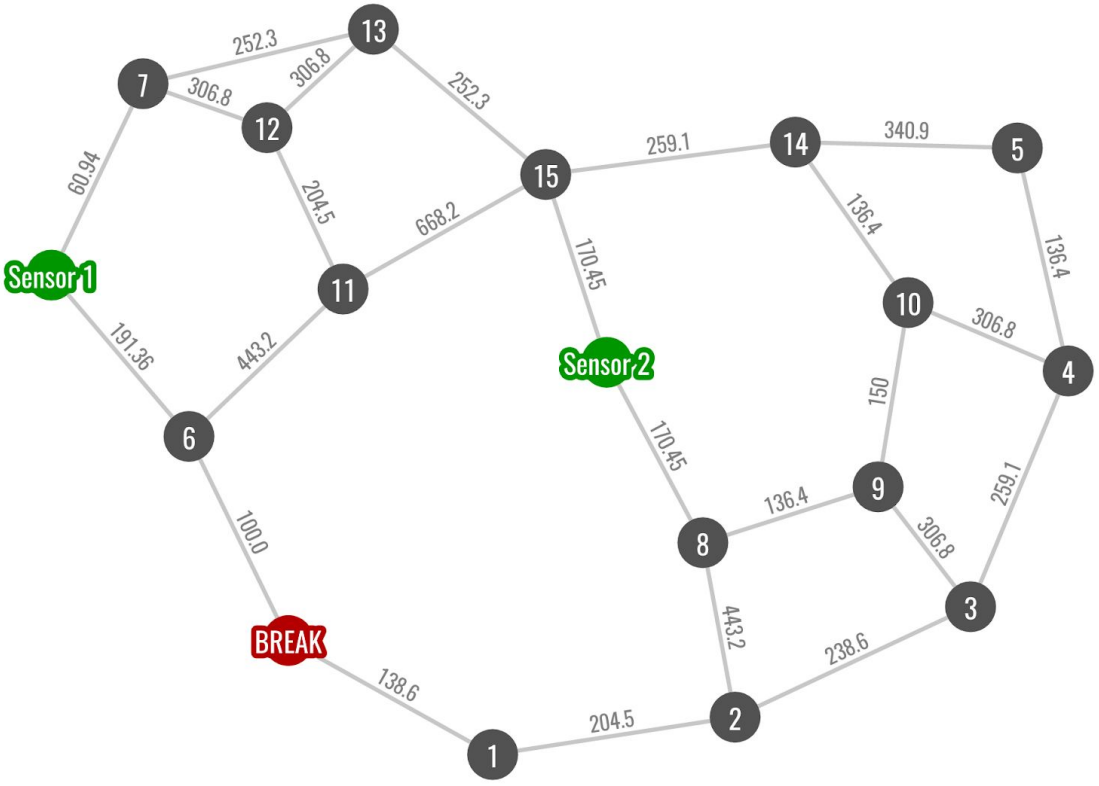
Creating a formula of some sort to find the pipe most likely to break is possible to do to some extent with this data, but would require a deeper understanding of physics, and plumbing!

I chose not to include the K_R value of the pipe in my model, because it isn’t relevant to a pressure wave’s movement. It describes the resistance to flow of rupture disks, which are “designed to rupture at a predetermined pressure” as a failsafe [1]. However, the resistance alone is only a “minor contribution to the overall system flow resistance” and will not impact where breaks happen or how the wave propagates [2].

Modelling the pipes

Abstract Data Type

I used a Graph ADT to model the pipe network, with edges representing the pipes and nodes representing the junctions between pipes, sensors or a break. An edge’s weight is its length in metres.



When a break occurs, the pressure wave will travel along the pipe in both directions. A user can select two nodes and give a distance to indicate where a break is. The edge in question is then removed (using REMOVEEDGE), and 2 edges and a node are added in its place (using ADDEDGE and ADDNODE respectively).

```
ADDNODE: Graph + Element → Graph
REMOVENODE: Graph + Element → Graph
ADDEDGE: Graph + Element + Element → Graph
REMOVEEDGE: Graph + Element + Element → Graph
NEIGHBOURS: Graph + Element → List
```

**only shows mentioned operations*

Justification

A graph works well when modelling networks of objects with relationships. It can very clearly show the links between the pipes. The NEIGHBORS operation is also useful when modelling where the water moves, as it returns the connected nodes the water can flow to. A downside of the graph ADT is that edges do not have weights, so the ADT must be modified in order to assign each edge a value. This modification is helpful because edges are ‘split’ when adding a sensor, so the length indicates where along the path the sensor is. In general, having the length as the weight makes the most sense visually.

Another way you could model this problem is using different layers of other ADTs, namely dictionaries and lists. Each pipe could be represented as a dictionary of attributes, similarly to how I have used dictionaries. You would then have an element of the dictionary be a list of connected pipes. While this model could accurately store the necessary data, it is not as practical when a wave needs to propagate the network, or when placing a sensor in a pipe. Both of these would require extra dictionary entries to store whether there is a wave, the direction the wave is moving, whether there is a sensor, and the location of the sensor. Using nodes on a graph also allows you to take advantage of various path-finding algorithms, such as Dijkstras or Floyd-Warshalls. This method is also a less visually appealing and less clear way of abstracting the problem than a graph.

Storing pipe attributes

Abstract Data Type

A graph alone cannot capture every aspect of the problem. I used the dictionary ADT to store the values for each edge (pipe). Each edge has a corresponding dictionary, which is changed via the SET operation. For example, to set the wave speed of a pipe to 120m, SET dict + “wave_speed” + 120. This can then be accessed with the key “wave_speed” using the GET operation.

```
SET: Dictionary + String + Element → Dictionary
GET: Dictionary + String → Element
```

**only shows mentioned operations*

Justification

The attributes, which are specified below, all have a name and a value. This fits the key/value pairs that dictionaries are made for, which is why it works so well.

Sources

1. http://www.wermac.org/valves/valves_rupture_disk.html
2. https://blog.zookdisk.com/blog/what-does-kr-value-mean