NetProg

A graph based visual programming language.

Ben Jefferson, March 2012

1 Introduction

Netprog is a concept for a graph-based visual programming language. I conceived this over ten years ago and tested it through numerous sketches on paper. The concept of a language where you could sketch a program on paper appealed to me as did the idea of a language where you could zoom in on elements, going from high level programming constructs right down to fundamental units in a relatively seamless manner. I also wanted to make a language that would be accessible to people with little or no programming experience.

Each program is made up of a network of interconnected nodes. Each node has a defined number of inputs and outputs. Outputs from one node are connected to inputs of another node. Nodes take packets of data on their inputs and based on the packets that arrive generate new packets of data on their outputs. A collection of interconnected nodes that performs a particular operation can be wrapped up into a module. A module is a network with defined input and output points. Modules can be included in other networks appearing as single node. Each node in a network can either be one of a small set of fundamental nodes, or a user defined module.

The current prototype is available at http://netprog.classics.co.uk/. This is built largely in Javascript, with a PHP back end for storage and retrieval of networks. This approach was chosen as it gives good cross-platform support and can be developed and deployed using open source tools.

As the concept for Netprog has developed it has struck me that this approach might prove to be a useful for programming massively parallel systems where the existing sequential paradigm fails to efficiently utilize resources. Writing sequential code for parallel systems requires the programmer to jump through hoops to maintain state and manage distribution and collection of data. Netprog is inherently parallel with state distributed throughout the network (in the packets which are traversing it) and data distribution and recombination defined in the network itself.

Unlike other visual graph based approaches to programming (such as HeNCE or CODE) Netprog completely does away with any sequential code with all programs built up from the small set of fundamental nodes. The intention here is that the fundamental nodes might be so simple as these could be implemented in hardware with the network defining the physical wiring between them. Ultimately the nodes could be implemented as a collection of gates on an FPGA such that an entire network (the gates which make up the nodes and the network connections which interlink them) can be flashed onto an FPGA. As a result Netprog might provide a easy to use approach to FPGA design.

Netprog is probably most similar to National Instruments LabVIEW (http://www.ni.com/labview/). However this has a strong emphasis on data acquisition, signal processing and automation as opposed to general purpose programming.

2 Aims

2.1 Accessible

One of my key aims was to implement a way of programming which would be intuitive to people with a graphical way of thinking. Conventional sequential programming is by no means intuitive and relies on the ability of the programmer to convert the words of code into some sort of understanding of how that code will function. This relies heavily on syntax. This approach is error prone and restricted to accessible only to those with a particular mind set. My aim with Netprog is to create a language in which you draw programmes instead of writing them.

2.2 Intuitive

In line with the intentions expressed above of creating a language that is accessible it is my intention to make Netprog as intuitive as possible. At this stage there is plenty more work to be done on this aspect. This is partly due to the requirement for input from a graphic designer and partly due to the influence of my own programming background. In particular the current naming scheme for the fundamental nodes requires work to move this away from conventional programming terms and more towards everyday concepts (e.g. a railway analogy of "points" and "carriages" may be better).

2.3 Simple

The desire for simplicity stems not only from a desire for accessibility and intuitiveness but also to ensure that the fundamental building blocks are so simple that they could be implemented in hardware

2.4 Modular

The ability for users to create there own custom nodes allows for code reuse and provides scope for repeated abstraction of complex code. At the highest level a program might consist of a single "do job" module, but zooming into this the user

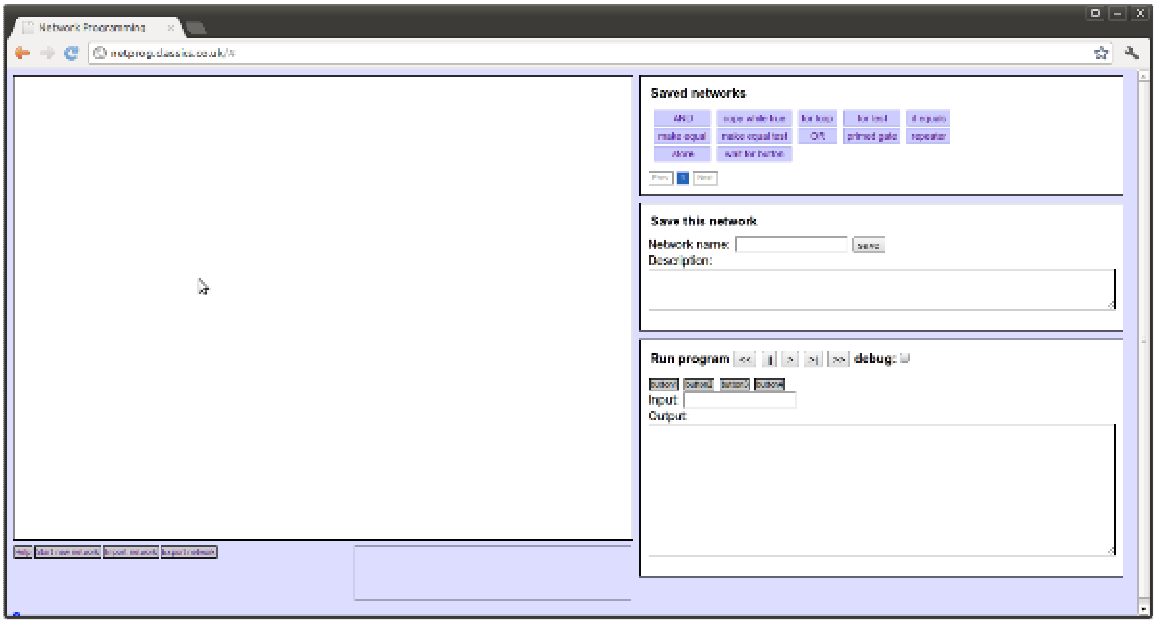
would find that this comprises a network of slightly lower level modules which have been wired together. The user could zoom in again and again to the point where the code consists only of fundamentals.

2.5 Getting Started

The current proof of concept has been developed using Google Chrome. It has not been extensively tested in other browsers but should also work fairly reliably in Firefox.

2.5.1 Visit http://netprog.classics.co.uk

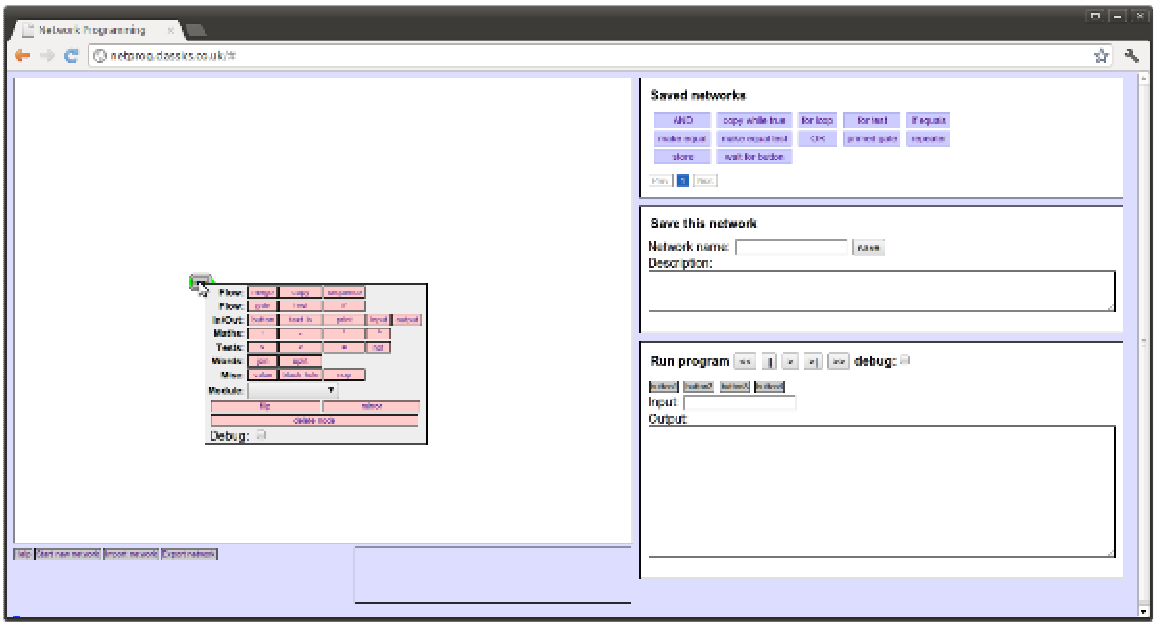
The initial screen layout is shown below:



- The main areas of the screen are as follows
- The main programming canvas on the left – this is where programs are drawn
- The saved network menu on the top right – here you can load networks that have been saved
- The "Save this network" box – here you can save the current network
- The "Run program" area – this allows you to run the current network, see the output from it and also interact with it through the 4 buttons (if it supports user interaction). User interaction is very limited at this stage.

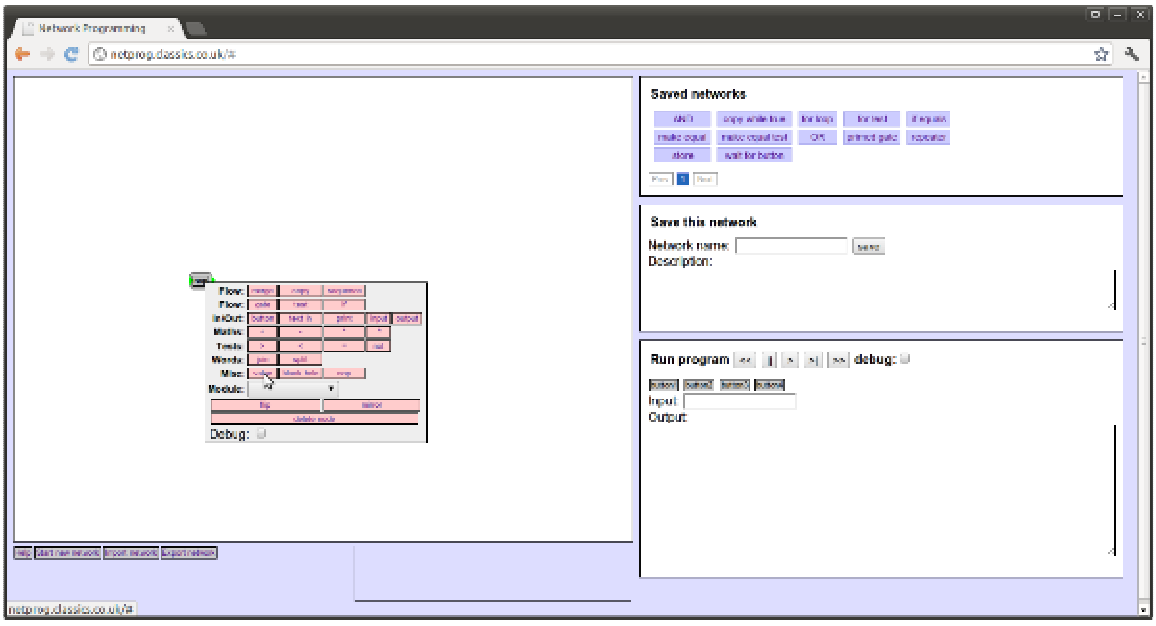
2.5.2 Right click on the programming canvas to create a new node

When you right click a new "No operation" (nop) node will be created on the canvas. The node control popup will also appear allowing you to change the node to something more useful. This is shown below:



2.5.3 Choose the node type

Click on the “value” box in the “Misc:” section as shown below:

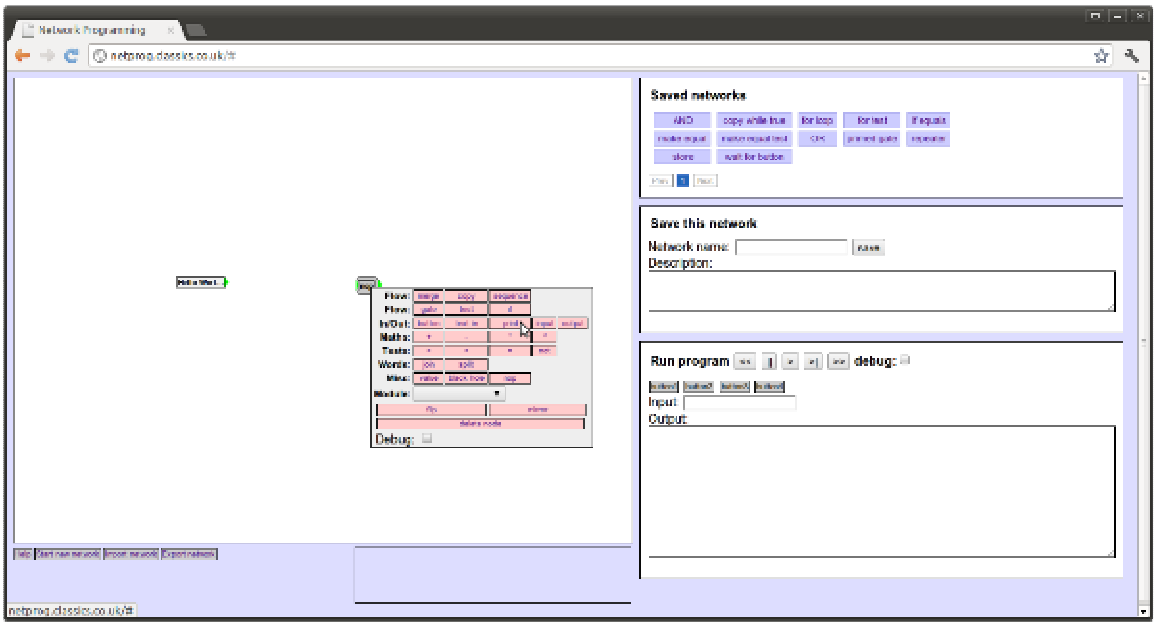


When you select this option you will be prompted to provide a value. In this case you can enter “Hello world” or anything else that takes your fancy!

The value node generates a single packet of data containing the specified value. Once this packet has been created the node effectively dies.

2.5.4 Create a “print” node

Right-click on the canvas again and choose “print” from the “In/Out” section as shown below:

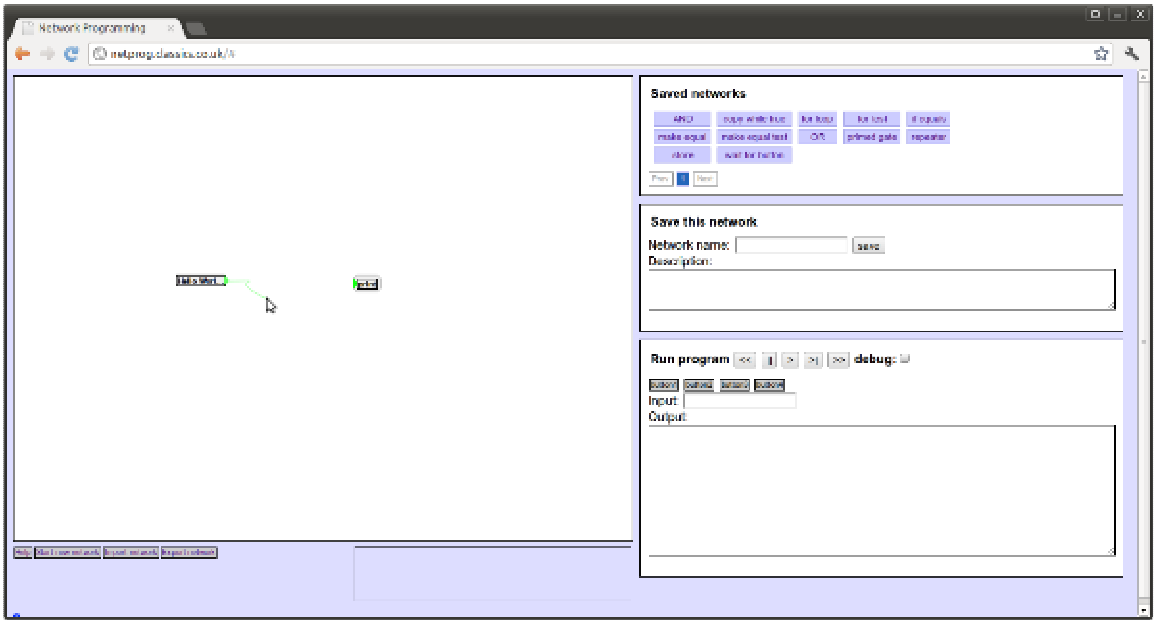


A print node causes the contents of any packets it receives to be displayed in the "Output" box in the "Run program area".

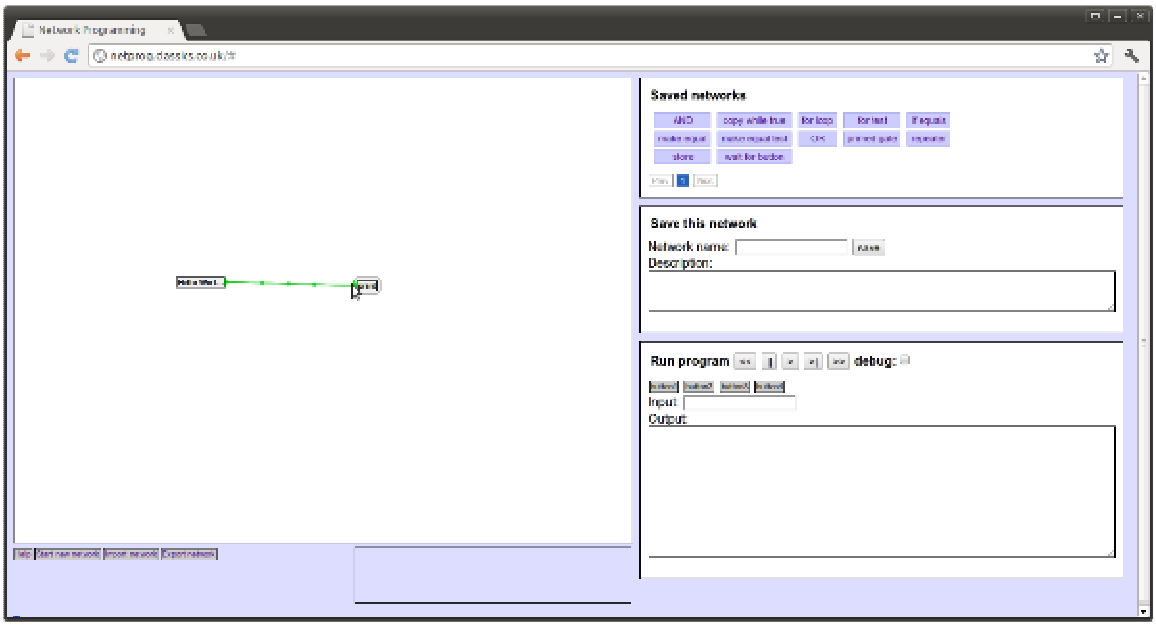
2.5.5 Create a link between the value and the print nodes

Click and drag from the green output arrow on the value (Hello World!) node. As you drag a light green tentative link line should appear. If a grey marquee appears then you have accidentally started drawing a group box. In this case release the drag before the group box gets very big and the group box will be abandoned. Grabbing the output arrow can be a bit fiddly.

The picture below shows the tentative link being dragged out:



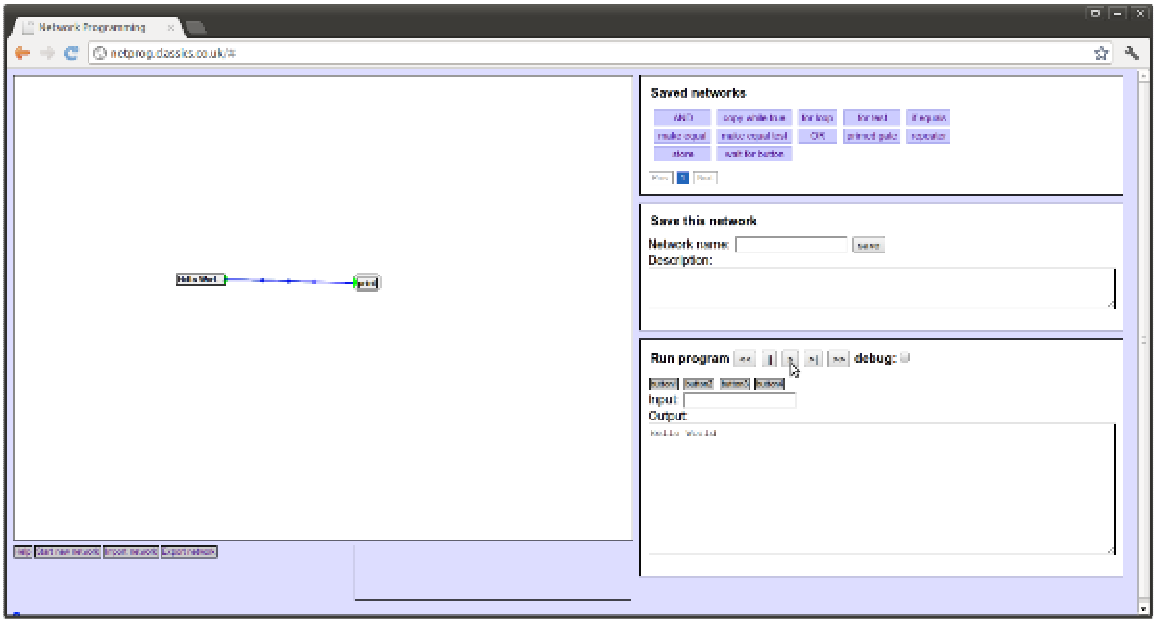
Drag the tentative link to the green input arrow of the print node – the link should snap to this when you are close. When the link has snapped to the input arrow you can stop dragging and a permanent link should appear as shown below:



The small green circles on the link are routing handles – you can drag these around to change the route that the link takes.

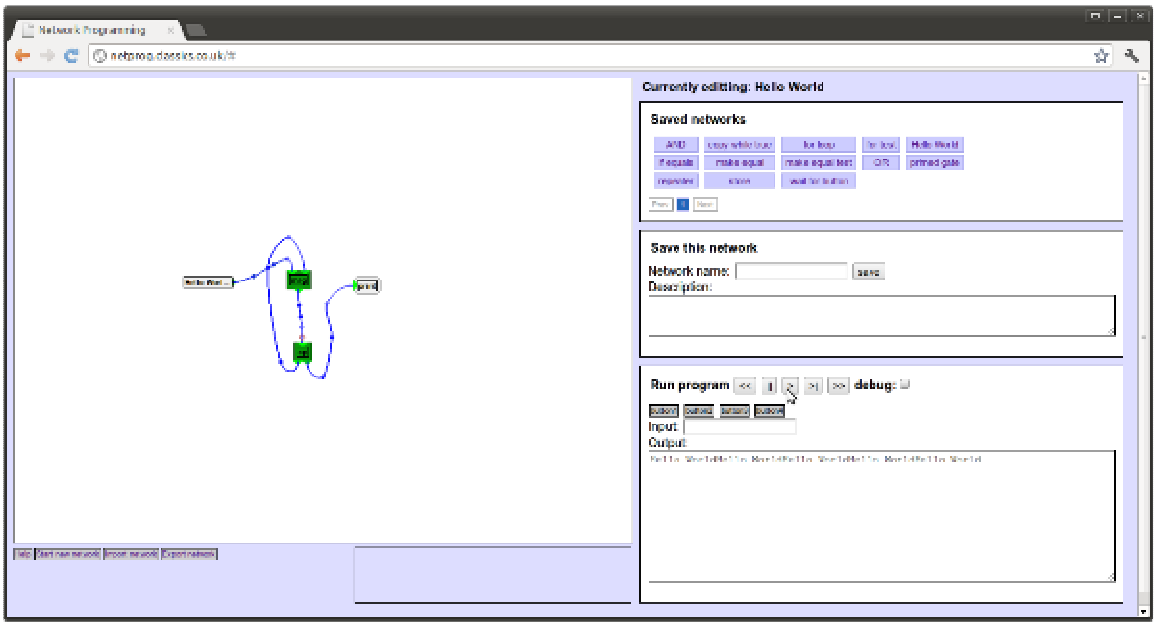
2.5.6 Run the program

To start execution press the ">" button in the Run program box. Before you do this take care to watch the link between the two nodes. When you press the start button you should see the data packet (shown as a small red circle) traversing the link.
When the data packet reaches the print node the words "Hello World!" will appear in the output box as shown below:



2.5.7 A slightly more complex network

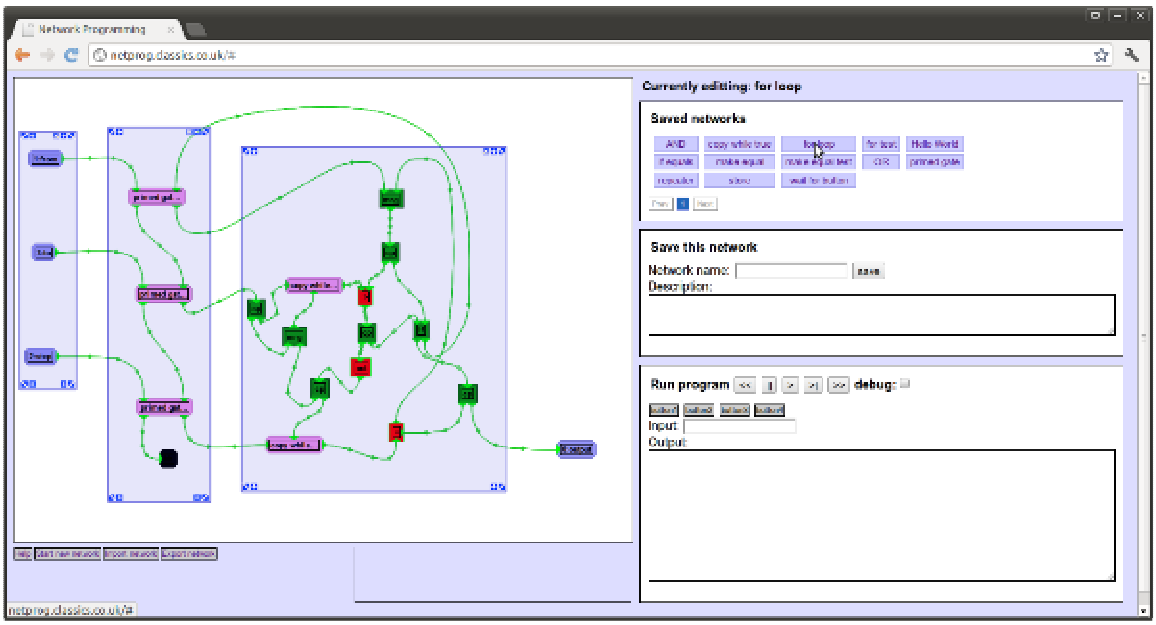
By adding a merge (mrg) and copy (cp) node you can create a network which prints hello world repeatedly as shown below:



To re-route existing links to different nodes simply drag from the green output arrow again – if a link exists then this will automatically be deleted when you start dragging.

2.5.8 Load an even more complicated network

Click on the "for loop" box in the "Saved networks" menu. This will load up a network which implements a for loop as shown below:



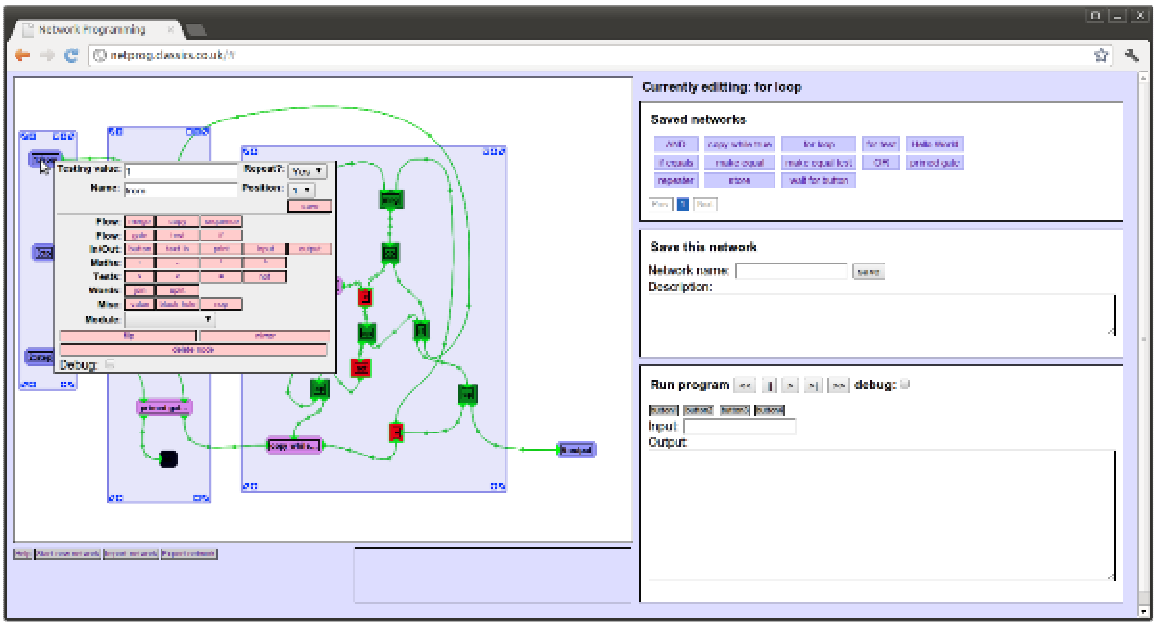
The light blue boxes are groups. These do not affect execution and simply provide a way to group nodes for aesthetic purposes to aid understanding. They also allow all nodes in the group to be moved at the same time.

The blue nodes are input and output – these turn a simple network into a module that can be embedded into other networks. This network includes a few other modules – these are the pink nodes ("primed gate" and "copy while true"). When a module is

loaded as a network for editing then output nodes behave as “print” nodes and input nodes behave as value nodes.

2.5.9 Set the values for the inputs

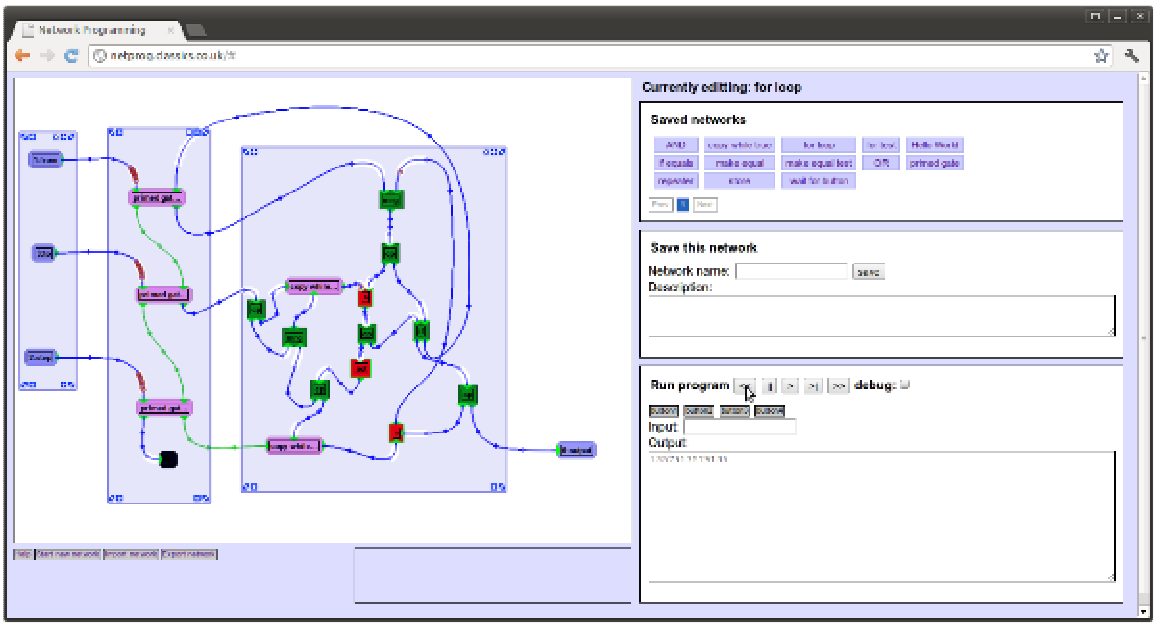
To set the test value that the input node will generate when it is run as a stand alone network, right-click on an input node. You will see the node control menu as shown below:



In this menu you can change the “Testing value” – remember to click save after changing this value.

In this case the test values for the for loop cause it to iterate from 1 to 10 in increments of 2.

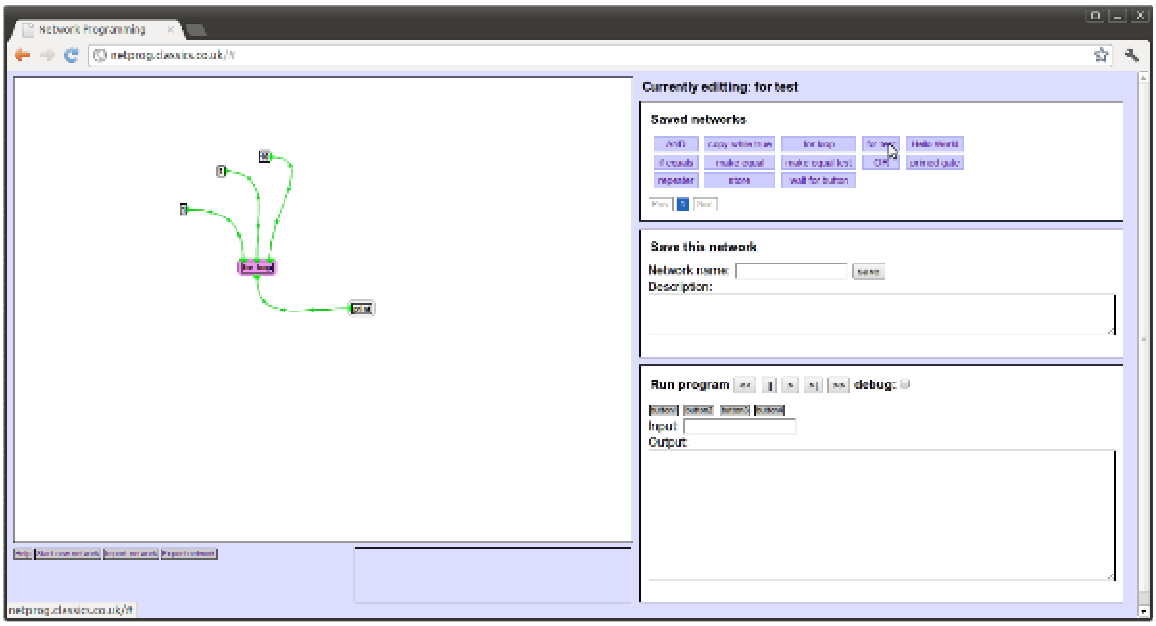
2.5.10 Run the “for loop” network



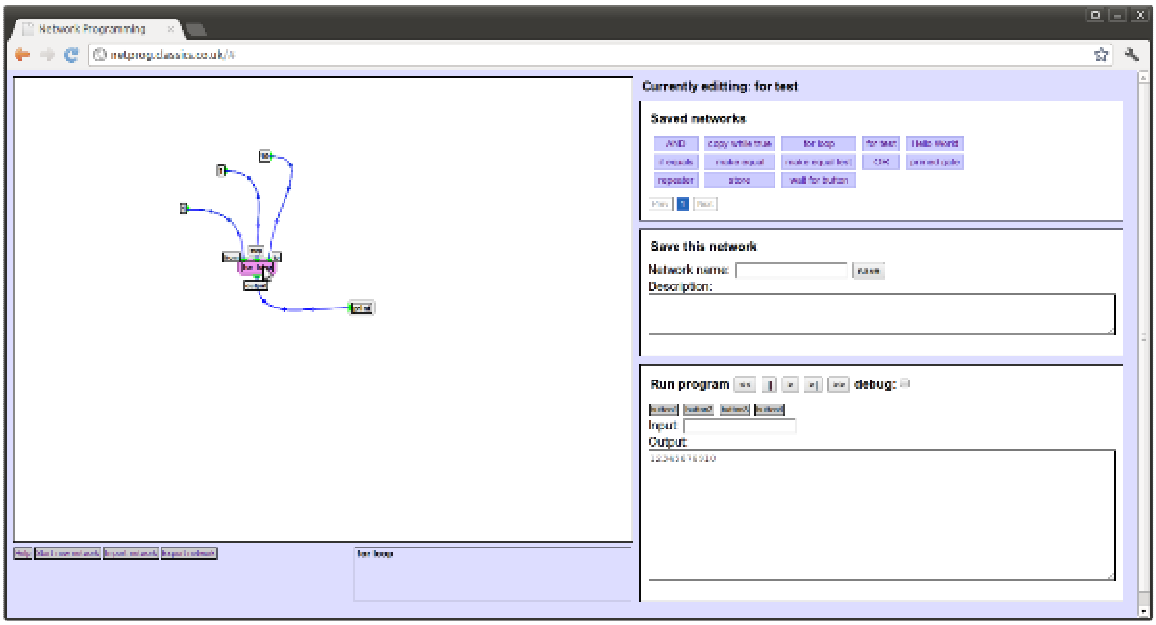
By clicking the ">" button in the "Run program" box you can test the for loop module. You will see the packets wiz round. You can pause execution with the "||" button and after pausing, step through execution one action at a time with the "|>" button. The execution of the network is artificially slowed down - you can speed up or slow down execution with the ">>" and "<<" buttons (the more you click them the faster/slower it goes).

2.5.11 Try the "for loop" module

Click the "for test" box in the "Saved networks" menu to load a network which uses the "for loop" module as shown below:



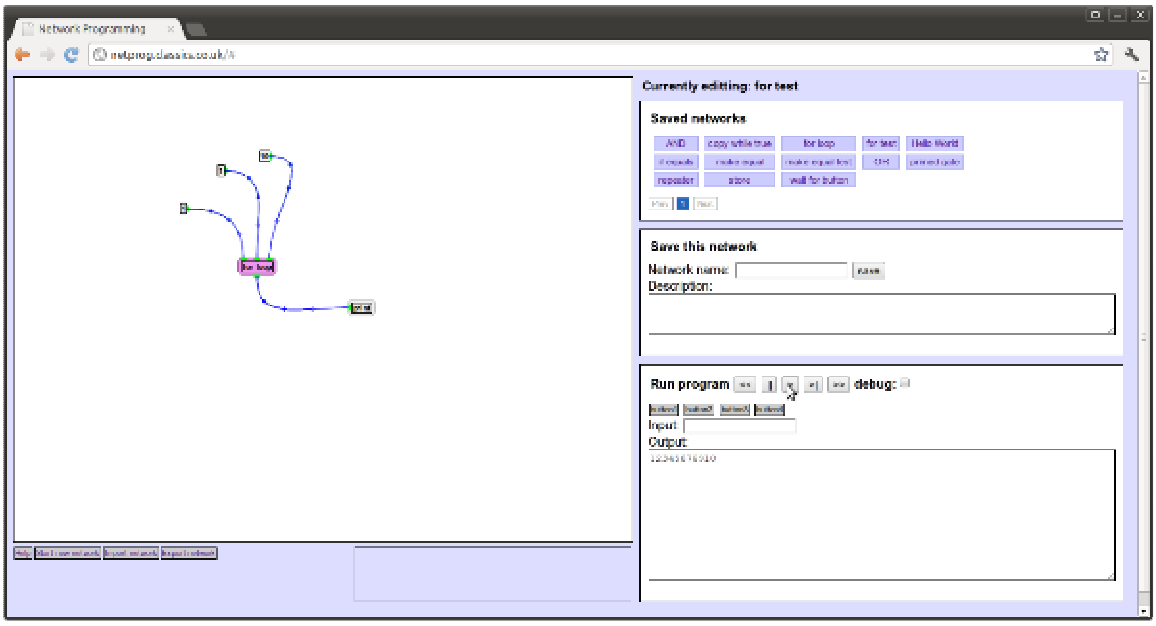
If you mouse over the "for loop" module node on the programming canvas the operation of the various inputs and outputs will be described by pop-up labels as shown below:



The descriptions of the inputs and outputs are taken from the names assigned to the input and output nodes in the "for loop" network design (see above). The position of these inputs and outputs around the edge of the node was also defined in the for loop network design (these are numbered 1 to 8 with 1 being on the left hand edge, 2,3,4 being along the top, 5 being on the right hand edge and 6,7,8 being along the bottom) – see node control menu shown in 2.5.9 above.

2.5.12 Run the "for loop" module

Click the ">" button in the "Run program" box to execute the program. The for loop will iterate from 1 to 10 in steps of 1 as shown below:



Because the value nodes (1,1 and 10) only produce a single packet each, the loop will only execute once. To run this multiple times, repeaters would need to be added in-line with each of the inputs.

3 Conclusion

I hope this document will have given a background to the aims of Netprog and an introduction to the workings of the proof of concept. This is obviously quite basic at this stage with very little scope for user interaction and no file I/O to name just 2 of the current limitations. However I hope that it will be sufficient to demonstrate that useful operations can be built from the small set of fundamental nodes.