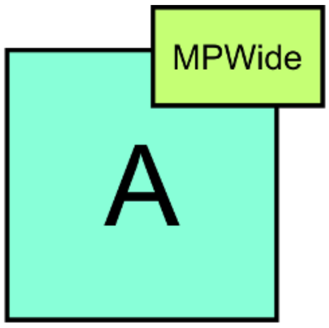
MPWide Tutorial

This tutorial has been made by Derek Groen. It is designed for MPWide 1.5d and later.

Part 1: First time setup



1. Visit http://castle.strw.leidenuniv.nl/software/mpwide.html and download the latest MPWide tarball file (you can also use 'wget http://castle.strw.leidenuniv.nl/~derek /mpwide-1.5d.tar.gz' directly if convenient).

2. Unzip the Tarball file on your local account on the grass1 or grass2 machine (or on any other Unix or Mac OSX machine of your liking). You should have obtained ar account for the grass cluster this morning, during the MUSCLE tutorial.

3. Go to the main directory where you unpacked MPWide, and type "make".

Once MPWide has been compiled, we can conduct several basic tests.

Using MPWide Responsibly

MPWide is a communication library which is quite low-level and fully geared towards delivering maximum performance. Similar to other Unix tools, it can be heavily customized but offers little in the way of automation, as such tasks are left to other tools (e.g., the application or framework using it). As such, MPWide can cause some disruption if used incorrectly on shared production
infrastructures. Here are some guidelines to ensure a correct and socially sensible usage of MPWide.

Ports

Most notably, when multiple users use MPWide on the same host, it is important that each user adopts its own range of ports, as overlapping port ranges can lead to unpredictable behavior. MPWTest by default adopts a base port number of 16256. You can find this on line 71 in Test.cpp:

```
int path_id = MPW_CreatePath(host, 16256, size);
```

If you are sharing one or more nodes with other people, please change that port to a different unique number. Preferably the number is somewhere between 6000 and 30000, and is not too close to that used by other people. The latter is important, because the port numbers used by you will increment up from that base number whenever you use multiple TCP streams.

A shorthand way to determine a unique base port number for you in this tutorial is to use the following recipe:

unique base port number = 16000 + ((the number of rows you in front of the one where you are sitting) * 8 + (the number of neighbours you have to your right)) * 256.

If you are unsure which ports are used by other MPWide instances or programs in general, you may be able to check this by typing "lsof -i". This lists all the ports that are currently in use.

Fair use of MPWide

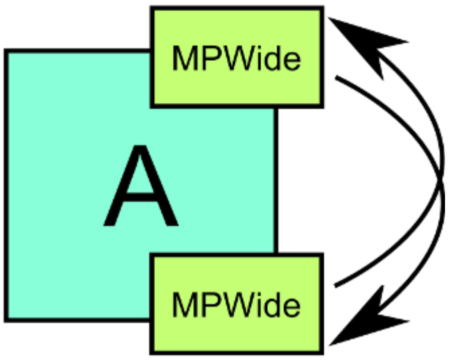
People seldom have exclusive use to clusters and supercomputers, and especially wide area networks. Since MPWide is able to deliver very high performance, extensive use of the library can influence the network performance for other users.

When you use MPWide within this tutorial, as well as for your own purposes on shared production resources, you must ensure that:

1. You use the software-based packet pacing at all times (it is turned on by default though).
2. You don't put high communication load on the various networks for extended periods of time.

For example, to use a single MPWTest script to pump over 16 MB or 64 MB per iteration is reasonable, as it will sleep for a second after each iteration. However, to run, say, 5 MPWTest scripts concurrently, each pumping over 1 GB or more per iteration will just result in noisy measurements on your side, and an impeded user experience for others.

Part 2: Local testing



This test involves using MPWide using the local loopback connection on your
machine. It's extremely basic, but it's a good way to check the MPWide functionality on its most basic level.

1. Open two terminals on the same machine.

2. In one terminal, switch to the MPWide directory and run:

```
./MPWTest 0.0.0.0 1 8
```

Here the first parameter is the endpoint of the MPWide test connection. A value of 0.0.0.0 means it won't try to connect to anything and will act as a server instead.

The second parameter indicates the number of tcp streams you choose to use.
In this case we choose to communicate over 1 stream only.

The third parameter indicates the size of the message (in kilobytes) we exchange. This should be set to the same value on both sides for this test case. Here we choose to exchange 8 kilobytes.

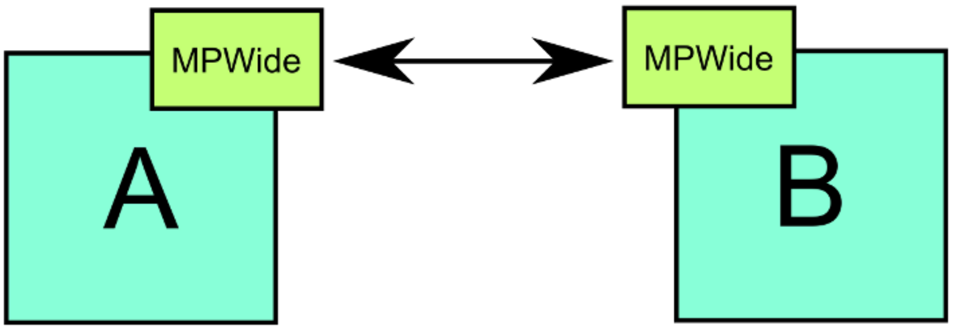
3. Leaving the first terminal open, also switch to the MPWide directory in the second terminal and run:

```
./MPWTest 127.0.0.1 1 8
```

At this point, output should start rolling down the screen, as messages are exchanged over your local loopback interface.

The source code of the MPWTest program can be found in Test.cpp. Feel free to take a look there, to see how MPWide works under the hood. MPWide can also be used from Python on several platforms, and a Python test script can be found in the Python subdirectory.

Part 3: MPWide between machines



1. Open two terminals on two *different* machines (which we call 'A' and 'B'). Ensure that the machines have direct connectivity (e.g., by performing a ping).

2. In the terminal connected to machine A, switch to the MPWide directory and run:

```
./MPWTest 0.0.0.0 1 8
```

3. In the terminal connected to machine B, switch to the MPWide directory and run:

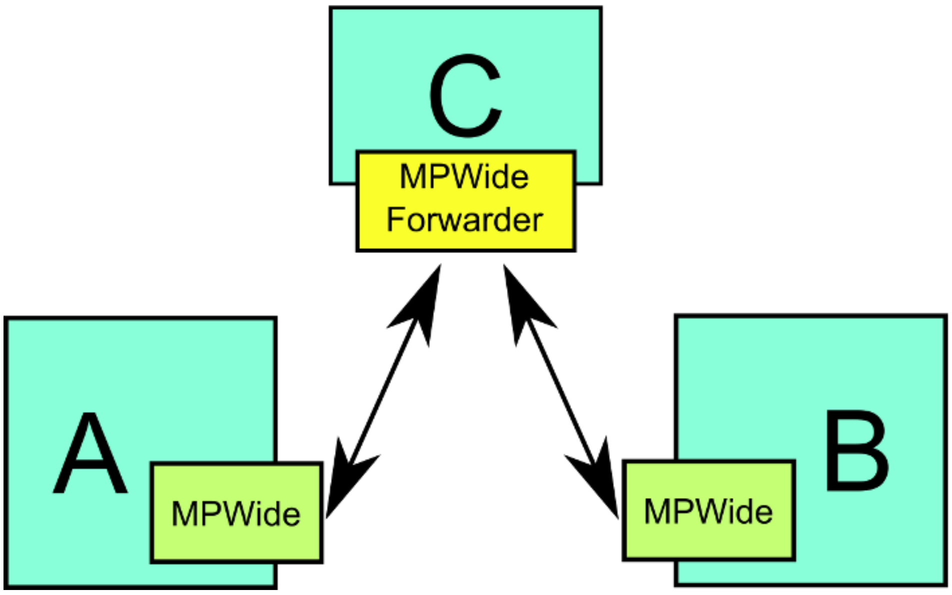
```
./MPWTest <hostname OR ip address of the other machine> 1 8
```

Does the MPWTest work now? Depending on the machines you picked, it might, or it might not. If the two end points do not connect, then it is quite likely that firewall policies are preventing you from connecting directly. The two most common ways to deal with such firewall constraints are:
i) Reverse the setup, running MPWTest in server mode on machine B and in client mode on machine A (so you run step 2 on machine B and step 3 on machine A)
ii) Identify a machine (named 'C') which you can connect to from both machine A and machine B. Then start a "Forwarder" process there. We will explain this in the next section.

4. If the basic test between machine described in step 2 and 3 works, try setting the message size (the third parameter) to a higher value, for example 8192. Then try to adjust the number of streams (the second parameter in each command, which is now set to 1) to a higher value, for example 4, or 16. Do you notice any difference in performance?

The performance impact of increasing the number of tcp streams is dependent on several factors:
i) Longer network paths generally tend to benefit from more streams, as the aggregate tcp buffer size tends to be higher.
ii) The beneficial effect tends to be higher on shared network paths (as multi-stream tcp trumps single-stream tcp), BUT:
iii) The beneficial effect tends to be lower on network paths that are concurrently used by other people doing this MPWide tutorial, as they incur an unpredictable background load!

Part 4: (Optional) Deploying the Forwarder to enable communication between Firewall-protected resources



The MPWide Forwarder is a program that provides tcp message forwarding using MPWide. It connects two port ranges from two different remote ip addresses, and does not proceed with data forwarding until both ends of a given forwarding channel are connected.

The Forwarder is a simple tool which is specifically developed to help users establish a network connection between two end points that cannot connect directly. Here we will explain the simplest case of using a Forwarder (as a rendez-vous server), but it is also straightforward to connect MPWide Forwarders to other Forwarders (in fact, we have used as many as 5 Forwarder processes in some distributed simulations).

The set up consists of the following steps:

1. Select a machine to which both of the end-points can connect, and install MPWide there if needed.

2. Configure the Forwarder on that machine by writing a forward.cfg file in the main MPWide directory. The layout of the configuration file is as follows:

```
<address1 ip address>
<address1 baseport>
<address2 ip address>
<address2 baseport>
<number of streams of the connection to be forwarded>
(...repeat this for any other forwarding route...)
done
```

For example, if you would like the Forwarder to act as a rendez-vous server on 1 stream using the default port (16256) on one side and a slightly different on the other (16320), you'd write

```
0.0.0.0
16256
0.0.0.0
16320
1
done
```

Importantly, the current version of the Forwarder does not forward any data communications until ALL connections are established.

3. Start the Forwarder using:

```
./Forwarder < forward.cfg
```

4. Ensure that MPWide uses different base ports on each end-point machines, and that those ports match with the ones you wrote in forward.cfg. In this example, you would leave the default base port (16256) on one machine, and change it to 16320 in Test.cpp on the other machine.

5. In each of the two terminals connected to the end-point machines, switch to the MPWide directory and run:

```
./MPWTest <hostname OR ip address of the rendez-vous server> 1
```

Part 5: (Optional) Ticket to Ride: The MPWide Edition

If you got this far you must either have a lot of time on your hands, or be really motivated! Not to worry though, because this part is actually a game. The rules are simple:

1. Connect two end points using MPWide and perform an MPWTest with a message size of no more than 512 Megabytes. Feel free to use Forwarders if they help!
2. Run MPWTest using no more than 64 streams, and measure the AVERAGE throughput in MB/s over 20 iterations.
3. Measure or estimate the length of the path between the two end points, for example by using Google Maps.
4. Calculate your track score. Your Score = Throughput in MB/s x Distance in KM (anything less than 1 km counts as 1).

Please not that you are allowed to overrule the autotuner to squeeze out even more performance. The full syntax of MPWTest is:

```
./MPWTest <ip address of other endpoint> <channels> <buffer [kb]> <pacing [MB/s]> <tcpwin [bytes]>
```

Here, you may be able to place custom values in the last two parameters to squeeze out that extra bit of performance.

The Mapper project receives funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-261507.