Install Mono

Here are my instructions for getting mono on your machine:

```
1) Create 'bin' directory.
  mkdir $HOME/bin;
2) Download current Mono (3.2.1):
  mkdir Mono; cd Mono;
  wget http://download.mono-project.com/sources/mono/mono-3.2.1.tar.bz2;
  tar –xf mono-3.2.1.tar.bz2 ;
  cd mono-3.2.1;
5) Compile and install Mono:
  ./configure --prefix=$HOME/bin LDFLAGS=-L$HOME/bin/lib \\
                        1>LOG.configure 2>ERR.configure;
  make 1>LOG.make 2>ERR.make;
  make install 1>LOG.make_install 2>ERR.make_install;
```

test whether command "mono" and "mcs" work in the folder.

Set up local environment

(Note:) this step may not work for certain clusters, but not affect later steps
Add this to .bashrc:

```
   # set PATH so it includes user's private bin if it exists
   if [ -d ~/bin ] ; then
       PATH=~/bin/bin:"${PATH}"
   fi
```

Make sure that .bashrc is sourced from .profile (it probably already is):

```
# include .bashrc if it exists
if [ -f ~/.bashrc ]; then
   source ~/.bashrc
fi
```

Test that it can find mono

```
~$ which mono
   ~/bin/bin/mono
```

Prepare newest Vsync.cs for experiment

```
Download Vsync.cs
mcs –target:library Vsync.cs
cp Vsync.dll ~/bin/lib
```

Or, write your Vsync application and compile with Vsync, this way:

```
mcs myApp.cs Vsync.cs
```

Run it like this:

```
mono myApp.exe
```

There are some situations in which you may wish to create a single binary with everything self-contained in it (this is best for clusters with batch schedulers, for example). To do that use mkbundle. It isn't at all hard or complicated to use. For example:

```
# be sure to use the directories where mono is installed.
export LD_LIBRARY_PATH=~/bin/bin:~/bin/lib:$LD_LIBRARY_PATH
PATH=~/bin/bin:$PATH
export PKG_CONFIG_PATH=~/bin/lib/pkgconfig

mcs VsyncTest.cs Vsync.cs -r:System.dll -r:Microsoft.CSharp.dll \\
                –r:Mono.Posix.dll -d__MonoCS__
mkbundle --static -o IdaTester IdaTester.exe --deps
rm IdaTester.exe
```

If at runtime your program (IdaTester in this example) complains that it can't find a required library, create symbolic links to the files in bin/lib in the directory within which you run your program. We've had problems of this kind on the batch launcher at LLNL and resolved them in this way. The issue centers on the value of LD_LIBRARY_PATH at runtime and you can also resolve it by making sure this path has the correct library path in it for the nodes on which your program is going to run. Obviously, if dynamically linked libraries are not accessible at runtime, or the wrong versions are present on some nodes, or the path names vary, you'll have challenges at this step.

More details:

mono application => Native application bundling information(mkbundle)
At Mono "Bundles" section

http://www.mono-project.com/Guide:Running_Mono_Applications

mono linker and mkbundle information
http://www.mono-project.com/Linker

mkbundle man page
http://linux.die.net/man/1/mkbundle

Sending run-time variables to your program:

Csh users can use the shell "setenv" command to pass values in to the various Vsync environment variables (VSYNC_HOSTS, VSYNC_UNICAST_ONLY, etc).

Bash users should use the "export" command instead. This is because we have seen bash append an equal sign ("=") to values set using setenv, although the command itself is available.

One more minor thing: Mono has a very long-standing bug that will cause it to print "Got a bad AF_PACKET" twice (it will reqport two non-equal lengths, perhaps 16 or 20 and 8, after the AF_PACKET).

This is observed when launching Vsync applications on certain version of Linux. If it occurs on your system, you can safely comment out the line that prints this error message, which you will find in:

```
mcs/class/System/System.Net.NetworkInformation/NetworkInterface.cs
```

Then remake and reinstall mono and it will behave itself when Vsync launches.