PNNL-SA-201800
PNNL-18428, Rev. 4

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

FLOWER Build Guide

FLOWER Version 06 (flr06)

June 2017

DS Curtis
LMS Curtis

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC06-76RL01830
with Battelle Memorial Institute

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <http://www.ntis.gov/about/form.aspx>
Online ordering: http://www.ntis.gov

This document was printed on recycled paper.
(8/2010)

PNNL-SA-201800
PNNL-18428, Rev. 4

FLOWER Build Guide

DS Curtis
LMS Curtis

June 2017

Prepared for
the U.S. Department of Energy
under Contract DE-AC06-76RL01830
with Battelle Memorial Institute

Pacific Northwest National Laboratory
Richland, Washington 99352

Acknowledgments

 PNNL-SA-120800

Battelle IPID: 17042-E

Prime Contract No.: DE-AC06-76RL01830 with Battelle Memorial Institute

Acronyms and Abbreviations

FLOWER	FLOW analyzer
GCC	GNU Compiler Collection
PNNL	Pacific Northwest National Laboratory
QSFP	Quad Small Form-factor Pluggable
RPM	RPM Package Manager

Contents

Acknowledgments iii
Acronyms and Abbreviations v
1.0 Introduction 1.1
2.0 Requirements 2.1
2.1 Hardware 2.1
2.2 Operating System 2.1
2.3 Developer Environment 2.1
2.3.1 Operating System (RHEL/CentOS 64-bit) development environment 2.1
2.3.2 PCAP development and runtime libraries 2.1
2.3.3 GNU C++11 or greater 2.2
2.3.4 Boost libraries 2.2
3.0 Building FLOWER 3.1
4.0 Testing FLOWER 4.1
5.0 Building RPMS 5.1
5.1.1 Build GCC RPM 5.1
5.1.2 Build Boost RPM 5.1
5.1.3 Build FLOWER RPM 5.2
6.0 Installing RPMs 6.3
Appendix A GCC Build Script A.1
Appendix B Boost Build B.1

1.0 Introduction

The FLOWER (FLOW analyzER) software was developed at PNNL and is intended for deployment on application sensors.

This guide is intended to assist an experienced developer to build and install the FLOWER software to deploy on a FLOWER appliance sensor. The instructions are written in a format that assumes a basic knowledge of Linux and C/C++ development. Short instructions are provided for each step along with command line or script references to complete the build. If you have any questions about the build, please contact FLOWER support at flower-support@pnnl.gov.

The FLOWER application is designed to start at boot time as a daemon process to summarize network flows.

FLOWER can also be used to read packets from a network interface or from packet capture (pcap) files that can be processed with the libpcap system library. See the FLOWER_Ops_Guide document for more detailed information.

2.0 Requirements

2.1 Hardware

The FLOWER application requires a 64-bit Intel based system with dual quad-core processors with a minimum of 4 GB of memory and 20 GB of disk.

The system requires a 1 Gigabit CAT5 or better Ethernet interface. For testing, it is recommended to have an additional 1 Gigabit CAT 5 or better Ethernet interface that is connected to a dedicated hub or switch. The hub/switch is needed to give the Ethernet interface an active link that can be used in isolation for testing. FLOWER has been tested and verified on an Intel XL710 40 Gigabit QSFP+ interface without any problems.

2.2 Operating System

The FLOWER application can be built and installed on Red Hat Enterprise Linux or CentOS 5.2 or newer.

- It is recommended that you build and install on RHEL/CentOS 7.x.

The Operating System can be installed to run as 32-bit only, 32-bit/64-bit mixed, or 64-bit only mode. You must build on a system that runs the same mode as the system you will be running the application on.

- It is recommended that you build only 64-bit mode. The previous modes are for legacy systems only.

2.3 Developer Environment

The FLOWER application requires a Linux development environment, a C++ compiler and Boost libraries. It is required to use at least the versions listed below or better.

2.3.1 Operating System (RHEL/CentOS 64-bit) development environment

To ensure that you have all the necessary development tools, install the standard development environment using the following command:

```
# yum groupinstall "Development Tools"
```

2.3.2 PCAP development and runtime libraries

The PCAP development and runtime libraries are required to build FLOWER. To install them on an RHEL/CentOS system, run the following commands:

```
# yum install -y libpcap-devel libpcap
```

2.3.3 GNU C++11 or greater

Appendix A contains a script to download and build GCC. It should work in most environments to simply copy, paste, and run.

Download GNU C and C++ version 5.3 from https://gcc.gnu.org/releases.html.

NOTE: While newer versions are available, FLOWER has been tested extensively with version 5.3, so this is the recommended version.

NOTE: Do not use the default compiler with RHEL/CentOS as a more recent build is required for FLOWER to properly build.

Set the build parameters to include:

```
--enable-shared
--enable-threads=posix
--enable-clocale=gnu
--disable-bootstrap
--disable-multilib
```

Build the GNU GCC compiler tools and install them in `/usr/local/gcc`.

Build both the C and C++ executables and libraries.

2.3.4 Boost libraries

Appendix B contains a script to download and build Boost. It should work in most environments to simply copy, paste and run.

Download Boost version 1.60 or greater from http://www.boost.org/users/history.

NOTE: While newer versions are available, FLOWER has been tested extensively with version 1.60, so this is the recommended version.

Important: Ensure that you are using the GCC 5.3 compiler tools and not the built in C compiler tools. To ensure that you are using the proper compiler, do the following:

- From the boost_1_60_0 directory, run the following command:
  ```
  # echo "using gcc : 5.3 : /usr/local/gcc/bin/g++ ; " >>
  tools/build/src/user-config.jam
  ```
- Make sure you have the `--toolset=gcc` defined when you run your b2 or bjam command.
- Set the following environmental variables:
  ```
  unset SSH_ASKPASS
  export PATH=/usr/local/gcc/bin:/usr/local/bin:$PATH
  export
  LD_LIBRARY_PATH=/usr/local/gcc/lib:/usr/local/gcc/lib64:/usr/local/l
  ib:/usr/lib:$LD_LIBRARY_PATH
  ```

```
export CC=/usr/local/gcc/bin/gcc
export CXX=/usr/local/gcc/bin/g++
```

- Set the build parameters to include:
  ```
  link=shared
  address-model=64
  runtime-link=shared
  threading=multi
  ```
- Build the libraries and install them in `/usr/local/boost`.

3.0 Building FLOWER

This section contains the instructions that build the FLOWER software. It assumes that all requirements in the previous section have been met, including the GNU C and C++ compiler tools and Boost are built and installing in `/usr/local`.

Download FLOWER 06 or greater from https://stash.pnl.gov.

```
# git clone https://$USER@stash.pnnl.gov/scm/flower/flower.git
```

Important: Ensure that the correct version of Boost and GCC executables and libraries are in your path.

Set the following environmental variables:

```
unset SSH_ASKPASS
export PATH=/usr/local/gcc/bin:/usr/local/bin:$PATH
export
LD_LIBRARY_PATH=/usr/local/boost/lib:/usr/local/gcc/lib:/usr/local/g
cc/lib64:/usr/local/lib:/usr/lib:$LD_LIBRARY_PATH
export CC=/usr/local/gcc/bin/gcc
export CXX=/usr/local/gcc/bin/g++
```

It is recommended that you use the included rebuild_from_scratch.sh script to build the FLOWER executables. Prior to running the script, edit and make the following changes:

- Change BOOST_HOME to `/usr/local/boost`
- Change PREFIX to the location that you would like to install FLOWER. It is recommended that you use `/opt/flower`.

After you run `rebuild_from_scratch.sh` with no errors, run "`make install`". This will install FLOWER in `/opt/flower`.

4.0 Testing FLOWER

After FLOWER is built and installed, test the application as follows:

- Add flower to your path
  ```
  # export PATH=/opt/flower/bin:$PATH
  ```
- Make sure the executable runs without error.
  ```
  # flower –v
  ```

The output should show detailed version information, including the version of FLOWER, the version of GCC, boost, and pcap that FLOWER was built with and the build date.

- View the FLOWER help information.
  ```
  # flower –h
  ```

This should show the FLOWER help information.

- Run the FLOWER test suite

  Identify a network interface that is up to run the tests against (e.g. eth0)

  Set the network interface variable for the test suite
  ```
  # export FLOWER_NIC=eth0
  ```

  Run the test suite from the source code directory
  ```
  # src/flower/api/test/flower.test
  ```

  You should see output similar to the following:
  ```
  Total Tests Files  = 12
  Total Test  Cases  = 62
  Total Tests Failed = 0 **
  Exit with Status   = 0 => PASSED
  ```

If these both work, proceed to Configuring FLOWER.

If any of the above tests do not work, the build did not complete correctly. If there are problems, potential areas to check are as follows:

- Ensure that the correct version of GCC is being used.
- Ensure that the correct version of Boost is being used.
- Ensure that the correct version of GCC was used during the build of Boost.
- Ensure that the PCAP and PCAP-devel libraries were installed during the build.

5.0 Building RPMS

This section includes the instructions to build RPMS.

Prior to building, ensure that you have the RPM build utilities installed on your machine.

```
# yum install rpm-build
# yum install rpm-build-libs
```

Create directories for RPM building in your home directory.

```
# mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
# echo '%_topdir %(echo $HOME)/rpmbuild' > ~/.rpmmacros
```

5.1.1 Build GCC RPM

Prepare the GCC components in preparation to package.

- Build the GCC compiler tools in `/usr/local/gcc` as instructed in GNU C++11 or greater.
  ```
  # mv /usr/local/gcc /usr/local/gcc-5.3
  ```
- Create a tar archive of this directory
  ```
  # cd /usr/local
  # tar cf gcc-5.3.tar gcc-5.3
  ```
- Copy this file into your RPM build SOURCES directory.
  ```
  # cp gcc-5.3.tar ~/rpmbuild/SOURCES
  ```
- Copy the `gcc.spec` file from the `flower/support/build/rpm` directory into the `~/rpmbuild/SPECS` directory.
  ```
  # cd ~/dev/flower
  # cp flower/support/build/rpm/gcc.spec ~/rpmbuild/SPECS
  ```
- Run the RPM build for GCC.
  ```
  # cd ~/rpmbuild/SPECS
  # rpmbuild –ba gcc.spec
  ```

You should now have the RPM: `~/rpmbuild/RPMS/x86_64/gcc-5.3-0.x86-64.rpm`

5.1.2 Build Boost RPM

Prepare the Boost components in preparation to package.

- Build the Boost libraries in `/usr/local/boost` as instructed in Boost libraries.
  ```
  # mv /usr/local/boost /usr/local/boost-1.60
  ```
- Create a tar archive of this directory
  ```
  # cd /usr/local
  # tar cf boost-1.60.tar boost-1.60
  ```
- Copy this file into your RPM build SOURCES directory.
  ```
  # cp boost-1.60.tar ~/rpmbuild/SOURCES
  ```

Copy the `boost.spec` file from the `flower/support/build/rpm` directory into the `~/rpmbuild/SPECS` directory.

```
# cd ~/dev
# cp flower/support/build/rpm/boost.spec ~/rpmbuild/SPECS
```

Run the RPM build for Boost.

```
# cd ~/rpmbuild/SPECS
# rpmbuild –ba boost.spec
```

You should now have the `RPM: ~/rpmbuild/RPMS/x86_64/boost-1.60-0.x86-64.rpm`

5.1.3 Build FLOWER RPM

Prepare the FLOWER components in preparation to package.

- Build the FLOWER software in `/opt/flower` as instructed in Building FLOWER.
- Copy this directory into your RPM build SOURCES directory and rename to flower-6.0.

```
# cd ~/dev
# cp -R flower ~/rpmbuild/SOURCES/flower-6.0
```

- Copy the flower support files from your flower source code into your RPM build SOURCES directory.

```
# cd ~/dev
# cp flower/support/build/rpm/flower-support.tar ~/rpmbuild/SOURCES
```

- Untar the support files into flower directory.

```
# cd ~/rpmbuild/SOURCES/flower-6.0
# tar xf ../flower-support.tar
# cd ..
```

- Create a tar archive of this directory.

```
# tar cf flower-6.0.tar flower-6.0
```

Copy the `flower.spec` file from the `flower/support/build/rpm` directory into the `~/rpmbuild/SPECS directory`.

```
# cd ~/dev
# cp flower/support/build/rpm/flower.spec ~/rpmbuild/SPECS
```

Run the RPM build for FLOWER.

```
# cd ~/rpmbuild/SPECS
# rpmbuild –ba flower.spec
```

You should now have the `RPM: ~/rpmbuild/RPMS/x86_64/flower-6.0-0.x86-64.rpm`

6.0 Installing RPMs

This section describes how to install the required RPMs that have been built in the previous sections.

Install GCC

```
# rpm –i ~/rpmbuild/RPMS/x86_64/gcc-5.3-0.x86-64.rpm
```

Install Boost

```
# rpm –i ~/rpmbuild/RPMS/x86_64/boost-1.60-0.x86-64.rpm
```

Install FLOWER

```
# rpm –i ~/rpmbuild/RPMS/x86_64/flower-6.0-0.x86-64.rpm
```

Appendix A

GCC Build Script

```
#!/bin/bash

export GCC_VERSION=5.3.0
export PROCS=`grep -c ^processor /proc/cpuinfo` || 1
export SED=sed


cd /usr/local

if [ ! -f "gcc-$GCC_VERSION.tar.gz" ]; then
  echo "################################################"
  echo "#         DOWNLOADING gcc_$GCC_VERSION"
  echo "################################################"
  wget -q http://www.netgull.com/gcc/releases/gcc-$GCC_VERSION/gcc-$GCC_VERSION.tar.gz
fi

if [ ! -d "gcc-$GCC_VERSION" ]; then
  echo "################################################"
  echo "#         UNPACKING gcc_$GCC_VERSION"
  echo "################################################"
  tar xzf gcc-$GCC_VERSION.tar.gz
fi

cd gcc-$GCC_VERSION                                     || exit 1

echo "################################################"
echo "#         DOWNLOADING Prerequisites"
echo "################################################"
#  NOTE:  This installs the mpc, mpfr, gmp, and zlib prereqs if needed
./contrib/download_prerequisites                        || exit 1
cd ..


#  NOTE: This creates a build directory so we don't overwrite the original source
mkdir -p gcc_$GCC_VERSION-build                         || exit 1
cd       gcc_$GCC_VERSION-build                         || exit 1


echo "################################################"
echo "#         CONFIGURE"
echo "################################################"
../gcc-$GCC_VERSION/configure                           \
    --prefix=/usr/local/gcc                             \
    --libdir=/usr/local/gcc/lib                         \
    --enable-shared                                     \
    --enable-threads=posix                              \
    --enable-clocale=gnu                                \
    --disable-bootstrap                                 \
    --disable-multilib                                  \
    --with-default-libstdcxx-abi=gcc4-compatible        \
    --enable-languages=c,c++                            || (cat config.log && exit 1)

echo "################################################"
echo "#         MAKE"
echo "################################################"
ulimit -s 32768                                         || exit 1
make -j $PROCS                                          || exit 1

make install                                            || exit 1
ldconfig

exit 0
```

Appendix B

Boost Build

```
#!/bin/bash

#  NOTE: Later versions may have new or different build and installation instructions.
#  we are using the bash shell on Linux
#  we will build the libraries in BUILD_DIR, /var/tmp
#  we will install the libraries in INSTALL_DIR, /usr/local
#  we are building the 64-bit versions of the libaries
#  we are building "shared" (LIB_MODE) version of the libraries
#  we are building "debug" (BOOST_VARIANT) versions of the libaries
export CXX_PATH=/usr/local/gcc
export INSTALL_DIR=/usr/local
export LIB_MODE="shared"              # Change to "shared" for shared libraries
export BOOST_VARIANT="debug"          # Change to "debug"  for debug  libraries
export VERSION=1.60.0
export SVERSION=`echo $VERSION | sed -e "s/\./_/g"`
export PROCS=`grep -c ^processor /proc/cpuinfo` || 1

cd /usr/local

if [ ! -f "boost_$SVERSION.tar.gz" ]; then
  echo "################################################"
  echo "#         DOWNLOADING boost_$SVERSION"
  echo "################################################"
  echo "wget --progress=bar:force
'https://sourceforge.net/projects/boost/files/boost/1.60.0/boost_1_60_0.tar.gz'"
        wget --progress=bar:force
'https://sourceforge.net/projects/boost/files/boost/1.60.0/boost_1_60_0.tar.gz'
fi

if [ ! -d "boost_$SVERSION" ]; then
  if [ ! -f "boost_$SVERSION.tar.gz" ]; then
    echo "################################################"
    echo "#         ERROR MISSING boost_$SVERSION.tar.gz"
    echo "################################################"
  else
    echo "################################################"
    echo "#         UNPACKING boost_$SVERSION"
    echo "################################################"
    echo "tar xzf boost_$SVERSION.tar.gz"
          tar xzf boost_$SVERSION.tar.gz  || exit 1
  fi
fi

echo "cd boost_$SVERSION"
      cd boost_$SVERSION  || exit 1

echo "./bootstrap.sh --with-libraries=all -without-libraries=mpi --without-icu"
      ./bootstrap.sh --with-libraries=all -without-libraries=mpi --without-icu    ||
exit 1

echo "./bjam -j$PROCS --layout=tagged --without-mpi --without-python --
builddir=../build_boost --build-type=co
mplete toolset=gcc --prefix=$INSTALL_DIR/boost optimization=speed link=$LIB_MODE
address-model=64 runtime-link
=$LIB_MODE variant=${BOOST_VARIANT} threading=multi install"
      ./bjam -j$PROCS --layout=tagged --without-mpi --without-python --
builddir=../build_boost --build-type=complete toolset=gcc --prefix=$INSTALL_DIR/boost
optimization=speed link=$LIB_MODE address-model=64 runtime-link=$LIB_MODE
variant=${BOOST_VARIANT} threading=multi install > build_boost.out 2>&1

exit 0
```

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

U.S. DEPARTMENT OF
ENERGY

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

www.pnnl.gov