Veins Development User Manual V0.4

1. Installation

System Requirements

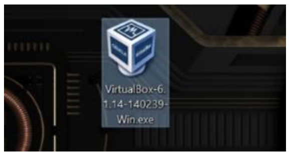
- 2 GHz dual core processor
- 4 GB RAM
- 25 GB of hard-drive space

*If using a newer AMD CPU virtualization is disabled by default. To have access to 64 bit Ubuntu virtualization go into your BIOS settings and enable SVM (mine was under overclocking settings).

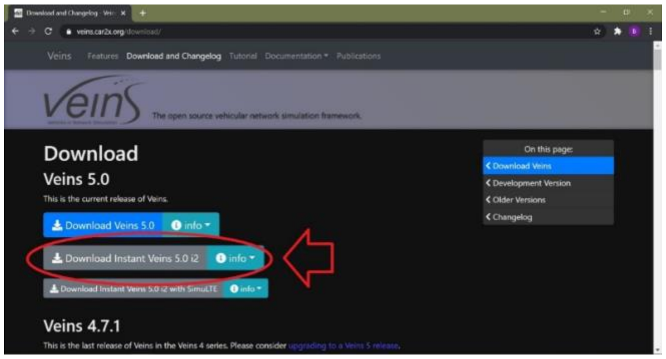
There are two ways of installing the simulation environment to your system. First is to download Instant Veins and run it as a Virtual Machine. This way is faster and easier. The second way is to manually install all of the required software in a ubuntu operating system. The installations steps for both methods are listed below starting with Instant Veins.

How to install Instant Veins

1. Download the most current version of VirtualBox for your operating system at https://www.virtualbox.org/wiki/Downloads. Once downloaded, locate the VirtualBox installation software on your device and run the setup (by double-clicking it) with default settings.



2. Go to https://veins.car2x.org/download/ and download the latest version of Instant Veins



Download

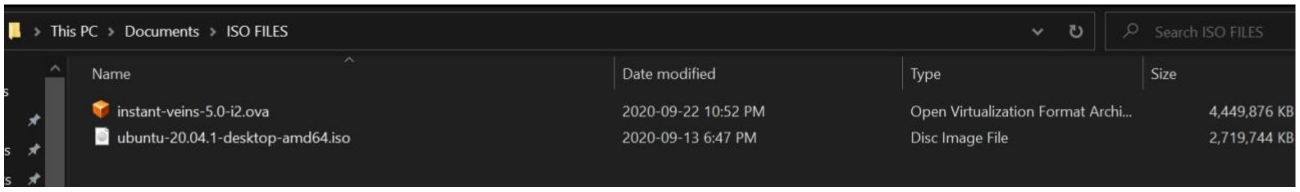
Veins 5.0

This is the current release of Veins.

Veins 4.7.1

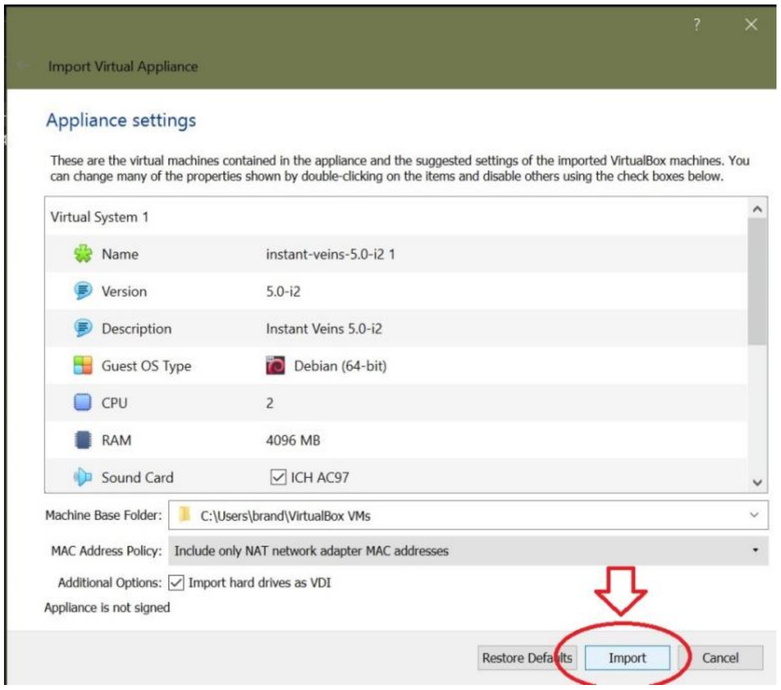
This is the last release of Veins in the Veins 4 series. Please consider upgrading to a Veins 5 release.

3. Once downloaded, move the Instant-veins-x.x-xx.ova to a safe location on your device


This PC > Documents > ISO FILES

Name	Date modified	Type	Size
instant-veins-5.0-i2.ova	2020-09-22 10:52 PM	Open Virtualization Format Archi...	4,449,876 KB
ubuntu-20.04.1-desktop-amd64.iso	2020-09-13 6:47 PM	Disc Image File	2,719,744 KB

4. Double-Click on Instant-veins-x.x-xx.ova and select **Import**.


Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1

Name	instant-veins-5.0-i2 1
Version	5.0-i2
Description	Instant Veins 5.0-i2
Guest OS Type	Debian (64-bit)
CPU	2
RAM	4096 MB
Sound Card	☑ ICH AC97

Machine Base Folder: C:\Users\brand\VirtualBox VMs

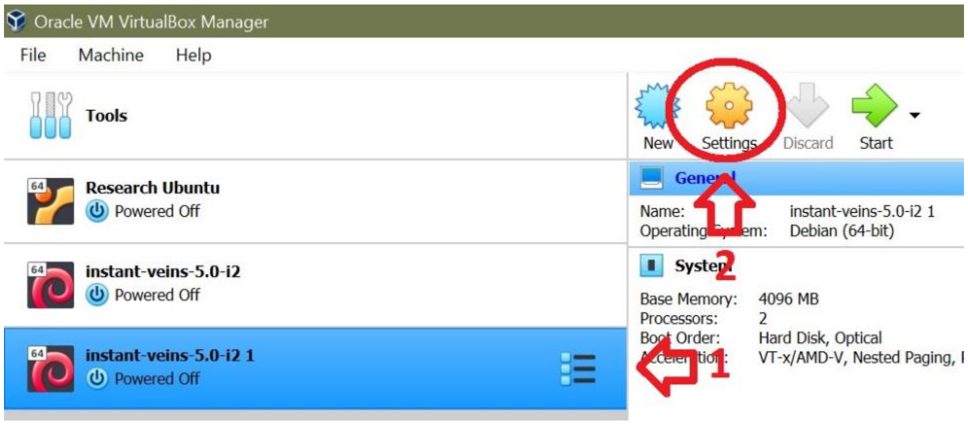
MAC Address Policy: Include only NAT network adapter MAC addresses

Additional Options: ☑ Import hard drives as VDI

Appliance is not signed

Once imported, open VM VirtualBox (it may automatically open once imported).

5. Before running, select the newly installed Instant Veins and select **Settings** at the top of the screen



Research Ubuntu
Powered Off

instant-veins-5.0-i2
Powered Off

instant-veins-5.0-i2 1
Powered Off

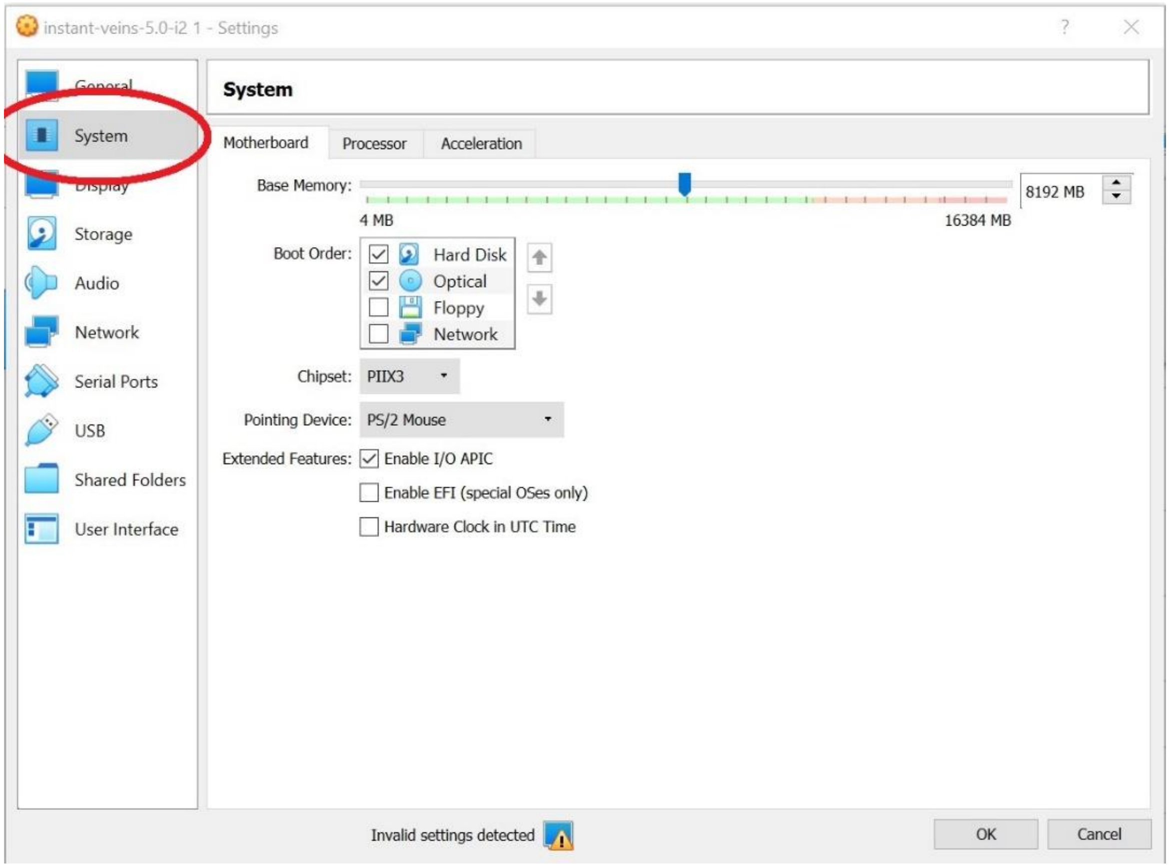
General

Name: instant-veins-5.0-i2 1
Operating System: Debian (64-bit)

System

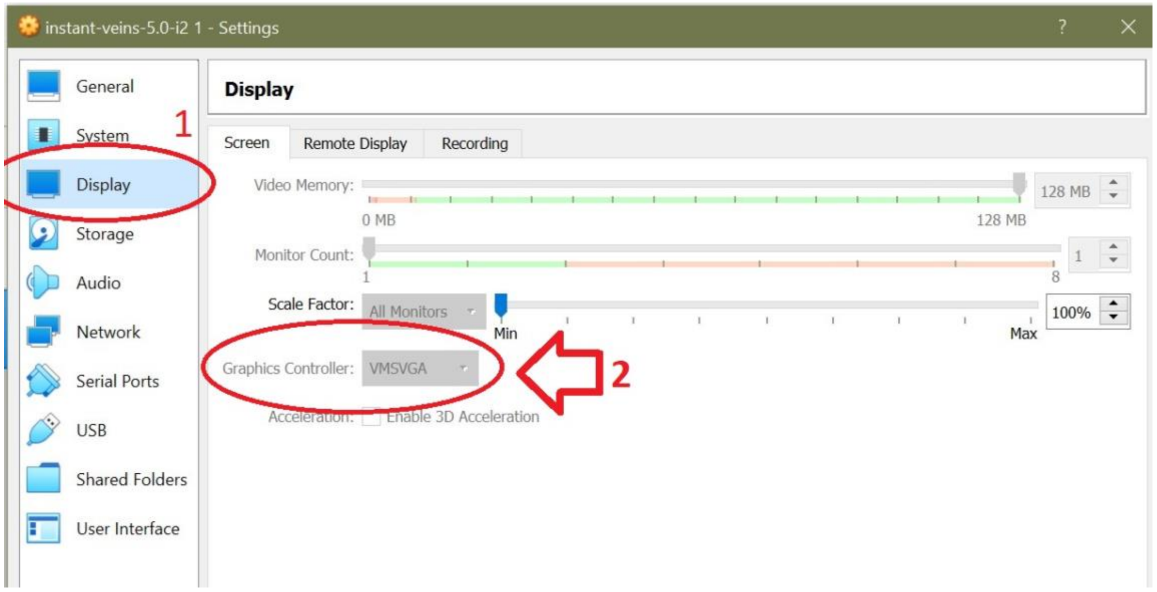
Base Memory: 4096 MB
Processors: 2
Boot Order: Hard Disk, Optical
Acceleration: VT-x/AMD-V, Nested Paging, I

6. Go to **System** and select an adequate amount of RAM and Cores to dedicate for your machine (make sure to not select all of it since your device still needs to function).



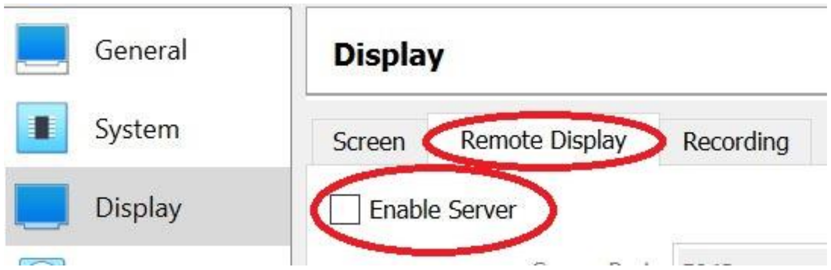
Base Memory: 8192 MB (4 MB – 16384 MB)
Boot Order: ☑ Hard Disk, ☑ Optical, ☐ Floppy, ☐ Network
Chipset: PIIX3
Pointing Device: PS/2 Mouse
Extended Features: ☑ Enable I/O APIC
☐ Enable EFI (special OSes only)
☐ Hardware Clock in UTC Time
Invalid settings detected

7. Then go to **Display** and changes **Graphics Controller** to **VMSVGA.** Then select an adequate amount of Video Memory for your device.

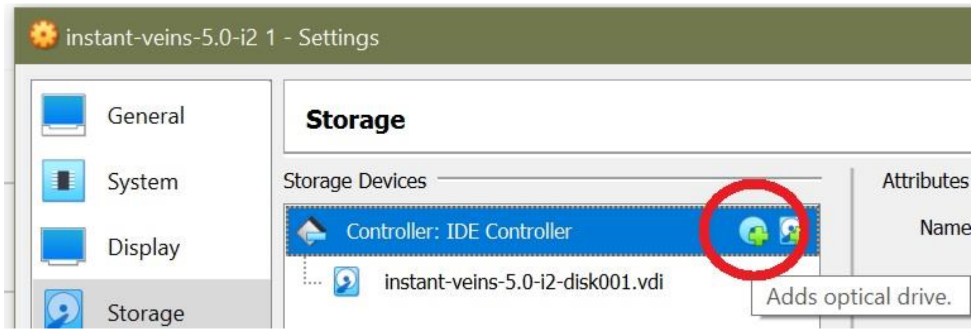


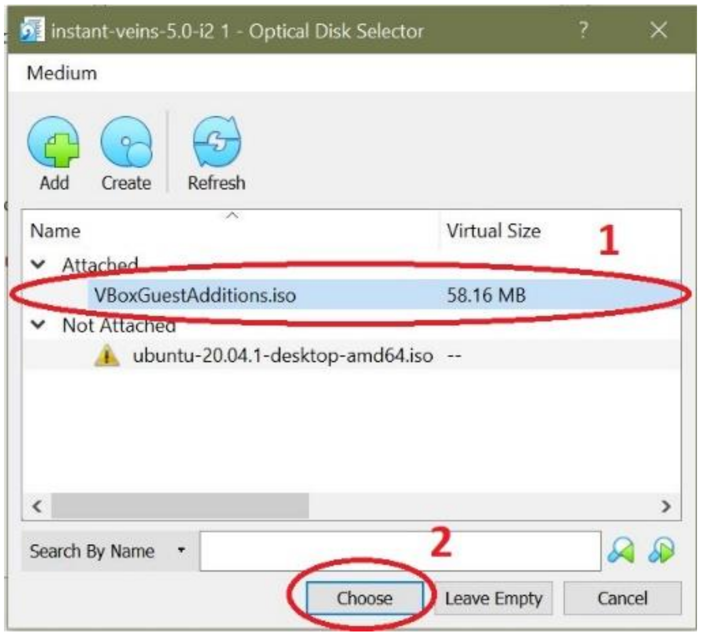
Video Memory: 128 MB (0 MB – 128 MB)
Monitor Count: 1 (1 – 8)
Scale Factor: All Monitors, 100% (Min – Max)
Graphics Controller: VMSVGA
Acceleration: ☐ Enable 3D Acceleration

8. While under **Display**, select the **Remote Display** tab and uncheck **Enable Server**



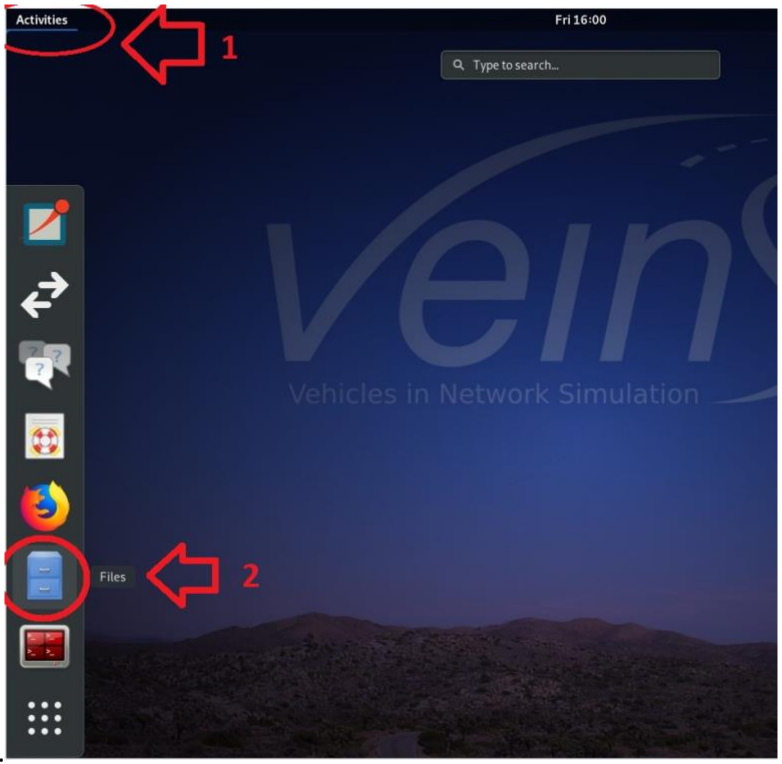
9. Finally, go to **Storage**, select the **add Optical Disk** icon, and choose VBoxGuestAdditions.iso. Once all these settings have been changed, click OK in the bottom right to close settings.


instant-veins-5.0-i2 1 - Settings
General
System
Display
Storage
Storage
Storage Devices
Controller: IDE Controller
instant-veins-5.0-i2-disk001.vdi
Adds optical drive.
Attributes
Name

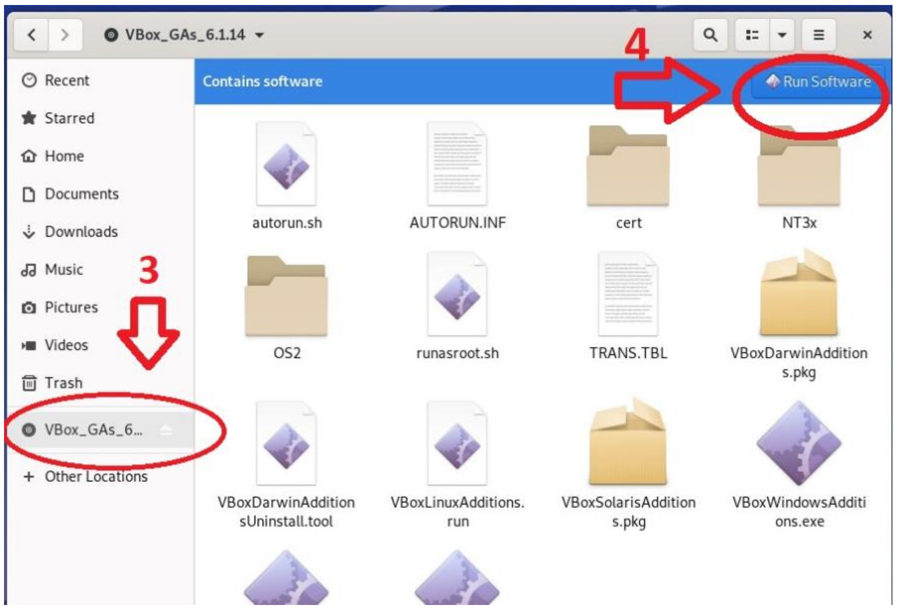


10. Select **Start** or double click the Instant Veins installation to enter the VM. At this point the VM is completely installed and ready for use, the remaining steps are for changing the screen size of the VM.

11. Once at your Instant Veins desktop, select **Activities** > **Files** > **VBox_GAs** disk and select **Run Software** at the top

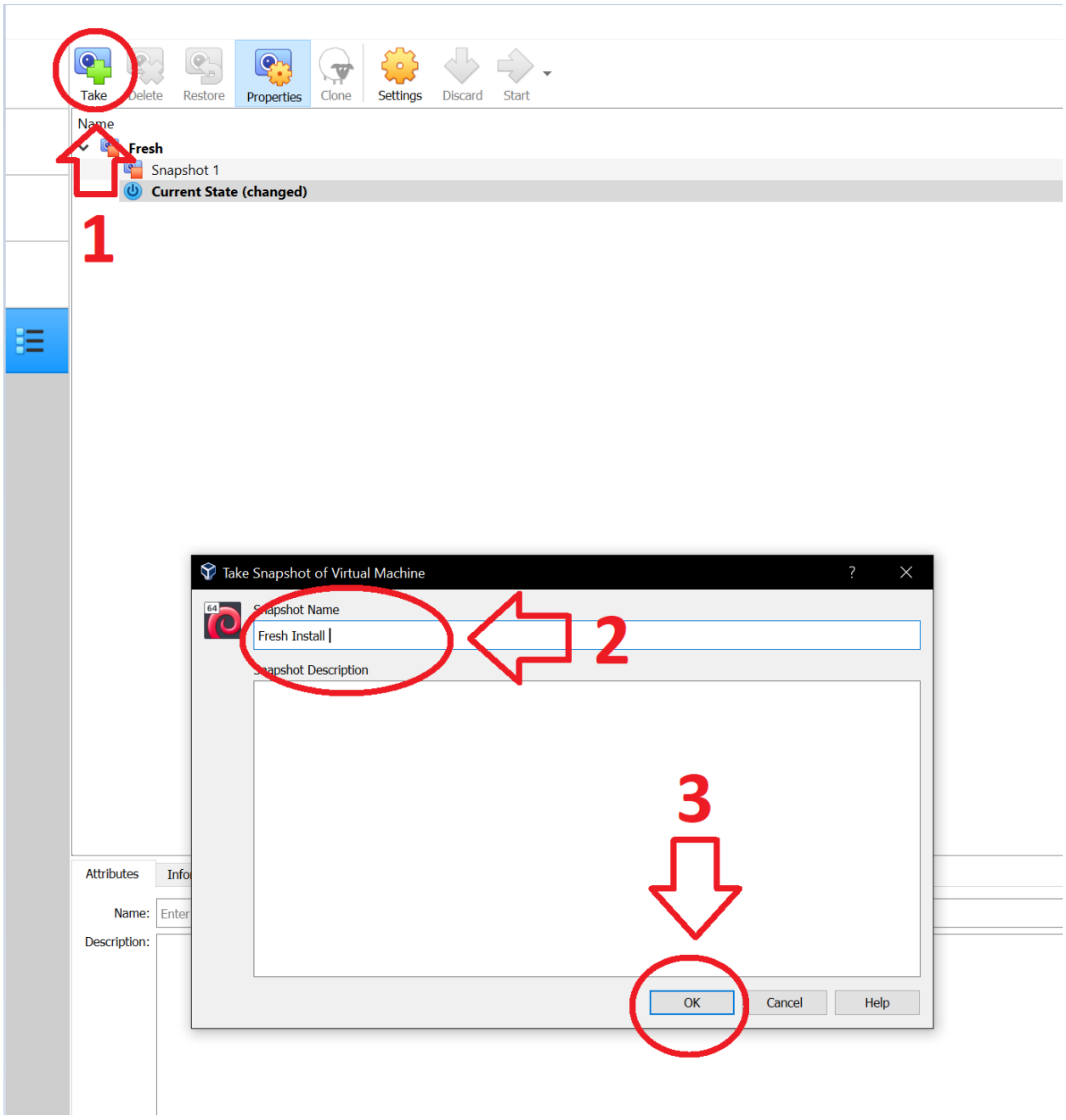


.



12. Once installed, restart Instant Veins and the screen should automatically resize to fit the window once on the desktop. To toggle between Fullscreen, press **Rctrl + f**.

13. The final step highly recommended to do is to save the current state of the virtual machine so it can be reset to a fresh workspace if needed. To do this exit the virtual machine once more and select “take snapshot”. Give it a name and select “ok”. Once this is done you can simply select the snapshot and select “Restore” to instantly clean your simulation environment to its current working form. Happy Simulating!



How to manually install Ubuntu and Veins

1.1 Installing VirtualBox (skip if Ubuntu is already an operating system on your device)

1.1.1 Download the most current version of VirtualBox for your operating system

https://www.virtualbox.org/wiki/Downloads

1.1.2 Locate the VirtualBox installation software on your device and run the setup with default settings

1.1.3 Download the latest Ubuntu ISO file

https://ubuntu.com/download/desktop

1.1.4 Open VirtualBox and create a new version of Ubuntu (32 or 64 bit, depending on your system). Make sure to allocate at least 10 gigabytes to the new virtual machine and enough ram to run smoothly without crashing your system.

1.1.5 Start the newly created Ubuntu virtual machine and select the location of the Ubuntu ISO file on your device when prompted. Ensure not to ignore the prompt as it is much harder to add the ISO file later.

1.2 Setting up the Ubuntu environment

1.2.1 When the Ubuntu ISO file has been successfully imported you will be presented with the installation screen. Select the installation settings to your personal preference as they do not affect the research in any way. Be sure to select "erase disk and install Ubuntu" during the "installation type" screen, it will not delete your system, it will only use the allocated space previously selected.

1.3 Installing OMNeT++

1.3.1 Download OMNeT++ 5.6.2 for Linux from https://omnetpp.org/download/old. Be sure to download version 5.6.2 as newer versions are incompatible.

1.3.2 Install python2 by right clicking, selecting open terminal and paste the following code:

```
sudo apt install python2
```

1.3.3 Install Java by pasting the following code in terminal:

```
sudo apt install openjdk-8-jdk
```

1.3.4 Extract the downloaded OMNeT++ 5.6.2 file to a desired location by right clicking the file, extract to, and select a location to work from.

1.3.5 Extract the downloaded OMNeT++ 5.6.2 file to a desired location by right clicking the file, extract to, and select a location to work from.

1.3.6 Navigate to the extracted OMNeT++ file and open the terminal there as done before. Then enter “pwd” to get the file path and copy it for later.

1.3.7 In the terminal paste the following code:

```
gedit ~/.bashrc
```

1.3.8 Then at the very bottom of the appeared file add the following two lines and replace the (paste)s with the path obtained earlier:

```
export PATH=$PATH:(paste)/bin
export OMNET_DIR=(paste)
```

1.3.9 Save and close the document and return to the terminal. Paste the following five codes into it in order from top to bottom:

1. `source ~/.bashrc`

2.
   ```
   sudo apt-get install build-essential gcc g++ bison flex perl \
       python python3 qt5-default libqt5opengl5-dev tcl-dev tk-dev \
       libxml2-dev zlib1g-dev default-jre doxygen graphviz libwebkitgtk-3.0
       sudo apt-get install openscenegraph-plugin-osgearth libosgearth-dev
   ```

3. ./configure

4. make

1.3.10 Once the terminal is done processing type “omnetpp” to start the program.

1.3.11 Once opened a prompt will appear to download additional framework and examples. Download them both. When completed at the top of the window select “Project > Build all” and wait until it is completed.

1.4 Installing Sumo

1.4.1 Install Sumo by entering the following into terminal:

```
sudo apt-get install sumo sumo-tools sumo-doc
```

1.4.2 Once installed make sure Sumo is an available application on your PC.

1.5 Installing Veins

1.5.1 Download Veins from the following link:

https://veins.car2x.org/download/

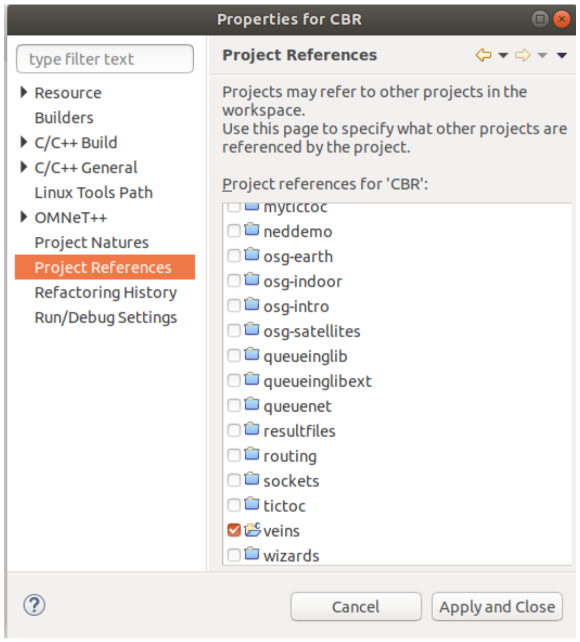
1.5.1 Once downloaded extract the file in the same workspace as OMNeT++

1.5.2 Open OMNeT++ by typing “omnetpp” into terminal and navigate to “File > Import > General > Existing project,” and select the extracted Veins file.

1.5.3 Select finish and finally build the project one more time by selecting “Project > Build all” and wait untill it is completed.

2. Configuration

2.1. Right-click on the project name, then click Properties, then Project References. Check the box for the veins folder but no others.



Properties for CBR

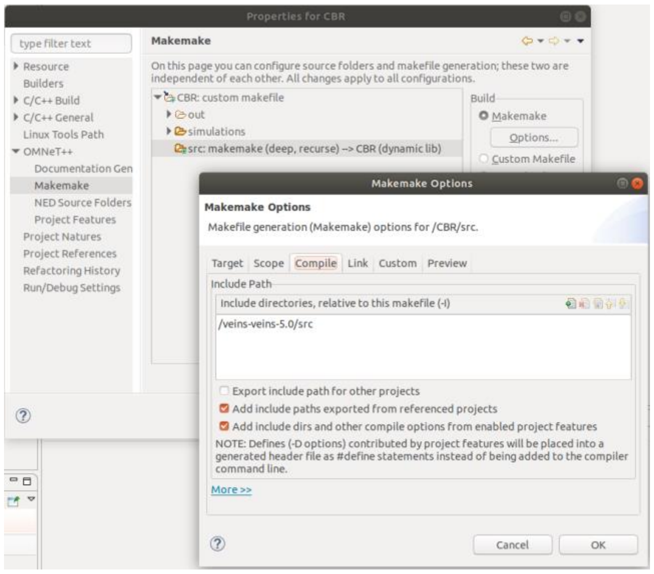
Project References

Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

Project references for 'CBR':

- [ ] mytictoc
- [ ] neddemo
- [ ] osg-earth
- [ ] osg-indoor
- [ ] osg-intro
- [ ] osg-satellites
- [ ] queueinglib
- [ ] queueinglibext
- [ ] queuenet
- [ ] resultfiles
- [ ] routing
- [ ] sockets
- [ ] tictoc
- [✓] veins
- [ ] wizards

2.2. Makemake Options: Click on the src folder. In the Build box on the right, click the Makemake button, then Options. Under the Compile tab, add the /veins-veins-5.1/src path and delete any other paths. Uncheck the ‘Export include path for other projects’ option.



Properties for CBR

Makemake

On this page you can configure source folders and makefile generation; these two are independent of each other. All changes apply to all configurations.

CBR: custom makefile
out
simulations
src: makemake (deep, recurse) --> CBR (dynamic lib)

Build: [✓ Makemake] Custom Makefile

Makemake Options

Makefile generation (Makemake) options for /CBR/src.

Include Path

Include directories, relative to this makefile (-I)

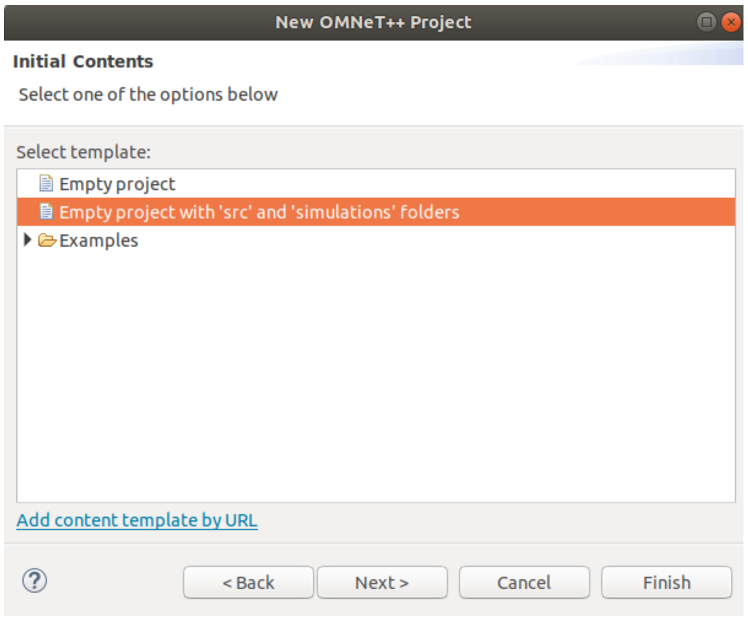
/veins-veins-5.0/src

- [ ] Export include path for other projects
- [✓] Add include paths exported from referenced projects
- [✓] Add include dirs and other compile options from enabled project features

NOTE: Defines (-D options) contributed by project features will be placed into a generated header file as #define statements instead of being added to the compiler command line.

3. Create a project

3.1. New OMNet++ project



New OMNeT++ Project

Initial Contents

Select one of the options below

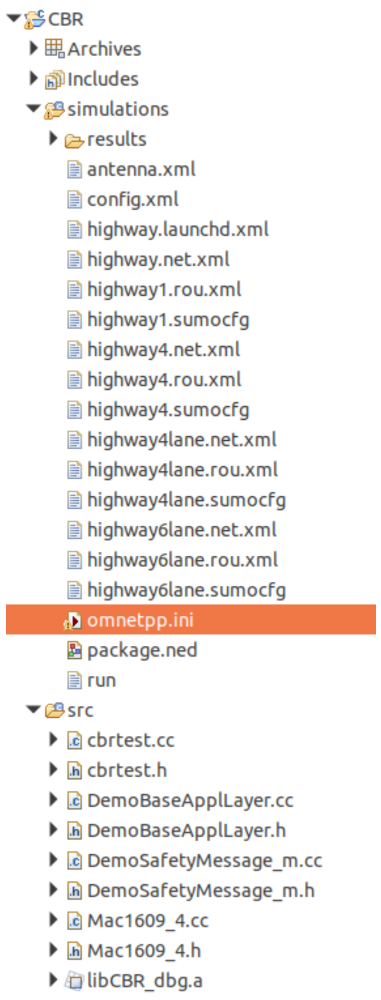
Select template:

- Empty project
- Empty project with 'src' and 'simulations' folders
- Examples

Add content template by URL

< Back | Next > | Cancel | Finish

3.2. Project structure



```
CBR
  Archives
  Includes
  simulations
    results
    antenna.xml
    config.xml
    highway.launchd.xml
    highway.net.xml
    highway1.rou.xml
    highway1.sumocfg
    highway4.net.xml
    highway4.rou.xml
    highway4.sumocfg
    highway4lane.net.xml
    highway4lane.rou.xml
    highway4lane.sumocfg
    highway6lane.net.xml
    highway6lane.rou.xml
    highway6lane.sumocfg
    omnetpp.ini
    package.ned
    run
  src
    cbrtest.cc
    cbrtest.h
    DemoBaseApplLayer.cc
    DemoBaseApplLayer.h
    DemoSafetyMessage_m.cc
    DemoSafetyMessage_m.h
    Mac1609_4.cc
    Mac1609_4.h
    libCBR_dbg.a
```

3.3. Create your own network
3.4. Create your own ned file and .c file
3.5. Necessary files (Give introduction of the main function for each file)

- DemoBaseApplLayer.cc
    - This file contains definitions of the methods defined in DemoBaseApplLayer.h. It contains different methods to be called depending on the type of message received, methods to update the vehicle's position depending on parking and movement, and methods to send a message with or without a delay.
- DemoBaseApplLayer.h
    - This file contains declarations of methods dealing with the sending and receiving of messages.
- DemoBaseApplLayer.ned
    - This file uses parameters and gates to describe the module structure. These parameters include a variable to tell the application layer to periodically send beacons, a variable that indicates the length of a beacon packet, a variable for the user priority of the messages, and a variable containing the interval between two messages.
- DemoSafetyMessage_m.cc
    - This is a generated file containing definitions of the methods defined in DemoSafetyMessage_m.h, including methods to get and set the four variables declared in DemoSafetyMessage.msg.
- DemoSafetyMessage_m.h
    - This is a generated file containing declarations of methods to get and set variables.
- DemoSafetyMessage.msg
    - This file contains a template for BSMs to be sent out, with variables to be set by the sending vehicle.
- Mac1609_4.c
    - This file contains definitions of the methods defined in Mac1609_4.h. It contains methods to deal with self-messages along with messages from the upper and lower layers and methods to change the status of the channel to busy or idle. There are also methods to change the transmission power and the modulation and coding scheme. Some of the variables include one for whether or not the channel is idle, along with variables containing the times when the channel was last busy and last idle, a variable for the total busy time of the vehicle, and another for the transmitting power of the vehicle.
- Mac1609_4.h
    - This file contains declarations of methods that involve the physical layer of the simulation, such as dealing with transmission power, messages, and the channel.

3.6. cbrtest.cc introduction

- `initialize(int stage)`
  - Creates the vehicle and adds it into the simulation. This method defines the variables `mobility` (the TraCIMobility module), `traci` (the command interface of `mobility`), `traciVehicle` (the vehicle command interface of `mobility`), `Tracscene` (the TraCI scenario manager), `mac` (the MAC layer of the simulation), and `myMac` (a variable using `mac` in order to access variables when calculating the CBR).
- `finish()`
  - Called when the vehicle exits the simulation. This method also contains code that can print the total number of messages sent and received by the vehicle. It can also print the number of messages that vehicle received from every other vehicle individually along with the average inter-packet delay for all sender-receiver vehicle pairs with that vehicle as the receiver.
- `onBSM(DemoSafetyMessage* bsm)`
  - Called whenever the vehicle receives a BSM (`bsm`) from another vehicle. In this method, `numReceived` (containing the total number of BSMs received by the vehicle) is incremented. Then a new BSM is created (`bsm2`), where `senderId` is set to be the vehicle's name (equal to `mobility`) and `refNum` is set to be a unique integer for that vehicle (i.e. the first message a vehicle sends has `refNum` = 1, the second has `refNum` = 2, etc.). The `populateWSM()` method is called, which fills in other attributes of the BSM, such as the coordinates of the vehicle sending the message. `bsm2` is then sent, and `numSent` (containing the total number of BSMs sent by the vehicle) is incremented. There are also calculations involving three vectors that can provide data; these are discussed in sections 5 and 6.
- `handleSelfMsg(cMessage* msg)`
  - Called whenever the vehicle receives a self-message. In this method, the `handleSelfMsg()` function in the DemoBaseApplLayer.cc file is called, which results in a new BSM (`bsm`) being created, populated with data in a similar way as in the above `onBSM()` method, and sent. `numSent` is then incremented here as well. Then the `CongAlgorithms()` method is called, which uses a `switch` statement and the `algorithmType` variable to run the algorithm that was chosen by the user prior to running the simulation.

3.7. How to set .ini file

4. How to get CBR

- CBR stands for channel busy ratio, which is a proportion of time that a channel is busy. For example, if the CBR is 0.1 for a period of 1 second, then the channel was sensed to be busy for a total of 0.1 seconds of that time.
- The `getCBR()` method returns the CBR for the period of time ranging from the last time the channel was sensed to be busy until the current simulation time.

Uncommenting the line under the one reading `// print CBR` in `handleSelfMsg()` will print the CBR value, the current simulation time, and the vehicle ID each time `handleSelfMsg()` is called.

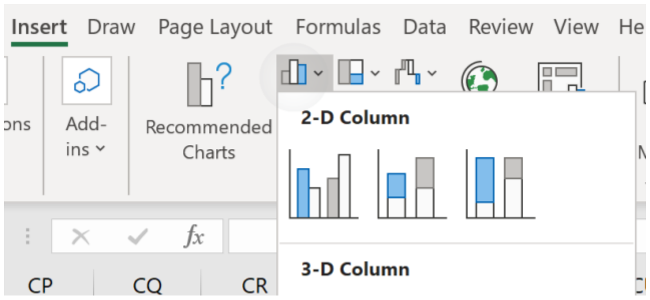
5. How to get IPD
   - IPD stands for inter-packet delay, which is the time between two consecutive BSMs being received within the same sender-receiver vehicle pair. For example, if vehicle B received a BSM from a vehicle A when the current simulation time is 1 second, and vehicle B receives a second BSM from vehicle A at 6 seconds, then the IPD is 5 seconds for the A-B sender-receiver pair, regardless of whether B received any BSMs from any other vehicles during that timeframe.
   - The vector `ipd` stores the IPD for each sender, while the vector `prevTimes` stores the last time the vehicle received a BSM from each sender. Both vectors create a new pairing if the vehicle receives a BSM from a new sender. If the block of code under the line reading `// IPD GRAPHS` is uncommented, then the method will print the coordinates of both the sending and receiving vehicles' position and the IDs of both vehicles each time `onBSM()` is called. If the sending-receiving pair has previously exchanged at least one message, then the gap between the last time the vehicle received a message from the same sender and the current time will be printed; otherwise, the word `First` will be printed.
6. How to get BSM data
   - In the `finish()` method of the cbrtest.cc file, if the two lines under `// SENT AND RECEIVED` is uncommented, then each time a vehicle exits the simulation, the number of messages they sent and received will be printed alongside the vehicle ID. These values can also be used to calculate the beacon receive rate (BRR), which is equal to the number received divided by the number sent.
   - In the onBSM() method, if the line of code under the `// BSM GRAPHS` line is uncommented, then each time a vehicle receives a BSM, the IDs of the sending and receiving vehicle are printed alongside the `refNum` of the BSM. The resulting output can then be copied and pasted into Excel or another similar program, where if you add up the number of received BSMs, counting each combination of `senderID` and `refNum` no more than once, you get the number of BSMs sent that were acknowledged by at least one vehicle. Subtracting this total from the total number of received BSMs, you end up with the number of lost BSMs. This value can then be used to calculate the beacon error rate (BER), which is equal to the number lost divided by the number sent.
   - For example, if one BSM got received by three vehicles, one BSM got received by one vehicle, and a third BSM got received by no vehicles, then the number sent is 3 and the number received is 4; however, the number of BSMs that were acknowledged by at least one vehicle is 2, and so the number of lost BSMs is equal to 3 – 2 = 1.
7. How to get busy time

- In the `handleSelfMsg()` method, if the line of code under `// BUSY TIME GRAPHS` is uncommented, then each time the vehicle receives a self-message, it will print the total busy time that vehicle has encountered so far. Using the data from all vehicles in the simulation, you can take busy time values from different periods of time from a selection of vehicles that combine to give an approximate value of the total busy time over the entire simulation.
- For example, in a simulation that lasts for 200 seconds where the first vehicle receives busy time data when time = 1 s, you could add the total busy time of vehicle A over the period of time from 1 second to 43 seconds, the total busy time of vehicle B from 43 seconds to 96 seconds, and the total busy time of vehicle C from time 96 seconds to 200 seconds to get an estimated value of the total busy time over the whole simulation, since the first vehicle added into the simulation may exit before the simulation is over.

8. How to apply different algorithm
9. How to run the simulation
10. How to get simulation result
    - For bar graphs in Excel:
        1. Create a table of the data. The photo on the right contains a table with the total number of BSMs received during the simulation for two different algorithms, each paired with two different traffic models; one being a highway with four lanes and one being a highway with six lanes.

Algorithm	Traffic Model	Received BSMs
10Hz	4 Lanes	623163
	6 Lanes	537251
OSC	4 Lanes	253642
	6 Lanes	282452

        2. To create a graph, click on the 'Insert' tab at the top and then the 'Insert Column or Bar Chart' button. Then click the 'Clustered Column' option (the first entry in the '2-D Column' section).



3. Right-click on the graph created (if no data was highlighted yet, the graph will look like a blank white rectangle) and click 'Select Data'.

4. In the next menu that pops up, select 'Add Series'.

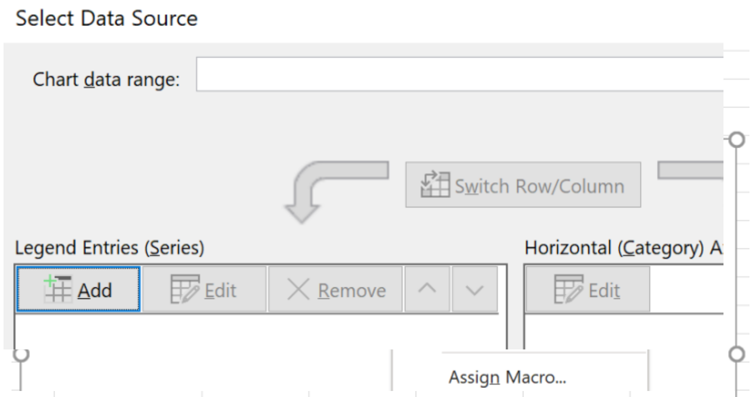
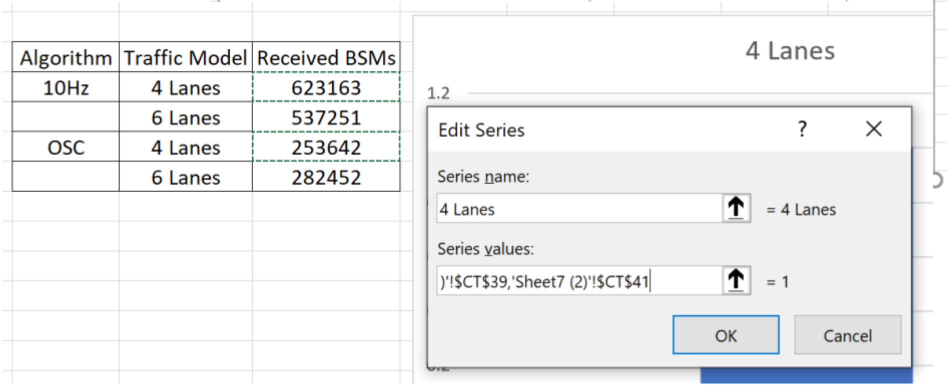

Select Data Source

Chart data range:

5. In 'Series name', enter the name of the first traffic model. In 'Series values', select all data cells containing values corresponding to that traffic model. Then press OK.

Repeat steps 4-5 for each traffic model.



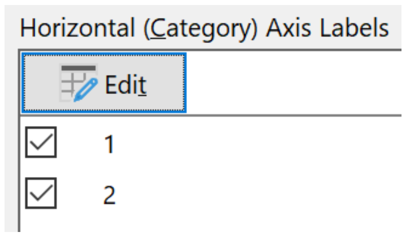
Algorithm	Traffic Model	Received BSMs
10Hz	4 Lanes	623163
	6 Lanes	537251
OSC	4 Lanes	253642
	6 Lanes	282452

Edit Series

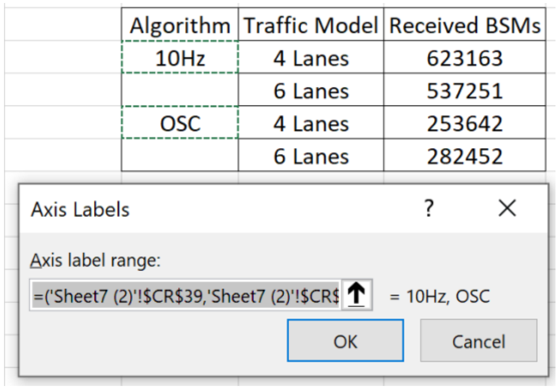
Series name: 4 Lanes = 4 Lanes

Series values:)'!$CT$39,'Sheet7 (2)'!$CT$41 = 1

6. Under the 'Horizontal (Category) Axis Labels' heading, select 'Edit'.



7. Select the individual cells containing the algorithm names. Hold the Ctrl key while selecting if the cells are not all connected. All algorithm names should be selected, not just

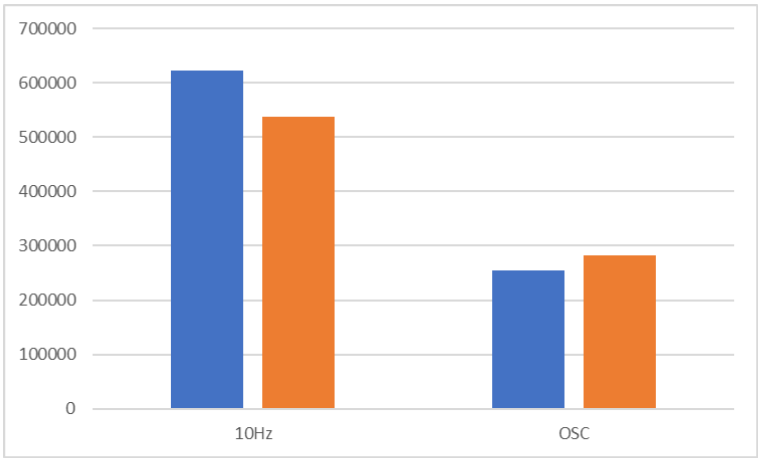


Algorithm	Traffic Model	Received BSMs
10Hz	4 Lanes	623163
	6 Lanes	537251
OSC	4 Lanes	253642
	6 Lanes	282452

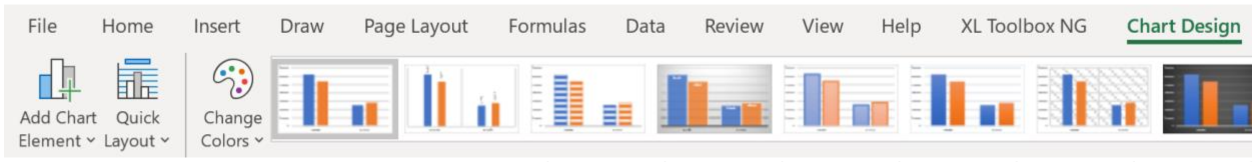
Axis Labels

Axis label range: =('Sheet7 (2)'!$CR$39,'Sheet7 (2)'!CR = 10Hz, OSC the first one. Then click OK, and then OK again.

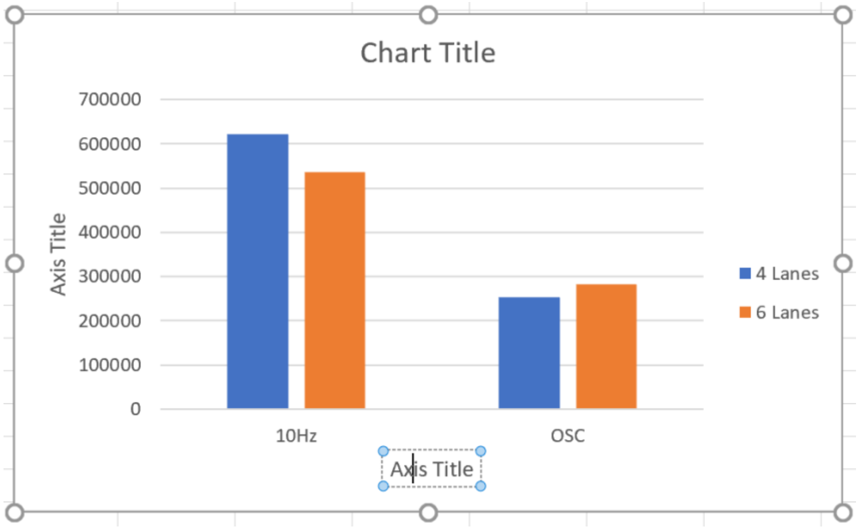
8. This is what your graph should look like now.



9. Next, click the ‘Chart Design’ tab at the top and then the ‘Add Chart Element’ button on the left.

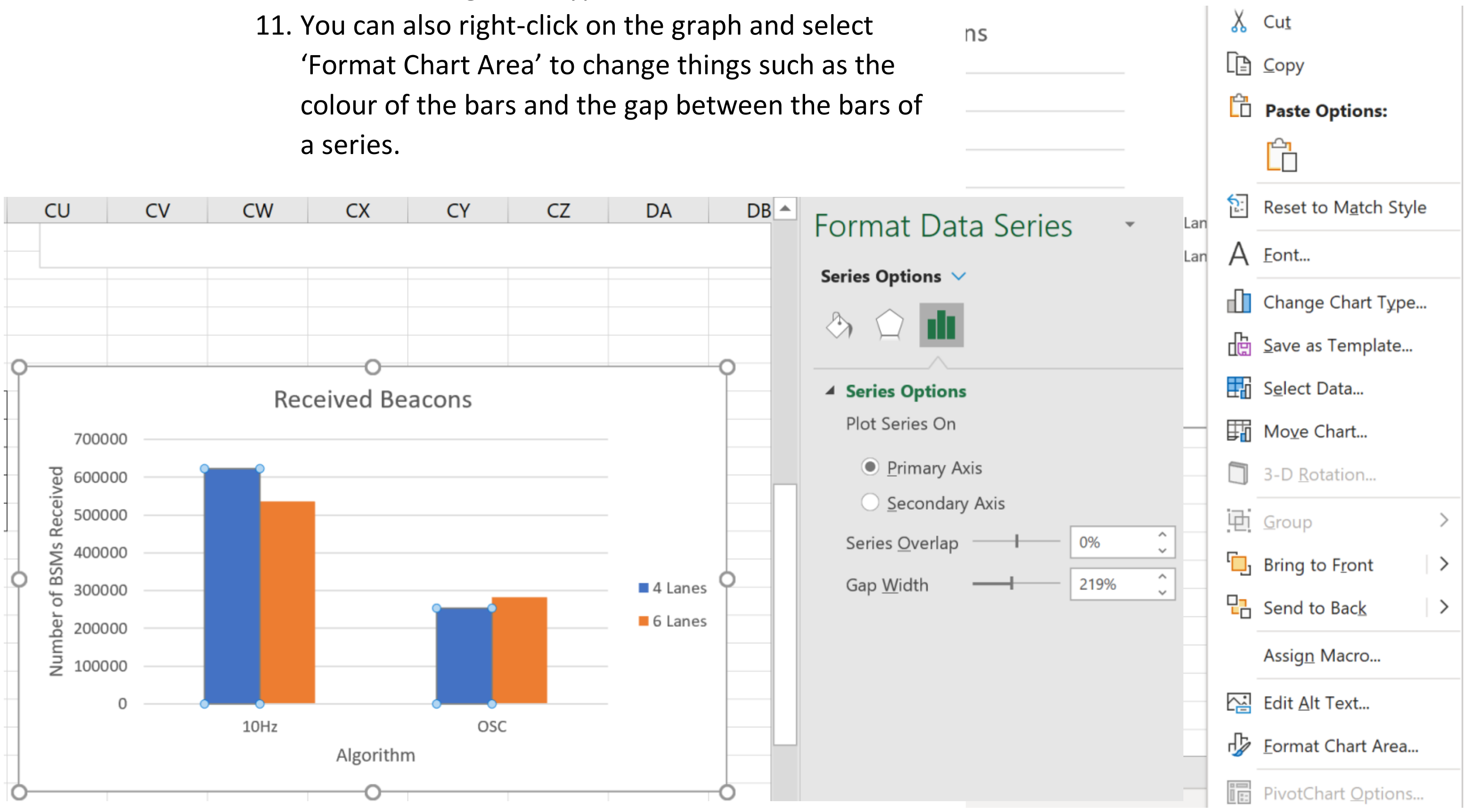


10. A drop-down menu will show up that will let you add a chart title, axis titles, and a legend. To change the chart title and axis titles, click on them in the chart and you can type whatever you want. You



can also change font type, size, and colour.

11. You can also right-click on the graph and select 'Format Chart Area' to change things such as the colour of the bars and the gap between the bars of a series.

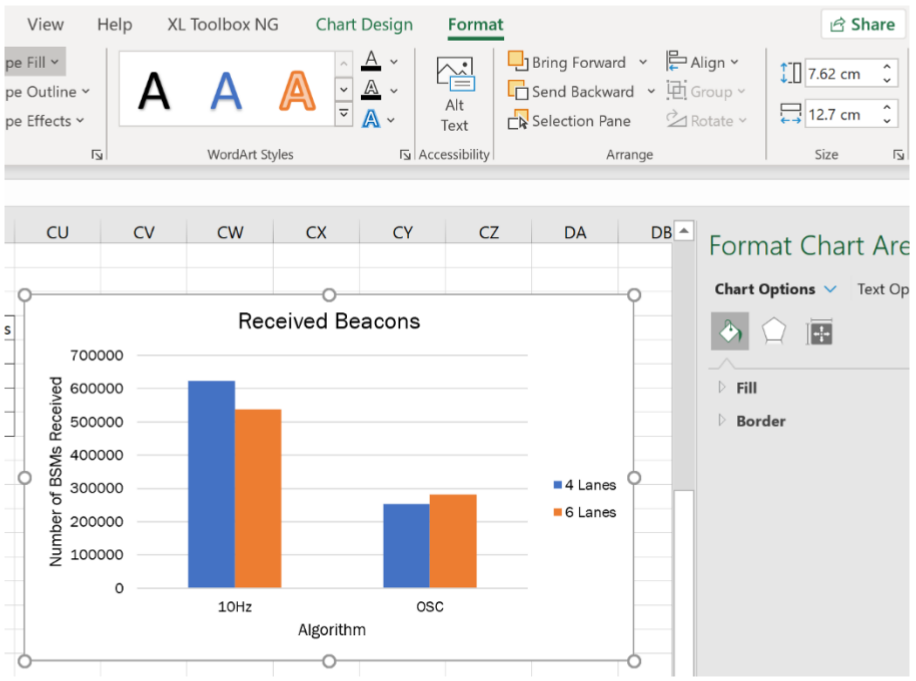


[✓ Primary Axis] Secondary Axis

Series Overlap 0%

Gap Width 219%

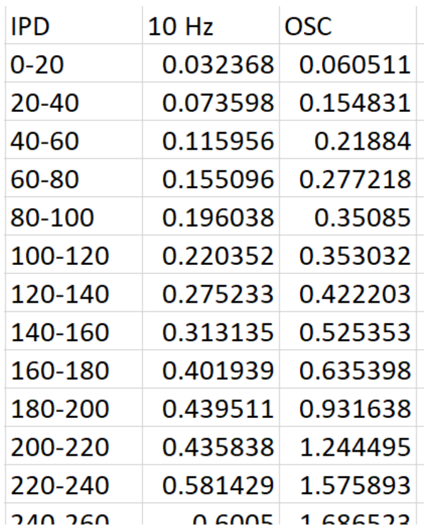
12. Under the 'Format' tab at the top, you can use the two boxes on the far right to change the height and width of your graph.



7.62 cm

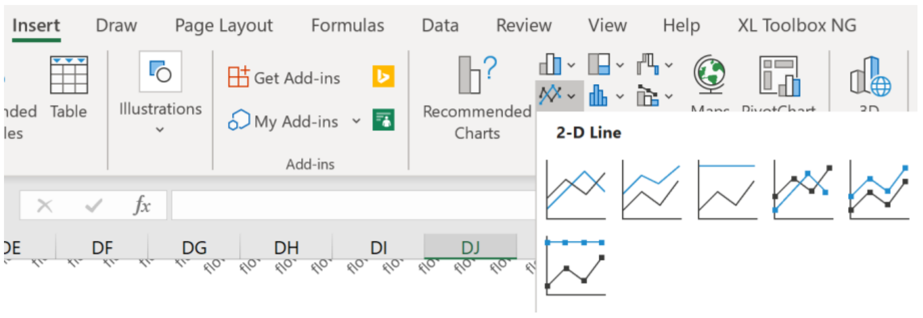
12.7 cm

- For line graphs in Excel:
  1. Create the table of data.

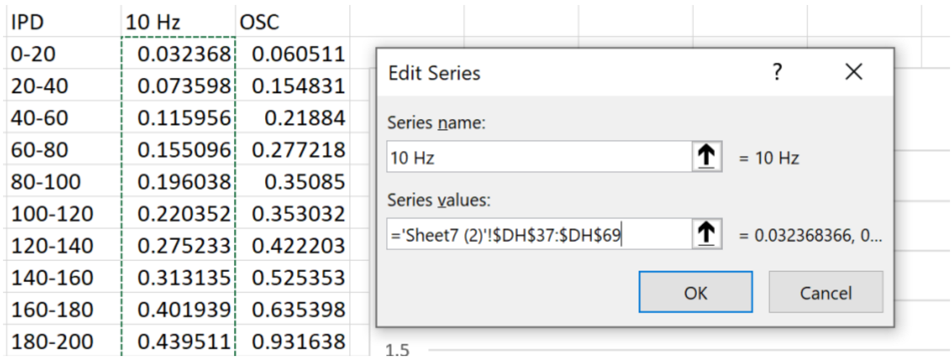


IPD	10 Hz	OSC
0-20	0.032368	0.060511
20-40	0.073598	0.154831
40-60	0.115956	0.21884
60-80	0.155096	0.277218
80-100	0.196038	0.35085
100-120	0.220352	0.353032
120-140	0.275233	0.422203
140-160	0.313135	0.525353
160-180	0.401939	0.635398
180-200	0.439511	0.931638
200-220	0.435838	1.244495
220-240	0.581429	1.575893
240-260	0.6005	1.686523

  2. Under the 'Insert' tab, select the 'Insert Area or Line Chart' button. Then select the 'Line' or 'Line with Markers' option (the first and fourth options respectively).



  3. Similar to the bar graph, right-click on the graph, click 'Select Data', and then 'Add'. In this example, we will be creating a graph for each traffic model, and each graph will display the lines for all of the algorithms. Enter the name of the first algorithm as the 'Series name' and the entire set of data relating to that algorithm in the 'Series values'. Repeat for all other algorithms.



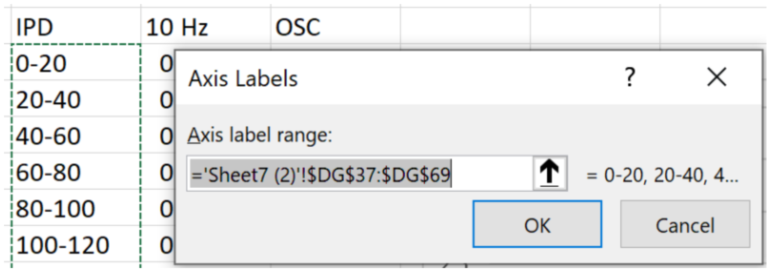
IPD	10 Hz	OSC
0-20	0.032368	0.060511
20-40	0.073598	0.154831
40-60	0.115956	0.21884
60-80	0.155096	0.277218
80-100	0.196038	0.35085
100-120	0.220352	0.353032
120-140	0.275233	0.422203
140-160	0.313135	0.525353
160-180	0.401939	0.635398
180-200	0.439511	0.931638

Edit Series

Series name: 10 Hz = 10 Hz

Series values: ='Sheet7 (2)'!$DH$37:$DH$69 = 0.032368366, 0...

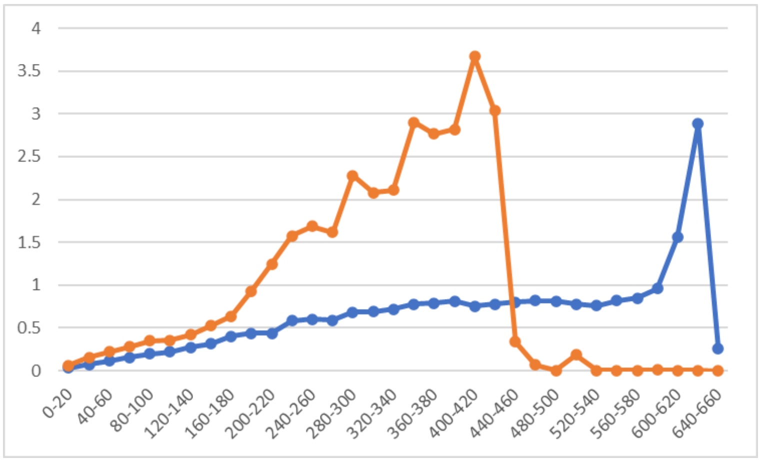
4. Under the ‘Horizontal (Category) Axis Labels’ heading, select ‘Edit’. Then for the axis label range, select the range of values to be placed on the x-axis.



IPD	10 Hz	OSC
0-20	0	
20-40	0	
40-60	0	
60-80	0	
80-100	0	
100-120	0	

Axis label range: ='Sheet7 (2)'!$DG$37:$DG$69 = 0-20, 20-40, 4...

5. This is what your graph should look like now.



6. To add additional items to the graph or customize it, consult numbers 9-12 under the bar graph section; line graphs work in the same way.

3. Using Veins Demo Project

omnetpp.ini

- This file contains the parameters for the simulation along with the selection of configurations you can use.
- To add your own configuration, add the line **`[Config config-name]`** at the bottom of the file, and add any parameters specific to your configuration below.
- One of the parameters you add to your configuration should include the .launchd file, which determines the traffic model that will be used during the simulation, using the following line:
  `*.manager.launchConfig = xmldoc("filename.launchd.xml")`
  - To add your own traffic model to the project, insert the .net.xml, .rou.xml, .sumocfg, and the .launchd.xml files into the simulations folder within the project.
  - If you want to change the traffic model used in your configuration, you can edit the names of the files in the .launchd.xml file to match the files of the model you want to use.
  - If you want BSMs to be automatically sent in your configuration, then you should also add the line **`*.node[*].appl.sendBeacons = true`** to your configuration.
  - When you run the simulation, an additional window will appear alongside the simulation window that will let you choose the configuration to run.

ConfigCollection.cc

- The **`onBSM()`** function is run when a vehicle receives a BSM, and this is where you can print the attributes of the BSM that was received to the console.
- The **`handleSelfMsg()`** function runs when it's time for the vehicle to send out another BSM. This is where you can print the current attributes of the vehicle to the console.
- The **`CongAlgorithms()`** function can perform different actions depending on the configuration selected, such as adjusting various attributes of the vehicle. To use this function, you must include the line **`*.node[*].appl.algo = x`** in the omnetpp.ini file, where **`x`** should be an integer. You can add a case in the switch statement for your value of **`algo`** and each vehicle will perform the actions for that case whenever it receives a self-message.

DemoSafetyMessage.msg

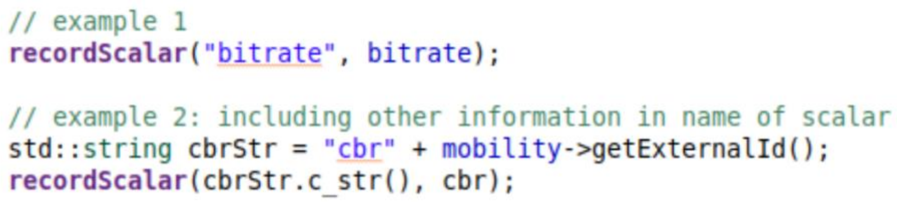
- To add any additional attributes to the BSMs being sent, add them in this file and then build the project. Get and set methods for those attributes will then be automatically generated in DemoSafetyMessage_m.cc/.h for you to use.

DemoBaseApplLayer.cc/.h

- If you add additional attributes to the DemoSafetyMessage.msg file, then in order to make sure those attributes get assigned a value when they are sent out, you can set those values in the **`handleSelfMsg()`** function in DemoBaseApplLayer.cc. You can also add an additional variable in DemoBaseApplLayer.h to store the value that you want to assign if needed.

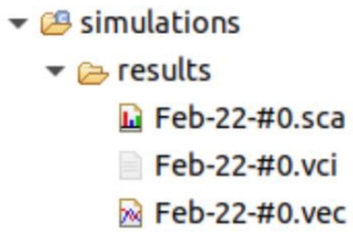
4. Exporting Data to Microsoft Excel

1. First, record the scalars that you want to use. This can be done by using the recordScalar() function, which takes the name and value of the scalar as parameters. If you want to reduce the number of scalars you record (it reduces the number of rows in your Excel file), you can append the values of other variables to your string as shown in example 2; keep in mind that if you do this you will have to use the c_str() function on your resulting string as shown below. Then run the simulation.



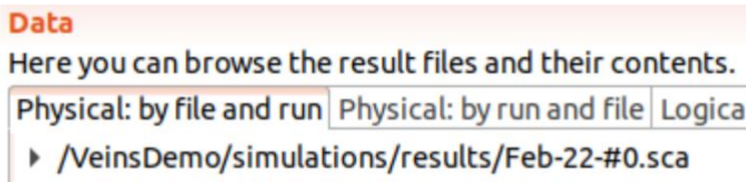
```
// example 1
recordScalar("bitrate", bitrate);

// example 2: including other information in name of scalar
std::string cbrStr = "cbr" + mobility->getExternalId();
recordScalar(cbrStr.c_str(), cbr);
```

2. After you finish running the simulation, double-click on the .sca file for your simulation that appears in the results folder; its name will match the configuration you used from the .ini file. An .anf file will be generated in the simulations folder; open the new .anf file (if you run the same configuration multiple times, it will use the same .anf file).



```
▾ simulations
  ▾ results
      Feb-22-#0.sca
      Feb-22-#0.vci
      Feb-22-#0.vec
```

3. Navigate to the 'Inputs' tab in the .anf file and check if the .sca file appears in the 'Data' section; if it does not, drag the .sca file over from the Project Explorer into the white box under the 'Data' heading.



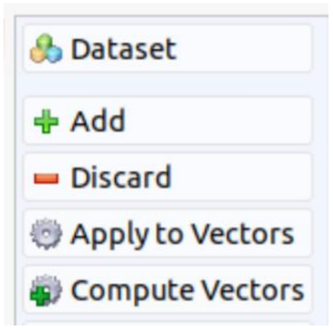
Data

Here you can browse the result files and their contents.

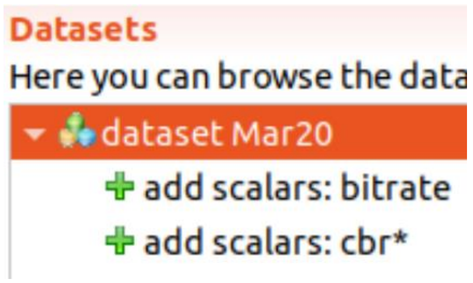
Physical: by file and run | Physical: by run and file | Logica

▸ /VeinsDemo/simulations/results/Feb-22-#0.sca

4. Navigate to the 'Datasets' tab in the .anf file. Click the 'Dataset' button on the right and enter a name for the dataset. You can leave the 'Based on' box blank. Then click Finish.



5. Now your new dataset has been created. To add a scalar to your dataset, click the 'Add' button on the right. The 'Source dataset' box can be left blank and the 'Data type' box should be set to 'scalar'. For 'Filter pattern', enter the name of the scalar. If your scalar has a name that varies (such as "cbr" + mobility->getExternalId(), in which the vehicle ID is part of the scalar name), then add an asterisk * in the location(s) where the difference in names can occur.

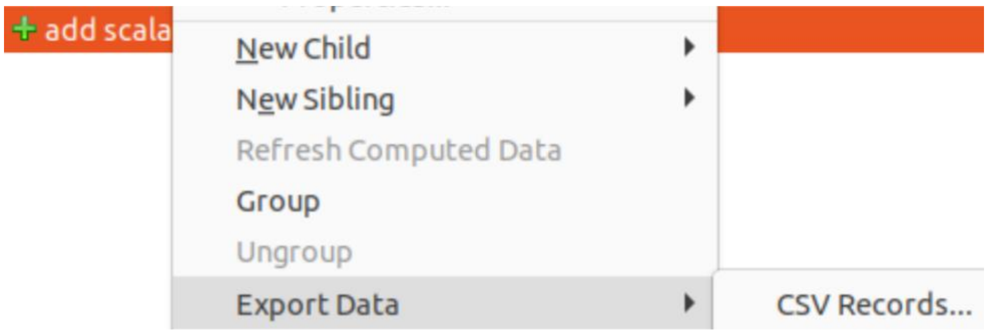


Datasets

Here you can browse the data

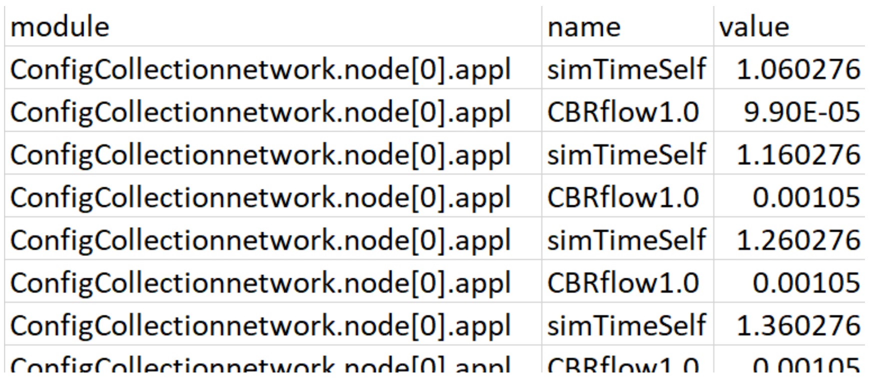
▾ dataset Mar20
    add scalars: bitrate
    add scalars: cbr*

6. Select the last scalar in your dataset and right-click on it. Then click Export Data → CSV Records. Choose the name and location of the file you want to save the data in and click Finish.



7. Now your .csv file has been generated. If your VirtualBox machine does not have Excel, you can email the file to yourself and download it on your host machine to open it. (Or you can use it with other programs; for example, the Ubuntu virtual machine may have LibreOffice in it.)

8. If you're using Excel, open the .csv file and then save it as an Excel (.xslx) file. Your file may contain several parameters from your .ini file, which can be removed (along with the attrname and attrvalue columns, which only hold the data for those parameters; the data for the scalars you recorded should be in the value column). Once the extra cells are removed, you should be left with a table that looks like the one above, which should have the name and value of the scalar, along with the vehicle that it applies to (the number inside the square brackets in the module column is linked to a unique vehicle in the simulation).



module	name	value
ConfigCollectionnetwork.node[0].appl	simTimeSelf	1.060276
ConfigCollectionnetwork.node[0].appl	CBRflow1.0	9.90E-05
ConfigCollectionnetwork.node[0].appl	simTimeSelf	1.160276
ConfigCollectionnetwork.node[0].appl	CBRflow1.0	0.00105
ConfigCollectionnetwork.node[0].appl	simTimeSelf	1.260276
ConfigCollectionnetwork.node[0].appl	CBRflow1.0	0.00105
ConfigCollectionnetwork.node[0].appl	simTimeSelf	1.360276
ConfigCollectionnetwork.node[0].appl	CBRflow1.0	0.00105