École Nationale Supérieure des Télécommunications
INFRES department.

Guide for installation and general use of the DropTailQueueExplorable queue-type in the Omnetpp Network Simulator.

Author:
Miguel Angel Cárdenas Araujo

Paris, December 2007

Table of Contents

1 Guide Presentation....3
2 Installing Omnetpp....4
2.1 Omnetpp-Notes....5
3 Installing INET....6
3.1 INET-Notes....6
4 Integrating new files in INET....8
4.1 Mandatory Contents of INI files that refer to non-native integrated files in INET....9
5 Necessary files for simulating with the DropTailQueueExplorable family files....10
6 Parameters to be set in INI files when using DropTailQueueExplorable queue-type....11
6.1 SVM-Table Fields....13
6.2 DropTailQueueExplorable simulations OUTPUT....15
7 Switch Interface between Tkenv & Cmdenv in INET....16

1 Guide Presentation

In the present guide will be explained how to install OMNETPP and INET, will also be explained how to integrate non-native INET files in INET, and finally will be explained the fields used in the novel queue-type for INET : DropTailQueueExplorable as well as the output fields given by this queue-type.

The mentioned novel queue-type could be used for instance in the Heavy-Hitters detection, could be also used for determining the flows-throughput that cross that queue, and the Output given by that type of queue could also be used in SuperVector Machines for forward uses.

An important remark is that you must already now how to build your Omnetpp simulations as well as with the INET integrated in order to well use the DropTailQueueExplorable module. You should also understand how and where to insert the parameters of this modules for instance in the INI files. Thus if your don't now how to use Omnetpp you can read its manual which can be found at: http://www.omnetpp.org/doc/manual/usman.html or at your doc/ folder in your Omnetpp installation folder.

2 Installing Omnetpp

In this section are given installation instruction for Omnetpp in Linux from the source files.

The programs TCL-TK, and BLT should be installed and working (i.e. to try tcl use the **tclsh** command and to try tk use the **wish** command). Then the Omnetpp program can be installed:

0. Copy the omnetpp archive to the directory where you want to install it (usually your home directory). Extract the archive using the command:

    tar zxvf omnetpp.tgz

1. Add the corresponding lines (with the right addresses) to your startup file (example for: .bashrc):

    export PATH=$PATH:/home/mcardena/simulator/omnetpp-3.3/bin **or** where the bin directory of omnetpp is

    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/mcardena/simulator/omnetpp-3.3/lib or where the Ld library is.

    export TCL_LIBRARY=/usr/local/lib/tcl8.4 **or** where the tcl lib is

        Note: remember do not add spaces between equalities.

2. restart the shell (close and restart the shell window or logout and login again).

3. execute **configure** comand adding the directories of TCL and TK for linus, ie: ./configure --with-tcl=/home/mcardena/simulator/tcl8.4.14/unix/build --with-tk=/home/mcardena/simulator/tk8.4.14/unix/build/ **or** with the directories of TCL and TK for linux

    Note: if any advise is given after executing the **configure** command (for instance when setting the TCL_LIBRARY environmental variable to another path), take it into consideration and apply it.

4. use the **make** command: make

(omnetpp is then installed.)

You should now test all samples and check they run correctly. As an example, the dyna example is started by entering the following commands:

```
cd /omnetppHomeDirectory/samples/dyna
./dyna
```

2.1 Omnetpp-Notes

The command "*opp_makemake -h"* OFFERS A SMALL HELP for the command, these is the same for many linux commands, that like this one doesn't have any man "manual page".

In the case you use don't use sh-like shells that do not accept the **export** command, a similar command to use could be: **setenv,** otherwise look at the command thatallows to make available the environmental variables and use it in the start-up file as in the example given.
example of use of setenv: "setenv TCL_LIBRARY /usr/local/lib/tcl8.4"

3 Installing INET

In this section are given installation instruction for INET:

1. Make sure you OMNeT++ installation works. (i.e. try running the samples)

2. Change to the INET directory and edit the inetconfig file
   to make sure it contains the right settings. You'll probably
   need to adjust the environmental variable ROOT to something like this:
   ROOT=/home/michaeilito2/installOmnetpp/INET-20061020 **or** the directory in where omnetpp was installed

3. Type "./makemake" to create the makefiles and the omnetppconfig file.

4. Type "make" to build everything.

5. Add INET/bin directory to the LD_LIBRARY_PATH.

6. If you modify INET and add/remove/rename directories, you'll need to modify
   "makemakefiles", and go to step 4.

That should be it.

Finally, try running the demo simulations. Change into Examples/ and type "./rundemo".

3.1 INET-Notes

The most general makefile of INET is in its ./bin directory.

INET is just a omnetpp simulation project like any other that you can create manually, but that has many directories and files. For using it, is enough to create the ned and ini files that will say how the network is configured.

Because INET is a program like any other written in omnetpp, it has just one executable files, which is called INET and is located in the /bin directory of Inet, i.e.:
/home/mcardena/simulator/INET-20061020/bin/**INET**

So a program should call him and must have that address. i.e.:
/home/mcardena/simulator/INET-20061020/bin/INET $*
   # $* indicates ALL the parameters passed will be included, this is a SHELL instruction.

Is advisable then, to add the INET executable address to one executable file in order to easily run INET and use the ini files created easily.
For instance:

1. Add the next line to a file named “manualExecutable.exe”:
   $INET_HOME/bin/INET $*   # where INET_HOME is an environmental variable that contains the HOME-PATH of INET.
2. run INET with your ini file like this:
   ./manualExecutable.exe -f myIniFile.ini

A good thing about omnetpp is that ned files can even be loaded dinamically, so its no necessary its automatic compilation with nedtool.

Modify INET files is like modifying the source files of any omnetpp program.

The advantage/disadvantage is , the INET is already compiled, so any modification could result in re-making all INET; But that's no necessary because omnetpp allows dynamic loading, so the new files are dynamically compiled, thus it is not necessary the compilation of all the INET.

4 Integrating new files in INET

To create simulations based on modifications of INET or simply adding new files that should be integrated with the INET files (for instance the simulations using the file DropTailQueueExplorable.cc) must be created dynamic shared libraries that OMNETPP/INET can use, thus everything will be loaded dynamically.

Therefore must be done the next steps:

1- Go to file omnetppHOME/configure.user and add in it the next line:
LDFLAGS="-Wl,-export-dynamic"

2- type and run **./configure**

3- type and run **make** #like this is recompiled omnetpp

4. Add the next line (with the right addresses) to your startup file (ie. .bashrc)
export INET_HOME=///The path of your INET home////

4.1Then logout and login to the shell.

5. Go to the directory in where the files you want to integrate are, (for instance: in our case the directory where is found the file queueExplorable.cc and run_when_changing_pc.exe)

6. Execute the file run_when_changing_pc.exe, it will generate the objects files and the shared library, in particular: *the shared-library (*.so) will have the same name of your present directory*:
./run_when_changing_pc.exe

If running that file throws errors or if you do not have that file at all, you should run then the next commands manually:

6.1 **rm *.o *.so**

6.2 **opp_makemake -s -f -N -I**dir1 **-I**dir2 **-I**etc

where the directories dir1, dir2.... are the directories in where can be found the *.h files used in INET, make sure to include all of those directories otherwise the program could not work.

As a general note, this command generates the object and shared libraries as well as the Makefile. In particular the share library will have the same name of your directory, and it accumulates the existent addresses of all the *.h files of INET, like this any new file (*.cc *.h) can include the *.h files of INET (using #include) without the need of writing a complete PATH.

6.3 do the make command: **make**

7 Now you are ready to run your simulation. Make sure that your ini file contain the lines :“**load-libs=…”** and **“preload-ned-files=..”** see the section 4.1 for more details about those lines.

Thus Run your INET with your ini file as normal. If you want see INET NOTES section for more details.

Note: if you modify the *.cc *. h files you need to do a make command every time you modify them: **make.**

4.1 Mandatory Contents of INI files that refer to non-native integrated files in INET

Any *.ini file that makes reference to an non-native integrated file in INET (i.e. DropTailQueueExplorable.cc). should have two important lines:

load-libs="mylib"

#where mylib is the name of the shared library created. Note: when running the *run_when_changing_pc.exe* executable file, the name of your shared library will be the same name of your present directory.

preload-ned-files = ./MyNed1.ned ./MYned2.ned @/INET-HOME/nedfiles.lst

where the **nedfiles.lst** file is a file normally found in the INET_HOME directory that contains the list of *.ned files used by INET by default, and **MyNed1.ned MyNed2.ned** are simply ned files in your present directory that you want to make reference to. That particular line allows the pre-load of the ned files that will be used in simulation

5 Necessary files for simulating with the DropTailQueueExplorable family files

In this section are introduced mandatory and non mandatory files needed in order to be able to use the queueType "DropTailQueueExplorable"

Mandatory files:

1. queueExplorable.h
2. queueExplorable.cc
3. DropTailQueueExplorable.h
4. DropTailQueueExplorable.cc
5. DropTailQueueExplorable.ned
6. manualExecutable.exe **or** executable file that includes the path to the INET executable.

Mandatory files but that can be different in each simulation:

1. testQueueExplorable.ned **or** file that have the ned configuration for your simulation.
2. testQueueExplorable.ini **or** INI file to be used in the simulation.

Not mandatory but recommendable file:

1. run_when_changing_pc.exe
   # file that erases the old *.so *.o files and executes the opp_makemake -s -f -N -Idir1 -Idir2 -Ietc command, which generates new object and shared libraries as well as the Makefile. In order to make this file work correctly, the environmental variable INET_HOME must be set.

6 Parameters to be set in INI fles when using DropTailQueueExplorable queue-type

First of all, must be remember that the *DropTailQueueExplorable* family files where created to generate a type of queue that is compatible and usable in any INET module that uses the standard INET queue-types.

So simply use in the INI files this new queue-type like any other but just with the parameters to be introduced and explained next.

Must also be said that as the *DropTailQueueExplorable* files are non-native INET files, they must be integrated in INET, follow instruction in section 4 in order to integrate them. Moreover all the INI files that use the *DropTailQueueExplorable* queue-Type must include the lines explained in section 4.1

Parameters:

queueType = "DropTailQueueExplorable".
#that is the value that specifies this queueType, when using it all the other parameters to be explained can be used.

frameCapacity= refers to the maximum queue length, after it the packets will be dropped.

HCP= refers to Historical Checks Period, is the period in which are done all the historical checks, and the information of the list is accumulated and sorted so then are printed in the SVM-Tables

append_SVMtablesOfDiffSimulations: is for appending the SVM-Tables of consecutive simulations, executed one after the other. For instance could be used when doing different RUNS (same simulation but using different seed for the Random Number Generator).
If value = 0, means that is not append the SVM-Table with old ones; this is, if there is a old SVM-TABLE with the same name of a New one it will be erased and created the new one.
If value = 1, means that the new SVM-Tables will be appended with the new ones; this is, if a new SVM-table to be generated has the same name of an old one, it will simply continue writing the values in the old SVM-table file.
NOTE: the SVM-TABLES are simply appended, the time and all other values restarted when new simulation are done, the objective of this appending is to have more statistics for a same scenario, simply looking at it with different values.

NHH: indicates the maximum length of records printed in the SVM-Table that is has not the Selected flow in a certain time instant, and or the maximum length

NHchks: indicates the number of times that will be done the Historical Checks to the queue inside the internal_throuput_time_interval. The separation of those checks is simply given by: HCP/NHchks

sortingType: Specifies the way the flows are sorted and/or the way is given relevance to the flows. With other words, it specifies which is the parameter gotten from the flows that decides which flow is more important than other and should take a relevant position, the relevance is given by higher value, so the flows with higher values of the selected parameter will be the most relevants.
The sorting types are:

1. actNumPkg_inQueue
2. oldNumPkg_inQueue
3. actNumPkg_inQueue + oldNumPkg_inQueue
4. hist_pkts_mean
5. hist_pkts_mean – hist_pkts_stdDev
6. hist_pkts_sum
7. meanFlowThrpt
8. actFlowThrouput

For instance the sortingType=6 that refers to the parameter *hist_pkts_sum,* gives relevance to the flow with higher value for that parameter, so a flow with *hist_pkts_sum=6* will be more important than another flow with *hist_pkts_sum=5.*

In the case the value of that parameter is the same for two flows the relevance then will be given in the next order (using this other parameters to compare): actNumPkg_inQueue, actFlowThrouput, oldNumPkg_inQueue,.oldFlowThrouput.

Example of what could be written in a INI file that uses this queueType:
i.e.

```
Network.Router.ppp[0].queueType = "DropTailQueueExplorable"
Network.Router.ppp[0].sortingType=1     #is used actNumPkg_inQueue as sortingType
Network.Router.ppp[0].NHchks=3
Network.Router.ppp[0].NHH=2
Network.Router.ppp[0].frameCapacity=100
Network.Router.ppp[0].HCP= 0.08192
Network.Router.ppp[0].append_SVMtablesOfDiffSimulations= 0
```

6.1 SVM-Table Fields

In this section are introduced the fields that can be found in the output of the SVM-Tables which is the name given to the output files generated when using the DropTailQueueExplorable queue-type. First of all will be introduced the concept of rank that will be used by the fields and then the fields will be explained.

rank: is the relative position of a certain flow respect to the other flows following some relevance, the relevance is given to the flows who have bigger *meanFlowThrpt* in the case for the *meanThRank* and bigger *actFlowThrpt* in the case of *actFlThRank*. The value of rank=1 is given to the most relevant flow according to the way the relevance was given.

IdSourceP: refers to the source port of the flow. It is a deprecated field that initially was used to Identify the flows in the case they had all different source ports.

simTime: refers to the actual simulation time, in where the data showed in the present line for the present flow was generated. Or in other words the instant of time in where was end the actual HCP.

actPcksQueue: is the number of packets sampled for a given flow in the last queue check of the present HCP.

oldPkgsQueue: is the number of packets sampled for a given flow in the last queue check of the last HCP (the HCP before the present one).

actFlowThrpt: is the throughput of the present flow when crossing the queue calculated only during the present HCP.

oldFlowThrpt: is the throughput of the present flow when crossing the queue calculated only during the last HCP.

meanFlowThrpt: is the mean throughput of the present flow when crossing the queue, calculated from its detected existence (in the queue) till the end of the simulation or only during the life of that flow. That option depends on the *isResetOfFlowMeanThr_enable* parameter. Thus if that *isResetOfFlowMeanThr_enable* is enabled (=1) the mean throughput of the flow is calculated from its creation till the moment flow does not sends more packets during a *minFlowDeathTimeOut*, thus if the flow sends packets again after that period of time the *meanFlowThrpt* will start again to be calculated. If the same parameter (*isResetOfFlowMeanThr_enable*) is disabled (=0), the *meanFlowThrpt* for the flow is calculated from its creation till the end of the simulation.

n: represents the sample number used to calculate the *meanFlowThrpt* for each flow, it gets incremented by one unit every HCP. As the *meanFlowThrpt* its value depends in the *isResetOfFlowMeanThr_enable* parameter. Thus if that *isResetOfFlowMeanThr_enable* is enabled (=1) the number of the sample grows from the moment of the flow creation till the moment flow does not sends more packets during a *minFlowDeathTimeOut*, thus if the flow

sends packets again after that period of time the sample number (n) will start again from 1 (is reset). If the same parameter (*isResetOfFlowMeanThr_enable*) is disabled (=0), the *the sample number (n)* for the flow is gets incremented from its creation till the end of the simulation.

meanThRank: is the rank given to the flow according to its *meanFlowThrpt*.

actFlThRank: is the rank given to the flow according to its *actFlowThrpt*.

actQueueLength: is the length of the queue in the last check of the present HCP.

oldQueueLength: is the length of the queue in the last check of the last HCP.

actvInQFlNmbr: indicates the number of flows present in the queue in the last check of the present HCP.

totalThrpt: represents the total throughput that incomes to the queue in the last check of the present HCP.

histPksMean: represents the mean of the packets for a given flow during the present HCP from the first check (inside this HCP) in where was detected till the end of the present HCP. The mean is calculated taking as samples the total number of packets for each of the checks involved.

histPksStdDev: represents the standar deviation of the packets for a given flow during the present HCP from the first check (inside this HCP) in where was detected till the end of the present HCP. The StdDev is calculated taking as samples the total number of packets for each of the checks involved.

histCtr: is used for the histPksMean and histPv, represents the current number of samples in where that mean is calculated. In other words the number of historical checks in the last HCP that where used to calculate the histPksMean and histPv

histPv: is the pseudo-variance of the packets for a given flow during the present HCP from the first check (inside this HCP) in where was detected till the end of the present HCP. The histPv is calculated taking as samples the total number of packets for each of the checks involved.

histPksSum: is the cumulative of the packets sampled in each check inside a HCP for a given flow.

flowID: refers to the fifth-tuple that identifies the flow. This is IP source address, IP destination Adress, Source Port, Destination Port, IP transport protocol type. As note the IP transport protocol type (=6) refers to a TCP flow and the (=17) refers to an UDP flow. An example of the fifth-tupla is : 10.0.0.1:1000@10.0.0.6:11000@17

6.2 DropTailQueueExplorable simulations OUTPUT

There are three two types of output the *DropTailQueueExplarable* queue-type generates, one is in the files omnetpp.sca and the otherare SVM-Tables generated.

Case of omnetpp.sca:

Will be output some scalars that indicates characteristics of the *DropTailQueueExplarable* queue-type. The name of those scalars indicate directly the property they are making reference to:

_period_drops_mean
_period_drops_variance
_period_drops_StandardDeviation
_period_drops_min
_period_drops_max
_medium_queue_length_mean
_medium_queue_length_variance
_medium_queue_length_StandardDeviation
_medium_queue_length_min
_medium_queue_length_max

Note: the word *period* in the above scalars refers to Historical Check Period (HCP).

Other scalars, common in all the queue-types (Not only in DropTailQueueExplorable):
"packets received by queue"
"packets dropped by queue"
"packets received by queue"
"packets dropped by queue"
"packets received by queue"
"packets dropped by queue"
"packets received by queue"
"packets dropped by queue"
"packets received by queue"

Case of SVM-Tables:
There are three types of output for SVM-Tables

queueHistoric: this prints at each simulation HCP at maximum the first *NHH* flows that are the most relevant according to the *sortingType*.

queueHistoric_Selected_MAX_ACTFlowThr: this prints at each simulation HCP the flow that has the maximum *actFlowThrpt* from the first *NHH* flows that are the most relevant according to the *sortingType*.

queueHistoric_Selected_MAX_MEAN_thrp: this prints at each simulation HCP the flow that has the maximum *actFlowThrpt* from the first *NHH* flows that are the most relevant according to the *sortingType*.

An example of the file output name could be:
IP_TCP_UDP_Network.RouterVector[0].ppp[5].queueHistoric_HH=6.dat

That file name indicates that the DropTailQueueExplorable used is found in the network “IP_TCP_UDP_Network” inside the router “RouterVector[0]” in the network interface “ppp[5]” and that uses as NHH the value of 6 (six) and that in its contents is printed the at each HCP at maximum the first *NHH* flows that are the most relevant according to the *sortingType*.

7 Switch Interface between Tkenv & Cmdenv in INET

Sometimes is convenient to deactivate the graphic interface (Tkenv) of Omnetpp in order to make run your simulations smoothly and faster, or when what you care is to obtain the output of a simulation and forget about the graphic interface. There's when becomes important to have a simple command line interface like the offerd by Cmdenv.

In order to swithc your interface you must do the next steps:

1- INET has to be already installed and working perfectly.

2- Go to the $INET-HOME/bin/ directory and modify the “Makefile” uncommenting the cmdenv and commenting the tkenv like the next (or the contrary if its your interest):

```
USERIF_LIBS=$(CMDENV_LIBS)
#USERIF_LIBS=$(TKENV_LIBS)
```

3- Save the file and Type: make

So then will be recompiled INET and created the INET executable that will run now in Cmdenv mode, so all the simulation of INET as use this executable they will be directly in Cmdenv.
To return to Tkenv just do the contrary, in commenting and uncommenting and remember to do a “make”.