Developer's Manual
v.1.1

Table of Contents

1. Description of Project

Pamvotis simulator is designed using a flexible architecture, in order developers to implement their own models or add their own extensions. For this purpose, a number of methods is created which are not all used by the simulator itself, but are useful for developers that want to extend the code.

The Pamvotis project is developed in Java 1.6. In order the executable file to be small and to reduce the resources required, only libraries included in JRE are used.

Pamvotis is created with Eclipse application. If you intend to extend the code of Pamvotis, we recommend that you use Eclipse. If you don't, keep in mind that you have to set the classpath correctly.

The whole project is constituted of four packages. The first (pamvotis.core) contains the classes required for the simulation execution. The second, (pamvotis.intf) contains the classes that implement the user interface in SWING. The third (pamvotis.sources) contains the classes that implement the various traffic sources. The last (pamvotis.exceptions) contains classes that implement exceptions related to Pamvotis project. It is obvious that developers are not interested in the intf package.

An extended description of each class and each method Pamvotis provides is formally documented in HTML format in the doc/classDoc folder. Open the file index.html existed in classDoc folder with your browser, or access the documentation online at the Pamvotis site.

2. Understanding and extending Pamvotis Simulator

In order to understand the way by which the Simulator class performs a simulation, a skeleton of the whole procedure is outlined in this section. However, if you want to extend the functionality of Pamvotis (implement your own model or add some other features) then you should also study the comments of the code.

The whole procedure by which a simulation is performed contains the following steps:

1. Read the simulation scenario parameters from the *config/ntConf.xml* file and store them to some internal, private variables.
2. Create and print the headers of the statistical results files.
3. Run the simulation for a specific time interval(e.g 0..10sec). Mean statistical results will be printed for this interval (0..10sec).
4. Repeat step 3 as many times as needed. For example, continue running the simulation for another time interval (e.g. 10..20 sec). Mean statistic results will be printed for 10..20sec. Run the simulation for 20..30sec. Statistic results will be printed for 20..30sec.
5. Print mean statistical results related to the total simulation time. In our example, mean values will be printed for 0..30sec.

For each of the above steps (excluding step 4 obviously), a public method exists that implements it. The first step is implemented with the *ConfParams* method, which reads the simulation scenario parameters from the *config/ntConf.xml* file and stores it to some global variables.

The second step is implemented by the *PrintHeaders* method, which creates some text files and prints some headers in it.

The third step, which is the most important, is implemented by the *simulate* method. This method takes two arguments; the start time and the end time, in milliseconds. It transforms the start time and the end time in slots. Then, for every slot, from start time to end time, it executes a for loop which does the following:

- Synchronizes the timer of each node.
- If a node has generated a packet, it adds it to the node's packet queue. This is implemented through the *putPacketToQueue* method.
- If a node has a packet to send in its queue, it takes this packet for transmission. This is implemented through *takePacketFromQueue* method.
- Begins to compete for the medium (*fightForSlot*). Three methods may be called in this step, depending on the medium's state in this slot (*emptySlot*, s*uccessfulTransmission*, *collision*).
- Calculates the statistics and add a horizontal line in the results files with the time instance and the statistic for each node. This is implemented through the *printStats* method.

Step 5 is implemented through the *printMeanValues* method, which prints the mean values of each statistic and for each node to a text file.

3. Creating Simulation Scenarios

The parameters of a simulation configuration are read from the config/ntConf.xml file. You create simulation scenarios by editing this file. Put another way, all the interface package does is to collect some user-input parameters and write them to the xml file. You can see the structure of the xml file if you create a simulation scenario run the simulation (it is not necessary to finish, stop it when it begins) and edit the *config/ntConf.xml* file. The xml file structure was kept as simpler as possible, in order people not familiar with xml to be able to edit it and modify it. Each of the xml parameters is explained below.

- *Seed*: An integer number from which the random number generator is initialized. Different values of seed produce different simulation scenarios.
- *Duration: The* duration of the simulation in seconds.
- *Values:* The number of collected values for each statistic, that will be written to the results files.
- *Node element:* We create as many node elements as the network nodes. Each node element has a number (ID) which is an integer number, unique for each node. Each element has the following characteristics:
    - *rate:* The node's data rate in bits.
    - *Coverage:* The coverage range of each node in meters.
    - *xPosition:* The horizontal (x) coordinate of each node in meters. If you do not care about hidden terminals and want all nodes to be in LOS, set the same values for *xPosition* and *yPosition* parameters. Do not leave them blank.
    - *yPosition:* The vertical (y) coordinate of each node in meters. If you do not care about hidden terminals and want all nodes to be in LOS, set the same values for *xPosition* and *yPosition* parameters. Do not leave them blank.
    - *AC*: The node's access category for IEEE 802.11e EDCA (0, 1, 2 or 3). Be sure to use only one of the above values. Any other value will be considered as AC-0 (best effort). Be also sure to set AC-0 for all nodes, if you do not want to use EDCA at all.
    - *Source element*: This element describes a traffic source for a node. Each node can have on or more traffic sources, each one represented by a source element. Each source element has an ID, which is a unique integer number for each source in a node, and a type, which can be one of the following: *generic, ftp, http, video*. See the description of each one of the elements, further on this section.
- *nodes:* The number of nodes in the network. This parameter is not used at all and may be omitted.
- *mixedNodes:* The number of IEEE 802.11b compliant nodes in a mixed 802.11b/g network. Be careful with this parameter. If you set it different than 0, be sure to set the *phyLayer* parameter to *m*.

- *phyLayer:* The physical layer type. (802.11 simple/a/b/g/mixed). s for simple, original 802.11, a for 802.11a, b for 802.11b, g for 802.11g and m for mixed mode 802.11b/g. Be sure to write only one of the above four characters in lowercase. Any other character will be handled as 802.11g layer. Be also sure to set the number of 802.11b compliant nodes (*mixNodes* parameter) if you selected the mixed mode network.
- *RTSThr:* The RTS threshold in bits. Set it to 0 if you want to be always enabled. Set it to 999999 if you want to be disabled, or set it to a value you desire.
- *CTSToSelf:* The CTS-to-Self parameter for 802.11g networks. Set it to y if you want it to be enabled or to n if you want it to be disabled. Be careful to select the correct type of physical layer. Enabling the CTS-to-Self parameter will have an effect only in 802.11g networks. Enabling CTS-to-Self and selecting another type of physical layer will lead to unexpected results.
- *EDCA element:* This element contains parameters for the 802.11e EDCA function for QoS. Leave the default values if you do not want to use it. Default values may be seen if you run Pamvotis Simulator and create a scenario without using 802.11e EDCA.
    - *CWMinFact0/1/2/3:* The division factor for the minimum contention window for AC-0, AC-1, AC-2 and AC-3 respectively. See the user's manual for a definition of the division factor.
    - *CWMaxFact0/1/2/3:* The division factor for the maximum contention window for AC-0, AC-1, AC-2 and AC-3 respectively. See the user's manual for a definition of the division factor.
    - *AIFS0/1/2/3:* The value of AIFS for AC-0, AC-1, AC-2 and AC-3 respectively. See the user's manual for a definition of AIFS.
- *ResultsPath:* The path where the results files will be created.
- *OutResults*: This is a tricky string parameter. It is used to check which results the user wants to be printed. If a user wants to collect values for a specific result then a specific substring is concatenated to the string parameter. For each result, the *PrintStats* method of the Simulator class checks to see if a specific substring is contained in the *outResults* string variable. If it does, then the result is printed. The substrings representing each result are shown below:
    - *tb*: Throughput in bits/s.
    - *tp:* Throughput in packets/s.
    - *ut:* Utilization.
    - *md:* Media access delay.
    - *qd:* Queueing delay.
    - *td:* Total delay.
    - *dj:* Delay jitter.
    - *ra:* Retransmission attempts.
    - *ql:* Queue length.

Thus, if we want to collect values for the first three results and for the last, we would have to define the outResults parameter as tb_tp_ut_ql_. Actually, the underscore is not necessary, but is useful for separating the substrings. If you do not want any results to be printed, do not leave this parameter empty. Just define it with an underscore or another character.

3.1. Description of source elements

In this subsection, a description of the parameters of each source element is outlined.

3.1.1. Generic source

This source represents a generic (abstract) source, producing abstract traffic, obeying specific parameters. Use this source by specifying *type="generic"* in the definition of the source element.

- *pktLngth:* The node's packet length mean value in bits. Can be a decimal number.
- *pktDist:* The distribution of the packet length: c for constant, u for uniform and e for exponential. Be sure to write only one of the above three characters in lowercase. Write only one of the predefined characters. Any other character will be handled as exponential distribution.
- *intArrTime*: The mean value of the packet generation rate in packets/sec. Can be a decimal number.
- *intArrDstr*: The distribution of the packet generation rate: c for constant, u for uniform and e for Poisson. Be sure to write only one of the above three characters. Write the characters in lowercase. Write only one of the predefined characters. Any other character will be handled as Poisson distribution.

3.1.2. FTP source

This source represents a source generating FTP traffic, as described in 3GPP TR 25.892 V6.0.0 (see Appendix). Use this source by specifying *type="ftp"* in the definition of the source element. For the explanation of the parameters that follow, please have a look at the appendix.

- *pktSize*: The MSDU packet size in bits. Can be a decimal number.
- *fileSizeMean*: The mean value of the file size in bytes. Can be a decimal number.
- *fileSizeStDev*: The standard deviation of the file size in bytes. Can be a decimal number.
- *fileSizeMax*: The maximum value of the file size in bytes. Can be a decimal number.
- *readingTime*: The reading time in seconds. Can be a decimal number.

3.1.3. HTTP source

This source represents a source generating HTTP traffic, as described in 3GPP TR 25.892 V6.0.0 (see Appendix). Use this source by specifying *type="http"* in the definition of the source element. For the explanation of the parameters that follow, please have a look at the appendix.

- *pktSize*: The MSDU packet size in bits. Can be a decimal number.
- *mainObjectMean, mainObjectStDev, mainObjectMin, mainObjectMax*: The mean, standard deviation, minimum and maximum values of the main object in bytes. Can be a decimal number.
- *embObjectMean, embObjectStDev, embObjectMin, embObjectMax*: The mean, standard deviation, minimum and maximum values of the embedded object in bytes. Can be a decimal number.
- *NumOfEmbObjectsMean, NumOfEmbObjectsMax*: The mean and maximum values of the number of embedded objects. Can only be integer values.
- *readingTime*: The reading time in seconds. Can be a decimal number.
- *parsingTime*: The parsing time in seconds. Can be a decimal number.

3.1.4. Video source

This source represents a source generating video traffic, as described in 3GPP TR 25.892 V6.0.0 (see Appendix). Use this source by specifying *type="video"* in the definition of the source element. For the explanation of the parameters that follow, please have a look at the appendix.

- *frameRate*: The frame rate in frames/sec. Can only be an integer number.
- *packetsPerFrame*: The number of packets per frame. Can only be an integer number.
- *pktSize*: The mean value of the packet size in bytes. Can be a decimal number.
- *pktSizeMax*: The maximum value of the packet size in bytes. Can be a decimal number.
- *pktIntArr*: The mean value packet interarrival time in seconds. Can be a decimal number.
- *pktIntArrMax*: The maximum value of the packet interarrival time in seconds. Can be a decimal number.

4. Using Pamvotis as an Embedded Simulator

Pamvotis Simulator was designed through a flexible architecture that enables developers to use it as an embedded simulator, inside their own simulator. For example, let's suppose that someone has developed a simulator for the IP layer (or perhaps for some layers above IP) and needs a simulator for the 802.11 MAC layer. Pamvotis can be used inside this simulator without any modification in the code. An other example is that someone has a UMTS simulator and needs to use an 802.11 simulator to simulate cross-system handovers or call access control algorithms. Pamvotis can be controlled through the other simulator without modification in the code.

All the above are feasible through some public methods that Pamvotis provides. All the user needs to do is to create an instance of the Simulator class and call the necessary methods. At the root of the Pamvotis folder, there exists a file named Example.java, which is an example of how to use Pamvotis as an embedded simulator. Another example of how to use Pamvotis is the interface itself. Take a look at the *run* method of the *SimThread* inner class, located in the *Run* class of the *pamvotis.intf* package.

An extended description of each class and each method Pamvotis provides is formally documented in HTML format in the doc/classDoc folder. Open the file index.html existed in classDoc folder with your browser, or access the documentation online at the Pamvotis site.

5. License and Contribution

It should be reminded again that Pamvotis may be distributed or modified under the terms of the GNU general Public Lisence.

Moreover, in order to extend and improve Pamvotis, it is kindly asked to contact with the author for any innovative modification performed in the code. References will be added to the web site and another version of Pamvotis will be released, based on the user's contribution.

For any information, please contact Dimitris Vassis:

support@pamvotis.org