An XML Experiment Description Language for ns–3

George Riley and Joshua Pekley
Department of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA. USA
{riley,jpelkey}@gatech.edu

ABSTRACT

While the *ns–3* tool has been shown to be an excellent tool for researchers and network designers to analyze and evaluate network performance, the use of *ns–3* in educational classroom environments has less success. The use of *ns–3* to illustrate network design principles to lower–level undergraduate students requires considerable skill in the C++ programming language, and appears to be beyond the ability of first or second year undergraduate students. Further, teaching faculty at the university level are also unlikely to have these skills, or the desire to obtain them to use *ns–3*. In order to allow novices to use *ns–3* to construct simple experiments and observe results, we have created the *ns3xml* front–end processor for *ns–3*, that reads a flat text file in the well–known *XML* format and then constructs and executes an *ns–3* experiment based on the *XML* description of the experiment. [1]

1. INTRODUCTION

The *ns–3* network simulator [4] has been designed with several goals in mind. These include extensibility, to allow researchers to add and experiment with new protocols and communications methods, efficiency to insure that experiments execute in reasonable time frames, and C++ coding standards and styles that help insure proper memory management and reduce chances of code crashes or other unexplained execution problems. The implementation of the core simulation framework and most of the networking models included with *ns–3* make extensive use of advanced C++ features such as templated classes and methods, smart pointers, and typesafe callback methods. These advanced coding methods are in conflict with another stated goal of the *ns–3* project, namely to be used in an educational environment to help undergraduate students understand basic networking concepts such as queuing, congestion control, routing, and others. It is unlikely that a first or second year undergraduate student will have the necessary experience and skill at C++ to construct, execute, and analyze various *ns–3* experiments to observe various networking phenomenon. Further, it is even more unlikely that teaching faculty at the university level will have the skills to use *ns–3* effectively in the classroom or to prepare demonstration experiments for out–of–class assignments.

To enable those with little or no C++ coding skills the ability to construct and demonstrate *ns–3* experiments, we have designed and implemented the *ns3xml* front–end environment for *ns–3*. Using our *ns3xml* processor, the *ns–3* experiments are created and executed by simply editing a flat text file in the well known *XML* format and providing that *XML* file as input to our *ns3xml* program. The *ns3xml* program reads the *XML* file, constructs the *ns–3* topology, random number generators, applications, and statistics collection objects specified therein, and executes the *ns–3* experiment. Depending on the type of statistics collection specified by the user, the results are written either on a flat text file in *comma separated value* format suitable for input in the popular *Excel* program, or in a format suitable for using the *ns–3* network animator. Using *ns3xml*, the experiment designer does not need any experience with C++ programming, compiling or linking.

We will describe in detail each of the *XML* elements used to describe an *ns3xml* experiment and an example of a complete *ns–3* input file. In section 2 will describe briefly the *XML* language. Next in section 3 we list the various *ns3xml* elements and the allowable attributes and sub–elements for each. In section 4 we describe how to reference *ns3xml* elements and sub–items. Section 5 gives a very brief discussion of statistics logging output files. Finally, section 6 provides some concluding remarks.

2. THE XML LANGUAGE

The *Extensible Markup Language* (*XML*) [2] is a widely used format for describing data and exchanging data between similar and dis–similar systems. It has it's origins in the 1970's with the development of the *Standardised Generalized Markup Language* (*SGML*) designed by Goldfarb, Mosher and Lorie [1]. In the late 1980s Berners Lee adopted most of the concepts from *SGML* in the design of the now popular *Hypertext Markup Language* (*HTML*) [3] used for designing and rendering pages from the World Wide Web. The *XML* specification grew out of a need for a more formal and concise definition of the syntax for *HTML* pages, and the *W3C* consortium was founded to advocate and maintain the *XML* standards.

An *XML* document consists of a number of *Elements*, each

[1]This work supported in part by the U. S. National Science Foundation grant number 0958015

WNS-3 2011, March 21, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245586

of which has zero or more *Attributes* and zero or more *Sub–Elements*. An example element is the widely used `img` element in *HTML* that identifies a graphical image to be displayed on a browser. An example `img` element might be:

```
<img src="/resources/aufflick-mark/
banner-bottom_01.png"
border="0" width="21" height="62">
```

In this example the `img` element has four attributes, namely `src`, `border`, `width`, and `height`. An example of an element with sub–elements might be:

```
<ul class="compact">
<li><a href="/">Mark's Web</a> </li>
<li><a href="/blog/">Mark's Blog</a> </li>
<li>Sensible Emacs usage</li>
</ul>
```

It this example, the `ul` (unordered list) element allows the `li` (list item) sub–element. Also illustrated by this example is the use of the *terminating* element, the `</ul>` in this example.

Although not used in *ns3xml*, the *XML* specification allows and encourages associated text values between the beginning and terminating instances of any element. For example:

```
<p><em><b>UPDATE 3</b></em></p>
```

here, the phrase `UPDATE 3` is the text value for the `b` (bold) element, which is a sub–element for both the `em` element and the `p` element.

The complete specification and use of *XML* is a complex and lengthy topic, and well beyond the scope of this paper. However, the *ns3xml* program uses only a small set of the features of *XML* as described in the next section.

3. *ns3xml* SYNTAX

The format of the *ns3xml XML* file is simple and easy to follow subset of the *XML* specification. Further, the format uses neither a *data dictionary* nor an *XML schema*. Rather, the *ns3xml* program enforces the set of allowable sub–elements for any given element, the set of allowed attributes for each element, and the set of required attributes for each element. Should any element be an improper sub–element, or should it have an disallowed attribute (or missing a required attribute) the *ns3xml* program produces an informative error message. Some further constraints are enforced for some of the elements, such as insuring the default configuration items for certain *ns–3* objects are recognized and valid.

The *ns3xml XML* file consists of a single top–level element called **ns3**. The allowed sub–elements for the **ns3** element are:

- **cmdline** specifies command line arguments.
- **config** specifies global *ns–3* configuration default values.
- **randomgenerators** which specifies the various random number generators used by this simulation.
- **topology** specifies the topology (nodes, links, queues, etc).
- **applications** specifies the various applications for this simulation.
- **animation** specifies the type of animation desired for this simulation.
- **statistics** specifies the various statistics to be gathered and reported for this run.

Each of these are described in more detail below.

3.1 The ns3 Element

The **ns3** element must be the top–level element in the *XML* file, and all other elements are sub–elements to it. This element has two optional attributes, as follows:

- The `repeat` attribute must have an integer attribute value and specifies the number of times the simulation experiment is to be repeated. For each repetition, the *run number* for the random number generators is incremented giving guaranteed uncorrelated random streams for each run.
- The `stopAt` attribute must have a floating point attribute value, and specifies the time at which the simulation is to terminate. If not specified, the simulation runs until the `Simulator::Stop` method is called.

3.2 The cmdline element

The **cmdline** element describes the various command line arguments that are supported by this simulation. The command line processing is identical to that used by the basic *ns–3* `CommandLine` objects. The **cmdline** element has three allowable attributes, all of which are required.

1. `name` is the name of the command line argument, that can later be used in subsequent sections to refer to the command line equivalenced value.
2. `default` is the default value for the argument, which is used if no command line equivalence is specified.
3. `help` specifies a descriptive phrase describing the usage of this argument.

3.3 The config element

The **config** element delineates a set of sub–elements (of type **default** described below) that specify new values for any of the *ns–3 object attributes*. The only allowable sub–element is the **default** element.

3.3.1 The default *element*

The **default** element species a new value of any of the pre–defined *ns–3* object attributes. This element has three allowable attributes, all of which are required. These are:

1. The `typeid` attribute specifies which *ns–3* object type the new attribute value applies to. An example might be `DropTailQueue`.
2. The `attr` attribute specifies which object attribute (within the object specified by the `typeid`) is to be changed.
3. The `value` attribute specifies the new default value for the specified value. For this attribute, the equivalence value can be a normal string value, or it can refer to any of the command line arguments by name by pre-pending a dollar sign to the command line argument name.

3.4 The **randomgenerators** element

The **randomgenerators** element delineates a list of random number generators required for this experiment. There are four possible sub–elements, described below. In addition, there are several allowable attributes for this element, none of which are required. Those are:

1. `runNumber` specifies the run number sub–stream for the random streams. If not specified it defaults to either 0 if no `repeat` is specified on the **ns3** element, or the repeat number if it is specified.
2. `seed` specifies the random seed for this run. The seed value for *ns–3* random number generators consists of six different 32–bit values. In this case the specified seed value is replicated six times.
3. `seed0`, `seed1`, `seed2`, `seed3`, `seed4`, `seed5` specify the six individual seed values for the *ns–3* underlying random number streams.

3.4.1 The **exponential** *element*

The **exponential** element must be a sub–element of the **randomgenerators** element. It specifies a random number stream with an exponential distribution. It has three allowable attributes, the first of which is required and the subsequent two are optional. Those are:

1. `name` specifies the name of this generator, which is later referred to by other elements needing random numbers.
2. `mean` specifies the mean value of this generator. If not specified the value of 1.0 is assumed.
3. `limit` specifies the maximum value of samples from this generator. If not specified, no maximum is assumed.

3.4.2 The **uniform** *element*

The **uniform** element must be a sub–element of the **randomgenerators** element. It specifies a random number stream with a uniform distribution. It has three allowable attributes, the first of which is required and the subsequent two are optional. Those are:

1. `name` specifies the name of this generator, which is later referred to by other elements needing random numbers.
2. `min` specifies the minimum value of this generator. If not specified the value of 0.0 is assumed.
3. `max` specifies the maximum value of this generator. If not specified the value of 1.0 is assumed.

3.4.3 The **normal** *element*

The **normal** element must be a sub–element of the **randomgenerators** element. It specifies a random number stream with an normal distribution. It has three allowable attributes, the first of which is required and the subsequent two are optional. Those are:

1. `name` specifies the name of this generator, which is later referred to by other elements needing random numbers.
2. `mean` specifies the mean value of this generator. If not specified the value of 0.0 is assumed.
3. `variance` specifies the variance of this generator. If not specified the value of 1.0 is assumed.

3.4.4 The **constant** *element*

The **constant** element must be a sub–element of the **randomgenerators** element. It specifies a random number stream with an constant distribution. It has two allowable attributes, both of which are required.

1. `name` specifies the name of this generator, which is later referred to by other elements needing random numbers.
2. `value` specifies the constant value of this generator.

3.5 The **topology** element

The **topology** element delineates a number of topology related sub–objects, and has five allowable attributes that control the node creation process. Those are:

1. `autoIPV4` is a Boolean value specifying whether each subsequently created `Node` object will automatically be assigned an *IPv4* protocol stack. If not specified the value of `false` is assumed.
2. `autoIPV4Base` specifies a base address for IP address assignment if the `autoIPV4` value is true. If not specified, a value of `172.16.0.0` is assumed.
3. `autoIPV4Mask` specifies the network mask for the IPv4 base address. This mask is used to increment the network address when needed. If not specified, a value of `255.255.0.0` is assumed.
4. `GODRouting` is a Boolean value specifying whether global routing table population is desired. If not specified, a value of false is assumed.
5. `NixVectorRouting` is a Boolean value specifying whether the *NIx–Vector* routing method is to be used. If not specified a value of true is assumed, unless the `GODRouting` value is true.

The allowable sub–elements for the **topology** element are described in the sub–sections below.

3.5.1 The **queue** *element*

The **queue** element defines a queue object that is later used when defining a point–to–point link. There are four allowable attributes as follows:

1. `name` is required and specifies a name for this queue that can later be used to reference it when creating links.
2. `type` is an optional attribute that specifies the type of queue. If it is not specified the default queue type (`DropTail`) is used. The allowable values are `DropTail` and `RED`.
3. `limittype` is an optional attribute that specifies the type of queue limit used for this queue. If not specified, the default value of `PACKETS` is assumed. The allowable values are `BYTES` and `PACKETS`.
4. `limit` is an optional attribute that specifies the maximum capacity of the queue, in units specified by the `limittype` attribute. If not specified, the default value of 100 packets is assumed.

3.5.2 *The* **p2plink** *element*

The **p2plink** element creates a point–to–point link that can later be used to connect two nodes. This element has several allowable attributes, most of which are required. Those are:

1. `name` is required and specifies a name for this link that can later be used to refer to this link during node connections.
2. `bw` is required and specifies the bandwidth of this link, using any of the well known and *ns–3* supported bandwidth specification strings.
3. `delay` is required and specifies the speed–of–light delay on this link, using any of the well known and *ns–3* supported time specification strings.
4. `queue` specifies the name of a previously defined **queue** element to use for the queue at both endpoints of this link. If not specified, both `queue1` and `queue2` must be specified. If this is specified, neither `queue1` nor `queue2` can be specified.
5. `queue1` specifies the name of a previously defined **queue** element to use for the queue at the *n1* to *n2* side of this link.
6. `queue2` specifies the name of a previously defined **queue** element to use for the queue at the *n2* to *n1* side of this link.

3.5.3 *The* **node** *element*

The **node** element describes a single *ns–3* `Node` object. It has four allowable attributes, the first of which is required. Those are:

1. `name` specifies a name for this node that can later be used to refer to this node when defining link connections or application assignments.
2. `repeat` specifies an optional repeat count, indicating how many nodes are to be created. To refer to a repeated node by name, append a dollar sign and an integer value after the node name. For example if a repeated node named `myNode` is created, the zero'th node in the repeat can be referred to as `$myNode$0`.
3. `x` specifies the x–coordinate of the node's position in the animation display.
4. `y` specifies the y–coordinate of the node's position in the animation display.

3.5.4 *The* **connectp2p** *element*

The **connectp2p** element is used to connect (using a point–to–point link) two previously defined **node** elements, using a previously defined **p2plink** element. This element has four allowable attributes, the first three of which are required. Those are:

1. `n1` specifies the name of this first of the two previously defined nodes to be connected.
2. `n2` specifies the name of this second of the two previously defined nodes to be connected.
3. `link` specifies the name of the previously defined **p2plink** element to use.

3.5.5 *The* **dumbbell** *element*

The **dumbbell** element creates a *dumbbell* topology, with leaf nodes on the left and right side, and two routers connecting the left and right leaf nodes. There are several allowable attributes. The `name` element is required, others may or may not be required depending on other attribute specifications as described below.

1. `name` is required and specifies the name of this **dumbbell** element. This name can later be used to refer to nodes within the dumbbell, as described in the discussion on *sub–items* later.
2. `link` specifies the name of a previously defined **p2plink** element. If this attribute is specified, it is the link used for all links in the dumbbell, including left–side leaf nodes, right-side leaf nodes, and the router–to–router connection. If not specified, all of the `leftlink`, `rightlink`, and `bottlenecklink` attributes must be specified.
3. `leftlink`, `rightlink`, and `bottlenecklnk` attributes specify the previously defined **p2plink** elements to be used for left–side links, right–side links, and bottleneck links respectively.
4. `nleaf` specifies the number of leaf nodes on both the left and right sides of the dumbbell.
5. `nleftleaf` and `nrightleaf` specify the number of left–side and right–side leaf nodes respectively. If these are specified, they override the `nleaf` value if it is specified. If none of the leaf count attributes are specified, the default value of 5 is used for both left and right side counts.
6. `x`, `y`, `w` and `h` specify the upper–left corner (`x` and `y`) and width/height (`w` and `h`) of a *bounding box* for this dumbbell. Node location values are computed to lay out the nodes in a visually appealing fashion for animation.

3.6 The **applications** element

The **applications** element delineates the list of application elements used for this simulation. This element has no attributes, and the allowed sub–elements are any of the application elements described below.

3.6.1 *The* **onoff** *element*

The **onoff** element describes one or more instances of an *ns–3* `OnOffApplication`, and adds the application to the specified node or nodes. There are several attributes, all of which are required excepting the `repeat` attribute. Those are:

1. `name` specifies the name for this application, that can later be referenced in the statistics gathering description.
2. `node` specifies the name of a previously defined **node** element. The `OnOffApplication` is assigned to the specified node.
3. `sockettype` specifies the type of *ns–3* socket to use. Allowable values are either `TCP` or `UDP`.
4. `rate` specifies the sending rate, when in the ON state, for this application.

5. `ontime` specifies the name of a previously defined random number generator to use for the ON time.

6. `offtime` specifies the name of a previously defined random number generator to use for the OFF time.

7. `remotenode` specifies the name of a previously defined node. The application uses the IP address of this node as the remote endpoint destination for the generated traffic.

8. `remoteport` specifies the port number on the destination node.

9. `startat` specifies a previously defined random number generator to be used to determine the start time of the application.

10. `repeat` specifies an optional repeat count, indicating how many applications are to be created.

3.6.2 The **bulksend** *element*

The **bulksend** element describes one or more instances of an *ns–3* `BulkSendApplication` and adds the application to the specified node or nodes. There are several attributes, all of which are required excepting `repeat` and `maxbytes`. Those are:

1. `name` specifies the name for this application, that can later be referenced in the statistics gathering description.

2. `maxbytes` specifies the maximum number of bytes this application should send before closing the connection. If not specified, infinity is assumed.

3. `node` specifies the name of a previously defined **node** element. The application will be attached to the specified node.

4. `remotenode` specifies the name of a previously defined node. The application uses the IP address of this node as the remote endpoint destination for the generated traffic.

5. `remoteport` specifies the port number on the destination **node** element.

6. `startat` specifies a previously defined random number generator to be used to determine the start time of the application.

7. `repeat` specifies an optional repeat count, indicating how many applications are to be created.

3.6.3 The **packetsink** *element*

The **packetsink** element describes one or more instances of an *ns–3* `PacketSinkApplication` and adds the application to the specified node or nodes. The application started automatically at time zero. There are several attributes, all of which are required excepting `repeat`. Those are:

1. `name` specifies the name for this application, that can later be referenced in the statistics gathering description.

2. `node` specifies the name of a previously defined node. The application will be attached to the specified node.

3. `port` specifies the port number on the specified node.

4. `repeat` specifies an optional repeat count, indicating how many applications are to be created.

3.7 The **animation** element

The **animation** element controls the creation of a log file that can later be used as input to the network animation program `NetVis`. The *NetVis* animator is a network visualizer that is designed to provide animations from a variety of network simulation tools. The input to *NetVis* is an *XML* file describing the topology (nodes and links), and all packet transmissions and receptions. It supports both wired and wireless simulations, and will be included in the contributed code portion of future *ns–3* releases. The **animation** element has two required attributes and one optional attribute. No sub–elements are allowed. The attributes are:

1. `format` specifies the format type of the animation output file. Allowable values are `XML` and `TXT`.

2. `filename` specifies the name of the animation output file.

3. `stoptime` is optional and specifies a simulation time at which the animation file will close. If not specified, the animation file is closed when the simulation completes.

3.8 The **statistics** element

The **statistics** element delineates a list of **log** sub–elements that specify what statistics are to be logged for this experiment. Many of the *ns3xml* elements have one or more sets of statistics that are automatically gathered, and can optionally be logged. An example is the total bytes received by the *PacketSinkApplication*. Another example is the running average queue occupancy for packet queues. Details of how to specify which statistics are desired are given in the following section on specifying sub–items. The **statistics** element has no attributes and allows only the **log** sub–element.

3.8.1 The **log** *element*

The **log** element specifies a single statistic item that is to be logged, possibly periodically, in the current experiment. There are several allowed attributes, the first two of which are required. Those are:

1. `name` specifies a meaningful name for the statistic that is noted on the statistics log file.

2. `stat` specifies the particular statistics to be logged.

3. `repeat` specifies the number of time the log element is to be repeated. This might be used to specify the same statistic is to be gathered on all applications created with a repeat count in a specific application sub–element.

4. `at` specifies the time at which the specified statistic is to be logged. If not specified it is logged at the end of the simulation.

5. `frequency` specifies the repeat interval for the statistic logging. If not specified the statistic is logged only once.

Refer to Appendix A for a simple, but complete, input file for *ns3xml*. This example uses most of the allowable elements and sub–elements, and illustrates the use of named *sub–items* described in the next section.

4. SPECIFYING ELEMENT SUB–ITEMS

In many instances, there must be some way to refer to *sub–items* within a given instance of an *ns3xml* element. An example is that when a **dumbbell** element is created there must be some way to later refer to an individual *ns–3* node in the dumbbell object. In *ns3xml*, this is accomplished by the use of element *names* and *sub–items*.

Many of the *ns3xml* elements have defined names that can be specified by pre-pending a dollar sign ($) to the element name. For example, if a **node** element is created with the name attribute `MyRouterNode`, it can later be referenced by the string `$MyRouterNode`.

Additionally, many of the *ns–3* elements have one or more *sub–items* that are specific to that element type. For example, **node** elements have a sub–item called `IPAddress` that refers to the IPv4 address of the zero'th interface assigned to that node. As another example, the **packetsinkapp** element has a sub–item denoted `totalBytesRx` that is used to specify the "total bytes received" statistic gathered by the `PacketSinkApplication`. To refer to a specific sub–item from a specific element, append a dollar sign to the element name followed by the sub–item name.

Several of the *ns3xml* elements allow a `repeat` attribute that specifies more than one *ns–3* object is to be created. For example, the **node** element allows `repeat` which results in multiple *ns–3* nodes being created. To refer to an individual *ns–3* node within a repeated **node** element, append a dollar sign followed by a numeric value after the name of the **node** element. To refer to the second node within a **node** element named `myRepeatedNode`, specify the string `$myRepeatedNode$2`.

Finally, it is occasionally necessary to refer to the *repeat index* when processing a repeated element. For example, to create a `PacketSinkApplication` on every node on the left side of a **dumbbell** named `myDumbbell`, the **packetsink** element would be given as:

```
<packetsink
  name = "mySink"
  repeat = 10
  node = "$myDumbbell$left$i />
```

In the descriptor above, the substring `$myDumbbell` refers to the dumbbell object, `$myDumbbell$left` refers to the set of left side nodes, and `$myDumbbell$left$i` refers to the i^{th} node in the set of left side nodes, where i is the repeat index for the **packetsink** repeat.

5. STATISTICS LOGGING

As discussed briefly earlier when describing the various *ns3xml* element types, the design supports logging of the performance metrics collected by the *ns–3* simulation. The **statistics** element and the various **log** elements specify the name and frequency of the desired statistics.

The format of the created statistics file is a simple *comma separated value* (*CSV*) file which is a simple standard compatible with several major analysis tools such as the popular *Excel* product. It has four fields, containing the run number, the time of the statistics collection, the name of the statistic being logged, and the actual value of the statistic.

There are a number of potential improvements that should be considered for this function. These improvements are future work in the continued development of *ns3xml*.

6. SUMMARY AND FUTURE WORK

We have described in detail the design of our *ns3xml* front–end processor for *ns–3*. The *ns3xml* tool will enable those students and faculty without the C++ programming skills necessary to master the complete *ns–3* environment to create simple yet instructional simulation experiments. While we do not yet have any experience with novice users attempting to construct *ns–3* simulations with *ns3xml*, we will insure that faculty at Georgia Tech who are teaching undergraduate students in introductory networking classes have access to our tool.

As of this writing, nearly all of the features described here have been implemented and tested. Still to be implemented are the statistics gathering function, the animation interface, and the sub–item naming and retrieval feature. We expect those to be fully completed by the end of calendar year 2010.

It is clear that the design and implementation of *ns3xml* is very tightly coupled to the models found in the *ns–3* environment. As new models and object are added to the *ns–3* simulator, additional elements and attributes must be added to *ns3xml* to enable the use of these new features. We plan to provide an updated release of *ns3xml* for each *ns–3* release that contains new object models of interest to the classroom teaching environment.

7. REFERENCES

[1] T. Anderson. Introducing XML. On-line: http://www.itwriting.com/xmlintro.php, 2004.

[2] T. W. Consortium. XML web page. On-line: http://www.w3.org, 2010.

[3] D. Raggett, J. Lam, I. Alexander, and M. Kmiec. A history of xml. On-line: http://www.w3.org/People/Raggett/book4/ch02.html, 1998.

[4] The NS3 Development Team. The NS3 network simulator. Software on-line: http://www.nsnam.org, 2010.

Appendix A - Sample *ns3xml* XML file

```
<ns3 repeat = "10" stopAt = "100.0">
  <cmdline name = "queueLim" default = "5000"
           help = "Size of DropTailQueue in bytes" />
  <cmdline name = "queueType" default = "DropTail"
           help = "Type of queue to use by default" />
  <cmdline name = "runNumber" default = "0"
           help = "Run number for RNG streams" />
  <config>
    <default typeid = "DropTailQueue" attr = "MaxBytes" value ="$queueLim" />
  </config>
  <randomgenerators runNumber = "$runNumber" seed = "123456" >
    <exponential name = "exp1" mean = "1.0" limit = "100.0" />
    <uniform     name = "uni1" min = "0.0" max = "1.0" />
    <normal      name = "norm1" mean = "0.0" variance = "1.0" />
    <constant    name = "const1" value = "0.0" />
  </randomgenerators>
  <topology autoIPV4 = "true" GODRouting = "true"
            autoIPV4Base = "1.0.0.0" autoIPV4Mask = "255.255.0.0">
    <queue name = "dtqueue1" type = "DropTail" limittype = "BYTES" limit = "10000" />
    <p2plink name = "fastlink" bw = "100Mbps" delay = "1ms" queue = "dtqueue1" />
    <p2plink name = "medlink"  bw = "10Mbps" delay = "1ms" />
    <p2plink name = "slowlink" bw = "1Mbps" delay = "1ms" />
    <node name = "starleaf" repeat = "10" />
    <node name = "starhub" x = "10" y = "10" />
    <connectp2p repeat = "10" n1 = "$starhub" n2 = "$starleaf$i" link = "$fastlink" />
    <dumbbell name = "mydb" link = "fastlink" nleaf = "10"
              x = "100" y = "100" w = "10" h = "10" />
  </topology>
  <applications>
    <onoff name = "onoffapp" sockettype = "TCP"
           rate = "1Mb" ontime = "$exp1" offtime = "$exp1"
           node = "$mydb$left$0"
           remotenode = "$mydb$right$0" remoteport = "10000"
           startat = "$uniform1" />
    <bulksend name = "bulksendapp" maxbytes = "100000000"
              node = "$mydb$left$1"
              remotenode = "$mydb$right$1" remoteport = "10000"
              startat = "$uniform1"/>
    <packetsink name = "packetsinkapp" repeat = "10"
                sockettype = "TCP" node = "$mydb$right$i"
                port = "10000" />
  </applications>
  <animation format = "XML" filename = "myanimfile.xml"
             stoptime = "100.0" />
  <statistics>
    <log repeat = "10" name = "sinkRx"
         stat = "$packetsinkapp$0$totalBytesRx" at = "1.0"
         frequency = "1.0" />
  </statistics>
</ns3>
```