QOMET User's Guide

by Razvan Beuran

June 2013

Hokuriku StarBED Technology Center
National Institute of Information and Communications Technology
1-12 Asahidai, Nomi, Ishikawa, 923-1211 Japan
http://www.nict.go.jp

QOMET User's Guide

Contents

Preface

Wireless network experiments have always presented challenges for researchers, which lead to the fact that real-world trials were replaced almost completely by simulation experiments. While there is a recent tendency to lay again emphasis on real-world trials, the predominance of simulation results in scientific papers is still obvious.

Our work has started in 2006 as an attempt to provide researchers with another practical approach for performing experiments, that trades off the advantages and disadvantages of real-world trials and simulation experiments. This alternative approach is that of *network emulation*.

QOMET is a set of tools for network emulation that makes it possible to run a wide range of realistic network experiments in controlled conditions. QOMET development started with focus on wireless networks, since that field has been considered as the one that would most benefit from employing network emulation. Many features were added over time, including support for wired networks and node mobility. QOMET development is still ongoing.

While the QOMET design, modeling, and software development were mainly done by the members of the StarBED Project, we could have not achieved the current state without all the users that contribute continuously to the improvement of QOMET through their feedback and suggestions. Many thanks to all of them.

R. Beuran

1 Introduction

This is the user's guide for QOMET, the set of tools for network emulation developed at the Hokuriku StarBED Technology Center[1] of the National Institute of Information and Communications Technology in Japan. The QOMET acronym stands for "Quality Observation and Mobility Experiment Tools" as an indication of the wide range of possible uses of these tools for network application and protocol evaluation in scenarios including mobility.

This user's guide refers to QOMET v2.1 that was released in June 2013. The release contains several new features with respect to the previous version, the most important of them being the addition of support for the WiMAX PHY computation. We plan to follow on this in 2014 with a release that will also contain the WiMAX MAC support, so as to make WiMAX emulation possible.

Other improvements in QOMET v2.1 include: the addition of fading model (AWGN and Rayleigh) for communication environments, better support for JPGIS map data, and the possibility to define communication disruptions by means of electromagnetic noise injection. The library for access point operation emulation, called `station`, was somewhat improved but is still under development. For detailed information referring to the development of QOMET v2.1 please consult the files HISTORY and CHANGES included in the distribution.

1.1 Overview

QOMET is based on our two-stage approach to wireless network emulation [3, 4, 5] which is represented in Figure 1.1.

In the first stage, a scenario representation is converted into a sequence of quality degradation (ΔQ) states of the network, termed ΔQ *description*. This conversion is done in a layer-oriented manner, as we compute first the physical layer effects that the changes in the communication channel have on the received signal power.

[1] Our center was named "Hokuriku Research Center" until April 2011.

Figure 1.1: Two-stage scenario-driven network emulation.

For WLANs, this is followed by an estimation of the frame error rate (FER) based on the received signal power and the noise power, and by taking into account the specificities of the 802.11 MAC layer. Finally, the network layer conditions are determined: packet loss, delay & jitter, and available bandwidth. Similar techniques are used for the other supported network technologies.

In the second stage, the ΔQ description, which accurately reproduces the wireless network behavior corresponding to the emulated scenario, is used to configure a link-level network emulator, such as `dummynet` [14], to recreate this behavior in a wired network.

We have performed several experiments using QOMET, with network setups ranging between a few and over one hundred PCs, each emulating one or multiple network nodes. Experimentation was mainly done on StarBED, the large-scale network experiment environment managed by our center [13]. The wireless network emulation testbed called QOMB (QOMet on starBed) integrates QOMET with StarBED so as to facilitate making large-scale experiments [6, 7].

1.1.1 Features

Initially focusing on IEEE 802.11a/b/g (WLAN) emulation, QOMET is being continually developed and improved by adding new features. In addition to WLAN, QOMET provides now support for other wireless network technologies, such as IEEE 802.15.4, for realistic 3D virtual environments, for node mobility generation, etc.

In order to make it possible to run network emulation experiments that, in addition to wireless networks, also involve wired networks (such as the network to which an access point is connected), we have added basic emulation capabilities for Ethernet networks operating at 10/100/1000 Mb/s. However, QOMET remains focused on the challenging area of wireless networks, including mobility scenarios.

A summary of the most important features of QOMET v2.1 is presented in Table 1.1. In addition to them, some new features are still under development. Such features are support for IEEE 802.16e WiMAX/LTE emulation, support for access point operation emulation, and support for execution on FreeBSD 8.2 operating system. It is expected that these features will be introduced in the next QOMET release.

Table 1.1: Summary of current QOMET features.

Category	Supported features
Network technologies	IEEE 802.11a/b/g (WLAN) Active RFID tag communication IEEE 802.15.4 (basis for ZigBee) IEEE 802.3 (Ethernet) at 10/100/1000 Mb/s
Topography	2D objects (including based on real map data) 3D objects (2D contour + height information)
Mobility	Linear Circular Rotation Random walk Behavioral motion
Operating systems	Red Hat Enterprise Linux 6 and derivatives, such as Scientific Linux or CentOS
Other features	Antennas (both 2D and 3D, omnidirectional and directional) Support for OLSR routing

1.1.2 Implementation

The QOMET software described in this user's guide implements first of all the conversion of the real-world wireless network scenario into the corresponding ΔQ description that characterizes the changing network states, as mentioned above. This function is achieved by means of the component called `deltaQ`.

The scenario representation used as input for `deltaQ` is written in XML format. It must include all the components of the scenario, such as nodes and objects, and their properties, including connectivity, mobility, etc. The `deltaQ` text output is a file that contains for each moment of time the most important ΔQ parameters: bandwidth, packet loss, and delay & jitter. Other parameters are included as well, e.g., the distance between nodes, the received power in dBm, etc. An equivalent output in binary format is also generated, providing a more compact representation for experiment purposes.

The output of QOMET is intended for driving a link-level network emulator to recreate the computed quality degradation in a real wired-network environment. QOMET supports this functionality by means of the component called `wireconf`. Starting with QOMET v2.x releases, this module drives the `ipfw3` portable implementation of `dummynet`, and can be used on modern Linux OSes.

Although our software implementation is mainly intended for being used as a stand-alone framework, most of the components, such as the wireless network

communication emulation core, can also be used as libraries, and linked to other programs. This makes it possible for users to tightly integrate our network emulation code with their own custom programs, for instance related to node motion generation. More details about this possibility are presented in Chapter 4.

1.2 Document structure

This user's guide is structured as follows. In Chapter 2 we give the reference of the XML-based QOMET scenario representation. In Chapter 3 we present the practical use of QOMET, including instructions on how to conduct experiments. Chapter 4 provides an overview of the software components of QOMET, such as the wireless network communication emulation library, and the other libraries included with QOMET. The document ends with a section of references.

1.3 Acknowledgment

We would like to acknowledge the contribution to the development and testing of the QOMET software, and the improvement of the current user's guide of (in alphabetical order): Kunio Akashi, Khin Thida Latt, Junya Nakata, Lan Tien Nguyen, Trung Tran Nguyen, and Takashi Okada.

1.4 Contact information

In case you want to make any suggestions, either about the present user's guide, or about QOMET itself, please send your comments or bug reports to the following e-mail address: `info@starbed.org`.

2 Scenario representation

This chapter describes the syntax of the scenario representation file used as QOMET input which is written using the XML format [1]. For parsing the scenario file we use the eXpat XML parser library [9]. The scenario representation is composed of a top-level element, `qomet_scenario`, and several second-level elements, representing nodes, objects, environments, motion elements, and connections. Third-level elements are used to represent lower-level details, such as interface properties for nodes, and object coordinates.

2.1 `qomet_scenario` element

The top-level element in the QOMET scenario representation file is `qomet_scenario`. All the other elements must be included in this top-level element. The `qomet_scenario` element provides basic configuration options for the overall emulation scenario.

2.1.1 Attributes

The possible attributes of a `qomet_scenario` element are presented next, together with their roles, expected data types and values:

`start_time` Specifies the starting time of scenario execution in seconds.

- Data type: double
- Value: any value (default = 0 s)

`duration` Specifies the duration of scenario execution in seconds.

- Data type: positive double
- Value: any value (default = 0 s)

step Specifies the time interval at which ΔQ calculation of scenario properties is performed, in seconds[1].

- Data type: positive double
- Value: any value (default = 0.5 s)

motion_step_divider Divider that should be applied to the ΔQ calculation `step` parameter for the case of motion computation, so that the motion time step can be sufficiently small to produce correct results in various topographical scenarios. For instance, assuming that `step` was assigned the value 0.5, then you need to set `motion_step_divider` to 2 in order to instruct QOMET to compute motion with a time step of 0.25 s.

- Data type: positive double
- Value: any value (default = 1.0)

coordinate_system Specifies the coordinate system used in the scenario.

- Data type: string
- Value: { `cartesian` | `lat_lon_alt` } (default = `cartesian`)

jpgis_file_name Specifies the name of the JPGIS[2] formatted file from which object coordinates are to be loaded; UTF-8 encoding is required. See also Section 2.4 for details on how to specify objects.

- Data type: string
- Value: any value (no default value)

2.1.2 Example

Below is an example of a scenario starting at time 0 s, with a duration of 60 s, that will be calculated with a step of 0.5 s. Element definitions are not specified in this example; please consult the following sections for details on defining scenario components.

```
<qomet_scenario start_time="0" duration="60" step="0.5">
      ...
</qomet_scenario>
```

2.2 node element

The `node` element defines the properties of the emulated devices, that we call "nodes".

[1] This step is also used by `deltaQ` as the time granularity of the generated output.
[2] JPGIS is a GIS (Geographic Information System) data representation format widely used in Japan.

2.2.1 Attributes

The possible attributes of a `node` element, their roles and expected data types and values are the following:

`name` Specifies the name of the node, which is used to refer to that node in other places in the scenario, such as in motion definitions.

- Data type: string
- Value: any value (no default value)

`x, y, z` Specify the initial coordinates of the node; the measurement unit depends on the coordinate system used.

- Data type: double
- Value: any value (default = 0, 0, 0, respectively)

`internal_delay` Specifies the internal fixed delay in milliseconds reflecting node operations that are unaccounted for in the communication model.

- Data type: positive double
- Value: any value (default = 0 ms)

Some attributes of the `node` element, that have been obsoleted and will be discontinued in the future, or that are currently not used, were omitted from the syntax description for clarity purposes. These attributes are:

- `type`, `ssid`, `connection`: The node type, network SSID and connection type are intended for supporting AP functionality in the future, but are not currently in use;
- `id`: The node id, which is used internally by QOMET, is now automatically generated, and this attribute is currently ignored when parsing a scenario;
- `adapter` and related attributes, such as `Pt`: Adapter-related parameters have been grouped as a separate element called `interface`. Thus it is recommended that they are set via this element, as indicated in Section 2.3; this is mandatory for multi-interface nodes, which cannot be defined otherwise. These attributes are currently maintained as node attributes for back-compatibility purposes, but will be removed in the future.

2.2.2 Example

Below is an example of a node named "node0" located at the initial position $(5, 10, 15)$. Other node properties can be specified via the `interface` element (see Section 2.3).

```
<node name="node0" x="5" y="10" z="15"/>
```

2.3 `interface` element

The `interface` element defines the properties of the interfaces of emulated nodes.

2.3.1 Attributes

The possible attributes of a `interface` element, their roles and expected data types and values are the following:

`name` Specifies the name of the interface, which is used subsequently in `connection` elements to refer to this interface.

- Data type: string
- Value: any value (default = no value)

`adapter` Specifies the adapter model which is used for the interface; should be correlated to the emulated wireless network technology used for connections from the interface.

- Data type: string
- Value: { `orinoco` | `dei80211mr` | `cisco_340` | `cisco_abg` | `jennic` | `s_node` } (default = `cisco_abg`)

`Pt` Specifies the power transmitted by the node in dBm.

- Data type: double
- Value: any value (default = 20 dBm)

`antenna_gain` Specifies the antenna gain of the node transceiver in dBi.

- Data type: positive double
- Value: any value (default = 0.0 dBi)

`azimuth_orientation` Specifies antenna orientation in the azimuth (horizontal) plane, in degrees.

- Data type: double in the range [0°, 360°)
- Value: any value (default = 0°)

`azimuth_beamwidth` Specifies antenna beamwidth in the azimuth (horizontal) plane, in degrees[3].

- Data type: double in the range [0°, 360°]

[3] Antenna beamwidth is the angle made by the directions on which power attenuation equals 3 dB (i.e., power is halved), and centered on the the maximum power direction.

- Value: any value (default = 360°, signifying an omni-directional antenna in azimuth plane)

elevation_orientation Specifies antenna orientation in elevation (vertical) plane in degrees.

- Data type: double in the range [0°, 360°)
- Value: any value (default = 0°)

elevation_beamwidth Specifies antenna beamwidth in elevation (vertical) plane in degrees.

- Data type: double in the range [0°, 360°]
- Value: any value (default = 360°, signifying an omni-directional antenna in elevation plane)

ip_address Specifies the IP address that will be assigned to this interface during the experiment. Note that this information is only used to generate the settings file that is used by `wireconf`, and not to effectively configure an IP address to the emulated node, an action that has to be done by the user.

- Data type: string
- Value: any value[4] (default = no value)

noise_source Flag that specifies the current interface is actually a noise source, and its signal will be treated as interference to other nodes. In this case `Pt` indicates the signal strength of the noise source[5].

- Data type: string
- Value: { `true` | `false` } (default = `false`)

noise_start_time, noise_end_time Specify the beginning and the end time in seconds of the interval in which the noise source is active. To define multiple intervals you need to create multiple noise sources that are each active in one of the desired intervals.

- Data type: double
- Value: any value

[4] The assigned valued should represent a valid IPv4 address, although this is not currently checked in QOMET.

[5] The frequency band of the noise source cannot be specified at this moment, and it is assumed to be the same with those of the operating nodes.

2.3.2 Examples

Below is an example of a node named "node0" that has two network interfaces; other properties of the node, such as its position are not specified here. The first interface is named "if0", uses an "orinoco" adapter, and is associated with the IP address "192.168.1.1". The second interface is named "if1", uses a "cisco_abg" adapter, and is associated with the IP address "192.168.2.1".

```
<node name="node0" ... >
      <interface name="if0" adapter="orinoco" ip_address="192.168.1.1"/>
      <interface name="if1" adapter="cisco_abg" ip_address="192.168.2.1"/>
</node>
```

The next example defines the interface "if0" of the node "node0" as a noise source with P_t equal 20 dBm. The noise source is active in the interval 10 to 20 s.

```
<node name="node0" ... >
      <interface name="if0" adapter="orinoco" Pt="20"
                 noise_source="true" noise_start_time="10" noise_end_time="20"/>
</node>
```

2.4 `object` element

The `object` elements represent structures, such as roads, buildings, or any other obstacles, such as hedges. Such objects may interfere with the wireless communication between nodes, therefore must be taken into account. The height of the object is important in this respect, since only objects that are high enough to obstruct communication are taken into account. Some objects, such as the roads (which are considered to have no height) have no effect on communication. Nevertheless, they can be used to restrict node motion within the designated road boundaries. In addition, motion computation can takes into account non-zero height objects as well, such as buildings, which are avoided by pedestrians. Note that the exact influence of objects on motion depends on the select motion model, and currently only the behavioral model considers objects during motion computation.

2.4.1 Attributes

The possible attributes of an `object` element are presented next, together with their roles, expected data types and values:

`name` Optionally specifies the name of the object, and is typically used to convey more information about the object, such as its role in the scenario.

- Data type: string
- Value: any value (default = `UNKNOWN`)

`type` Specifies the type of the object. This information is used by QOMET when loading JPGIS data to determine the type of data that must be read.

- Data type: string
- Value: { building | road } (default = building)

environment Specifies the name of the environment associated to this object; the respective environment cannot be dynamic (see Section 2.6).

- Data type: string
- Value: any value (no default value)

x1,y1,x2,y2 Specify the coordinates of a rectangular object; the measurement unit depends on the coordinate system used.

- Data type: double
- Value: any value (no default value)

height Specifies the height of an object in meters. This attribute only has to be specified when the user wishes the scenario to be processed in 3D. The height is usually zero for objects such as roads, and non-zero for buildings or other obstacles interfere with signal propagation.

- Data type: positive double
- Value: any value (default = 0 m)

load_from_jpgis_file Specifies that the object coordinates should be loaded from the JPGIS-format file specified by the scenario attribute jpgis_file_name.

- Data type: string
- Value: { true | false } (default = false)

make_polygon Specifies that the current object should be made into a polygon by automatically connecting its last and first vertexes.

- Data type: string;
- Value: { true | false } (default = false)

load_all_from_region Specifies that all the objects in the region indicated by x1, y1, x2, y2 should be loaded from the JPGIS-format file specified by the scenario attribute jpgis_file_name.

- Data type: string
- Value: { true | false } (default = false)

Currently all 2D objects must be given to QOMET as polygons. This can be done easily by using the make_polygon attribute mentioned above. Alternatively, the last vertex coordinate can be the same with that of the first vertex. This mimics the convention used to represent polygons in JPGIS-format topology descriptions.

2.4.2 Examples

Below is an example of an object named "building" associated to an environment called "building_env" (not defined here). The object contour is a rectangle defined by the coordinates $(-10, -10)$ and $(10, 10)$.

```
<object name="building" environment="building_env"
        x1="-10" y1="-10" x2="10" y2="10"/>
```

The next example defines a set of objects whose name will be auto-generated starting from the base name "Roads". The objects will be created by loading all the objects of type "road" in the JPGIS file associated to the scenario that are fully contained within the region defined by the points (`x1`, `y1`) and (`x2`, `y2`) specified in latitude-longitude coordinates.

```
<object name="Roads" type="road" environment="env" height="0"
        load_from_jpgis_file="true" load_all_from_region="true"
        x1="35.540" y1="139.685" x2="35.550" y2="139.695" make_polygon="false"/>
```

2.5 `coordinate` element

The `coordinate` element represents coordinates of points, such as object vertexes. This element is currently used only to represent the 2D coordinates of a building or road contour, for example as it could be seen on a map. The object attribute height can be used to construct a 3D object.

2.5.1 Attributes

The possible attributes of a coordinate element are presented next, together with their roles, expected data types and values:

`name` Optionally specifies the name of the coordinate, for instance in order to convey more information about the coordinate, such as its index.

- Data type: string
- Value: any value (default = `UNKNOWN`)

The actual value of the coordinate is given by pairs of floating-point numbers enclosed between the start and end tags of the coordinate element. These numbers can represent either Cartesian or latitude & longitude coordinates, depending on the value of the `coordinate_system` attribute of the top-level `qomet_scenario` element (see Section 2.1). In order to associate some coordinates to a certain object, the corresponding coordinate element definitions must be included between the start and end tags of that object element.

2.5.2 Example

The example below shows the definition of an object element representing a triangle with coordinates $(0.0, 0.0)$, $(0.0, 10.0)$, and $(10.0, 0.0)$. The attributes of the object are not included in this example. Note that the last coordinate is repeated to form a polygon. The object attribute `make_polygon` could have been set to `true` instead.

```
<object ... >
        <coordinate> 0.0 0.0 </coordinate>
        <coordinate> 0.0 10.0 </coordinate>
        <coordinate> 10.0 0.0 </coordinate>
        <coordinate> 0.0 0.0 </coordinate>
</object>
```

2.6 `environment` element

The `environment` element describes a communication channel (environment) through which nodes can communicate using radio signals. In order to associate an environment with two nodes, the `connection` element must be used (see Section 2.8).

2.6.1 Attributes

The possible attributes of an `environment` element are presented next, together with their roles, expected data types and values:

`name` Specifies the name of the environment, which is used to refer to this element later in the scenario, for instance in object definitions.

- Data type: string
- Value: any value (no default value)

`is_dynamic` Specifies whether the environment is *dynamic* (i.e., its properties will be computed at each step depending on scenario topology) or *static* (i.e., its properties will never change).

- Data type: string
- Value: { `true` | `false` } (default = `false`)

`alpha`, `sigma`, `W` Specify the parameters α (attenuation constant), σ (shadowing parameter) and W (wall attenuation) of the log-distance path loss propagation model; α is unit-less, and σ and W are both expressed in dB — see [4] for details.

- Data type: positive double
- Value: any value (default = 0, 0 dB, 0 dB, respectively)

noise_power Specifies the power of the noise in dBm, as it would be sensed by a receiver located in this environment. This models uniform noise floors, as if coming from an infinitely-remote noise source.

- Data type: double
- Value: any value (default = -100 dBm, representing the thermal noise level for 802.11b/g operating frequencies)

fading Specifies which type of fading should be used for the environment.

- Data type: string
- Value: { `AWGN`[6] | `Rayleigh` } (default = `AWGN`)

The following environment attributes are not currently used, and may be removed in the future:

- `type`: Intended for specifying the type of the environment, such as indoor or outdoor.

2.6.2 Examples

Below is an example of an environment named "env", with values for the parameters α, σ and W being 5.5, 1 dB and 0 dB, respectively. The noise power for this environment is -100 dBm.

```
<environment name="env" alpha="5.5" sigma="1" W="0" noise_power="-100"/>
```

The next example defines an environment called "env_R" which is identical with the previous one except for the fact that the "Rayleigh" fading model will be used for propagation calculation instead of AWGN:

```
<environment name="env_R" alpha="5.5" sigma="1" W="0" noise_power="-100"
             fading="Rayleigh"/>
```

2.7 `motion` element

The `motion` element describes the motion pattern of a node. Note that, except for the behavioral motion in which objects are avoided, other motion types do not take into account objects, and node trajectory follows strictly the defined parameters in those cases.

[6] Additive White Gaussian Noise.

2.7.1 Attributes

The possible attributes of a `motion` element, their roles, expected data types and values are:

`node_name` Specifies the name of the node to which the current motion should be applied.

- Data type: string
- Value: any value (no default value)

`type` Specifies the type of the motion.

- Data type: string
- Value: { `linear` | `circular` | `rotation` | `random_walk` | `behavioral` | `qualnet` } (default = `linear`)

`speed_x`, `speed_y`, `speed_z` Specify for **linear** motion the speed on the $0x$, $0y$, and $0z$ axes, in meters per second.

- Data type: double
- Value: any value (no default value)

`center_x`, `center_y` Specify for **circular** motion the center in a horizontal plane with respect to which the motion is to be performed. The measurement unit depends on the coordinate system used.

- Data type: positive double
- Value: any value (no default value)

`velocity` Specify for **circular** motion the tangential velocity in the horizontal plane in meters per second. Positive velocity values signify anti-clockwise rotation, whereas negative values indicate clockwise rotation.

- Data type: double
- Value: any value (default = 0 m/s)

`rotation_angle_horizontal`, `rotation_angle_vertical` Specify for **rotation** motion the rotation angles in horizontal and vertical plane, respectively, in degrees.

- Data type: double in the range [0°, 360°)
- Value: any value (default = 0°)

`min_speed`, `max_speed` Specify for **random walk** motion the minimum and maximum movement speeds, in meters per second.

- Data type: positive double
- Value: any values, with `max_speed` ≥ `min_speed` (default = 0 m/s, and 1 m/s, respectively)

`walk_time` Specifies for **random walk** motion the walking time in seconds.

- Data type: positive double
- Value: any value (default = 5 s)

`destination_x`, `destination_y`, `destination_z` Specify the motion destination, and let QOMET compute the needed speed to reach that destination in the allocated time. These parameters can only be given for the **linear** and **behavioral** models. Measurement units depend on the coordinate system used.

- Data type: double
- Value: any value (no default value)

`mobility_file_name` Specifies the name of the file from which mobility data generated by an external tool should be loaded. The data must be stored in the QualNet format, and the type of the motion should be set to `qualnet`.

- Data type: string
- Value: { `linear` | `circular` | `rotation` | `random_walk` | `behavioral` | `qualnet` } (default = `linear`)

`start_time`, `stop_time` Specify the motion start and stop time in seconds.

- Data type: double
- Value: any value (default = 0 s, and 0 s, respectively)

The following should be considered when defining **random walk** motion. This motion type uses the parameters `min_speed` and `max_speed` to define the range of the randomly generated speed in this motion model. The moving direction is also randomly generated. The parameter `walk_time` specifies how long the node moves before changing both speed and moving direction.

The QualNet mobility format supported by QOMET specifies motion data in plain-text format, with each line having the following content:

```
NODE_ID TIME (X, Y, Z) AZIMUTH ELEVATION
```

where the meaning of each field is as follows:

- `NODE_ID`: The id of the node to which the entry should be applied (when used with QOMET, the id in the QualNet mobility file should match the auto-generated id for the node to which the mobility data is to be associated);
- `TIME`: The time in s for which the entry should be used;

- X, Y, Z: The coordinates of the node at the specified moment of time, either as (x, y, z) values or as latitude/longitude/altitude values;
- AZIMUTH, ELEVATION: Optional parameters specifying the azimuth and elevation of the mobile node (default values are) for both parameters).

2.7.2 Examples

Below is an example of a motion to be applied to the node "node0". The motion is a linear displacement with speed $(0.5, 0.0, 0.0)$. The start time is 0 s, and the stop time is 30 s.

```
<motion node_name="node0" type="linear"
        speed_x="0.5" speed_y="0.0" speed_z="0.0" start_time="0" stop_time="30"/>
```

The next example specifies that the mobility trace for the node "node1" should be loaded from the file "qualnet.mobility". The values from the trace with time values between 0 and 60 s will be used.

```
<motion node_name="node1" type="qualnet" mobility_file_name="qualnet.mobility"
        start_time="0" stop_time="60"/>
```

2.8 connection element

The connection element describes the connection between two nodes by means of a communication channel (environment). Connections are important elements in a scenario representation, since the ΔQ description is only computed for those nodes appearing in connection fields.

2.8.1 Attributes

The possible attributes of a connection element are presented next, together with their roles, expected data types and values:

from_node Specifies the name of the transmitting node.

- Data type: string
- Value: any value (no default value)

to_node Specifies the name(s) of the receiving node(s); multiple names should be separated by a space character.

- Data type: string
- Value: any value (no default value). The auto_connect value has a special meaning, as it instructs QOMET to connect the corresponding from_node to all the nodes that were defined before this connection[7].

[7] If such a connection has a dynamic environment, then all the necessary environments needed for the automatically-generated connections will be created as well.

through_environment Specifies the name of the environment through which the nodes communicate.

- Data type: string
- Value: any value (no default value)

standard Specifies the wired or wireless network technology used by this connection.

- Data type: string
- Value: { `802.11a` | `802.11b` | `802.11g` | `eth_10` | `eth_100` | `eth_1000` | `active_tag` | `zigbee` } (default = `802.11b`)

rate Specifies whether the wireless network operating rate is adaptive or fixed. For the adaptive case, the Auto Rate Fallback (ARF) algorithm is used [11].

- Data type: string
- Value: { `adaptive` | `1Mbps` | `2Mbps` | `5.5Mbps` | `11Mbps` | `6Mbps` | `9Mbps` | `12Mbps` | `18Mbps` | `24Mbps` | `36Mbps` | `48Mbps` | `54Mbps` } (default = `adaptive`);

channel Specifies the channel on which the transceiver operates (if applicable, i.e. only for wireless technologies that use multiple channels).

- Data type: positive integer (including zero)
- Value: any value between 0 and 200 (default = 1 for 802.11b/g, 36 for 802.11a, and 11 for 802.15.4, respectively)

RTS_CTS_threshold Specifies the packet size above which the RTS/CTS mechanism in IEEE 802.11 standard is enabled (if applicable).

- Data type: positive integer
- Value: any value between 0 and 2347 (default = 2347, which completely disables the RTS/CTS mechanism)

packet_size Specifies the average packet size in bytes *at network layer* that is transmitted through this connection. For IEEE 802.11a/b/g emulation, the actual average packet size will be computed dynamically during the experiment, and the value specified here is used only for the initial offline calculations.

- Data type: positive integer
- Value: any value (default = 1024 bytes)

bandwidth Specifies the average bandwidth for this connection in bits per second. This is a way for users to control precisely the emulated bandwidth if they desire to do so.

- Data type: positive double
- Value: less or equal 1,000,000,000 bps (default = *automatically computed*)

loss_rate Specifies the average packet loss rate of this connection as a probability. This is a mechanism for users to control precisely the emulated loss rate if they desire to do so.

- Data type: positive double
- Value: less or equal 1 (default = *automatically computed*)

delay, jitter Specify the average delay and jitter of this connection in milliseconds. This represents a way for users to control precisely the emulated delay and jitter if they desire to do so.

- Data type: positive double
- Value: any value (default = *automatically computed*)

consider_interference Specifies whether interference between nodes is to be taken into account when computing *offline* the communication conditions for 802.11a/b/g emulated network. We stress that this parameter only refers to the offline computation, as a way to estimate worst-case conditions, but interference and contention are accounted for anyway dynamically during experiment execution.

- Data type: string;
- Value: { `true` | `false` } (default = `true`)

The following should be considered when defining connections:

- The `auto_connect` option of connections *only* generates connections to the nodes that were defined *before* that connection definition, and ignores the nodes that will be defined after it in the scenario representation.
- If a parameter such as `bandwidth`, `loss_rate` or `delay` are statically specified for a connection, then the corresponding constant values will be used for the entire duration of the experiment instead of the values that are computed by QOMET. This is particularly useful for defining the conditions of Ethernet connections, or to control the emulated network conditions for various experiment purposes. When one desires to have varying conditions during an experiment, the alternative `fixed_deltaQ` element should be used (see Section 2.9).
- Typical settings for IEEE 802.11a/b/g channels in Japan are between 1 and 13 for 802.11b/g, and within the set of values {36, 40, 44, 48, 52, 56, 60, 64} for 802.11a.
- Typical channel settings for IEEE 802.15.4 are between 11 and 26.

2.8.2 Example

Below is an example of a connection from node "node0" to the nodes "node1" and "node2" through the environment named "env0". The wireless network standard used is 802.11g, and the estimated average packet size is 200 bytes:

```
<connection from_node="node0" to_node="node1 node2" through_environment="env0"
            standard="802.11g" packet_size="200"/>
```

2.9 `fixed_deltaQ` element

The `fixed_deltaQ` element is used to specify fixed network conditions for a certain period of time. This is an extension of the statically-defined network conditions by using the attributes `bandwidth`, `loss_rate`, etc. of the `connection` element. This extension provides more flexibility to the users. A potential use of this element is in the context of fault injection, to introduce temporary network faults at specified moment of time in a scenario. The `fixed_deltaQ` element should be included in the `connection` element to which it refers.

2.9.1 Attributes

The possible attributes of a `fixed_deltaQ` element are presented next, together with their roles, expected data types and values:

`start_time`, `end_time` Specifies the time moments between which the specified fixed ΔQ conditions should be applied.

- Data type: double
- Value: any value (default = 0 s, and 0 s, respectively)

`bandwidth` Specifies the fixed average bandwidth for this connection in bits per second.

- Data type: positive double
- Value: less or equal 1,000,000,000 bps (no default value)

`loss_rate` Specifies the fixed average packet loss rate for this connection as a probability.

- Data type: positive double
- Value: less or equal 1 (no default value)

`delay`, `jitter` Specify the fixed average delay and jitter for this connection in milliseconds.

- Data type: positive double
- Value: any value (no default value)

2.9.2 Example

Below is an example of a connection that uses fixed ΔQ parameters (the connection properties are not specified here). In the interval from 0 s to 10 s and in the interval from 20 s to 30 s, the connection will have a 10 Mb/s bandwidth, zero packet loss probability, 5 ms delay and 1 ms jitter. However, in the interval from 10 s to 20 s, the connection will have a 0.5 Mb/s bandwidth, packet loss probability equal to 0.9, a 25 ms delay and a 5 ms jitter, representing a significant network failure in that interval:

```
<connection ... >
      <fixed_deltaQ start_time="0"  end_time="10"
                    bandwidth="10e6" loss_rate="0.00" delay="5" jitter="1"/>
      <fixed_deltaQ start_time="10" end_time="20"
                    bandwidth="0.5e6"loss_rate="0.90" delay="25" jitter="5"/>
      <fixed_deltaQ start_time="20" end_time="30"
                    bandwidth="10e6" loss_rate="0.00" delay="5" jitter="1"/>
</connection>
```

2.10 Remarks

We give below several clarifications regarding the information provided in the previous sections. Some of the restrictions mentioned here apply to the current version of QOMET, but may change in future versions.

1. The strings used to define element attribute names, names of nodes, topology objects and motion elements are all *case sensitive*.

2. Except for the `qomet_scenario` element, which must be the first element defined (as it is the top-level XML document entity), for all the other elements the order in which they are defined is inconsequential. However, some elements can only be defined within other elements, such as `interface` being used within `node` elements, or `coordinate` being used within `object` elements.

3. At the moment of writing this document, there are only some types of checks performed at parse time, such as checking that mandatory attributes were assigned values, checking that attribute values have the correct data type and data range, and checking that nodes, objects or environments with the same name were not defined multiple times. Other incongruities, such as defining a motion for an nonexistent node, will however be signaled as warnings during scenario processing.

4. Currently, the number of elements of each type in a scenario (nodes, interfaces, objects, coordinates, environments, motions, and connections) is limited to fixed maximum values. These parameters can be changed in the source code if required (specifically in the file “deltaQ/scenario.h”). An error will be displayed in case the current data structures cannot fit all the elements of a

certain scenario. Note, however, that using larger values increases the memory requirements of QOMET, and sufficient RAM must be present on the PC used to run the software.

5. For all internal processing operations, node names are used to identify nodes. However, in the text output file, only node ids are included. This is to facilitate post-processing and plotting by using shell scripts and Matlab. The same holds for the binary output file.

3 Utilization

QOMET is a set of wireless network emulation tools provided as a collection of stand-alone programs and libraries. This chapter provides more details on how to employ the QOMET set of software tools in practice. We focus here on the usage of the stand-alone programs, which are recommended for typical users. Nevertheless, for advanced users, details about how to employ directly the provided libraries, for instance the wireless network emulation library for computing ΔQ parameters, will be given in Chapter 4.

3.1 Overview

The use of QOMET involves a scenario-driven approach that consists of two main stages, as previously mentioned in Section 1.1:

1. Offline processing of the scenario representation that describes the emulation experiment to create the corresponding ΔQ description;

2. Performing the effective emulation experiment by using the generated ΔQ description to configure the communication conditions in the wired network.

Figure 3.1 summarizes this two-stage process, while also highlighting the names of the programs used in each of them: `deltaQ` for the first stage, and `wireconf` for the second one.

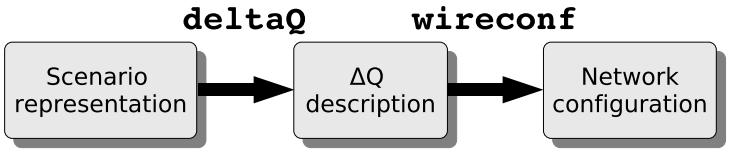

Figure 3.1: Using QOMET tools for two-stage scenario-driven network emulation.

Before doing any processing, the user has to create the scenario representation describing the conditions that he or she wants to emulate. At present this has to be done by creating an XML file by following the syntax presented in Chapter 2. No special tools are required for the scenario representation creation, and it can be done by using any text editor[1]. For reference, an example scenario is provided later in Figure 3.6.

In what follows we describe in detail the two main stages related to the use of QOMET.

3.2 Offline processing

The scenario created using the syntax given in Chapter 2 needs to be processed before actually performing an emulation experiment. This offline processing serves several purposes:

- *Scenario validation*: Determine whether the scenario representation syntax is correct, and whether the created representation is consistent.
- *Pre-computation of fixed parameters*: Compute the data that does not change during the experiment (such as the trajectory of the nodes, in case it is fixed). This offloads some processing that would otherwise have to be done in real-time when running the experiments, and thus makes it possible to run larger-scale experiments.
- *Pre-computation of ΔQ description*: Compute the estimated ΔQ description for the emulated scenario, so that the user can assess in advance whether it takes place as intended in terms of communication range, node mobility, etc.

This offline processing is done by using the stand-alone program called `deltaQ`, which uses the library having the same name to perform computations (see Section 4.1). The `deltaQ` program was implemented in C language. It makes use of the eXpat XML parser library [9] that needs to be compiled and installed in advance on the platform on which `deltaQ` is to be run. Compiling `deltaQ` should be done using the provided Makefile. So far `deltaQ` has been compiled mainly on UNIX systems such as Linux and FreeBSD. Other UNIX operating systems such as Sun Solaris, or Windows with Cygwin/Visual C compiler are only partially supported.

3.2.1 Command execution

Once `deltaQ` is compiled and ready to use, the user must execute it as follows:

```
> deltaQ <scenario_file.xml>
```

[1] In the future, a graphical user interface may also be made available to facilitate the scenario creation task.

where <scenario_file.xml> is the name of the file in which the XML scenario representation was stored.

Command-line options can be provided to deltaQ to control its execution. The options are summarized in Figure 3.2. By default, deltaQ will do ΔQ computation, and will generate both text and binary ΔQ output, but no motion data.

```
General options:
-h, --help            - print this help message and exit
-v, --version         - print version information and exit
-l, --license         - print license information and exit

Output control:
-t, --text-only       - enable ONLY text deltaQ output (no binary output)
-b, --binary-only     - enable ONLY binary deltaQ output (no text output)
-n, --no-deltaQ       - NO text NOR binary deltaQ output will be written
-m, --motion-nam      - enable output of motion data (NAM format)
-s, --motion-ns       - enable output of motion data (NS-2 format)
-j, --object          - enable output of object data
-o, --output <name>   - use <name> as base for generating output files,
                        instead of the input file name

Computation control:
-d, --disable-deltaQ  - disable deltaQ computation (output still generated)
```

Figure 3.2: Summary of deltaQ command-line options.

3.2.2 Output files

As a result of executing the deltaQ command, the scenario is parsed and processed. By default, the output is stored in a file with the name created by appending specific extensions to the scenario name, as follows: the extension ".out" for the text format (hence the resulting file will be named something like "scenario_file.xml.out") , the extension ".bin" for the binary format, and the extension ".motion" for the motion data.

At this moment, the following node properties are written in the text output file for each defined connection at each moment of time (one line per entry):

```
time from_node_id from_node_x from_node_y from_node_z to_node_id to_node_x
to_node_y to_node_z distance Pr SNR FER num_retr op_rate bandwidth loss_rate
delay jitter
```

Binary output is generated in a file with the appended extension ".bin" (e.g., "scenario_file.xml.bin"). The advantage of the binary output is that the file size is significantly reduced when compared to the text output, since only part of the information existing in the latter is written, and this is done in a binary form. In addition, if communication conditions do not change in a certain interval, no binary records are written for that interval, thus further reducing file size. The binary

format is the only format supported for effectively running emulation experiments, as it will be discussed next.

Starting with QOMET v2.x releases, the settings file that is required for making an emulation experiment is also automatically generated with the appended extension ".settings" (e.g., "scenario_file.xml.settings"). The generated settings file contains in the first column the names of the nodes, in the second column the names of the interfaces, and in the third column the automatically generated unique id that corresponds to a certain node and interface pair. The fourth column contains the IP address that will be assigned by the user to that interface. If no IP address was associated with certain interfaces (by using the `ip_address` attribute of the `interface` element), the IP address field is assigned the value "aaa.bbb.ccc.ddd" which should be replaced by the user with the correct value by editing the generated file. An example generated settings file will be shown later in Figure 3.7.

3.3 Effective emulation

We shall illustrate the effective emulation process with a simple setup for wireless network emulation, as depicted in Figure 3.3. The experiment hosts in Figure 3.3 play the role of wireless nodes, even though they are connected via a wired network switch, shown in the center of the figure. The communication conditions in the wired network are changed in a controlled fashion so as to reproduce the emulated wireless network. This is achieved by means of QOMET, which is denoted by the letter "Q" in the figure. This approach is called *distributed emulation* because all the nodes participating to the emulation experiment are in charge of emulating their own communication conditions. Another approach is that of *centralized emulation*, in which a computer is in charge of emulating the communication conditions for all the other experiments hosts. However, since the central computer may become a bottleneck for large experiments, we use almost exclusively the distributed approach. For a detailed introduction to network emulation and the related concepts see [2].

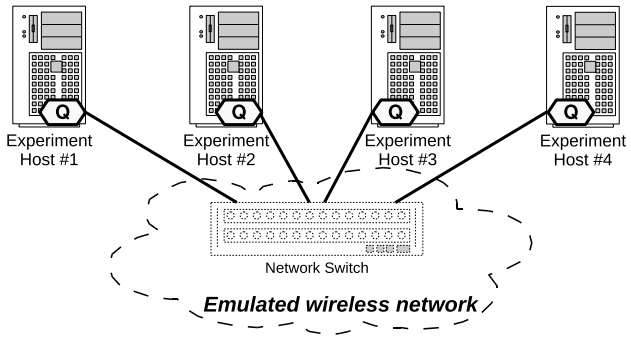

Figure 3.3: Distributed emulation setup.

In practice, QOMET uses the component called `wireconf`, which uses the library having the same name (see Section 4.2) to "drive" a link-level network emulator by periodically changing its configuration parameters (bandwidth, packet loss, delay). This makes it possible to reproduce the changing communication conditions of a wireless network in the wired network over which the experiment is actually carried out. Compiling `wireconf` should be done using the provided Makefile, and it requires the following libraries to be already compiled:

- `ipfw3`: Needed for network emulation features. This library is based on the open-source code available at [14], with minor modifications to integrate it with QOMET. Compilation should be done by using the included Makefile.
- `timer`: Needed for event timing support. Compilation should be done by using the included Makefile.

Mainly due to the requirements of these two libraries, `wireconf` can now only be used on Red Hat Enterprise Linux 6 based systems, with support for FreeBSD 8.2 being under development.

3.3.1 Command execution

Once the `wireconf` program is compiled and ready to use, it should be executed as follows:

```
> wireconf -q <qomet_output.bin> -i <current_id> -s <settings_file>
```

where `<qomet_output.bin>` is the name of the file in which the `deltaQ` binary output[2] is stored, `<current_id>` is the automatically generated id[3] of the node and interface pair to which the current `wireconf` instance is associated, and `<settings_file>` is the name of the file which describes the IP address settings for the hosts involved in the emulation experiment.

The full list of configuration options for `wireconf` is given in Figure 3.4.

3.3.2 Using SpringOS

The command `wireconf` can be executed directly by the user for each experiment host (manually or, better, through shell scripts). However, for a large number of hosts this may lead to timing problems, as synchronous execution of commands becomes difficult. Therefore, for large-scale and repeatable experiments a an alternative approach should be used, as discussed next.

[2]Note that starting with the QOMET v2.x releases the text format output of `deltaQ` is not supported by `wireconf` anymore, and it is only intended for viewing and analysis with external programs (scripts, Matlab, etc.).

[3]The value of this id can be retrieved from the settings file created by `deltaQ`, as explained in Section 3.2.2.

```
General options:
 -h                     - print this help message and exit
 -v                     - print version information and exit

Emulation parameters:
 -q <qomet_output.bin>  - provide deltaQ output for the emulation
 -i <current_id>        - set the id of the current node/interface
 -s <settings_file>     - provide various emulation settings
 -m                     - use MAC addresses in settings file instead of IP
 -b <broadcast_address> - (optional) provide broadcast address

Debugging options:
 -n                     - disable real-time adjustment of deltaQ parameters
                          (e.g., due to contention)
```

Figure 3.4: Summary of `wireconf` command-line options.

SpringOS is the experiment-support software for StarBED. By using the tools of SpringOS one can easily perform experiment control functions, such as powering on and off the experiment hosts, reading and writing operating system images to them, configuring the network topology by using VLANs, and so on.

Another function of SpringOS is to manage the experiment execution. By using SpringOS the user can define the tasks of each of the experiment hosts, and also can start the experiment itself. By using this mechanism it becomes possible to run experiments with a large number of hosts.

In order to use SpringOS, one has to write a specific script that describes the experiment execution. An example of such a SpringOS script is provided later in Figure 3.9. For more information on SpringOS and its utilization please see the corresponding user manual and tutorial [8, 12].

Experiment execution via SpringOS is effectively started by using the `master` command in SpringOS, as follows:

```
> master -S <preamble.sc> <springos_script.sc>
```

where `<preamble.sc>` is a configuration script for various settings of the experiments, including user credentials, and `<springos_script.sc>` is the name of the file containing the SpringOS script for that experiment..

3.4 Usage example

In this section we shall present a utilization example, detailing all the necessary actions when using QOMET. In particular, the following steps are generally required in order to run a SpringOS-driven emulation experiment:

1. Write the QOMET scenario for the target experiment;
2. Process the QOMET scenario using the `deltaQ` command;

3. Write the SpringOS script for the experiment;

4. Distribute the QOMET output and other required files to the experiment hosts;

5. Run the experiment by using SpringOS.

Even if a user chooses not to use SpringOS, an equivalent mechanism, such as shell scripts, is required at step 3 above, and the experiment has to be executed using remote access tools such as `ssh` at step 5.

In what follows we shall illustrate each of these steps for the particular case of a simple target experiment involving two nodes, one fixed and one mobile, located at an initial distance of 5 m. The mobile node moves from time 0 to 30 s with speed 0.5 m/s from left to right, and then from time 30 to 60 s in the opposite direction with the same speed. The experiment setup is shown in Figure 3.5.

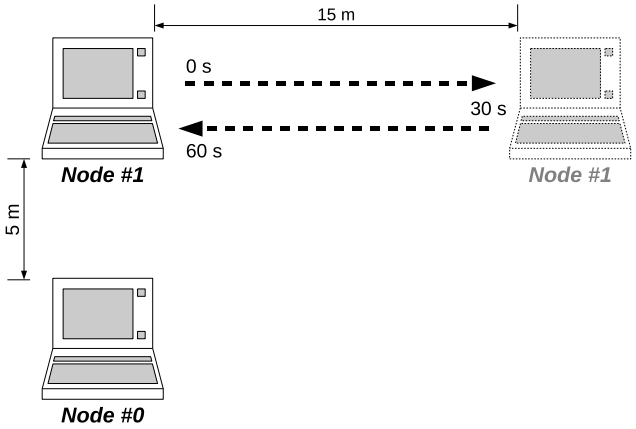

Figure 3.5: Setup of the example two-node experiment.

3.4.1 QOMET scenario

The QOMET scenario for an experiment such as the one discussed here must specify all the elements involved, such as:

- Scenario properties (duration, etc.);
- Wireless nodes
- Communication environment;
- Connections between nodes;
- Node movement.

In more complex situations, the user must also specify the topography of the virtual world in which the experiment takes places, such as streets and buildings. All these should be done according to the scenario representation syntax detailed in Chapter 2.

In Figure 3.6 we show the corresponding QOMET scenario representation for the two-node experiment example introduced in Figure 3.5 (line numbers are only shown for the reader convenience, and are not part of the scenario itself). Let us detail the scenario representation is what follows:

- *Line 1*: Specifies the global properties of the scenario, such as duration (60 s) and processing step (0.5 s);
- *Lines 3-4*: Define the properties of the wireless nodes, such as their name ("node0" and "node1"), the initial position ($(0, 0, 0)$ and $(0, 5, 0)$), and the transmit power (20 dBm);
- *Line 6*: Specifies the properties of the communication environment, such as its name ("env"), the propagation attenuation, shadowing component standard deviation and wall attenuation (5.6, 0 dBm and 0 dBm, respectively), and the noise power (-100 dBm);
- *Lines 8-11*: Define the properties of the motion for "node1", such as the speed (0.5 or -0.5 m/s on the $0x$ axis, 0 otherwise), and the interval between which motion takes place (from 0 to 30 s for the first motion element, and from 30 to 60 s for the second one);
- *Lines 13-16*: Specify the properties of the connections between the two nodes, such as the environment used ("env"), the wireless network standard ("802.11b") and the default packet size (1024 bytes). These properties must be defined for both directions, from "node0" to "node1", and from "node1" to "node0", and they can be different if the user decides so;
- *Line 18*: Close the top-level XML element `qomet_scenario` that was started on line 1.

3.4.2 Scenario processing

Assuming that the above scenario is saved into the file "scenario_file.xml", processing the scenario means running the `deltaQ` command as indicated in Section 3.2.1, and providing the scenario name as its argument:

```
> deltaQ scenario_file.xml
```

This command execution will validate the scenario syntax, compute the trajectory of the mobile node "node1", and will pre-compute the communication conditions between the two nodes. As a result of the processing, the files "scenario_file.xml.out"

```
 1  <qomet_scenario duration="60" step="0.5">
 2
 3  <node name="node0" x="0" y="0" z="0" Pt="20"/>
 4  <node name="node1" x="0" y="5" z="0" Pt="20"/>
 5
 6  <environment name="env" alpha="5.6" sigma="0" W="0" noise_power="-100"/>
 7
 8  <motion node_name="node1" speed_x="0.5" speed_y="0" speed_z="0"
 9          start_time="0" stop_time="30"/>
10  <motion node_name="node1" speed_x="-0.5" speed_y="0" speed_z="0"
11          start_time="30" stop_time="60"/>
12
13  <connection from_node="node0" to_node="node1" through_environment="env"
14              standard="802.11b" packet_size="1024"/>
15  <connection from_node="node1" to_node="node0" through_environment="env"
16              standard="802.11b" packet_size="1024"/>
17
18  </qomet_scenario>
```

Figure 3.6: QOMET scenario representation for the two-node experiment.

and "scenario_file.xml.bin" will be created. The text output file "scenario_file.xml.out" should be inspected to make sure that the scenario representation matches the intended scenario. A technical computing environment such as Matlab can be used to plot different communication and network parameters; this is especially recommended for larger scenarios, since it makes it easier to spot errors. The binary output file "scenario_file.xml.bin" is the one that will be effectively used later for emulation experiments.

As already explained in Section 3.2.2, `deltaQ` also generates the settings file that is required for running emulation experiments. This file will be created with the name "scenario_file.xml.settings" for the QOMET scenario in our example. The generated settings file can be used either directly, or as a template that is to be modified by the user, depending on the settings given in the scenario representation. In the particular case of our two-node example, the generated settings file will look as shown in Figure 3.7.

```
node0 interface0 0 aaa.bbb.ccc.ddd
node1 interface0 1 aaa.bbb.ccc.ddd
```

Figure 3.7: Generated settings file for the two-node experiment.

The settings file contains in the first column the names of the nodes, and in the second column the names of the interfaces, which are automatically assigned in our case. The third column contains the automatically generated unique id that corresponds to a certain node and interface pair. The fourth column contains the IP address that will be assigned by the user to that interface. In our example, since no IP address was associated with any of the interfaces (by using the `ip_address`

attribute of the `interface` element), the IP address field is assigned the value "aaa.bbb.ccc.ddd", which should be replaced by the user with the correct IP address by editing the generated file. An example correct settings file for the two-node experiment is shown in Figure 3.8.

```
node0 interface0 0 192.168.3.10
node1 interface0 1 192.168.3.11
```

Figure 3.8: Generated settings file for the two-node experiment.

3.4.3 SpringOS script

In order to run an emulation experiment with the support of SpringOS, the users must also write a SpringOS script that will be used to drive the experiment. This script will be used by SpringOS to invoke the experiment execution on all the hosts. A SpringOS script must contain instructions for the following items:

- Define the actions to be executed by the experiment hosts;
- Select the experiment hosts through their IP addresses (optional);
- Synchronize execution through message passing.

The SpringOS script for the two-node experiment is shown in Figure 3.9. Understanding this script requires at least some basic knowledge of the SpringOS script syntax, and users are referred to [8] for details. Users should also bear in mind that using SpringOS on the StarBED testbed implies a registration procedure as well as making a reservation for the desired number of experiment hosts and VLANs, as detailed in [10].

Astute readers will notice that the SpringOS script in 3.9 uses the built-in command `callw` to call a shell script named "run_experiment_node.sh". This shell script is therefore executed on each of the two hosts on which our two-node example experiment is run. A possible such shell script is shown in Figure 3.10. The shell script starts the `wireconf` program, and uses the `iperf` command to send traffic between the two experiment hosts.

3.4.4 File distribution

When using SpringOS, the operating system of the experiment hosts is typically written by saving the image of a template host and writing it to all the other experiment hosts. This image should include the basic OS files, and all the other tools necessary for making the experiment. However, one often uses different scenarios on the same experiment setup, and repeating the imaging process is not very practical in such cases.

For updating the files that are particular to a certain experiment run in a more convenient way, we recommend using a distribution mechanism to propagate new or changed files to the experiment hosts. For this purpose basic UNIX tools such as `scp` or `rsync` are very well suited. Such a distribution mechanism could be used for propagating changes in the case of:

- Newly-generated QOMET binary output files;
- Modified programs and shell scripts;
- Modified configuration files.

3.4.5 Experiment execution

Effective experiment execution should be done as indicated in Section 3.3.2. For our particular example, assuming the SpringOS scenario in Figure 3.9 was named "springos_script.sc", execution is started with the command:

```
> master -S preamble.sc springos_script.sc
```

For details on how to create the file "preamble.sc" which is specific to each registered StarBED user and project, please see the SpringOS tutorial [12].

Once experiment execution finishes, the user should gather any logs or data files that may have resulted from running the experiment. The UNIX command `scp` can again be used for this purpose.

```
assure num_nodes=2
export num_nodes
nodeclass client_class
{
  method "thru"
  partition 2
  ostype "FreeBSD"
  scenario
  {
    # initialize constants
    test_name="two_node_test"
    test_duration="60"
    packet_size="1024"
    offered_load="200k"
    # receive my_id
    recv my_id
    # signal setup is finished
    send "setup_done"
    # wait for start message
    recv start_msg
    # run experiment
    callw "/bin/sh" "run_experiment_node.sh" test_name my_id \
      offered_load test_duration packet_size > "/tmp/scenario.log"
  }
}

# define the clients
nodeset clients class client_class num num_nodes
# set the IP addresses of the clients
for(i=0; i<num_nodes; i++)
{
  clients[i].agent.ipaddr = "172.16.3." + tostring(10+i)
  clients[i].agent.port = "2345"
}

# global experiment scenario
scenario
{
  # send id to nodes
  for(i=0; i<num_nodes; i++)
  {
    send clients[i] tostring(i)
  }
  # wait for all clients to finish setup
  sync
  {
    multimsgmatch clients "setup_done"
  }
  # send start message to all clients
  multisend clients "start"
  # wait for clients to end execution
  sleep 60
}
```

Figure 3.9: SpringOS script for the two-node experiment.

```
# command line arguments
test_name=$1
node_id=$2
offered_load=$3
test_duration=$4
packet_size=$5

# pre-defined values
first_node_id=0
IP_base=192.168.3

# start emulation program
sudo -b ../wireconf/wireconf -q $test_name -i $node_id -s node_settings.txt

# start iperf (node0 is server, node1 is client)
if [ $node_id -eq $first_node_id ]; then
  # starting iperf server
  iperf --server --udp --interval 0.5 --format k --len $packet_size &
  sleep $test_duration
  killall -INT iperf
else
  # starting iperf client
  iperf --client $first_node_IP --udp --interval 0.5 --format k \
    --len $packet_size --bandwidth $offered_load --time $test_duration
fi
```

Figure 3.10: Shell script to be called by SpringOS for the two-node experiment.

4 Software components

This chapter presents details about the QOMET implementation. This information can be used to directly employ the various QOMET libraries without the assistance of the provided stand-alone programs.

In certain situations, one may wish to use the `deltaQ` wireless network communication emulation library for computing ΔQ parameters independently from the `deltaQ` stand-alone program (see Section 4.1). This may be useful in cases when the node motion is not known in advance, for instance, when it is being decided as the nodes communicate with each other, as in the case of mobile robots or inter-vehicular communication.

Other QOMET libraries can also be used independently, such as the `wireconf` link-level emulator configuration library intended for facilitating experiments (Section 4.2), or the `timer` library used for periodic execution of tasks at accurate time intervals (see Section 4.3).

4.1 Network emulation library: deltaQ

The network emulation library `deltaQ` is represented by the file "libdeltaQ.a", and it can be used independently from the stand-alone program with the same name. Actually the stand-alone program itself uses this library, hence its source code, "deltaQ.c", and the associated Makefile, are good examples of how to use the library.

The most important functions of the `deltaQ` library are given below together with a short description of their role:

`scenario_deltaQ` The main function of the library that computes the ΔQ parameters for all the connections in the scenario. The properties of dynamic environments and other varying parameters (such as the distance between nodes) are all automatically calculated.

scenario_init_state The state initialization function that should be called at least once before calling scenario_deltaQ, so as to perform preliminary initializations of the internal state of the scenario. This function should also be called before scenario_deltaQ in all the situations when the scenario was modified (e.g., nodes or connections were added), so as to update the internal state of the scenario.

xml_scenario_parse The function that loads the initial conditions of a scenario from the XML QOMET scenario representation file. The loaded scenario will be available as the structure field "xml_scenario->scenario".

For more details about the usage of each function, please see the source code of the deltaQ library, in particular the main program file "deltaQ/deltaQ.c".

4.2 Link-level emulator configuration library: wireconf

In order to allow users to configure with ease a link-level network emulator, we introduced a QOMET library called wireconf. Using wireconf and its companion library timer for accurate time measurement (see Section 4.3), one can run emulation experiments in a precise and convenient manner. At this moment only ipfw3/dummynet [14] is supported as link-level emulator, but functionality can be extended to other emulators such as NetEm by writing wrappers for the corresponding API functions.

The most important functions of the wireconf library are given below together with a short description of their role:

add_rule_and_pipe Adds a dummynet rule that contains a configurable pipe to the ipfw3 firewall configuration.

configure_pipe Configures a pipe associated to a dummynet rule so as to introduce artificial bandwidth limitations, packet loss, and delay. This function can be called multiple times, as needed.

delete_pipe, delete_rule Deletes a dummynet pipe or rule, respectively. These functions can be called independently from each other if needed.

These functions should be called in the order in which they are presented in order to successfully carry out network emulation. See the source code of the standalone program wireconf available in "wireconf/wireconf.c" for a usage example.

4.3 Time measurement library: timer

The link-level emulator configuration library, wireconf, uses a companion library for accurate time measurement, called timer. At this moment the timer library is only available for Linux and FreeBSD.

The most important functions of the `timer` library are given below together with a short description of their role::

`timer_reset` Sets the relative time origin for the subsequent wait operations.

`timer_wait` Causes the calling program to wait for a specified time instant to occur. The time instant is expressed in microseconds, and is given with reference to the previously set relative time origin. This function can be called multiple times, as needed.

These functions should be called in the order in which they are presented in order to successfully carry out the execution of timed events. See the source code of the stand-alone program `wireconf` available in "wireconf/wireconf.c" for a usage example.

4.4 Other tools

In this section we introduce a few other QOMET components, that are not essential for network emulation purposes, but were provided for the user convenience will be introduced in this section.

4.4.1 Scenario generator

When working with large-scale scenarios, a useful feature is the ability to automatically generate the QOMET scenario that describes the experiment. For this purpose, we created an example C source file called "extras/generate_scenario.c", whose compilation produces the program named `generate_scenario`. Executing this program outputs to `stdout` a QOMET scenario file that can be saved to a file and provided as input to the `deltaQ` program.

Depending on the value of constants in the source code, one can generate, for example, a scenario in which 1000 pedestrians starting from areas at the periphery of a 2 x 2 km square start moving using the behavioral model towards destinations located in the center of the square following a cross like structure of roads. The file "generate_scenario.c" can be changed by users to suit their own needs. Bear in mind though that this file is provided just as an example, and development based on it is not supported in any way.

4.4.2 Test suite

The QOMET distribution includes a test suite that can be used to validate the operation of QOMET after making changes to the source code. The scenarios in the test suite also serve as examples for the various features of QOMET.

In order to run the test suite, the following command should be executed in the "test_suite" directory:

```
> ./run_test_suite.sh
```

As the test suite is executed, it reports for each test whether it succeeded or failed. The scenarios that have failed can then be analyzed to understand what caused execution to fail. Note that it is possible that more scenarios fail due to the same problem. It is also possible that some unforseen errors do not cause any scenarios to fail.

The test suite compares the QOMET text output with an output considered valid to determine whether a test succeeded or failed; hence, it is a high-level validation procedure. In the future we aim to completely integrate a unit testing framework that we created with the QOMET source code, so that a low-level checking of the source code is done. However, as this is an extensive process, it will be done gradually.

4.5 Software distribution

This user's guide is distributed as part of an archive containing all the source files for the QOMET set of tools for wireless network emulation, as well as additional items such a validation suite, examples, etc.

The software distribution is organized into the following directories:

deltaQ/ The directory containing the `deltaQ` wireless network communication emulation library source files and stand-alone program;

extras/ The directory containing files provided for user convenience, such as a scenario generator, a basic QOMET scenario, etc.;

ipfw3/ The source code of the `ipfw3/dummynet` link-level network emulation tools which was integrated with QOMET;

test_suite/ The validation framework for QOMET;

timer/ The timer library used in the real-time execution of QOMET;

wireconf/ The directory containing the `wireconf` wired-network emulator configuration library and stand-alone program.

Bibliography

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan (ed.), *XML 1.1 (Second Edition)*, W3C Recommendation, 16 August 2006.

[2] R. Beuran, *Introduction to Network Emulation*, Pan Stanford Publishing, to appear in 2012.

[3] R. Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, Y. Shinoda, *A Multipurpose Wireless Network Emulator: QOMET*, 22nd IEEE International Conference on Advanced Information Networking and Applications (AINA 2008) Workshops, FINA 2008 symposium, Okinawa, Japan, March 25–28, 2008, pp. 223–228.

[4] R. Beuran, L. T. Nguyen, K. T. Latt, J. Nakata, *Wireless LAN Emulation*, Research Report, IS-RR-2006-015, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan, October 2006.

[5] R. Beuran, L. T. Nguyen, K. T. Latt, J. Nakata, Y. Shinoda, *QOMET: A Versatile WLAN Emulator*, in Proc. of IEEE Intl. Conf. on Advanced Information Networking and Applications (AINA 2007), Niagara Falls, Ontario, Canada, May 21–23, 2007, pp. 348–353.

[6] R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K. Chinen, Y. Tan, Y. Shinoda, *QOMB: A Wireless Network Emulation Testbed*, IEEE Global Communications Conference (GLOBECOM 2009), Honolulu, Hawaii, USA, November 30–December 4, 2009.

[7] R. Beuran, L. T. Nguyen, Y. Shinoda, *QOMB Wireless Network Emulation Testbed: Evaluation and Case Study*, 5th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH 2010), in conjunction with MobiCom 2010, Chicago, Illinois, September 20–24, 2010.

[8] K. Chinen, *SpringOS Version 1.5 Manual*, March 2011, http://www.starbed.org/documents/spo15manual-110325a.pdf

[9] James Clark, The Expat XML Parser v2.0.1, http://expat.sourceforge.net.

[10] Hokuriku StarBED Technology Center, *Regarding the Use of Hokuriku StarBED Technology Center Facilities* , http://www2.nict.go.jp/collabo/collabo/q262/3105/riyou/usage-en.html

[11] A. Kamerman, L. Monteban, *WaveLAN[R]-II: A high-performance wireless LAN for the unlicensed band*, Bell Labs Technical Journal, vol. 2, no. 3, pp. 118133, August 1997.

[12] T. Miyachi, *SpringOS Tutorial for Scenario Execution*, 2010, http://www.starbed.org/documents/SpringOS-Scenario-Tutorial-eng-0.2.pdf

[13] T. Miyachi, K. Chinen, and Y. Shinoda, *StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software*, Intl. Conf. on Perf. Evaluation Methodologies and Tools (Valuetools 2006), ACM Press, Pisa, Italy, October 2006.

[14] L. Rizzo, *ipfw3/dummynet network emulator*, http://info.iet.unipi.it/ luigi/ip_dummynet.