QOMET User Manual

Răzvan Beuran

March 31, 2009

National Institute of Information and Communications Technology (NICT)

Hokuriku Research Center
1-12 Asahidai, Nomi, Ishikawa, 923-1211 Japan

http://www.nict.go.jp

Japan Advanced Institute of Science and Technology (JAIST)

Internet Research Center
1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan

http://www.jaist.ac.jp

QOMET User Manual

Table of Contents

1 Introduction

This is the user manual for QOMET, our set of tools for wireless network emulation. QOMET stands for "Quality Observation and Mobility Experiment Tools". This user manual refers to QOMET v1.5 that was released in March 2009.

Initially focusing on WLAN, 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 active tag or ZigBee, for realistic 3D virtual environments, mobility generation, etc.

The QOMET software is an implementation of the first stage of our approach to wireless network emulation [1-3]. This approach is layer oriented, as it computes first the physical layer effects the changes in the communication channel have on the received signal power. 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 MAC layer. Finally, the network layer effects are determined: packet loss, delay & jitter, and available bandwidth. Similar techniques are used for the other supported networks.

Through the use of the wireless network model that is summarized above, the real-world wireless network scenario is converted into a sequence of quality degradation (ΔQ) states of the network, termed *ΔQ description*. The ΔQ description accurately reproduces the wireless network behavior corresponding to the emulated scenario, and can be subsequently used to configure a wired-network emulator to recreate this behavior for network experimentation.

In order to make it possible to run 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, the main focus of the emulator remain the wireless networks.

1.1 Implementation overview

The QOMET software described in this user manual implements first of all the conversion of the real-world wireless network scenario into a sequence of quality degradation (ΔQ) states of the network. The software takes a scenario representation as input, and produces the ΔQ description as output. QOMET currently supports the following wireless communication technologies:

- IEEE 802.11a/b/g WLAN;
- Active RFID tag communication;
- IEEE 802.15.4 (base for the ZigBee communication protocol).

The scenario representation used by QOMET is written in XML format. An example scenario is provided in this document as an appendix. The text output of the program is a file that contains the most important computed parameters as columns, such as the ΔQ parameters: bandwidth, packet loss and delay & jitter. Other parameters are included as well, e.g., the time, the distance between nodes, the received power in dBm, etc. An equivalent output in binary format can also be generated.

The output of QOMET is intended to be used to drive a wired-network emulator, such as `dummynet` [4], to recreate the computed quality degradation in a real network environment. QOMET supports this functionality as well, by means of a specific library.

We have performed several experiments using QOMET, with network setups ranging between a few and over one hundred PCs, each emulating multiple real nodes [5-7]. Experimentation is mainly done on StarBED, the large-scale network experiment environment at the National Institute of Information and Communications Technology (NICT) Hokuriku Research Center in Ishikawa, Japan [8]. Recently we have started a related project called QOMB (QOMet on starBed), that aims at better integrating QOMET with StarBED so as to facilitate users' activity of making experiments.

Although our implementation is mainly intended for being used as a stand-alone software, the wireless network communication emulation core can also be used as a library, and linked to other programs. In fact, the main QOMET executable uses this library itself and integrates it with the scenario parsing and node motion generation code. More details about this aspect are available in Section 3.2.

1.2 Document structure

This user manual is structured as follows. In Chapter 2 we give the reference of the XML scenario representation. In Chapter 3 we present various issues related to the practical use of our implementation, including instructions on how to use the QOMET output to drive a wired-network emulator and conduct experiments; this chapter also provides an overview of the use of the wireless network communication emulation library and other libraries included with QOMET. Chapter 4 summarizes the conclusions related to our current implementation of the wireless network emulator, and is followed by a section of references.

Then, in Appendix A we give an example input scenario file with brief explanations. In Appendix B we show graphical representations of QOMET's output for the input scenario presented in Appendix A. In Appendix C we provide an example `bash` shell script that can be used to run a simple experiment on a FreeBSD platform by directly configuring `dummynet`. In Appendix D a similar shell script is provided, this time intended to run the same experiment in a more user-friendly manner by using the included `wireconf` library (see Section 3.3).

1.3 Acknowledgments

We would like to acknowledge the contribution to the development and testing of the software, and the improvement of the current user manual of: Junya Nakata, Lan Tien Nguyen, Takashi Okada, and Khin Thida Latt.

1.4 Contact information

In case you want to make any suggestions, either about this user manual, or about QOMET itself, please send your comments or bug reports to `info@starbed.org`.

2 Scenario representation

This chapter describes the structure of the scenario representation file used as QOMET input. The file is written in XML format [9]. For parsing the scenario file we used the eXpat XML parser [10]. 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. A third-level element can be used to represent object coordinates.

2.1 `qomet_scenario` element

The top-level element in the 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 scenario.

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

- `start_time`
  - Role: specify the starting time of scenario execution in seconds;
  - Data type: double;
  - Value: any value (default = 0 s);
- `duration`
  - Role: specify the duration of scenario execution in seconds;
  - Data type: positive double;
  - Value: any value (default = 0 s);
- `step`
  - Role: specify the time intervals at which ΔQ calculation of scenario is performed, in seconds[1];
  - Data type: positive double;
  - Value: any value (default = 0.5 s);
- `motion_step_divider`
  - Role: divider that should be applied to ΔQ calculation step; used for motion computations, so that motion time step can be sufficiently small to produce correct results, independently on the ΔQ calculation step;
  - Data type: positive double;
  - Value: any value (default = 1.0);
- `coordinate_system`
  - Role: specify which is the coordinate system used in the scenario;
  - Data type: string;
  - Value: {`cartesian` | `lat_lon_alt`} (default = `cartesian`);

1 This step is also used by QOMET as time granularity for the generated output.

- `jpgis_file_name`
  - Role: specify the name of the JPGIS-format[2] file from which object coordinates are to be loaded (UTF-8 encoding is required)[3];
  - Data type: string;
  - Value: any value (no default value).

Below is an example of a scenario starting at time 5s, with a duration of 60 s, that will be analyzed with a step of 0.5 s. Element definitions are not specified in this example; please consult the following sections for details on defining elements.

```
<qomet_scenario start_time="5" duration="60" step="0.5">
    ... element definitions ...
</qomet_scenario>
```

2.2 node element

The `node` element defines the properties of emulated devices, that in QOMET are called *nodes*.

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

- `name`
  - Role: specify the name of the node;
  - Data type: string;
  - Value: any value (no default value);
- `type`
  - Role: specify the type of the node;
  - Data type: string;
  - Value: {`regular` | `access_point`} (default = `regular`);
- `id`
  - Role: specify an id for the node;
  - Data type: positive integer;
  - Value: any value (no default value);
- `ssid`
  - Role: specify the ssid for the network to which the node should connect;
  - Data type: string;
  - Value: any value (default = `UNKNOWN`);
- `connection`
  - Role: specify the type of connection the node uses;

2 JPGIS is a GIS (Geographic Information System) data representation format widely used in Japan.
3 See also Section 2.3 for details on how to specify objects.

- Data type: string;
- Value: {`ad_hoc` | `infrastructure` | `any`} (default = `any`);

- `adapter`
  - Role: specify the adapter model which is used for the node; should be correlated to the emulated wireless network technology used for connections from the node;
  - Data type: string;
  - Value: {`orinoco|dei80211mr|cisco_340|cisco_abg|jennic|s_node`} (default = `cisco_abg`);
- `x, y, z`
  - Role: specify the initial (*x*, *y*, *z*) coordinates of the node; the measurement unit depends on the coordinate system used;
  - Data type: double;
  - Value: any value (default = 0, 0, 0, respectively);
- `Pt`
  - Role: specify the power transmitted by the node in dBm;
  - Data type: double;
  - Value: any value (default = 20 dBm);
- `antenna_gain`
  - Role: specify the antenna gain of the node transceiver in dBi;
  - Data type: positive double;
  - Value: any value (default = 0.0 dBi);
- `azimuth_orientation`
  - Role: specify antenna orientation in the azimuth (horizontal) plane, in degrees;
  - Data type: double in the range [0°, 360°);
  - Value: any value (default = 0°);
- `azimuth_beamwidth`
  - Role: specify antenna beamwidth in the azimuth (horizontal) plane, in degrees[4];
  - Data type: double in the range [0°, 360°];
  - Value: any value (default = 360°, signifying an omni-directional antenna in azimuth plane);
- `elevation_orientation`
  - Role: specify antenna orientation in elevation (vertical) plane in degrees;
  - Data type: double in the range [0°, 360°);

4 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 = 0°);
- `elevation_beamwidth`
    - Role: specify 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);
- `internal_delay`
    - Role: specify the internal fixed delay in milliseconds that is specific to the node operation;
    - Data type: positive double;
    - Value: any value (default = 0 ms).

Below is an example of a node named "node1" with id = 1, defined as a regular node supporting ad hoc connections. The initial position of the node is (5, 10, 15), and its transmitted power equals 20 dBm.

```
<node name="node1" type="regular" id="1" connection="ad_hoc"
       x="5" y="10" z="15" Pt="20"/>
```

2.3 `object` element

The `object` element represents structures, such as roads, buildings, or other obstacles (for example, hedges). Such objects may interfere with the wireless communication between nodes, therefore must be taken into account. Objects such as roads are important to restrict node motion within the designated road boundaries[5]. Motion also can take into account buildings, which are avoided by pedestrians.

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

- `name`
    - Role: specify the name of the object;
    - Data type: string;
    - Value: any value (default = `UNKNOWN`);
- `type`
    - Role: specify the type of object; this 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`
    - Role: specify the name of the environment associated to this obstacle; the respective environment must not be dynamic (see Section 2.5);

5 The height of the object becomes in this case important. Road objects should have zero height, thus not interfering with communication, but only restricting motion.

    - Data type: string;
    - Value: any value (default = `UNKNOWN`);
- `x1, y1, x2, y2`
    - Role: 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`
    - Role: specify the height of the 3D object in meters, if needed;
    - Data type: positive double;
    - Value: any value (default = 0 m);
- `load_from_jpgis_file`
    - Role: specify 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`
    - Role: specify that the object should be made into a polygon by connecting its last and first vertices[6];
    - Data type: string;
    - Value: {`true` | `false`} (default = `false`).

Below is an example of an object named "building" associated to an environment called "building_env" (not defined here). The object 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"/>
```

2.4 `coordinate` element

The `coordinate` element represents point coordinates. This element is currently only 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. This attribute is usually zero for objects such as roads, and non-zero for buildings or other obstacles that need to be treated as 3D for signal propagation calculations.

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

- `name`
    - Role: specify the name of the coordinate (optional);

6 Alternatively, the last vertex coordinate can be equal to that of the first vertex. This mimics the convention used to represent polygons in JPGIS-format topology descriptions.

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

The value of the coordinate is given by pairs of double numbers enclosed between the start and end tags of the `coordinate` element. These numbers can represent either (*x*, *y*) Cartesian coordinates or (*latitude*, *longitude*) coordinates, depending on the value of the `coordinate_system` attribute of the `qomet_scenario` element (see Section 2.1). In order to associate coordinates to an object, the `coordinate` element definitions must be included between the start and end tags of the `object` element.

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 ignored 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>
```

2.5 `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 element `connection` must be used (see Section 2.7).

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

- `name`
  - Role: specify the name of the environment;
  - Data type: string;
  - Value: any value (no default value);
- `type`[7]
  - Role: specify the type of the environment;
  - Data type: string;
  - Value: {`indoor` | `outdoor`} (default = `UNKNOWN`);
- `is_dynamic`
  - Role: specify whether the environment is dynamic (its properties will be computed at each step depending on scenario topology) or static (its properties will never change);
  - Data type: string;

7 This attribute is not currently used.

    - Value: {`true` | `false`} (default = `false`);
- `alpha, sigma, W`
    - Role: specify the parameters α (attenuation constant), σ (shadowing parameter) and *W* (wall attenuation) of the log-distance path loss environment model; α is unitless, and σ & *W* are both expressed in dB – see [1] for details;
    - Data type: positive double;
    - Value: any value (default = 0, 0 dB, 0 dB, respectively);
- `noise_power`
    - Role: specify the power of the noise in dBm, as it would be sensed by a receiver located in this environment;
    - Data type: double;
    - Value: any value (default = −100 dBm).

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

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

2.6 `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.

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

- `node_name`
    - Role: specify the name of the node to which this motion object should be applied;
    - Data type: string;
    - Value: any value (no default value);
- `type`
    - Role: specify the type of the motion;
    - Data type: string;
    - Value: {`linear` | `circular` | `rotation` | `random_walk` | `behavioral`} (default = `linear`);
- `speed_x, speed_y, speed_z`
    - Role: specify the speed for linear motion on the 0*x*, 0*y*, and 0*z* axes, in meters/second;
    - Data type: double;

    - Value: any value (no default value);
- `center_x, center_y`
    - Role: specify the center for circular motion horizontal plane with respect to which the motion is performed; the measurement unit depends on the coordinate system used;
    - Data type: positive double;
    - Value: any value (no default value);
- `velocity`
    - Role: specify the tangential velocity for circular motion in the horizontal plane;
    - Data type: double;
    - Value: any value[8] (default = 0 m/s);
- `min_speed, max_speed`
    - Role: specify the minimum and maximum speeds for random-walk motion;
    - Data type: positive double;
    - Value: any values, with `max_speed` ≥ `min_speed` (default = 0 m/s, 1 m/s, respectively);
- `walk_time`
    - Role: specify the walking time for random-walk motion, in seconds;
    - Data type: positive double;
    - Value: any value (default = 5 s);
- `destination_x, destination_y, destination_z`
    - Role: specify the motion destination; used only with linear and behavioral models; the measurement unit depends on the coordinate system used;
    - Data type: double;
    - Value: any value (no default value);
- `rotation_angle_horizontal, rotation_angle_vertical`
    - Role: specify the rotation angles in horizontal and vertical plane, respectively, in degrees;
    - Data type: double in the range [0°, 360°);
    - Value: any value (default = 0°);
- `start_time, stop_time`
    - Role: specify the motion start and stop time in seconds;
    - Data type: positive double;
    - Value: any value (default = 0 s, 0 s, respectively).

8 Positive velocity values signify anti-clockwise rotation, whereas negative values indicate clockwise rotation.

Below is an example of a motion to be applied to the node named "node2". The motion is a linear displacement with speed (0.5, 0, 0) in the (*x*, *y*, *z*) space. The start time is 0 s, and the stop time is 30 s.

```
<motion node_name="node2" type="linear"
    speed_x="0.5" speed_y="0" speed_z="0"
    start_time="0" stop_time="30"/>
```

2.7 `connection` element

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

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

- `from_node`
  - Role: specify the name of the transmitting node;
  - Data type: string;
  - Value: any value (no default value);
- `to_node`
  - Role: specify 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 value "`auto_connect`" has a special meaning, as it instructs QOMET to connect the `to_node` to all the nodes that were defined *before* this connection[9];
- `through_environment`
  - Role: specify the name of the environment through which the nodes communicate;
  - Data type: string;
  - Value: any value (no default value);
- `standard`
  - Role: specify the wireless network standard 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`);
- `channel`
  - Role: specify the channel on which the transceiver operates (if applicable);
  - Data type: positive integer (including zero);

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

- Value: any value between 0 and 200 (default = 1 for 802.11b/g and 36 for 802.11a, respectively);

- `RTS_CTS_threshold`
  - Role: specify the packet size above which the IEEE 802.11 WLAN RTS/CTS mechanism is enabled;
  - Data type: positive integer;
  - Value: any value between 0 and 2347 (default = 2347, which completely disables the RTS/CTS mechanism for WLAN);
- `packet_size`
  - Role: specify the average packet size in bytes (at network layer) transmitted through this connection;
  - Data type: positive integer;
  - Value: any value (default = 1024 bytes);
- `bandwidth`
  - Role: specify the average bandwidth of this connection in bits per second;
  - Data type: positive double;
  - Value: less or equal 1,000,000,000 bps (default = *automatically computed*);
- `loss_rate`
  - Role: specify the average packet loss rate of this connection as a probability;
  - Data type: positive double;
  - Value: less or equal 1 (default = *automatically computed*);
- `delay, jitter`
  - Role: specify the average delay and jitter of this connection in milliseconds;
  - Data type: positive double;
  - Value: any value (default = *automatically computed*);
- `consider_interference`
  - Role: specify whether interference between nodes is to be taken into account when computing communication conditions;
  - Data type: string;
  - Value: {`true` | `false`} (default = `true`).

Below is an example of a connection from node "node1" to the nodes "node2" and "node3" through the environment named "env1"; the wireless network standard used is 802.11g WLAN, and the average packet size is 200 bytes:

```
<connection from_node="node1" to_node="node2 node3"
   through_environment="env1" standard="802.11g"
   packet_size="200"/>
```

2.8 Remarks

2.8.1 Scenario definition issues

This subsection contains several clarifications regarding the information provided in previous sections.

1. Node connection type can be either `ad_hoc` (connect just with `regular` nodes), `infrastructure` (connect only with access points) or `any` (connect with any other node);
2. The `random_walk` motion uses the parameters `min_speed` and `max_speed` for the range of the randomly generated speed. The moving direction is also randomly generated. The parameter `walk_time` specifies how long the node moves before a change of speed and moving direction;
3. If a parameter such as bandwidth, loss rate or delay are specified for a connection, then the corresponding constant values will be used for the entire duration of the experiment, and will not be computed anymore. This is particularly useful for defining the conditions of Ethernet connections;
4. Typical settings for 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;
5. The strings used to define element attribute names, names of nodes, topology objects and motion elements are all *case sensitive* (this is a change compared to previous versions).

2.8.2 Restrictions

The restrictions mentioned in this subsection apply to the current version of QOMET, but may change in future versions.

1. 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, the "`auto_connect`" option of connections only generates connections to the nodes that were defined *before* that connection definition;
2. At the moment of writing this document, there are only some types of checks performed at parse time (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/id were not previously defined). Other incongruities, such as defining a motion for an inexistent node, etc., will however be signaled as run-time warnings;
3. Due to the fact that the network parameters are computed in advance, QOMET can have no knowledge about the size of the packets that will actually flow through the emulated wireless network. However, the computation of all the ΔQ parameters, bandwidth, packet loss, and delay & jitter, depends on packet size. At the moment it is possible to provide the average packet size for the connection, and this should be changed according to each particular user scenario. For example, packet size can be set to 200 bytes for VoIP traffic. The longer term solution we envisage is to provide to `wireconf` packet-size-independent values

for bandwidth, packet loss and delay & jitter, and then `wireconf` will compute on-the-fly the corresponding parameters for each incoming packet, depending on its size, traffic conditions and congestion, etc.;

4. At the moment the number of elements of each type in a scenario (nodes, objects, coordinates, environments, motions, and connections) is limited to fixed maximum values. These values are: 250 for nodes, objects and motions, and 62500 for environments and connections. Each object can have up to 35 vertex coordinates. The complexity of dynamic environment computation is limited to a maximum of 100 segments. These parameters can be changed in the source code if required; note however that using larger values increases the memory requirements of QOMET, and sufficient RAM must exist on the PC used to run the software;
5. For all internal processing operations, node names are used to identify nodes; however, in the output file only node ids are included. This is to facilitate post-processing and plotting by using shell scripts and Matlab.

3 Utilization

This chapter provides more details on how to employ the QOMET software in practice. We describe both the usage of the stand-alone version (Section 3.1), as well as how to use the deltaQ wireless network communication emulation library for computing ΔQ parameters (Section 3.2). We also present the `wireconf` wired-network emulator configuration library intended for facilitating experiments (Section 3.3). Other library descriptions follow in Sections 3.4 (`timer`) and 3.5 (`chanel`).

3.1 Stand-alone usage

Although QOMET represents a set of distinct wireless network emulation tools, for convenience we created a program called `qomet` that can be used as a stand-alone software in order to access most of the computation features of QOMET (ΔQ description, motion, etc.). Details of this usage are given next.

3.1.1 Execution

QOMET was implemented in C language. It makes use of the eXpat XML parser library [5] that needs to be compiled and installed in advance on the platform on which QOMET is to be run. Compiling QOMET should be done using the provided Makefile. So far QOMET has been compiled mainly on UNIX systems (Linux, FreeBSD). Other UNIX operating systems such as Sun Solaris, or Windows with cygwin or the Visual C compiler are only partially supported.

Once QOMET is compiled and ready to use, the user must execute the following command line:

```
./qomet <scenario_file.xml>
```

where `<scenario_file.xml>` is the name of the file in which the XML scenario representation was stored. An example scenario is provided in Appendix A.

Command-line options can be provided to QOMET to control execution. The options are summarized in Figure 1. By default, QOMET will do deltaQ computation, and generate both text and binary deltaQ output, but not motion data.

3.1.2 Output files

As a result of executing the command mentioned in the previous section, the scenario is parsed and processed. By default, the output is stored in a file having the same name with the scenario and the additional extension ".out" (e.g., "scenario_file.xml.out").

The graphical representation of QOMET output is provided in Appendix B. At this moment the following node properties are written for each defined connection at each moment of time:

```
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 additional extension ".bin" (e.g., "scenario_file.xml.bin"). The advantage of the binary output is that the file size is

significantly reduced, since only part of the information existing in the text output is written. In addition, if communication conditions do not change, no binary records are written, thus further reducing file size. The binary format is recommended for large-scale experiments, and is fully supported by the `chanel` library (see Section 3.5).

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)
-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 1: Summary of `qomet` command-line options.

3.1.3 Effective emulation

Starting from the output file discussed above, a wired-network emulator, such as `dummynet`, can be used to effectively reproduce the wireless network effects in a wired network. We will illustrate this process here with a very simple setup for WLAN emulation, as depicted in Figure 2.

In the setup in Figure 2 the end PCs play the role of the mobile nodes in the scenario, and they can use any operating system. The emulated WLAN is represented by a third PC, on which runs a wired-network emulator such as `dummynet`. This leads to some restrictions on the operating system, which needs to be FreeBSD if using `dummynet`. For the purpose of this document the QoS Meter block that measures the end-to-end QoS parameters, and the UPQ meter that measures the User-Perceived Quality for the tested application are irrelevant, but they should be used in practice when performing tests regarding application performance assessment.

The conceptual setup in Figure 2 can been simplified for illustration purposes. The scripts provided in Appendix C can be used to effectively run a test on a simplified setup in which only one PC is required. This PC (running FreeBSD) is the WLAN emulator and in the same time sends and receives traffic using the loopback interface. However, for experiments that generate a significant amount of traffic, we recommend using separate PCs for the end nodes and the emulator so that running applications don't interfere with emulation timing. Another possibility is to use a distributed emulation solution, in which the emulator on each PC is in charge of reproducing only the communication conditions

between the nodes associated to that PC and all the other nodes. We used this solution for experiments related to robot emulation, for example [6].

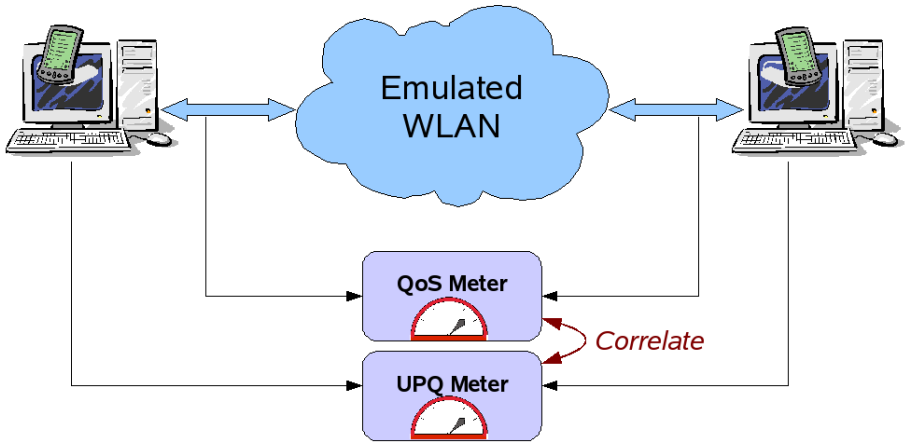

Figure 2: Experimental setup for tests using WLAN emulation.

3.2 Network emulation library: `deltaQ`

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

The main function of the library is:

```
int scenario_deltaQ(scenario_class *scenario);
```

This function computes the ΔQ parameters for all the connections in the scenario. The dynamic environments and other varying properties (such as the distance between nodes) are automatically updated.

The function:

```
int scenario_initialize_state(scenario_class *scenario);
```

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

Another useful function is:

```
int xml_scenario_parse(xml_scenario_class *xml_scenario);
```

which can be used to easily load the initial conditions of a scenario from an XML 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 itself.

3.3 Wired-network emulator configuration library: `wireconf`

In order to allow users to configure with ease at regular intervals a wired-network emulator, we introduced a QOMET library called `wireconf`. Using `wireconf` (and its companion library for accurate time measurement, `timer`) one can run QOMET

emulation experiments in a more precise and convenient manner than by using shell scripts. At the moment only `dummynet` on FreeBSD is supported as wired-network emulator.

The most important functions of the `wireconf` library are:

```
int add_rule(int s, uint16_t rulenum, int pipe_nr,
      char *src, char *dst, int direction);
```

which adds a `dummynet` rule to the `ipfw` firewall configuration in FreeBSD,

```
int configure_pipe(int s, int pipe_nr, int bandwidth,
      int delay, int lossrate);
```

that configures a pipe associated to a `dummynet` rule, and

```
int delete_rule(uint s, u_int32_t rule_number);
```

which deletes a `dummynet` rule.

For running our own we created the program `do_wireconf` that effectively uses the `wireconf` library when configuring `dummynet`. The program supports two usages, one for making individual settings, and one for making grouped settings in conjunction with OLSR routing support.

The main input argument of `do_wireconf` is the QOMET output for a scenario file, denoted by `<qomet_output_file>`. The other configuration options of `do_wireconf` are summarized in Figure 3.

(1) *Configure a specified pair* <from_node_id> (IP address <from_node_addr>)
<to_node_id> (IP address <to_node_addr>) using the dummynet rule
<rule_number> and pipe <pipe_number>, optionally in direction 'in' or 'out'.
Usage: ./do_wireconf -q <qomet_output_file>
-f <from_node_id> -F <from_node_addr>
-t <to_node_id> -T <to_node_addr>
-r <rule_number> -p <pipe_number>
[-d {in|out}]

(2) *Configure all connections from the current node* <current_id> given the
IP settings in <settings_file> and the OPTIONAL broadcast address
<broadcast_address>; Interval between configurations is <time_period>.
The settings file contains on each line pairs of ids and IP addresses.
Usage: ./do_wireconf -q <qomet_output_file>
-i <current_id> -s <settings_file>
-m <time_period> [-b <broadcast_address>]

NOTE: If option '-s' is used, usage (2) is inferred, otherwise usage (1) is assumed.

Figure 3: Summary of `do_wireconf` usage manners.

3.4 Time measurement library: `timer`

The wired-network emulator configuration library, `wireconf`, uses a companion library for accurate time measurement, `timer`. At the moment the `timer` library is only available for FreeBSD.

The most important functions of the `timer` library are:

```
void timer_reset(timer_handle *handle);
```

that resets the timer to zero, thus setting a relative time origin for subsequent wait operations, and:

```
int timer_wait(timer_handle *handle, uint64_t time_in_us);
```

which waits for a time to occur (expressed in microseconds). The time is given with reference to the previously set time origin.

3.5 Communication channel emulation library: `chanel`

In some cases non-IP systems applications are to be tested. This is the case, for example, of active tag or ZigBee systems. In these circumstances `dummynet` cannot be used anymore, since it is a network emulator dedicated to IP traffic. For such circumstance we wrote a communication channel emulation library, called `chanel` that integrates with RUNE [11], the experiment support software for StarBED. This library was used for experiments related to active tag emulation [7].

3.6 Scenario generator

When working with large-scale scenarios, an 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 "generate_scenario.c". By compiling this file, the program "generate_scenario" is created. Running "generate_scenario" outputs to *stdout* a QOMET scenario file. Depending on some constants in the source code, one can generated, 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. At the moment this file is provided just as an example, and development based on it is not supported in any way.

3.7 Software distribution

The present document is the user manual for our set of tools for wireless network emulation, QOMET. This implementation takes an input scenario representation given as an XML file, and produces an output file that contains the most important properties of each scenario node at each moment of time.

This user manual is distributed as part of an archive containing all the needed files in order to run a provided example scenario, assuming that a PC running FreeBSD with `dummynet` support is available.

The archive contains the following files and directories, listed in alphabetical order:

- chanel/ – the directory containing the source files of the communication channel

emulation library for non-IP applications;

- deltaQ/ – the directory containing the `deltaQ` wireless network communication emulation library source files;
- dummynet_load.sh, dummynet_unload.sh – example scripts of how to load and unload `dummynet`;
- generate_scenario.c – source file for the example scenario generator;
- HISTORY – review of changes between versions;
- LICENSE – QOMET's copyright statement
- Makefile – the main QOMET make file;
- nap.pl – Perl script used to enforce millisecond-level delays;
- post-processing.sh – `bash` script that creates the intermediary files used to effectively run the experiment;
- qomet-1.5_manual.pdf – this user manual
- qomet.c – the main QOMET source file;
- README – step by step instructions on how to perform an experiment;
- routing/ – the directory containing the routing-related source files;
- run_experiment_api.sh – `bash` script used to run a QOMET experiment that makes use of the `wireconf` library (Appendix D of this document);
- run_experiment.sh – `bash` script used to run a simple QOMET experiment (Appendix C of this document);
- scenario_file.xml – example scenario file (also shown in Appendix A of this document);
- scenario_file.xml.bin – example output file in binary format;
- scenario_file.xml.out.bandwidth, scenario_file.xml.out.loss, scenario_file.xml.out.delay – example files obtained after post-processing;
- scenario_file.xml.nam – example motion output file in NAM[10] format;
- scenario_file.xml.out – example output file in text format;
- wireconf/ – the directory containing the `wireconf` wired-network emulator configuration and `timer` libraries source files.

10 NAM is the network animator (i.e., visualizer) companion of NS-2.

References

[1] R. Beuran, L. T. Nguyen, K. T. Latt, J. Nakata, “Wireless LAN Emulation”, research report, *Japan Advanced Institute of Science and Technology (JAIST)*, Ishikawa, Japan, October 2006.

[2] R. Beuran, L. T. Nguyen, K. T. Latt, J. Nakata, Y. Shinoda, "QOMET: A Versatile WLAN Emulator", *21st IEEE International Conference on Advanced Information Networking and Applications (AINA-07)*, Niagara Falls, Ontario, Canada, May 21-23, 2007, pp. 348-353.

[3] R. Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, Y. Shinoda, "A Multi-purpose Wireless Network Emulator: QOMET", *22nd IEEE International Conference on Advanced Information Networking and Applications (AINA2008) Workshops, FINA2008 symposium*, Okinawa, Japan, March 25-28, 2008, pp. 223-228.

[4] Dummynet FreeBSD network emulator, http://info.iet.unipi.it/~luigi/ip_dummynet.

[5] R. Beuran, K. Chinen, K. T. Latt, T. Miyachi, J. Nakata, L. T. Nguyen, Y. Shinoda, Y. Tan, "Application Performance Assessment on Wireless Ad Hoc Networks", *Asian Internet Engineering Conference (AINTEC) 2006*, Springer-Verlag LNCS 4311, Bangkok, Thailand, November 28-30, 2006, pp. 128-138.

[6] R. Beuran, J. Nakata, T. Okada, Y. Tan, Y. Shinoda, "Real-time emulation of networked robot systems", 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008), Marseille, France, March 3-7, 2008.

[7] R. Beuran, J. Nakata, Y. Suzuki, T. Kawakami, K. Chinen, Y. Tan, Y. Shinoda, "Active Tag Emulation for Pedestrian Localization Applications", 5th International Conference on Networked Sensing Systems (INSS08), Kanazawa, Ishikawa, Japan, June 17-19, 2008, pp. 55-58.

[8] T. Miyachi, K. Chinen, Y. Shinoda, “StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software”, VALUETOOLS’06, Pisa, Italy, October 11-13, 2006.

[9] 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.

[10] The Expat XML Parser v2.0.0, http://expat.sourceforge.net.

[11] J. Nakata, T. Miyachi, R. Beuran, K. Chinen, S. Uda, K. Masui, Y. Tan, Y. Shinoda, “StarBED2: Large-scale, Realistic and Real-time Testbed for Ubiquitous Networks”, TridentCom 2007, Orlando, Florida, U.S.A., May 21-23, 2007.

Appendix A

This appendix contains a full scenario definition, which is to be found in the archive under the name “scenario_file.xml”. The overview of the scenario is the following:

- The scenario has a duration of 60 s and is evaluated with a step of 0.5 s;
- There are two participating nodes, “node_ap”, an access point, and “node_r”, a regular WLAN node;
- Communication media is modeled by one static environment, “env”;
- From its initial position, node “node_r” moves to the right on the 0*x* axis for 30 s with speed (0.5, 0, 0), then to the left for another 30 s with speed (−0.5, 0, 0);
- Node “node_ap” is connected to “node_r” through the environment “env” using 802.11b WLAN standard.

```
<qomet_scenario duration="60" step="0.5">

  <node name="node_ap" type="access_point" id="1"
    x="0" y="0" z="0" Pt="20" internal_delay="1"/>
  <node name="node_r" type="regular" id="2"
    connection="infrastructure" adapter="orinoco"
    x="0" y="10" z="0" Pt="20" internal_delay="1"/>

  <environment name="env" alpha="5.6" sigma="0" W="0"
    noise_power="-100"/>

  <motion node_name="node_r" speed_x="0.5" speed_y="0"
    speed_z="0" start_time="0" stop_time="30"/>
  <motion node_name="node_r" speed_x="-0.5" speed_y="0"
    speed_z="0" start_time="30" stop_time="60"/>

  <connection from_node="node_ap" to_node="node_r"
    through_environment="env" standard="802.11b"
    packet_size="1024"/>

</qomet_scenario>
```

Appendix B

The output file in text format for the example scenario presented in Appendix A, "scenario_file.xml.out", is also included in the distribution. The output is also written in binary format in the file "scenario_file.xml.bin". The following is the header of the file "scenario_file.xml.out":

```
%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
```

Next we give the graphical representation of the output data. Figures 4, 5, and 6 show the dependency between the following metrics and time: distance between nodes and received power, bandwidth and packet loss, delay and jitter, respectively.

Note in Figures 5 and 6 the level of emulation detail that can be achieved using QOMET. We have shown in various papers how the quality variation associated to rate changes impacts on application performance, for example in the case of VoIP.

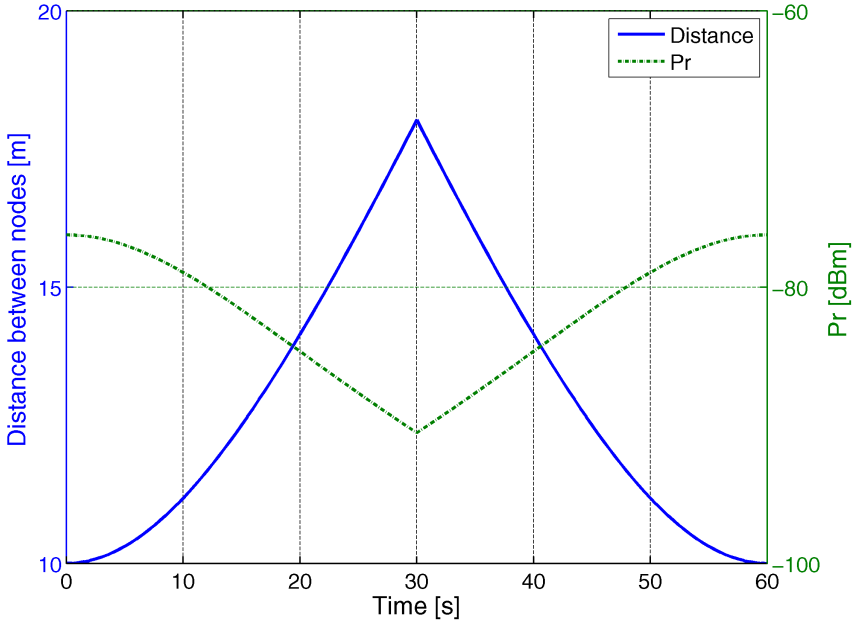

Figure 4: Distance between nodes and received power versus time.

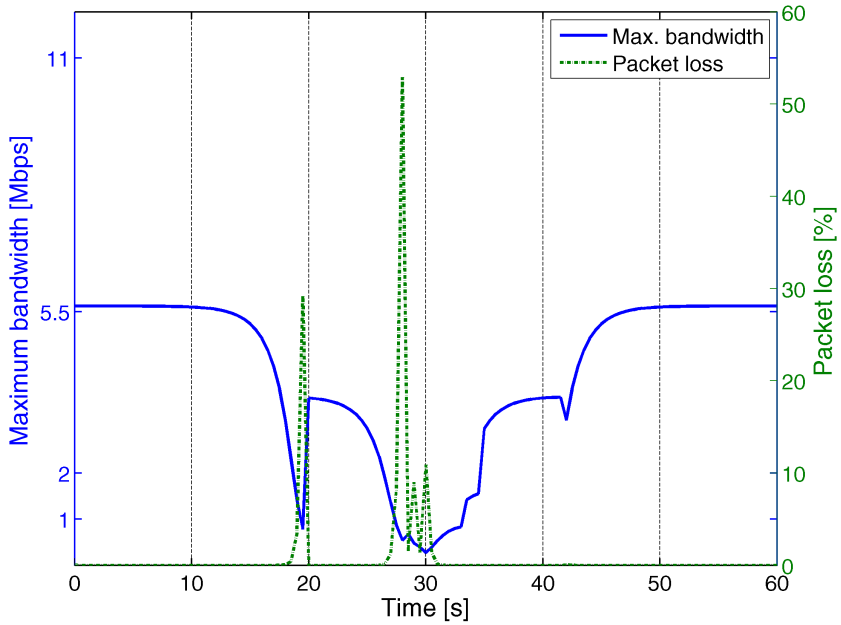

Figure 5: Maximum bandwidth and packet loss versus time.

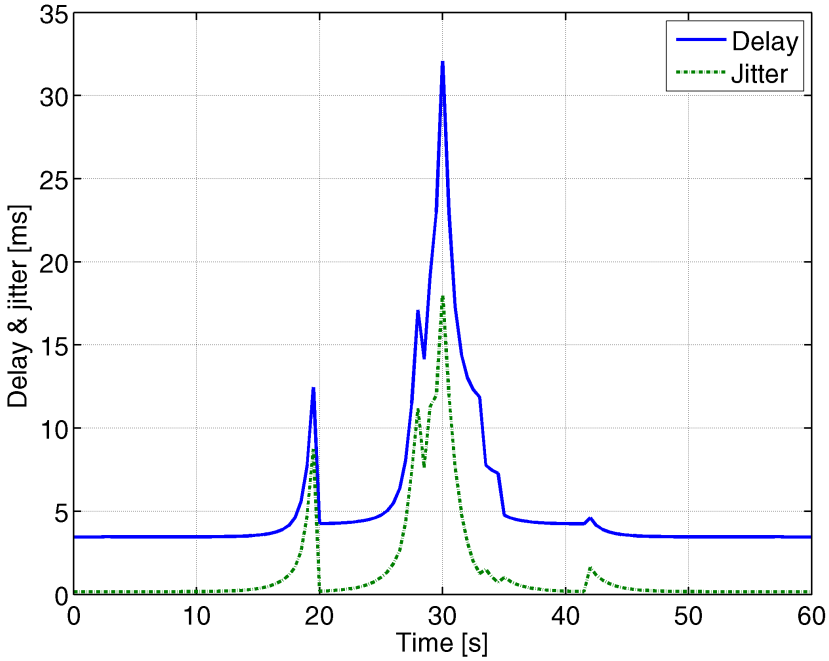

Figure 6: Delay and jitter versus time.

Appendix C

This appendix provides the script "run_experiment.sh" that can be used to effectively run an emulation using the output of QOMET on a single PC running FreeBSD. We assume that all the required software is installed and the script is run from the current directory. See the "README" file in the archive for more practical details.

```
# Run QOMET-based WLAN emulation experiment

# Define variables
scenario_file=$1

bandwidth_values_file=${scenario_file}.bandwidth
loss_values_file=${scenario_file}.loss
delay_values_file=${scenario_file}.delay

test_duration=60
nap_duration=460
#number_iterations=$(($test_duration*2))


########################################
# Prepare the system by loading dummynet
########################################

echo
echo -- Loading dummynet and doing basic configuration...
sudo ./dummynet_load.sh
sudo ipfw add 100 pipe 100 ip from 127.0.0.1 to 127.0.0.1 out

####################
# Run the experiment
####################

echo
echo -- Running experiment...
# Load loss and delay values from files
echo -n Loading $bandwidth_values_file, $loss_values_file, and
$delay_values_file ...
declare -a bandwidth_array
bandwidth_array=( `cat "$bandwidth_values_file"`)
declare -a loss_array
loss_array=( `cat "$loss_values_file"`)
declare -a delay_array
delay_array=( `cat "$delay_values_file"`)
element_count=${#bandwidth_array[*]}
echo " done (element count = $element_count)"

# Start server application
echo
echo "* Starting server..."
/usr/local/netperf/netserver

# Wait for server to start
sleep 3

index=0
bandwidth=${bandwidth_array[0]}
loss=${loss_array[0]}
delay=${delay_array[0]}

# Prepare the emulator for the first time period
```

```
echo
echo "* Dummynet configuration #$index: bandwidth=${bandwidth}bit/s loss=$loss
delay=${delay}ms"
sudo ipfw pipe 100 config bw ${bandwidth}bit/s delay ${delay}ms plr $loss

# Start client application
echo "* Starting client..."
/usr/local/netperf/netperf -l $test_duration &

echo ----------------------------
date
echo ----------------------------
for((index=1; index < $element_count; index++)); do
#for((index=1; index < $number_iterations; index++)); do
    # Sleep for the previous configuration
    perl nap.pl $nap_duration

    # Prepare the emulator for the next time period
    bandwidth=${bandwidth_array[$index]}
    loss=${loss_array[$index]}
    delay=${delay_array[$index]}
    echo "* Dummynet configuration #$index: bandwidth=${bandwidth}bit/s loss=
$loss delay=${delay}ms"
    sudo ipfw pipe 100 config bw ${bandwidth}bit/s delay ${delay}ms plr $loss
done

# Do a last sleep for the last configuration
perl nap.pl $nap_duration

echo ----------------------------
date
echo ----------------------------

################
# Final actions
################
echo
echo -- Ending experiment...

# Kill processes
#echo Killing client...
#killall netperf
sleep 3
echo Killing server...
killall netserver

# Process output files
#echo Processing output files...

# Clean the system
echo Unloading dummynet...
sudo ipfw delete 100
sudo ./dummynet_unload.sh
```

Appendix D

This appendix provides the script "run_experiment_api.sh" that can be used to effectively run an emulation using the output of QOMET on a single PC running FreeBSD. We assume that all the required software is installed and the script is run from the current directory. See the "README" file in the archive for more practical details.

```
# Run QOMET-based WLAN emulation experiment

# NOTE: This script uses an API-based program to configure dummynet
#       and to have precise timing control for step execution
#       These functions are provided by the "wireconf" library
#       and the example program "do_wireconf"

if [ $# -lt 1 ]
then
    echo "ERROR: Not enough arguments were provided!"
    echo "Command function: Run QOMET-based WLAN emulation experiment"
    echo "Usage: $(basename $0) <qomet_output_file>"
    exit 1
fi

# Define variables
test_name=$1
output_name=${test_name}
from_node=1
to_node=2
client_address=127.0.0.1
server_address=127.0.0.1
rule_number=100
pipe_number=200
packet_size=1200
test_duration=60
offered_load=10M

########################################
# Start tcpdump
########################################

#echo
#echo -- Starting tcpdump...
#sudo tcpdump -n -i lo0 -w ${output_name}_tcpdump.out &

########################################
# Configure alias interfaces and routes
########################################

#sudo ifconfig lo0 inet 127.0.0.2 netmask 255.0.0.0 alias
#sudo route add -host 127.0.0.2 127.0.0.1 0
#sudo ifconfig lo0 inet 127.0.0.2 -alias

########################################
# Prepare the system by loading dummynet
########################################

#echo
#echo -- Loading dummynet...
# the command below is not needed if dummynet is compiled into the kernel
#sudo ./dummynet_load.sh

####################
# Run the experiment
####################
```

```
echo
echo "-- Running experiment..."

# Start server application
echo
echo "* Starting server..."
/usr/local/bin/iperf -s -u -i 0.5 -f k > ${output_name}_iperf_s.out -l
$packet_size&

# Wait for server to start
sleep 1

# Start client application
echo "* Starting client..."
#/usr/local/bin/iperf -c $server_address -B $client_address -u -i 0.5 -l
$packet_size -b $offered_load -t $test_duration -f k > $
{output_name}_iperf_c.out &
/usr/local/bin/iperf -c $server_address -u -i 0.5 -l $packet_size -b
$offered_load -t $test_duration -f k > ${output_name}_iperf_c.out &

# Start ping
echo "* Starting ping..."
sudo ping -i 0.25 -s $packet_size $server_address > ${output_name}_ping.out &

echo
echo ----------------------------
date
echo ----------------------------

# Start the QOMET emulation
echo "* Starting QOMET emulation..."
sudo ./wireconf/do_wireconf -q $test_name -f $from_node -F $client_address -t
$to_node -T $server_address -r $rule_number -p $pipe_number -d out

echo ----------------------------
date
echo ----------------------------

################
# Final actions
################
echo
echo -- Ending experiment...

# Kill processes

echo Terminating ping...
sudo killall -INT ping

sleep 2
echo Killing iperf server...
killall -INT iperf

#echo Killing tcpdump...
#sudo killall tcpdump

# Process output files
#echo Processing output files...

# Clean the system
#echo Unloading dummynet...
#sudo ./dummynet_unload.sh
```