Real-time emulation of networked robot systems

Razvan Beuran
National Institute of Information and Communications Technology, Hokuriku Research Center
Japan Advanced Institute of Science and Technology
razvan@nict.go.jp

Junya Nakata
National Institute of Information and Communications Technology, Hokuriku Research Center
Japan Advanced Institute of Science and Technology
jnakata@nict.go.jp

Takashi Okada
Japan Advanced Institute of Science and Technology
National Institute of Information and Communications Technology, Hokuriku Research Center
tk-okada@jaist.ac.jp

Yasuo Tan
National Institute of Information and Communications Technology, Hokuriku Research Center
Japan Advanced Institute of Science and Technology
ytan@jaist.ac.jp

Yoichi Shinoda
National Institute of Information and Communications Technology, Hokuriku Research Center
Japan Advanced Institute of Science and Technology
shinoda@jaist.ac.jp

ABSTRACT
In this paper we present a methodology for the evaluation of networked systems communicating using WLAN technology. We show a case study of goal-oriented cooperating robots, for which our approach is particularly useful. Developing robots is expensive; hence emulation can be employed in the first part of the development cycle to study robot software implementations in realistic conditions at a reduced cost. Our methodology is based on the emulation of both the robots and the WLAN communication technology. The robots we consider cooperate in order to efficiently reach a destination while avoiding collisions with obstacles and other robots. The WLAN communication emulation engine QOMET is deployed in the emulated robots to recreate network conditions similar to those occurring in a real WLAN environment. The experiments are run on a large-scale network experiment environment, StarBED, using the support software RUNE. Currently, over one hundred emulated robots can be run simultaneously during an experiment on our testbed.

Categories and Subject Descriptors
I.6.3 [**Simulation and Modeling**]: Applications.

General Terms
Measurement, Performance, Design, Experimentation, Verification.

SIMUTools 2008, March 3–7, 2008, Marseille, France.

Keywords
Real-time system emulation, networked robots, WLAN emulation, large-scale testbed.

1. INTRODUCTION
The study of networked systems through real-world tests is often hard to perform. The reasons for this can be one or more of the following:

1. The networked systems are expensive, hence difficult to obtain in a sufficient number, or they are still in development phase, therefore impossible to obtain at all;
2. The networked systems are complex, hence their behavior and motion are hard to manage with sufficient precision for obtaining reproducible results;
3. The network used by the systems under study is in itself difficult to control, either because it is shared with other users (e.g., the Internet), or because it is a shared environment by nature (such as a wireless medium, in which undesired interferences can perturb experimental results at any time).

As a consequence, analytical modeling and, often, simulation are extensively used to study networked systems. However, analytical modeling is an abstract technique that doesn't allow objective measurements, but only rough predictions of general system behavior. Simulation is very popular because of its low cost, and the fact that algorithm implementations can be studied. Although closer to reality, simulation is still relatively abstract, given that during a simulation experiment only models of real systems interact with each other in logical time.

A solution for the study of network systems in general, and wireless systems in particular, which gained popularity in recent years is that of emulation. Network emulation combines the advantages of real-world experiments and simulation. As in the case of real-world tests, emulation allows researchers to use the same implementations that are already or will be deployed on target systems. Hence, their observations are readily applicable to practical situations. Moreover, problems that may occur in real environments can be detected in the early stages of development, before the expensive deployment on real systems. This is possible since a wide range of controllable conditions can be studied through emulation, similar to the case of simulation. Only the need to run real applications on a potentially large number of computers may be considered as a disadvantage of emulation when compared to simulation. Nevertheless, the ability to study realistic conditions usually compensates this disadvantage.

So far network emulation was mainly used in conjunction with traditional network applications, such as network protocols. We adapted this methodology to the study of complex systems, such as networked robots cooperating in order to accomplish a task. In our approach the robot behavior implementation runs on standard PCs and the emulated robots are located in a virtual space. Hence it becomes easy to manage the conditions of the experiment and the motion of the robots in this virtual space. A specific requirement of such an approach is to also emulate the virtual environment in which the emulated robots are situated, so that they can "behave" in this virtual space as they would do in reality. For example, a GPS emulator will provide to the software implementation the robot position in the virtual space, just as a GPS card would do so in the real world. Concerning the network aspects, through the use of WLAN emulation the wireless medium effects can be fully controlled, so as to study the desired scenarios with ease and free of interferences.

Researchers implementing or evaluating autonomous robots have the choice of simulation, by using a product such as Webots [1], co-developed by the Swiss Federal Institute of Technology in Lausanne, Switzerland. Software simulators model robot components, sensors and surrounding environment. Such simulators require writing simulator-specific modules, and the results obtained may differ with respect to those that would be observed in a real system. Special small-size robot testbeds also exist, such as the free-flying robot testbed at the Stanford Aerospace Robotics Laboratory [2], which uses up to three robots moving on an air bearing over a 3x4 m flat surface. Although realistic, this testbed doesn't allow studying interactions between a larger number of robots and obstacles, which is essential for projects such as motion-planning development. Other researchers use real-world tests, as reported in [3]. However, using real environments is not a feasible approach at all times, for reasons of high costs, system unavailability, management difficulties. The approach of emulation that we envisage allows testing the real robot implementations in an emulated environment (including inter-robot communication), and therefore bridges the gap between software simulation and real-world tests for complex co-operating robot systems.

Robots are by their nature mobile; therefore, in order to communicate they must use a wireless network technology such as Wireless LAN (WLAN). Using real WLAN testbeds such as that at Emulab [4] for experiments is not a valid choice for robot research, since interferences cannot be controlled, and motion effects are not taken into account. As a consequence, emulating WLAN communication is mandatory in our approach. A survey of existing WLAN-related real-world and simulation testbeds is available in [5]. Previous approaches to WLAN emulation are oversimplified in general. Some emulators, for instance Seawind [6] or Empower [7], introduce network layer effects, such as bandwidth limitation, delay, packet loss. However these effects are directly provided by the user who configures the emulator; this means that the connection between these effects and reality is the user's task, and may not be accurate. There are also attempts to develop emulators that recreate by themselves network conditions that correspond to real events. Such is the case, for example, of W-NINE [8] and the wireless-network emulation extension of SDNE [9]. These two implementations both start from a description of node positions and movements. However the accuracy of the conditions they recreate is relatively low because of the simplicity of the employed models. For example, W-NINE uses tables to associate IP throughput to received signal levels, packet loss probability is considered to be either 0 or 1, etc. MobiNet [10] is another wireless network emulator; it focuses on ad hoc routing, but the detail level of the wireless communication emulation itself is still reduced.

In order to cope with these problems we developed QOMET (Quality Of applications in transforMing network Environments Testbed), which is intended to be a versatile WLAN emulator that accurately reproduces in a wired network the WLAN conditions that correspond to a user-defined scenario. Our approach is inspired by [8] and [9] in the sense that QOMET is a two-stage scenario-driven design. Therefore it can be run on top of any wired-network emulator. Our implementation can be used either standalone, to study predefined user scenarios, or in a library form, integrated into more complex systems, such as the networked-robot case study discussed in this paper. Moreover QOMET's modular architecture makes it possible to easily extend its functionality to other wireless environments.

The evaluation methodology we present in this paper is not limited to robots, but can be applied to the extended case of ubiquitous networked systems. There are already a number of implementations of emulators and testbeds for ubiquitous networks. TOSSIM [11] is a TinyOS [12] simulator implemented in Java which aims to simulate TinyOS applications in a virtual environment. ATEMU [13] too is able to emulate TinyOS applications, and has a flexible architecture to support other platforms. However, none of these tools provides the user with a method to describe the surrounding environment, or an interface with real nodes, which would allow tested systems to interact with real environments. This is of great use, especially in the final phases of system development.

Our methodology addresses these needs through the use of the software tool nicknamed RUNE (Real-time Ubiquitous Network Emulation environment). RUNE's role is to enable running tests involving a large number of nodes in a flexible and efficient manner. Currently, RUNE is being developed as an experiment-support software for StarBED and its successor StarBED2 [14], the large-scale network experiment environment at the National Institute of Information and Communications Technology (NICT), Hokuriku Research Center in Ishikawa, Japan. Using RUNE is an integral part of our experiment methodology.

2. METHODOLOGY OVERVIEW

Our general experimentation approach is to use software implementations of networked systems, such as robots, running on ordinary PCs, in order to evaluate them. A virtual environment is created for these systems, so that they can operate in quasi-real conditions. The overall system architecture of our methodology, shown for the particular case of networked robot systems, is depicted in Figure 1.

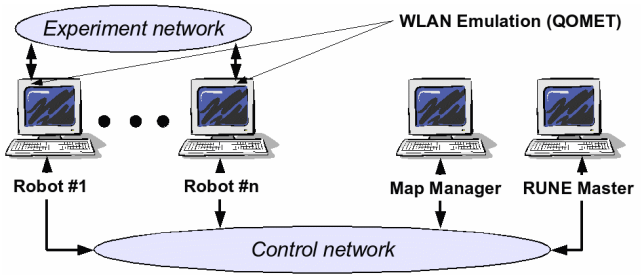

Figure 1. Overall system architecture.

The networked robots in our approach are considered to be equipped with various sensors, GPS equipment, and communication devices based on WLAN technology. Although running on standard PCs, the emulated systems are seen as located in a virtual space, whose topology and conditions are defined by the user. The characteristics of the virtual space are taken into account by the robot implementation, for example in the motion-planning algorithm. This allows testing algorithms before deployment in various realistic scenarios, including obstacles and large-scale topologies. For this purpose, some aspects of the virtual environment need to be emulated. A GPS module, for instance, will provide the software implementation with the robot position in the virtual space (see Section 3 for more details).

The emulated robots use a virtual WLAN connection to exchange information related to the accomplishment of their task. In practice they communicate through the "Experiment network" for the entire duration of the test. An additional network is available, the "Control network", which is used for management purposes. To make this possible, all experiment PCs are equipped with at least two network cards (100 Mbps or 1 Gbps type). By creating such a separation between the experiment network and the control network we ensure that control traffic does not interfere with experimental results.

In the case study we describe, the access to the experiment network is mediated by the use of the QOMET WLAN emulator, for which an overview is provided in Section 4. The WLAN communication emulation library is integrated into the software running on the experiment PCs, and allows recreating conditions similar to those that would occur in a real WLAN environment. The characteristics of the virtual space in which the emulated systems are located (topology of the area, distance between robots, etc.) are taken into account during WLAN emulation. This process is assisted by the module "Map Manager" that runs on a different PC than the systems under test, and communicates with a special module on those systems using the control network. The function of Map Manager is to keep an up-to-date robot system status, including robot positions and trajectories. This information, communicated through the control network, is used by the WLAN emulation module of each robot to calculate in real time the network conditions that correspond to the given virtual-environment configuration.

In order to perform such complex experiments we use the network testbed called StarBED, as mentioned in Section 1. StarBED has a large number of PCs (currently around 700) interconnected by a high-speed network. To simplify the manner in which users perform experiments on StarBED, an experiment-support software is available, called SpringOS. We currently develop an extension to SpringOS, named RUNE (Real-time Ubiquitous Network Emulation environment), which provides additional functionality, and is targeted at large-system emulation, such as that of sensor systems. RUNE was used in this paper for experiment integration and the "RUNE Master" module in Figure 1 manages the entire system we describe, as it will be detailed in Section 5.

3. ROBOT EMULATION

In disaster areas or office buildings autonomous robots can act instead of human beings. Rescue robots are able to accomplish many tasks in dangerous places where humans cannot enter, such as sites where harmful gases or high temperatures are present. Cleaning robots can save costs by performing various routine tasks. In all these examples robots have to move to their destination in order to perform their task. For this purpose they need to be able to recognize the environment around them, and use a motion-planning method in order to avoid collision with obstacles or other robots.

The experiments for evaluating such research are difficult to perform, since the cost of these real autonomous robots is high. This is particularly true if researchers want to experiment with more than a few robots, and need to test systems with tens or even hundreds of robots. The solution of simulation is a quick way to perform experiments or evaluate algorithms. However, the results obtained from simulators may not always be reliable, since during simulation only models of the systems and their components interact with each other in logical time. This difference is important, for instance, when testing algorithms for robots that co-operate with each other using communication protocols. In such a case it is preferable to study in real time the performance of software implementations, in which realistic order of events and timing takes place.

3.1 General Architecture

In this paper we propose a methodology that can be used for the evaluation of large-scale autonomous networked robot systems. The core of this methodology is the idea of emulation. In this approach the robot behavior implementation, including aspects of motion planning and communication with other robots, is run on a standard PC. Various modules are connected to this implementation with the goal of allowing it to "behave" as it would do in a real-world environment (see Figure 2). For example, a WLAN emulation module ensures that a robot can communicate with other robots using a real protocol implementation in conditions similar to those that would occur in a real WLAN environment. A GPS emulation module provides the current coordinates in a virtual space to the robot behavior implementation, exactly as a GPS card would do so in a real environment. The robot can be equipped with various other sensors, each requiring a specific emulator to be present. For instance, we can use a thermal field emulation module to recreate temperature variation in the virtual space; to enable the use of

visual sensors, a visual environment emulation module is required, and so on.

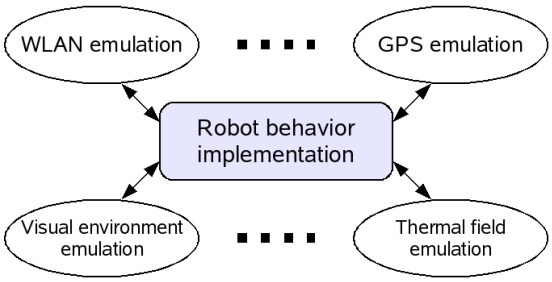

Figure 2. General architecture for robot emulation.

One important aspect of the general architecture we propose is that the robots we emulate are able to communicate realistically, in this case by means of emulated 802.11 wireless networks (WLAN). Mobile robots equipped with WLAN transceivers can transfer point-to-point information to each other, and thus cooperate. If an ad hoc protocol is used, such as those of MANET (Mobile Ad hoc NETworks), several robots can autonomously form a network and exchange data over larger distances. In order to fulfill their different individual tasks, robots have to move to different locations and accomplish their missions. Based on the initial information, robot trajectories are pre-planned. However, if the robots are able to communicate and share information, they can also respond to unexpected changes in the topology, and avoid the other robots in a dynamic and intelligent manner. Such autonomy is essential for various tasks, including safety-critical and mission-critical ones, such as tactical operations, rescue missions, national security, and so on. WLAN capabilities also make it possible for human users to remotely control robots, so as to coordinate them in the field, provide new tasks, etc.

3.2 Motion Planning

In this paper we shall illustrate the use of our experiment methodology by studying a motion-planning algorithm through emulation. Many path-planning algorithms have been proposed to date. The performance of a motion-planning algorithm can be characterized by the following three main properties: speed, completeness, and optimality. In a dynamic and unknown environment, robots must re-plan their motion many times, because the environment changes. Therefore, when robots need to continuously replan on-the-fly their trajectory, algorithm speed is one of the most important properties.

3.2.1 Probabilistic Roadmap Planner

The probabilistic roadmap (PRM) planner is a very popular motion-planning algorithm because of its speed. A PRM planner randomly samples the set of locations where a robot can move. PRM then registers the collision-free locations as possible milestones. Next it tries to connect pairs of these milestones, and saves the collision-free connections as possible robot trajectories. In the context of PRM, "probabilistic roadmap" signifies the undirected graph composed of collision-free connections, in which the edges are the trajectories of robots, and the nodes are the milestones. The planner uses Dijkstra's algorithm to find the optimal path. For this purpose weights can be assigned to edges.

The PRM method that we use is based on the "Path Planning in Expansive Configuration Spaces" algorithm [15]. This algorithm iteratively and alternately executes two basic steps, expansion and connection, until either a path is found or the maximum number of iterations is reached. The main idea of this PRM planner is illustrated in Figure 3.

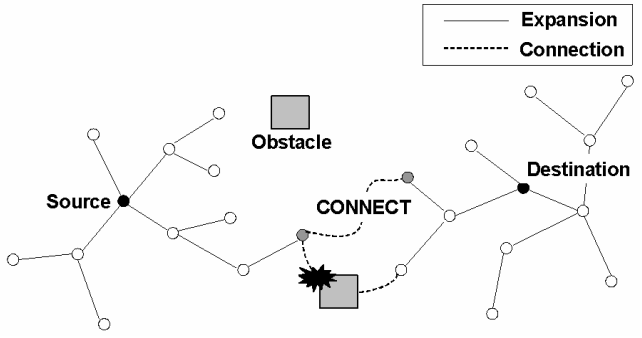

Figure 3. Probabilistic roadmap motion planning.

The algorithm in [15] uses trees instead of graphs. In what follows we use the standard notation for trees, $T = (V,E)$, where V and E are the sets of vertices and edges of the tree T, respectively.

Expansion is the first step of the algorithm, by which the planning method builds two trees: $T_{Source} = (V_{Source}, E_{Source})$, and $T_{Destination} = (V_{Destination}, E_{Destination})$, starting from the robot initial position (source) and the robot destination, respectively. When building each of these trees, the planner picks from the existing milestones a node x, which is chosen with a probability proportional to $1/w(x)$. The value $w(x)$ represents the weight of node x, and is equal to the number of neighbor milestones of x plus 1 (the node x itself). Then the planner samples the neighborhood of x to determine a number of points effectively reachable from x, and adds them to the tree.

Connection is the second step of the algorithm, in which PRM tries to connect the two previously built trees, T_{Source} and $T_{Destination}$. For every node in the set of vertices of the corresponding trees, V_{Source} and $V_{Destination}$, PRM checks whether they "see" each other, i.e. one is reachable from the other on a straight-line trajectory which is not obstructed by obstacles or other robot trajectories. Given that some nodes were already checked in the previous iteration, only newly-added nodes in V_{Source} are checked against all the nodes in $V_{Destination}$, and only newly-added nodes in $V_{Destination}$ are checked against old nodes in V_{Source}. When finding the first pair of milestones which are reachable from each other, and which are separated by a distance inferior to $D_{Connection}$, the milestones are used to connect the two trees and the algorithm ends. Otherwise the expansion step is repeated.

3.2.2 PRM for Dynamic Environments

In the case study we show in this paper, robots are considered to be placed in environments that are unknown or change in a dynamic manner. The PRM method described above is not well suited for such a case. For example, the previous PRM planner cannot predict whether there are any collisions or not in the tree built from destination, $T_{Destination}$. This is because the planner has no way to know the time when the robot will reach the milestones in this tree. Hence the planner cannot at all expand $T_{Destination}$ in this case.

To cope with the conditions of unknown and dynamic environments, we adapted the PRM planner as follows:

1. In the expansion phase, the only tree that is grown is T_{Source}. For this tree the planner can detect collisions with known robots or obstacles. Collision checking is dynamic. This means that the robot can avoid collisions not only with static obstacles, but also with moving robots and obstacles;
2. In the connection phase, the planner uses only the newly-added milestones to the (unique) tree T_{Source} to check whether there is a connection between the milestones and the destination. As a collateral advantage this reduces significantly the number of comparisons. Moreover, to speed up trajectory finding, the parameter $D_{Connection}$ is not used anymore. The first milestone from which the destination is reachable is the solution of the algorithm, and the search ends;
3. Given that the environment is dynamic, as soon as a robot becomes aware of a change in the known conditions (position of obstacles, positions and trajectories of other robots), the PRM for dynamic environment is restarted. The "source" for the new motion planning is the future position of the robot after the time interval T_{Think}, the time estimated as necessary to find a new path to destination (since the robot doesn't stop moving while thinking).

While building the T_{Source} tree, our algorithm selects K points from the set $N_{dist}(x)$, where $N_{dist}(x)$ is given by the formula:

$$N_{dist}(x) = \{q \in C \mid d_{\min} < dist_C(q,x) < d_{\max}\}. \qquad (1)$$

In Equation (1) C represents the space in which the robot is located; $d_{\min}$ and $d_{\max}$ represent the minimum and maximum distances at which a point can be selected. Of the selected K points, only those which are effectively reachable are selected and added to the tree T_{Source}.

4. WLAN EMULATION: QOMET

The scenario-driven architecture we propose for WLAN emulation has two stages. In the first stage, from a real-world scenario representation we create a network quality degradation (ΔQ) description that corresponds to the real-world events (see Figure 4).

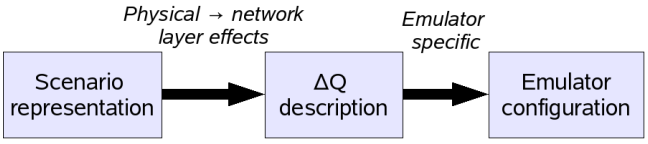

Figure 4. Two-stage WLAN emulation.

By quality degradation we mean the change in network service quality between two measuring points; we denote this degradation by the shorthand ΔQ. Since the ΔQ description represents the varying effects of the network on application traffic, the WLAN emulator's function is to reproduce it. The ΔQ description calculated in the first stage is therefore converted into an emulator configuration that is used during the effective emulation process to replicate the user-defined scenario in a wired network. This makes it possible to study the effects of the scenario on the real application under test.

The following is a summary of the features of the current QOMET implementation. QOMET enables emulation of 802.11a/b/g WLANs [16-19] using a model based on receive-sensitivity thresholds available from most manufacturers, and that includes effects of background noise. Support exists for 802.11g stations operating in compatibility mode (i.e., when in the presence of 802.11b stations). Our model also takes into account interference between neighboring nodes, either resolved through the CSMA/CA (Collision Sense Multiple Access with Collision Avoidance) mechanism, or regarded as noise if the interfering signal is too weak to be detected.

The QOMET experiment scenario consists of an XML-based description of the WLAN nodes, topology elements, motion patterns, and communication environment properties. For WLAN nodes one can specify the WLAN adapters and their properties (transmitted power, antenna gain, etc.). The virtual environment topology and objects are defined by the user, and the communication environment is computed depending on the varying node positions, either off-line or dynamically.

The WLAN emulation model that we propose is an aggregation of several models used at the various steps of the conversion of the scenario representation to the network ΔQ description which is needed to recreate those scenario conditions. QOMET models were initially presented in detail in [20]. Since then some improved models were designed that supersede those in [20], and that we shall describe in this paper. For brevity purposes, at times, we skip here some modeling details; please refer to [20] for a more complete description of QOMET. The following subsections describe the main aspects of QOMET models at each level of the conversion: real world scenario to physical layer, physical layer to data link layer, and, finally, data link layer to network layer. Modeling stops at network layer because it is at this level that we introduce the quality degradation using a wired network emulator.

4.1 Real-World Scenario to Physical layer

In order to calculate the effects of real-world scenario events on the physical layer of a WLAN station, it is necessary to determine first the signal attenuation due to the distance between the communicating stations, interposed obstacles, etc.

For this purpose we use the log-distance path-loss model [21]. This model gives the received power, P_r, expressed in dBm (decibel-milliwatt), as function of the received power at the distance of 1 m, P_{r0}, and the distance, d, between receiver and transmitter. The communication environment is described by the parameters α (the path-loss coefficient), σ (shadowing parameter; the standard deviation of the zero centered Gaussian distributed random variable X_σ), and W (wall attenuation; considered equal to zero for indoor environments):

$$P_r(d) = P_{r0} - 10 \cdot \alpha \cdot \log_{10}(d) - W + X_\sigma . \qquad (2)$$

The following step is computing the frame error rate (FER) corresponding to a received power strength. For this purpose we created a model that doesn't rely on theoretical or manufacturer-measured bit error rate (BER) versus signal-to-noise ratio (SNR) dependencies. Theoretical dependencies are too abstract if one wishes to study a realistic case, and measured BER data is only

available from a small number of manufacturers (e.g., Intersil [22]). Using it in our approach would strongly limit the number of real adapters we can emulate.

As a consequence we decided to use a model that is based on receive sensitivity, information that is provided by most manufacturers as part of the specifications of WLAN adapters. We also included noise in our model, since such interference has a significant effect of WLAN performance. For modeling we used the information in [16, 23] regarding the procedure of measuring receive sensitivity. Based on a negative exponential model, we can determine the frame error rate (*FER*) corresponding to a received power strength. This P_r-threshold-based model computes *FER* as function of the adapter-specific receive sensitivity threshold for the current operating rate, *S*, the received power, P_r, the background noise, *N*, and the thermal noise N_{th}, as follows:

$$FER = FER_S \cdot e^{\gamma(S-(P_r-N)-N_{th})}, \quad (3)$$

where γ is a constant to be determined by calibration for each adapter (at the moment we use the default value 1), and FER_S is the frame error rate when P_r reaches the threshold *S*. According to [16, 17], FER_S equals 0.08 for 1024-byte frames for the DSSS encoding used by 802.11b/g, and 0.1 for 1000-byte frames for the OFDM encoding used by 802.11a/g.

Note that the difference (P_r–*N*) in Equation (3) actually represents the connection SNR. N_{th} is the thermal noise at room temperature, and approximately equals –100 dBm if considering the 22 MHz operating frequency bandwidth of 802.11a/b/g networks. Note that the FER given by Equation (3) must be limited at 1, since the result represents a probability.

The background noise *N* in Equation (3) has two sources. One is the man-made electro-smog from devices operating in the same frequency band (for instance microwave ovens or cordless phones when speaking about the 2.4 GHz band of 802.11b/g). This parameter can be configured by the QOMET user. The other noise source is represented by the signals generated by other WLAN devices. Such signals are treated as noise if the signal strength of the received power is inferior to the lowest receive-power sensitivity threshold of the emulated adapter. This interference type can cause frame errors at reception that will lead to retransmissions, hence delays, and possibly packet loss. In the opposite case the signal is considered to be properly detected by the receiver, and therefore induces transmission delays due to the use of the CSMA/CA mechanism. This is the second effect of interference, and it is dealt with separately (see Section 4.3).

In order to determine the power of the received signal from interfering WLAN sources we use Equation (2) as well. However, interfering sources can transmit on different channels that the channel on which the receiver listens. In this case the inter-channel attenuation must be computed. For this purpose we used the requirements in the IEEE 802.11 specifications regarding the transmit spectral mask [16, 17]. Figure 5 shows our model's attenuation characteristics depending on channel distance (i.e., the absolute difference between channel ids). Note that in 802.11a networks effectively-used channels are at a minimum distance of 4, which reduces considerably interferences, since attenuation will always exceed 30 dB. For 802.11b/g WLANs, distance between different channels can be 1, making it a more difficult situation. Subsequently this attenuation is subtracted from the computed power of interfering signals; the result is then compared to the receive sensitivity level, as discussed in the previous paragraph.

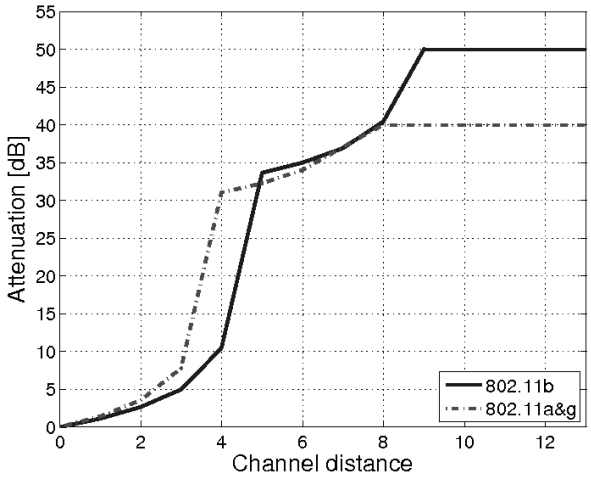

Figure 5. Inter-channel attenuation model.

4.2 Physical Layer to Data Link Layer

The frame error rate computed in the previous section will have various effects on the behavior of the 802.11 MAC layer. One of them is related to operating rate adaptation, which for most WLAN adapters is based on the ARF (Auto-Rate Fallback) mechanism [24]. This features makes it possible to dynamically determine the operating rate of the WLAN adapters, and is taken into account by QOMET.

The next step is to use a delay model for determining the delay, *D*, and the jitter, *J*, introduced at data link layer by the interaction between the MAC retransmission mechanism and the frame error rate. The formula we propose below computes the delay as the weighted average of the delays induced to frames undergoing a number of *i* retransmissions before being received, D_i, with *i* from 0 to *r*, where *r* is the maximum number of retransmissions (in addition to the initial first transmission of a frame). Default values for *r* are 6 and 3, depending whether the RTS/CTS (Request To Send/Clear To Send) mechanism in IEEE 802.11 MAC is disabled, or enabled, respectively [16].

$$D = \frac{\sum_{i=0}^{r}(1-FER)\cdot FER^i \cdot D_i}{\sum_{i=0}^{r}(1-FER)\cdot FER^i}, \quad FER \neq 0 \wedge FER \neq 1; \quad D\big|_{FER=0} = D_0; \quad D\big|_{FER=1} = \infty. \quad (6)$$

The weights included in Equation (6) represent the probabilities for a frame to undergo *i* retransmissions. Delay values, D_i, are computed as described in [20] using T_{SIFS}, T_{ACK}, T_{DIFS}, T_{Frame}, which represent the time needed for SIFS (Short Inter Frame Space), the ACK (acknowledgement) frame, DIFS (Distributed coordination function Inter Frame Space), and the frame payload itself, respectively. Note that if the RTS/CTS mechanism is enabled, additional terms must be considered, namely T_{RTS} and

T_{CTS}, which represent the time needed to transmit an RTS and a CTS frame, respectively, as well as twice more T_{SIFS}.

Jitter is computed using a similar weighted average formula with that for delay. The jitter values for frames undergoing a number of i retransmissions before being received, J_i, can be computed using the following formula:

$$J_i = |D_i - D|, i = \overline{0, r}, FER \neq 0 \wedge FER \neq 1. \quad (7)$$

4.3 Data Link Layer to Network Layer

After data link layer parameters are computed, we can proceed to the last step: calculating the network layer parameters. These parameters are the output of the first stage of the emulator, and they can be used to configure a wired-network emulator so as to reproduce the WLAN conditions associated to the given scenario.

Packet loss rate, *PLR*, is computed from *FER* by taking into account the 802.11 MAC retransmission mechanism:

$$PLR = FER^{r+1}. \quad (8)$$

The delay and jitter at network layer are the same with those discussed in Section 4.2.

The other important parameter at network layer is bandwidth. The bandwidth model we propose to determine the effectively available bandwidth, *B*, as "perceived" at network layer is given by the equation:

$$B = \frac{T_{Frame}}{D} \cdot R, \quad (9)$$

where *R* is the current operating rate of the WLAN station.

Note that the formulas given here for the computation of delay and bandwidth only take into account the environment effects on communication. However, if multiple users share the wireless media, additional quality degradation occurs because of the use of the CSMA/CA (Collision Sense Multiple Access with Collision Avoidance) mechanism of 802.11. To account for these effects we use an equation inspired by the analytical model given in [25], as discussed in [20].

5. EXPERIMENT INTEGRATION: RUNE

Our goal is to develop a methodology for performance assessment of networked systems in general, not just computer networks. Networks composed of heterogeneous elements, usually called ubiquitous networks, have different properties than computer networks in many aspects, such as: high node variability and network media variety, huge number of nodes, importance of the interaction with the surrounding environment, as well as that of geographical information, changing network topology, etc.

For these reasons testbeds for ubiquitous network emulation must meet the following requirements:

1. Emulate the surrounding environments and provide an interface between the emulated nodes and these environments;
2. Provide an interface between the physical space and the logical space, so as to make possible virtual and real mixed setups;
3. Support the numerous nodes of ubiquitous networks;
4. Emulate the various architectures of nodes and networks that form typical heterogeneous networks (processors, middleware, etc.);
5. Provide a multi-level emulation layer;
6. Provide an emulation support system that enables execution of experiments in a controlled manner through an automated-execution mechanism.

5.1 General Description

In order to meet the aforementioned requirements on the StarBED2 testbed, the experiment-support software RUNE is being currently developed. A detailed account of RUNE architecture and implementation is given in [26]; we outline here only its main features. RUNE provides an API set for controlling experiments. The fundamental goal of RUNE is to implement a test environment in which a number of "spaces" that emulate each experiment target can run on either single or multiple nodes. RUNE provides a reasonably abstracted interface for easily implementing emulation targets as spaces without much concern about the interaction between emulation nodes. RUNE has the following roles: (i) experiment environment setup/cleanup and progress management; (ii) procedure invocation; (iii) interaction between spaces; (iv) time synchronization; (v) mutual exclusion (not implemented yet).

Figure 6 shows the structure of an experiment implemented using RUNE. The "RUNE Master" module manages the configuration of each experiment, and controls the progress of the experiment. The execution of all spaces deployed on multiple nodes is initiated by RUNE Master via modules called "RUNE Manager". The RUNE Manager is deployed on every emulation node, and mediates communication between spaces through objects called "conduits". Spaces implementing emulation targets exist on emulation nodes in the form of shared objects, loaded dynamically by the RUNE Manager.

5.2 Emulation Process

The emulation process performed by RUNE takes place as follows. First of all RUNE Master is compiled with the experiment definition file, which includes the information regarding spaces and conduits. When run, RUNE Master sends the instruction "attach process" to the RUNE Managers executed on each node. A space then returns its entry point information to the RUNE Manager, which includes pointers to the available functions.

When the RUNE Manager notifies the RUNE Master of the completion of the "attach" process, the latter indicates the "initialize" process of all spaces to RUNE managers on each node. After the initialization of all spaces is finished, the RUNE Master instructs the managers to start the iterated invocation of the "step" function, which represents the main body of a space. Accordingly, spaces start to execute the emulation step-by-step, and inform the corresponding RUNE Master of execution status. At the end of the experiment, the RUNE Master starts the "finalization" process by notifying all nodes. Subsequently, spaces release the work area allocated in the initialization process.

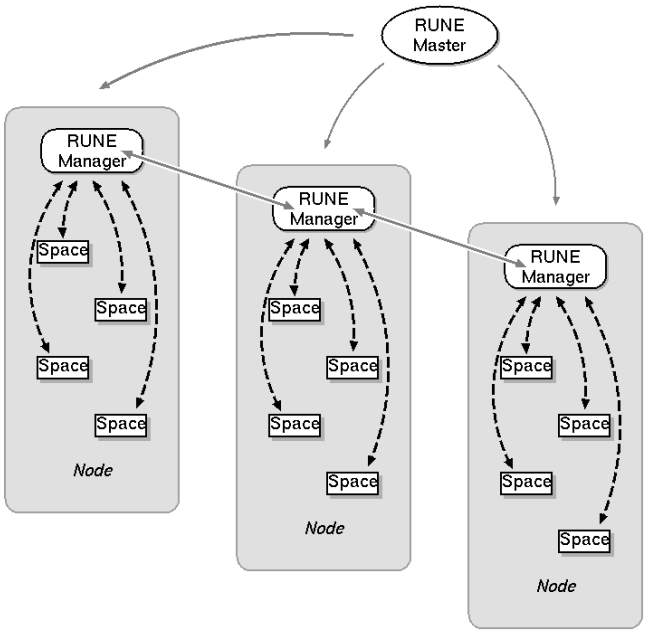

Figure 6. Structure of experiments using RUNE.

6. EXPERIMENTAL RESULTS

The range of applications that we can emulate using the two tools we described above, QOMET and RUNE, is very large, both in indoor and outdoor environments. VoIP experiments with reference to rescue worker communication in emergency conditions were presented in [20]. The emulation of ubiquitous home environments with application to room temperature control was described in [26, 27]. At the moment we pay particular attention to emulation of robots that cooperate using wireless communication in order to achieve various tasks. The implementation of such robot systems in reality incurs high costs since they need to be equipped with sensors, motors, WLAN cards. These costs become prohibitive if systems with tens or hundreds of robots are to be tested, and it is useful to employ emulation environments such as ours in the initial design and implementation phases.

In what follows we present an experiment carried out using our networked-system evaluation methodology. In this experiment we study through emulation a motion-planning algorithm that we designed for the case of dynamic environments. The setup used for the experiments described here is the same with that shown previously in Figure 1. RUNE controls the start of experiment, its execution and finalization on all the participating nodes, including the Map Manager.

The virtual space in which the robots are considered to be located is the basement of a building in which 10 robots are assigned some tasks at various locations. They must travel from the initial starting position to their individual destination while making sure no collisions occur with the obstacles present (building pillars) or with the other robots. The virtual topology is depicted in Figure 7. Note that we can perform experiments with over 100 emulated robots, but we show here only a small-scale example for the sake of clarity.

In the emulated environment robots are initially located at 15 meters with respect to their neighbors. For Robot #*n*, with *n* from 1 to 10, its source and destination are denoted by *S#n* and *D#n*, respectively. Robots are considered to have a radius of 1 m and a constant speed of 0.5 m/s. Obstacles also have a radius of 1 m; they are represented by dark discs in Figure 7. The robots are assigned priorities equal to their id; the robot with the highest priority is Robot #10. The range of visual sensors (represented by an omni-directional camera) is considered to be 10 m; this is the maximum distance from which robots can visually detect obstacles and other robots. The time in which robots assume to be able to find a trajectory, T_{Think}, has an initial value of 1 s. If it is determined that this time is not sufficient, the value is gradually increased.

We considered that robots are equipped with 802.11b transceivers, which consequently must be emulated. In the experiment, the QOMET WLAN communication emulation library computes in real time the characteristics of WLAN communication between robots at each processing step (currently 250 ms). Then network degradation is enforced using *dummynet* [28], by applying to the wired network the WLAN characteristics calculated previously. This takes place in real time on the testbed. The emulated WLAN environment used in our experiment had the following main parameter values: $\alpha = 5.6$, $\sigma = 3.1$, $N = -100$. This corresponds to bad propagation conditions, and gives a communication range of approximately 18 m.

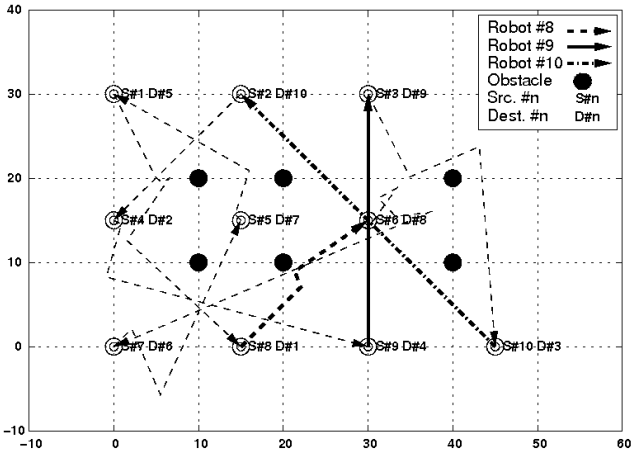

Figure 7. Robot initial positions and trajectories.

In Figure 7, for simplicity reasons, we emphasized only the trajectories of three robots, those with ids 8, 9, and 10. Analyzing the trajectories of Robot #8 and Robot #10, one can notice that their initial trajectories starting at S#8 and S#10, respectively, lead them to collision at position (30,15). However, when Robot #8 approaches the point of coordinates (22,7) it enters in the communication range of Robot #10. As a consequence, they are able to exchange information about each other's trajectory. This leads to the fact that Robot#8 (the lower priority robot) re-plans its trajectory to avoid collision. The higher priority robot, Robot #10, continues moving on the initial trajectory all the way to its destination at (15,30). Assuming the robots would not have been able to communicate, the trajectory change would have occurred later (when robots enter each other's visual range), and probably for both robots, since no priority mechanism could have been enforced.

Another remark about Figure 7 concerns Robot #9. This robot followed the simplest trajectory from its starting point S#9, located at (30,0), to the destination D#9 at (30,30). The fact that it could communicate with the other robots in its neighborhood made it possible for Robot #9 to know their intended trajectory and predict when they will arrive at the conflict location, (30,15). Consequently the robot decided correctly to continue its route without unnecessary detours. In this case too a different decision may have been taken if only visual sensor information was used, since in the absence of communication one robot has no way to know whether a robot trajectory will conflict with its own.

We shall not discuss here the trajectories of all robots, but one can see in Figure 7 how the obstacles and other robots' trajectories influence the path of each robot. For example Robot #3, starting at (30,30) needs to change its trajectory from the initially planned one when it observes the obstacle located at (40,10).

Sometimes a robot trajectory may interfere with the trajectories of other robots, and the robot must stop to "think" in order to find a trajectory. In our initial tests this triggered a motion-planning algorithm implementation bug. We managed to identify the cause of this problem, and fixed it, which would have been difficult to accomplish without running large-scale real-time experiments. When robots interact with each other in real time and in complex scenarios, implementation issues may be revealed, issues that would otherwise go unnoticed. In Figure 8 we show (for illustration purposes only) a caption of our experiment visualization interface for the case of 100 simultaneously emulated robots. From this figure one can understand the complexity of the situations that can be encountered in experiments of this scale.

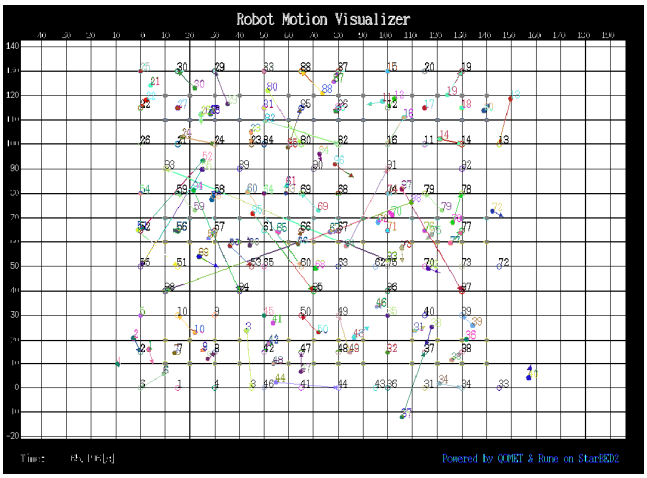

Figure 8. Emulation experiment with 100 robots.

7. CONCLUSIONS

The approach we propose for the assessment of networked systems through real-time emulation makes it possible for users to objectively evaluate software implementations of such networked systems under realistic conditions. The software implementations are afterwards readily deployable on real target systems; therefore their evaluation through emulation is an essential way of reducing the development cost of networked systems. This is especially true for the case of robot systems, which are particularly expensive.

The two tools that enable the usage of the proposed methodology are the QOMET WLAN emulator, and the RUNE emulation environment, all running on the large-scale network experiment testbed, StarBED. QOMET allows the transformation of a user-meaningful real-world representation of a WLAN environment (termed "scenario representation") into a network quality degradation description (termed "ΔQ description"). The ΔQ description obtained is sufficient to subsequently configure a wired-network emulator and effectively reproduce in real time an environment that corresponds accurately at network level to the emulated WLAN scenario. RUNE is a flexible experiment-support environment that can be used to perform ubiquitous network tests on StarBED in a straight-forward manner.

We illustrated the practical use of our approach on the particular case of networked robot emulation, for which we designed and implemented a specific framework. We provided a detailed analysis of a case study of robot motion planning for an algorithm we developed based on Probability Roadmap Planner, and adapted to dynamic environment conditions. The experimental results showed how one can assess the properties of such an algorithm through emulation in complex scenarios. Our experiments helped us identify and correct several implementation issues.

Our methodology proved very effective, and we are now in the process of using it to support the development of a pedestrian localization system using active tags in collaboration with Panasonic System Solution Company. As future work regarding the development of QOMET we intend to add features to it so as to make it possible for users to define scenarios in a more realistic way, including streets and buildings. More advanced ad hoc network features, such as routing will also be included in future versions. Concerning RUNE, a high priority is given to adding support for processor and middleware emulation, since RUNE is intended for the general case of ubiquitous network emulation. Other desired features are, for example, more strict synchronization and mutual exclusion.

8. ACKNOWLEDGMENTS

We would like to acknowledge the contribution to the ideas presented in this paper of Lan Tien Nguyen, Khin Thida Latt, Toshiyuki Miyachi, Saber Zrelli, Assoc. Satoshi Uda, and Assoc. Ken-ichi Chinen.

9. AFFILIATION INFORMATION

National Institute of Information and Communications Technology (NICT), Hokuriku Research Center, 2-12 Asahidai, Nomi, Ishikawa, 923-1211 Japan. Tel: +81-761-51-8118.

Japan Advanced Institute of Science and Technology (JAIST), 1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan. Tel: +81-761-51-1111.

10. REFERENCES

[1] Webots. Fast Prototyping and Simulation of Mobile Robots. http://www.cyberbotics.com

[2] Stanford Aerospace Robotics Laboratory. http://arl.stanford.edu

[3] Chaimowicz, L., *et al*. 2005. Deploying Air-Ground Multi-Robot Teams in Urban Environments. In Proc. of the 2005 International Workshop on Multi-Robot Systems (Washington DC, U.S.A., March 2005). 223-234.

[4] University of Utah, School of Computing. Emulab – Network Emulation Testbed. http://www.emulab.net

[5] Kropff, M., Krop, T., Hollick, M., Mogre, P.S., Steinmetz, R. 2006. A Survey of Real-World and Emulation Testbeds for Mobile Ad hoc Networks. In Proc. of TridentCom 2006 (Barcelona, Spain, March 2006).

[6] Kojo, M., Gurtov, A., Manner, J., Sarolahti, P., Alanko, T., Raatikainen, K. 2001. Seawind: A Wireless Network Emulator. In Proc. of 11th Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB) (Aachen,Germany, September 2001).

[7] Zheng, P., Ni, L.M. 2003. EMPOWER: A Network Emulator for Wireline and Wireless Networks. In Proc. of IEEE Infocom 2003 (San Francisco, U.S.A., April 2003)

[8] Perennou, T., Conchon, E., Dairaine, L., Diazet, M. 2004. Two-Stage Wireless Network Emulation. In Proc. of WCC2004 (Toulouse, France, August 2004).

[9] Bateman, M., Allison, C., Ruddle, A. 2003. A Scenario Driven Emulator for Wireless, Fixed and Ad Hoc networks. In Proc. of PGNet2003 (Liverpool, U.K., June 2003). 273-278.

[10] Mahadevan, P., Rodriguez, A., Becker, D., Vahdat, A. 2005. MobiNet: A Scalable Emulation Infrastructure for Ad hoc and Wireless Networks. In Proc. of the International Workshop on Wireless Traffic Measurements and Modeling (WiTeMe 2005).

[11] Levis, P., Lee, N., Welsh, M., Culler, D. 2003. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In Proc. of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).

[12] TinyOS embedded networked sensor operating system. http://www.tinyos.net

[13] Polley, J., Blazakis, D., McGee, J., Rusk, D., Baras, J.S. 2004. ATEMU: A Fine-grained Sensor Network Simulator. In Proc. of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004).

[14] Miyachi, T., Chinen, K., Shinoda, Y. 2006. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In Proc. of International Conference on Performance Evaluation Methodologies and Tools (Valuetools2006) (Pisa, Italy, October 2006).

[15] Hsu, D., Latombe, J.-C., Motwani R. 1997 Path Planning in Expansive Configuration Spaces. International Journal of Computer Geometry and Applications, Vol. 9, No. 4/5 (1997). 495-512.

[16] ANSI/IEEE Standard 802.11. 2003. 1999 Edition, Reaffirmed 2003.

[17] ANSI/IEEE Standard 802.11a. 1999.

[18] ANSI/IEEE Standard 802.11b. 1999.

[19] ANSI/IEEE Standard 802.11g. 2003.

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

[21] Rappaport, T.S. 2002. Wireless Communications: Principles and Practice. Prentice Hall PTR, 2nd edition. (2002)

[22] Intersil. 2002. HFA3861B: Direct Sequence Spread Spectrum Baseband Processor. Intersil data sheet. (February 2002)

[23] Intersil. 2000. Measurement of WLAN Receiver Sensitivity. Intersil technical brief. (February 2000)

[24] Kamerman, A., Monteban, L. 1997. WaveLAN-II: A high-performance wireless LAN for the unlicensed band. Bell Lab Technical Journal (1997). 118-133.

[25] Gupta, P., Kumar, P.R. 1999. Capacity of wireless networks. Technical report, University of Illinois, Urbana-Champaign. (1999)

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

[27] Beuran, R., Nakata, J., Okada, T., Miyachi, T., Chinen, K., Tan, Y., Shinoda, Y. 2007. Performance Assessment of Ubiquitous Networked Systems. 5th International Conference on Smart Homes and Health Telematics (ICOST2007) (Nara, Japan, June 21-23, 2007, pp. 19-26).

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