Video Simulation through 802.11.e using NS-2 Documentation

The EuQoS Project
C-Lab, Paderborn
Vadim Kataev vkataev@upb.de

September 17, 2008

This document presents an example of setting up an environment for video traffic simulation through the wireless networks.

1 Overview

NS2

NS2 is an object-oriented, discrete event driven network simulator. It is possible to setup simulation networks of many kinds, because NS2 is a set of well documented C++ classes. NS2 is Open Source, so that its user can change and extend existed classes very easily.

Especially for the research in computer networks, NS2 is preferable because any adjustments of its internal behavior are easy to implement directly by changing proper parts of its source code. There are exist also some functional extensions of basic NS2 environment. They provide NS2 with improved functionality for many different types of simulation.

Our objecitve requires the support of 802.11e HCCA and EDCA for wireless network simulations. We want also to simulate video traffic and we are interested in the extension of NS2 with such desirable functionality.

802.11e simulation in NS2

Existing 802.11 support in basic NS2 environment is disapointing for a number of reasons. For example, at this momemnt it does not provide 802.11e HCCA and EDCA standards. One good implementation of 802.11 standard for NS2 is developed at French national research institution (INRIA).

Its sourcecode is freely accessible. It is released under the GPL2 software license. [http://yans.inria.fr/ns-2-80211/]

802.11 extension implements new 802.11 PHY and MAC layers for NS2. 802.11 extension, develped at INRIA supports mandatory aspects of 802.11e:

- access differentiation through "Access Categories".
- access differentiation through TSPEC granting by the QoS AP

After 802.11 implementation is integrated into basic NS2 environment, it is possible to associate different network packets with different priority classes.

One important note is that NS2 tracing mechanism does not work with the 802.11 module. It is to use the module's debugging tracing macros. In each 80211/*.cc file, there is a self-explained macro definition at the top of the file, that can be commented, if required. Information about packet loss and performance is available after simulation in the form of the properly arranged text files which are easy to process with different GNU text processing tools.

Another characteristic of INRIA's implementation is that simulation that uses 802.11e standard, requires specially written scenario script which is different from an usual NS2 scenario script.

MPEG simulation in NS2

Basic NS2 environment supports researchers with CBR (Constant Bit-Rate) and VBR (Variable Bit-Rate) traffic methods. Both methods cannot represent a video traffic per-packet model simulation precisely. Because our tasks require simulation of video transmissions, we have integrated special simulation framework called EuQoS Video based on the well-known Evalvid framework. EuQoS Video is developed to allow precise simulation of video traffics.

In this simulation method, researchers can use video traffic trace files to evaluate the performance of the proposed mechanisms. EuQoS Video cares about binding video packets with the network frames, tracing relevant events in the simulation time. It adds a set of new Agents and one new Application into NS2 framework.

Both, 802.11 and EuQoS Video extensions are inserted into the basic NS2 environment. They represent new classes and small changes in existed classes. The former are instantiated in our simulation tests.

2 Architecture

We have developed a heterogeneous system of the simulation with the feedback. It consists of four major components: Simulator, Traffic generator, Fuzzy Admission Controller, and Simulation scenario.

Simulator

Simulator is NS2 simulator with extended functionality, i.e. it includes 802.11e and MPEG simulation support as described in previous section above.

General changes are done in the EuQoS Video classes in order to they take on the behaviour required by our objectives. Socket connection between EuQoS Video and MLLFScheduler will be established after the source and the sink agents are instantiated and sink's method *socket_init* is called. This method takes care about initialization of all required interprocess communication support mechanisms. Such techniques and data format conversions are integrated seamlessly into the classes of EuQoS Video framework, so that a scenario could stay basically the same as before new changes in EuQoS Video are employed. No additional functions are required in the OTCL script (simulation scenario) as well.

It must be taken into account that the same simulation processes can be described in the different functions in the different classes. In order to avoid such possible conflicts in fuctional behaviour of simulation processes, all changes in simulation processing are done in EuQoS Video framework classes only. No other new class is introduced beyond EuQoS Video source code. Simulation scenario cares only about simulation environment setup, but extended simulation processing is implemented in the EuQoS Video classes.

Traffic generator will be started from the body of *socket_init* method. Note that *socket_init* method includes short 5 seconds delay that should ensure the Simulator against a case when the Traffic generator is still not ready. This delay may be changed to conform with a slower or faster simulation computer.

Traffic generator

Timing, frame type, and priority information for the MPEG-packets to be transmitted are generated by the programm called MLLFScheduler. This is in Java written application. Its function is to generate traffic. It generates

a set of parameters, representing streams of video packets according some predefined and some real time conditions. It decides constantly in the real-time, which packets are better to send at this time, making it possible to improve the quality of the video transmission.

Simulation scenario

Simulation scenario is small TCL-Script, which communicates with MLLF-Scheduler through a socket connection in the simulation time. Each MPEG-packet, which is either I, P, or B Frame, generated by MLLFScheduler, is processed by the Testbench. New time value, according a new value of the simulation time on the *receiving a packet* event, has to be given back to the traffic generator.

A simulation scenario describes different network clients, connections and traffics. Some relevant parameters are set in the mpeg-stream script. For instance, they can define which one of the different simulation schedulers should be activated for current simulation scenario.

3 Simulation Scenario

The simulation scenario (simulation scheduling) and its processing is briefly described here. As it is already mentioned above, a simulation scenario is to be described in a OTCL script. OTcl is Object-orieneted extension of Tcl scripting language. To run a scenario, a script is to be run with NS2 as its argument in the following way: "ns scenario.tcl".

Each properly described scenario script is considered to have exactly one "Simulator" object, and at least two Agents, one for a Source and one for a Sink. All general events such as start simulation time and end simulation time are to be set as well.

Simple Simulation Example

Script called mpeg-stream.tcl contains simple simulation scenario. Tnis scenario describes the communication between three network nodes A, B, and C. Node A is the source. It generates MPEG stream using MLLFScheduler and transmit it to the B node, which is the sink.

We also want to have a background traffic between some nodes. Because of throughput without background traffic is always higher than throughput with background traffic, we can vary it to observe the packet loss, which

means that an original video picture sometimes will be accepted by an observer as visually disturbed. Background traffic may be established between A and C nodes as required by uncommented relative lines in OTCL scenario script.

4 Running Simulation

In order to run presented Simulation System there must be done some simple steps:

- go to the testbench directory
- adjust run.sh file if it is required
- run run.sh file. Simulation should start. It takes time until simulation will be finished.
- many relevant information will be showed on the end of simulation. If you need full information about results of simulation, you have to check following files:
  - 80211e.trace has very detailed description of what happens in the interiors of the full Simulation System, including important messages from EuQoS Video framework
  - sink.output has information on each packet, departured from the Source Node
  - source.output has information on each packet, arrived to the Sink Node as well as connection speed in bits per second
  - MLLF/logs/MLLF.log consists of the messages from the MLLF-Scheduler

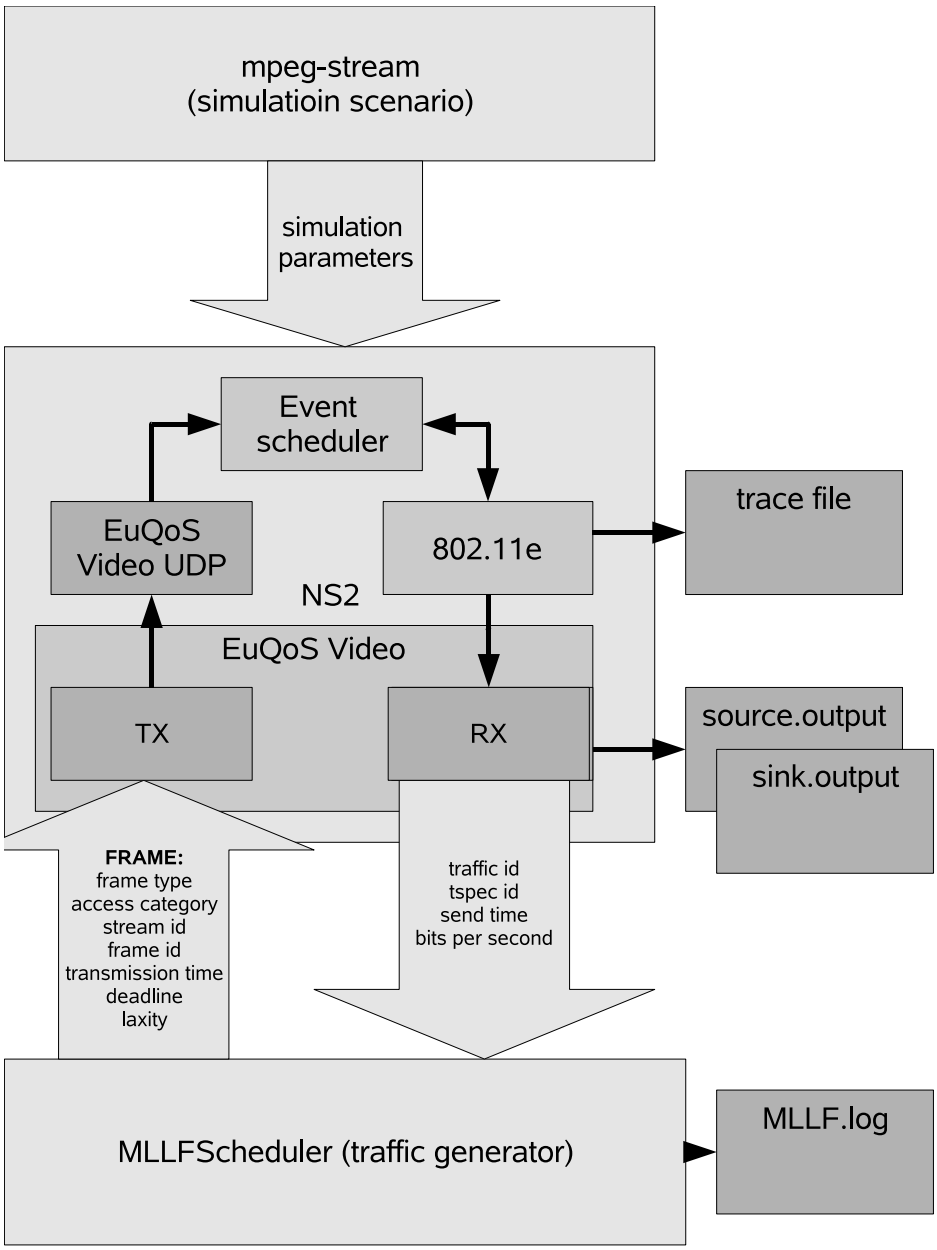
  Alternatively, you can change run.sh so that all needed information will be shown automatically after a simulation is over.

If something goes wrong, please be sure that you have all required software and libraries installed. We have tried to avoid innecessary or depricated software dependencies, but there is no warranty that our system if been fresh installed in any different Operating System Environment will work as it ought to be. You can find all software dependencies required in the Appendix to this documentation.

5 Results

Results of the simulation are available after simulation is finished. They are represented in the form of text files called source.output and sink.output. Difference in the length between them could indicate possible packet loss. Each successfully transmitted packet is provided with detailed information on the time of departure from a source and the time of arrival to a sink, as well as with unique ID, type of the network protocol and the bit length of a packet. Speed of communication, measured in bits per second is also presented on the per-packet base. It means that this speed was calculated at the time of that packet has arrived to the sink agent. A packet, that is not received but was sent, should be considered as been dropped.

A Appendix. Architecture



B Appendix. Dependencies

List of libraries and software programmes which are required to be installed before a simulation started.

- GNU tools for text processing (usually installed in all Linux systems by default)
- Bash interpreter
- Python programming language
- Java virtual machine
- Java compiler (required if MLLFScheduler recompilation is needed)
- GCC C/C++ compiler (required if EuQoS Video, INRIA 802.11e, or NS2 recompilation is needed)