User-level Performance Evaluation of VoIP Using ns-2

A. Bacioccola, C. Cicconetti, G. Stea
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Via Diotisalvi, 2 – 56100 Pisa, ITALY
{a.bacioccola,c.cicconetti,g.stea}@iet.unipi.it

ABSTRACT

Voice over IP (VoIP) is gaining an ever increasing popularity. As such, it nowadays represents one of the most studied test applications in the performance evaluation of wireline and wireless networks. However, a sound performance analysis of VoIP communications should be carried out at the user level, by computing perceptive metrics like the Mean Opinion Score (MOS) or the E-Model. In this paper, we present enhancements to the popular Network Simulator 2 (ns-2) that allow a reliable VoIP user-level performance analysis to be carried out through simulation. We show that computing performance measures at the IP level, which is usually done in ns-2, often leads to inaccurate results. Our code is publicly available at http://info.iet.unipi.it/~cng/ns2voip/.

Keywords

Simulation, ns-2, VoIP, MOS, E-Model, QoS

1. INTRODUCTION

Voice over IP (VoIP) applications are gaining an ever increasing popularity in the Internet community, favored by the massive deployment of wireless access technologies. For instance, more than eighty million users have already subscribed to Skype [31], the most popular VoIP commercial application for personal use, roughly 10% of which are estimated to be simultaneously online at any time. While it is not clear whether VoIP will ultimately replace traditional telephony, its massive diffusion may act as the main driving factor for the actual deployment of Quality of Service (QoS), both in the Internet backbone and in the (wired or wireless) access segments. For this reason, using VoIP as a test case in the performance evaluation of new QoS components, such as (to name a few) scheduling, resource reservation, admission control, traffic policing, traffic engineering, etc., has become a common practice. Unlike classic data applications, in which easily quantifiable, data-related performance metrics (e.g., throughput and mean packet delay) most often represent meaningful evaluations, the actual performance of VoIP applications depends on user perception (a concept often referred to as *Quality of Experience*, *QoE*) [2]. For this reason, the ITU-T has established a computational model, called the Emodel, [15, 16, 17], which defines a quality factor - the so-called *R score* — to capture the effect of mouth-to-ear delay and losses in packet-switched networks. The R score can then be mapped to the *Mean Opinion Score* (MOS) [10], which in turn can be converted into subjective quality levels (e.g. "good", "poor"). Despite this, assessing the VoIP performance through measures taken at the IP level – rather than taking into account the user perception – is often the norm in QoS literature. However, it can be shown that a sound assessment of VoIP quality has to take into account several factors which extend *beyond* the IP level. For instance, playout buffers, which come as part of a VoIP application, play a crucial role [28]: packets that are successfully delivered within a given deadline at the IP level can in fact be delayed or dropped at the playout buffer. On the other hand, playout buffers dampen the jitter, so that evaluating the jitter at the IP level (rather than after the playout buffer) often overestimates it. Let us consider, for instance, the works presented at the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM 2007, [14]), which are focused on the performance analysis of VoIP traffic. Among seven such works, only one [3] both takes playout buffers into account and uses MOS as a metric. Others ([4], [19], and [22]) neglect playout buffers (thus possibly over-estimating the MOS), while a third set ([30], [1], and [32]) only considers IP-level metrics, such as average/90th percentile of the delay, average packet loss and packet inter-arrival time.

The *Network Simulator* (ns-2) [25] is the de-facto standard simulation tool for the networking community, as shown by recent surveys led on the proceedings of important networking conferences [20]. Ns-2 [25] is an open-source simulator, continuously enhanced and extended thanks to the contribution of a large community of researchers. It includes a large number of network protocols, applications, algorithms, devised for wired and wireless networks. A new version of the simulator (known as *ns*-3) is currently being developed at the time of writing [26]. However, ns-2 lacks a sound and flexible simulation model of a VoIP application, as well as routines for processing events so as to evaluate VoIP performance according to the Emodel. The contribution of this paper is therefore a software framework that enhances ns-2, thus allowing a sound simulation and performance evaluation of VoIP applications. More specifically, we developed a VoIP application, which includes support for different codecs, Voice Activity Detection (VAD), aggregation of multiple voice frames into the same IP packet, and support for playout buffers. All the above components are defined in a modular way, so as to make it easy to specialize them to add new functionalities (e.g. to add a new codec or playout buffer algorithm). Furthermore, a set of routines that allow Emodel statistics to be computed directly as an outcome of simulation runs has been designed. Practical applications of the above framework are evaluated, so as to provide evidence, on one hand, that neglecting some of the simulated characteristics of VoIP applications can actually lead to unrealistic simulation scenarios, and, on the other hand, that assessing the performance

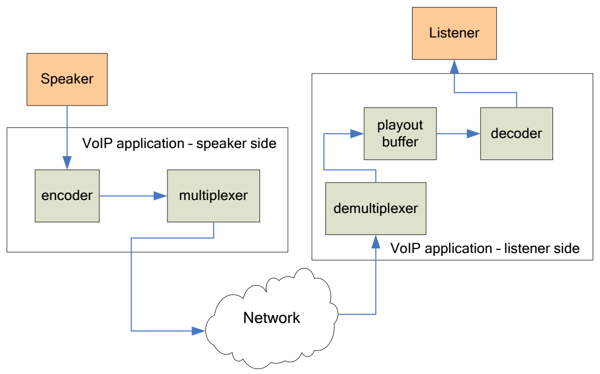

Figure 1. Scheme of a VoIP application

of VoIP without considering the full set of characteristics captured by the Emodel leads to erroneous conclusions.

The rest of the paper is organized as follows: Section 2 describes the necessary background in VoIP applications and performance evaluation, whereas in Section 3 we describe our software framework. Section 4 reports the performance evaluation, and Section 5 concludes the paper.

2. BACKGROUND

In this section we provide the necessary background on the structure of VoIP applications and on their modeling and evaluation.

A VoIP application is composed of several building blocks, as shown in Fig. 1. At the sender side, the first component is an *encoder*, which periodically samples the voice signal. A large variety of encoders are available, representing different trade-off points in the balance between quality and bandwidth consumption. Encoders can be either *sample based* or *frame based*. The former, (e.g., G.711) code individual speech samples periodically The latter, instead, (e.g. G.729) group a certain number of samples within a time window (i.e. a frame) of some milliseconds, and then code them together. For this reason, frame based encoders often achieve higher compression and smaller data rates than sample based ones, though at a higher encoding/decoding complexity. Since, for the scope of this paper, we do not need to distinguish sample-based and frame-based encoders, hereafter we refer to both *samples* and *frames* using the phrase *speech frames* for the sake of readability. The generation of speech frames can either occur at periodic interval, or, more commonly, be modulated by *voice activity detection* (*VAD*). VAD capitalizes on the natural alternation of *talkspurt* and *silence* periods in a single (unidirectional) stream of a bi-directional conversation. During silence periods, either no speech frame is produced at all, or they are produced at a reduced rate and/or using a reduced number of bits, so as to convey some *comfort noise* to the listener. Comfort noise is used by one conversation party to feel the liveliness of the other one. A number of speech frames can be multiplexed into the same packet payload, so as to reduce the overhead of transport, network and MAC headers, though at the expense of increasing the transmission delay. The VoIP payload is typically encapsulated into RTP/UDP/IP packets.

At the receiver side, speech frames are de-multiplexed and inserted into a *playout buffer*. A playout buffer enforces speech frames to be decoded at the same interval at which they were generated by the encoder. To do so, it might re-order, delay or even drop them if they arrive after their expected playback time. Playout buffers can be either *fixed* or *adaptive*. The former assume the network delay to be constant during a conversation, and therefore delay the first packet of a talkspurt of a fixed amount of time. The latter, instead, strive to dynamically adapt the playback point to the changing network conditions, normally on a per-talkspurt basis. The playout buffer delivers speech frames to the decoder, which actually playbacks them. Some decoders implement *packet loss concealment* (*PLC*) techniques (see, e.g. [12]), which allow missing speech frames to be somewhat reconstructed by interpolating (correctly received) surrounding frames. PLC techniques can obviously mask a limited number of losses. However, it is shown in [23] that they effectively reduce the impairment from loss perceived by a listener.

As far as user behavior modeling is concerned, it is common practice in the literature to model a single stream of voice as a two-state process. The sojourn time in the two states, *silence* and *talkspurt,* is usually drawn from exponential [5] or Weibull [7] distributions. In a bi-directional conversation, however, the two conversation streams are not independent of each other. More specifically, the conversation between A and B can be in any of four states: *A speaking*, *mutual silence, B speaking* and *double talk.* In [18], the mean sojourn times and the state transition probabilities are computed, based on analysis of multi-lingual conversations. In [13] authors show that the sojourn times and the state probabilities are somewhat affected by the end-to-end delay, so that at higher delays the occurrence and length of mutual silence periods increases at the expenses of the *A speaking* and *B speaking* periods.

The evaluation of VoIP conversational quality has been the subject of several works. The most widely used evaluation framework for VoIP is the so-called Emodel, standardized by ITU-T [15, 16, 17], which computes a predictive estimation of the subjective quality of the packetized voice from transmission parameters. The output of an E-model computation is a scalar number, called the *R factor*, computed as a function of delays, packet loss, equipment impairment factors, and user quality call expectation as follows:

$$R = R_0 - I_s - I_d - I_{e,eff} + A\,, \qquad (1)$$

where R_0 is the basic signal-to-noise ratio (received speech level relative to circuit and acoustic noise), I_s accounts for the impairments which occur with the voice signal, I_d sums all impairments due to delay and echo effects, $I_{e,eff}$ is the effective equipment impairment factor, taking into account the codec and its tolerance to random packet losses. Furthermore, A is a "bonus" factor that models the user expectation of the technology employed. For instance, the A value is greater in satellite networks than in classical circuit-switched networks, because user expectations in satellite networks are lower than those in wired networks. The typical range for the A factor is [0, 20] and the example values proposed by the ITU are reported in Table 1.

Table 1. *A* factor values proposed by the ITU.

Communication System	*A* factor
Wired phone	0
Cellular in building	5
Cellular in moving vehicle	10
Access to hard-to-reach geographical zones (many satellite hops)	20

Finally, $I_{e,eff}$ can be computed as:

$$I_{e,eff} = I_e + (95 - I_e) \cdot P_{pl} \big/ \left(P_{pl} + B_{pl}\right),$$

where I_e is the equipment impairment factor and is used to characterize the behaviors of the codec with a low bit rate, P_{pl} is the packet loss probability and B_{pl} is the codec packet loss robustness factor. Once an R factor is obtained, it can be directly mapped to an estimated MOS. The value of the above parameters for the most used codecs can be found in [17]. The above model is used, for instance, in [23] to assess the performance of the current Internet backbone as a VoIP infrastructure.

In [27], the authors propose a QoS control scheme to adapt the rate of the GSM AMR codec so as to adapt the sending rate to the current network conditions. Adaptation is based on the MOS measured by the VoIP decoder, which reports feedback information to the VoIP encoder via Real-Time Control Protocol (RTCP) messages. The proposed scheme was evaluated by means of the ns-2.1b9a simulator. In this work, instead, we focus specifically on the performance evaluation of the quality of VoIP calls, and provide a publicly available software framework for the latest ns-2 version (ns-2.31). Our contribution will thus benefit researchers, who will be able to evaluate the performance of their proposed solutions, such as that in [27], without the burden of writing and testing the code to enhance the ns-2 simulator and collect MOS-based performance measures.

Finally, a recent work [6] evaluates VoIP user satisfaction by computing a User Satisfaction Index (USI) based on an analysis of the call duration. While this technique allows VoIP calls to be analyzed relying only on passive measurement, i.e. without involving user surveys, it is of no use in a simulation environment, in which calls are artificially generated.

3. DESCRIPTION OF THE FRAMEWORK

In this section we describe the simulation model of a VoIP application and its implementation in ns-2. We model all the relevant features of a VoIP application described in the previous section. More specifically, we model the *sender* and the *receiver* side separately. The sender side includes:

- a customizable codec, which generates generic *speech frames* (the latter being either voice samples of voice frames, depending on the codec);
- a multiplexer, which aggregates several speech frames into one payload.

The receiver side, instead, includes

- a customizable *playout buffer* algorithm;
- a demultiplexer.

Moreover, user activity is modeled as a series of *talkspurt/silence* commands given to the sender. This allows for easy integration of one-way and two-way (coordinated) user activity models. Finally, as far as data modeling is concerned, we model a speech frame by storing its size, its generation timestamp, the talkspurt the frame is associated to, and its position within the talkspurt.

The ns-2 implementation of the above model includes several inter-operable modules, which are described separately in the following. The big picture illustrating all the functional modules is depicted in Fig. 2.

First of all, in order to implement speech frames, we cannot simply extend the ns-2 *packet* C++ class. The latter, despite their name suggests network-layer transmission units only, are used in ns-2 in a broader sense, which includes upper-layer units, e.g. TCP segments, and lower-layer units, e.g. MAC frames. The actual layer to which an ns-2 packet belongs depends on the packet type, which is thus used for encapsulation/decapsulation. However, packets are not flexible enough to represent speech frames too, since there is not an easy way to combine them into chunks, which is required for frame multiplexing and demultiplexing. Therefore, we created a specific data structure, called `VoipFrame`, with the following fields:

- `talkspurt`: sequential numerical talkspurt identifier;
- `nframes`: number of frames in the current talkspurt;
- `frame`: sequential numerical identifier of the frame within the talkspurt where it was generated;
- `timestamp`: time when this frame was generated;
- `size_`: frame size, in bytes.

Therefore, VoIP frames are simulated by means of empty packets, as is almost always the case in ns-2, which thus do not contain the actual data that would have been produced by a real VoIP encoder. While we believe that this assumption is adequate for most studies of the performance VoIP traffic in communication networks, feeding simulations with real-time speech data produced by VoIP applications is being considered as a future work.

VoIP frames are packed into a specific container, called `VoipPayload`, which is simply a list of `VoipFrame` objects. A `VoipPayload` is conveyed by an ns-2 packet through the `AppData` pointer. The latter is specifically designed to enable end-to-end applications to be developed in ns-2. In fact, network modules, such as agents and nodes, are supposed not to mess with encapsulated data, which is only recovered by the destination application through the `process_data()` function. This feature is optional and is seldom used in network simulation with ns-2, which are instead usually focused on the performance evaluation from the network perspective. An example of end-to-end application that is distributed together with the official set of modules is *web caching*, which implements basic HTTP functions to simulate user caches.

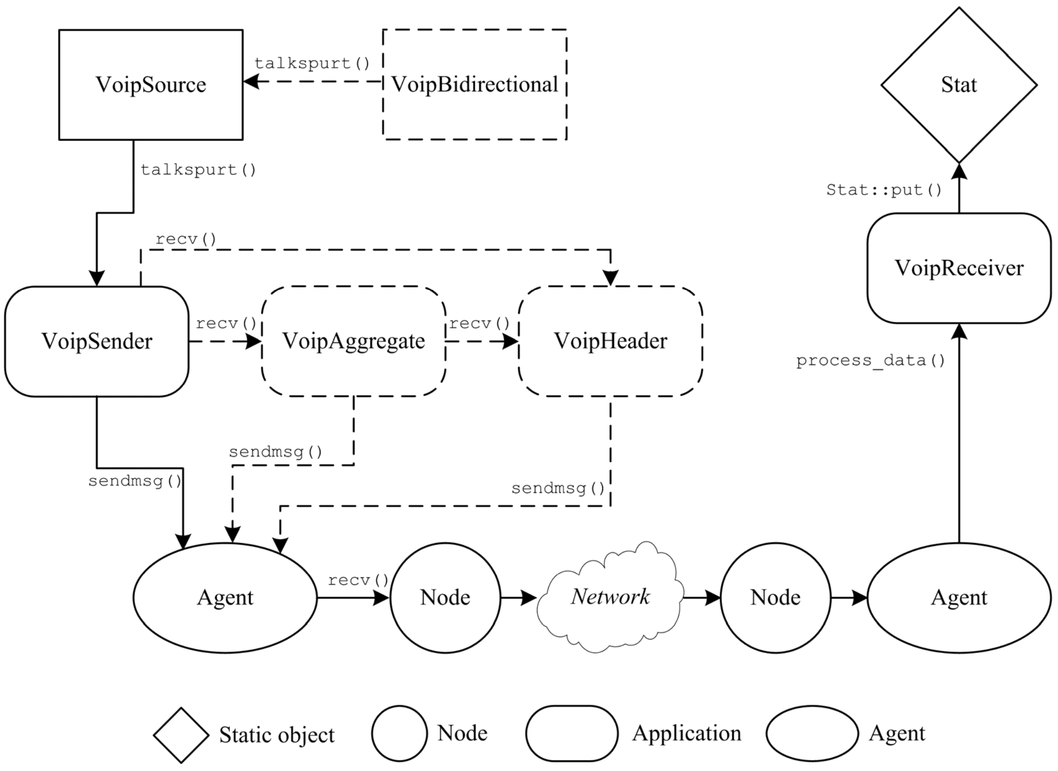

Figure 2. Modules for VoIP simulation in ns-2. Dashed objects are optional..

In the following we describe in detail each module depicted in Fig. 2. First, user speaking is modeled by means of alternating talkspurt and silence periods, which are generated by the `VoipSource` module. Specifically, the base version of `VoipSource` can be configured to draw the duration of the talkspurt and silence periods from uncorrelated exponential [5] or Weibull [7] distributions. In particular, the latter are pre-configured to fit the following voice application scenario: one-to-one conversation, multiple partners conference call, lecturer/audience speaking. At the beginning of each talkspurt, the `VoipSource` module calls the `talkspurt()` function of the underlying `VoipSender` object, also specifying the duration of the forthcoming talkspurt and silence periods. When a new talkspurt is triggered by `VoipSource`, the `VoipSender` object starts generating a sequence of `VoipFrame` items, whose size and generation rate depend on the simulated codec. The most common codecs employed in network simulation (e.g. G.711, GSM.AMR) are supported by the `VoipSender`, while others can be easily added without requiring full knowledge of the implementation details of the VoIP modules.

Depending on whether frame multiplexing and/or header inclusion are enabled or not, the recipient of the generated VoIP frames will be different. If multiplexing is enabled, a `VoipAggregate` object is created and bound to the `VoipSender`, which calls the `recv()` function at each frame generation to pass the frame to the multiplexer object. The `VoipAggregate` waits until a specified number of frames are received before packing them together into a `VoipPayload` object and sending them to the `VoipHeader` (if enabled) or to the transport layer, i.e. the agent, in ns-2 terminology. Note that a talkspurt may produce a number of frames that is not a multiple of the payload size, in which case the last ns-2 packet will contain fewer frames than the others. On the other hand, if there is no multiplexing of frames into a single IP datagram, the `VoipSender` object directly generates a `VoipPayload` item for each VoIP frame, which is sent to the `VoipHeader`, if enabled. Otherwise, the `VoipPayload` is sent to the agent straight away. The `VoipHeader` receives a `VoipPayload` object from the `VoipSender` or the `VoipAggregate`, depending on whether multiplexing is enabled. The `VoipHeader` adds the RTP/UDP/IP headers to the `VoipPayload` and supports header compression. After the header has been added to the `VoipPayload`, the payload is sent to the agent. This explains why `VoipSender`, `VoipAggregate`, and `VoipHeader` are implemented as ns-2 applications.

The sending agent then encapsulates the `VoipPayload` object into the ns-2 packet, which traverses the simulated network without the VoIP fields ever being accessed, possibly experiencing delay, loss, re-ordering.

Eventually, ns-2 packets that are not lost arrive on the receiving agent, which passes the encapsulated data to the `VoipReceiver` application. The latter retrieves all the `VoipFrame` objects listed into the `VoipPayload` and performs playout buffering. Three different buffering policies are implemented in the current version of our VoIP package, although more can be seamlessly added: *no buffering*, to be used as a reference

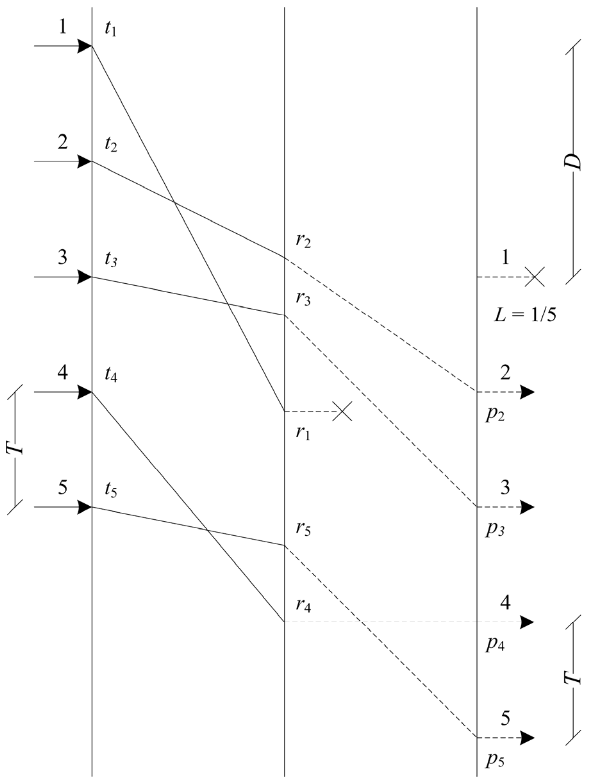

Figure 3. VoIP frame re-ordering due to (non-causal) playout buffering. Solid lines represent network delays, dashed lines playout buffer delays.

for test only, *static buffering*, and *optimal buffering*. Playout buffers are described in a separate sub-section below. Both static and optimal buffering output VoIP frames at a constant rate, which is suitable for the R score to be computed correctly. The metrics evaluated within the `VoipReceiver` follow:

- *MOS*, as the average of per-talkspurt MOS values, and as the aggregate of per-conversation MOS value;
- *VoIP frame delay*, defined as the interval between the time when VoIP frame was generated by the encoder and the time when the frame is played out; this closely approximates the mouth-to-ear delay;
- *frame loss*, defined as the ratio between the number of frames that have not been played out and the number of frames that have been generated by the encoder;
- *cell outage*, defined as the ratio of users that are not satisfied with the call quality, depending on a configurable MOS threshold.

All the above metrics except the last one are collected on a per VoIP flow basis by `VoipReceiver` objects, each identified by means of a numerical identifier set via Tcl by the users. The *cell outage*, instead, is computed on a set of VoIP flows, which are assumed to be established in the same cell or network domain. Therefore, the user is requested to select which VoIP flows belong to any set, by assigning the same numerical cell identifier.

Measures are collected by the *Stat* module, which has been first described in [13], whose current version, bundled with the VoIP patch, includes several functional and performance enhancements.

3.1 Playout Buffers

As far as playout buffers are concerned, we implemented two different algorithms: a simple *static* playout buffer, and a, non-causal *optimal* playback algorithm. The first one simply delays frames of a selectable but fixed amount of time before delivering them to the decoder. The second one capitalizes on the fact that, in a simulated environment, non-casual behavior can actually be implemented to achieve optimal results. More specifically, the optimal playout buffer waits for the whole set of frames of a given talkspurt to arrive at the receiver, and then it selects the playback delay – based on the pattern of arrivals – according to which the best possible speech quality would have been achieved. The purpose of including such an optimal playout buffer in the simulation framework is twofold: on one hand, it acts as a benchmark for possible *causal* playout buffer algorithms. On the other hand, it will be used to show that computing the MOS *without* including a playout buffer yields even *better* results than those obtained using such an ideal playout buffer scheme. Hereafter, we describe its implementation in more detail.

Let us consider the example illustrated in Fig. 3, which shows the transmission, buffering and playing phases of a talkspurt with five VoIP frames. Let t_i, r_i, p_i be the time when the i-th frame of a talkspurt is generated by the encoder, received by the playout buffer, and passed to the decoder respectively. Now, $p_i - p_j = t_i - t_j$, for each i, j that are actually played. We define the *network delay* of frame i as $d_i = r_i - t_i$, and the *playout delay* D as the interval between when first frame was generated by the encoder and when the playout buffer begins passing frames to the decoder (see Fig. 3). Without loss of generality, we assume that $r_i = \infty$ for each frame i that is lost due to the network. The playout buffer discards all frames that are received too late, i.e. such that $d_i > D$, which implies that all frames with $d_i = r_i = \infty$ are discarded, regardless of the value of D. Frames that are discarded by the playout buffer contribute to the loss rate L, which is defined as the ratio between the number of discarded frames and the number of frames in the talkspurt. In the example of Fig. 3, $L = 1/5$ in because all frames but the first one are received "on time".

Therefore, the only degree of freedom of the optimal playout buffer is the playout delay D. We define the *optimal playout delay* $D_{opt} \in \{d_i\}$ as the value of the playout delay that maximizes the MOS. As already pointed out, the actual formula to compute the latter depends on the specific codec used. However, by definition, the MOS is always obtained via a non-increasing function of D and L. Furthermore, L is itself a non-increasing function of D. Therefore, there exists an optimal value D_{opt} that achieves the highest MOS for the talkspurt, which can be computed through a logarithmic search in the sorted set of the network delays in a talkspurt $\{d_i\}$. Such a playout buffer algorithm is *adaptive*, since it sets the playback point on a per-talkspurt basis. Furthermore, it is obviously *non-causal*, since it requires the delay of all speech frames to be known before selecting the playback point. Finally, our optimal playout buffer does not consider the phenomenon of *collision* [24]: it can happen that, because of the optimal choice of *D*, two (or more) talkspurt periods partially overlap in time. Clearly, this situation never arises in practice, since the decoder can reproduce one talkspurt at a time. Instead, we collect measures as if it were actually possible to playback different talkspurts in parallel. This implies that the MOS that is obtained using the optimal playout buffer may not be achievable with a causal playback algorithm, no matter how effective.

Furthermore, we have implemented a simple static buffer with a fixed configurable length. When a frame is received the static buffer checks if the frame can be accommodated or if it should be dropped. An incoming frame is not added to the buffer if its playout time has already elapsed. The buffer implements the following dropping policy: when the buffer is full the first frame to be dropped is the *next* frame to be played out. In other words, the buffer discards the oldest frames, keeping the more recent ones. As regards the playout policy, the buffer delays the first frame of a talkspurt by a fixed, though configurable, amount of time. When the above initial playout delay is elapsed a new frame is played out every sample interval until the talkspurt ends. Finally, the buffer possibly reorders the received frames so that they are played out in the order in which they were generated by the encoder.

3.2 Interface

We now briefly describe how to use the above described framework. The source code is documented inline and can be converted into a stand-alone manual through the Doxygen [11] tool. Additionally, sample Tcl scripts are provided in bundle with the VoIP package at [33].

All the `Voip*` modules can be created both via C++ and Tcl, and are bound together in the standard ns-2 way, i.e. by means of the Tcl/C++ `command()` interface function. For instance, the following Tcl code creates a `VoipSource`, a `VoipSender` and a UDP agent, and connects them:

```
set src [new VoipSource]
set snd [new Application/VoipSender]
set agt [new Agent/UDP]
$src encoder $snd
$snd attach-agent $agt
```

The commands required to connect objects follow a standard syntax and are not reported here. The interested reader can found the whole manual at [33]. Hereafter, we show instead the configuration commands.

<u>VoipSource</u>

```
$voip_source start|stop
```
Starts/stops the generation of talkspurts.

```
$voip_source model
      exponential $on $off |
      one-to-one |
      one-to-many |
      many-to-one |
      many-to-many
```
Enables a specified model to generate talkspurt and silence periods. The 'exponential' model requires the average talkspurt and silence durations to be specified, while the others are based on Weibull distributions, as specified in [7].

<u>VoipSender</u>

```
$voip_snd codec G.711 | G.729.A |
    GSM.AMR | G.723.1 | GSM.EFR
```
Selects the codec to be used to encoder VoIP frames.

<u>VoipAggregate</u>

```
$voip_aggr nframes $n
```
Configures the aggregate object so that all ns-2 packets contain `$n` VoIP frames, possibly except the last VoIP payload of the talkspurt.

<u>VoipReceiver</u>

```
$voip_rcv id $ID
```
Associates the numerical identifier `$ID` to the VoIP receiver in order to distinguish a specific VoIP traffic flow among others in the output statistics.

```
$voip_rsv cell-id $cell
```
It is used to associate the VoIP decoder to a specific cell, so as to be able to gather cell-wide cell outage metrics.

```
$voip_rsv emodel $ie $bpl $a $ro $th
```
Configure the E-Model based on its parameters, described in Sec. 2, i.e. equipment impairment factor (`$ie`), packet loss robustness factor (`$bpl`), expectation factor (`$a`), transmission rating factor (`$ro`), respectively. Plus, the cell outage threshold is defined (`$th`).

```
$voip_rcv emodel G.711 | G.729 | GSM.AMR
```
Configure the E-Model equipment impairment factor (`$ie`) and packet loss robustness factor (`$bpl`), based on the codec used by the transmitter. The above value has been obtained according to [17].

Furthermore, only for the VoIP receiver with static playout buffering policy, the following parameters can be specified.

```
$voip_rcv buffer-size $n
```
The maximum number of VoIP frames that the buffer can accommodate.

```
$voip_rcv initial-delay $delay
```
The delay, introduced by the buffer, related to the first frame of the talkspurt.

```
$voip_rcv playout-rate $time
```
The time, in ms, between the generation of two frames at the transmitter. This value depends on the codec being used and will be used by the buffer to playout the samples with the same rate as they were generated by the encoder.

<u>VoipHeader</u>

```
$voip_compression $size | nocompression
```
Set the IP/UDP/RTP header size. If not specified, the default value is `nocompression`, which results in a header size of 40 bytes (= 20 bytes/8 bytes/12 bytes, respectively for standard IP/UDP/RTP headers). Otherwise, if the

`compression` command is given, the compressed header size is specified by the parameter `$size`.

3.3 Requirements

We complete this section by describing the computational requirements of the proposed framework. With regard to the VoIP sender modules, the overhead incurred by using the proposed framework is negligible. As a matter of fact, the only module that requires additional storage is `VoipAggregate`, which is expected to keep a small number of VoIP frames, typically between 2 and 5. All the other modules only perform simple computations, e.g. creating a `VoipFrame`, changing the IP packet size, drawing random numbers, thus not impairing the capability of ns-2 to manage large network simulations.

On the receiver side, instead, the computational overhead highly depends on the algorithm that is employed to play out VoIP frames at a constant rate. Specifically, with *static* buffering, each `VoipReceiver` keeps at most a configurable number of VoIP frames (`buffer-size` Tcl command), which is expected to be in the order of tens, plus the state variables to keep updated the average VoIP frame delay and loss of the current talkspurt. The latter are used at the end of each talkspurt to derive the MOS based on the *R* factor computed according to (1). As far as the *optimal* playout algorithm is concerned, both temporal and spatial overhead may significantly increase with respect to the *static* playout buffer. In fact, for each talkspurt, the value of the playout delay that maximizes the MOS is to be selected. This can be done through a logarithmic search among the delay samples collected for that talkspurt, where each step requires the computation of the average VoIP frame delay and loss. The delay samples of *all* the VoIP frames within a talkspurt have thus to be stored at the VoIP receiver and processed to derive the optimal playout delay and related MOS of the talkspurt. This requires $\mathcal{O}(n \log_2(n))$ operations in the worst case, *n* being the number of VoIP frames in a talkspurt. Therefore, we argue that, even with *optimal* playout buffering, the overhead, both in time and space, can hardly become a bottleneck, unless a very large number of VoIP applications are simulated simultaneously.

4. PERFORMANCE EVALUATION

In this section we present simulation results, which are meant as a proof of concept of how the contributed simulation framework can be exploited for a sound and simple performance evaluation of VoIP applications in ns-2. We therefore purposefully set up a very simple networking environment, which only consists of only one VoIP flow established between two ns-2 nodes connected through an 8 Mb/s wired link with a constant 10 ms propagation latency. The VoIP sender employs a GSM AMR codec at 12.2 kb/s, with one-to-one VAD model, without header compression. Frame aggregation is not used, unless specified otherwise.

In order to be able to vary the delay through the network and to introduce losses, we filter the ns-2 packets generated by the UDP sender through an ad hoc connector object, which is able to:

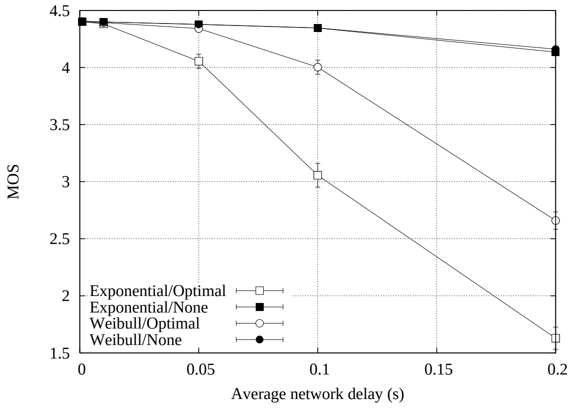

Figure 4. MOS vs. average network delay, PLR = 0.

- drop packets at a specified *packet loss rate* (*PLR*) based on a uniform random variable;
- delay packets according to a configurable delay distribution; in the following we report results obtained with exponential and Weibull (*shape* = 2) random variables. We call *average network delay* the mean value of the distribution.

The simulation duration is set to 100 s, with a 10 s warm-up period during which statistics are not collected. The simulation experiments have been carried out using the method of independent replications [21], using the framework for statistics gathering in ns-2 described in [9]. Confidence intervals are not reported in figures whenever negligible. In this analysis, we only consider the MOS.

We begin by comparing the performance obtained with the optimal playout buffer (‘*Optimal*’ in the figures) and without playout buffer (‘*None*’ in the figures). In Fig. 4 we show the MOS when the average network delay increases from 1 ms to 200 ms, and packets are never discarded at the tagger, i.e. *PLR* = 0. Nonetheless, packets may be discarded by the playout buffer, when present, if they arrive after their playback time. Results are shown when packets are delayed according to both an exponential and a Weibull distribution. As can be seen, without playout buffer, the MOS always lies above 4, i.e. “high-to-best” quality perceived by VoIP users according to the scale in [15]. More specifically, the MOS does not depend on the packets delay distribution employed, and it is only slightly affected by the average network delay. However, such a result is utterly misleading since the upper bound on the performance achievable with *any* playout buffer, measured by means of our optimal non-causal playout policy, is significantly smaller than that. This becomes more evident as the delay introduced by the network gets higher. Furthermore, the MOS instead heavily depends on how the network delays are distributed. For instance, with an exponential distribution with an average equal to 100 ms, nearly all users would be dissatisfied according to the E-model considered.

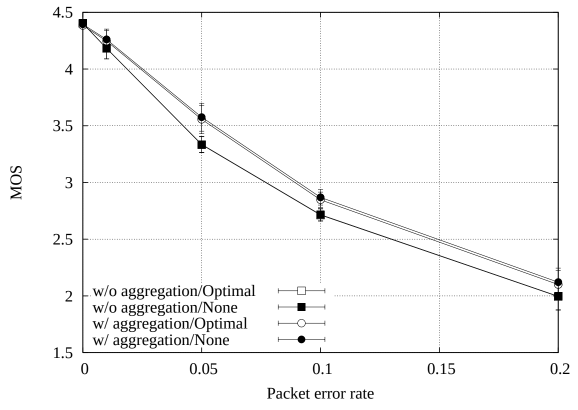

Figure 5. MOS vs. PLR, avg. network delay = 1 ms.

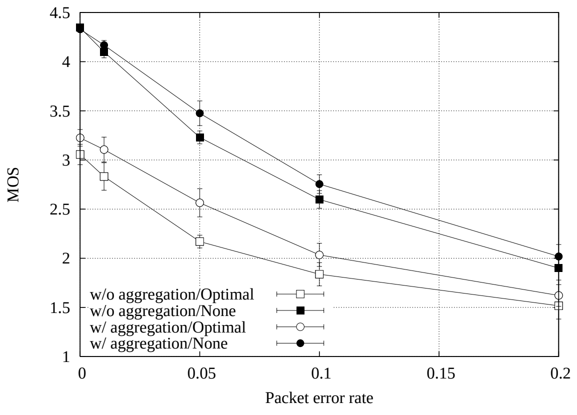

Figure 6. MOS vs. PLR, avg. network delay = 100 ms.

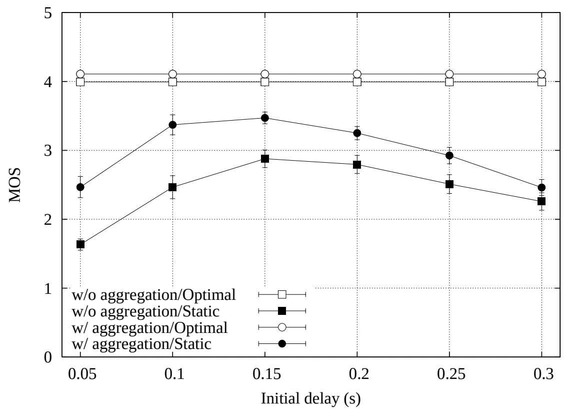

Figure 7. MOS vs. initial playout delay (ms), average network delay = 30 ms, PLR = 0.01.

In Fig. 5 we assume that network delays are drawn from an exponential distribution, with an average equal to 1 ms, and we instead vary the PLR (in the graph the curves with solid symbols are almost overlapping with those with empty symbols, thus they cannot be distinguished easily). Two different aggregation policies are considered here, i.e. one frame per payload ('w/o' in figures) and three frames per payload ('w/' in figures). In this specific scenario, aggregation seems to play a negligible role in the overall performance. Furthermore, employing an optimal playout buffer exhibits approximately the same performance as neglecting the playout buffer, which is apparently in contrast with the results obtained above when varying the average network delay. However, if the average network delay is increased, e.g. to 100 ms, as shown in Fig. 6, the results are radically different. On one hand, the performance evaluated without playout buffers appears to be largely over-estimated, even with respect to a theoretical optimum. Furthermore, aggregating VoIP frames (up to a certain extent, as shown below) actually increases the MOS. This counter-intuitive result can be explained as follows. On the one hand, aggregating frames introduces an additional delay at the sender side. On the other hand, all VoIP frames aggregated into the same payload will experience the very same network delay, i.e. the least possible delay variation. This last effect entails a gain in terms of MOS, which counterbalances the minor performance decrease due to the (small) additional delay at the source. In fact, playout buffers are very sensitive to frame re-ordering, which happens less frequently when VoIP frames are aggregated.

We now evaluate the performance of static playout buffering. As discussed in Sec. 3.1, the latter needs be configured in terms of *buffer size* and *initial delay*. In Fig. 7 we show the results obtained by varying the latter, with the former set to a fixed value of 20 VoIP frames. Results with optimal playout buffering are reported as a reference. As can be seen, the MOS curves reach a maximum, for an optimal initial delay. In fact, increasing the initial delay first reduces the probability that packets are discarded by the playout buffer. However, after some value of the initial delay, the playout delay becomes the limiting factor of the VoIP performance in terms of MOS.

To conclude the analysis we show in Fig. 8 the MOS obtained with all the playout buffer policies that we have implemented, with an increasing number of VoIP frames aggregated into the same payload. The static playout buffer is configured with a buffer of 20 VoIP frames and an initial delay of 150 ms. The results presented above have shown that aggregation entails a performance gain. However, the delay due to the aggregation increases with the number of VoIP frames packed into the same payload, which eventually leads to a performance degradation. In the scenario evaluated, the best performance, in terms of MOS, is obtained aggregating 5 VoIP frames into the same payload, with both the optimal and the static playout buffer. Once again, note that the results obtained without employing a playout buffer not only largely over-estimates the call quality, but are also misleading about the *qualitative* system behavior. For instance, the 'none' curve in Fig. 8 is almost constant irrespective of the frame aggregation, which is not true with the 'optimal' and 'static' policies.

5. CONCLUSIONS

In this paper we presented extensions to ns-2 that allow a sound and efficient simulation of VoIP applications. More specifically, we implemented a flexible and extensible VoIP application, with support to various codecs and playout buffer algorithms, allowing for voice activity detection and for correlated streams in a bi-directional conversation. Each module can be easily extended to support specific algorithms. Furthermore, we devised and implemented an optimal, non causal playout buffer algorithm, to be used as a reference for performance evaluation of VoIP applications. Finally, we implemented – within the standard framework for statistics gathering in ns-2 described in [9] – the computation of meaningful user metrics, such as the Emodel and the Mean Opinion Score (MOS).

We showed through simulation that the performance of a VoIP stream, in terms of the MOS, may heavily depend on the playout buffer employed. In particular, if playout buffers are neglected, which is often the case in the simulation studies found in the literature, the performance of VoIP streams is largely overestimated. Furthermore, we observed that aggregating frames into a single packet payload can increase the MOS.

Future work includes developing similar extensions to ns-2 for other applications involving user perception, such as video conference and video streaming.

6. ACKNOWLEDGMENTS

This work has been carried out in the framework of the *NADIR* project financed by the Italian Ministry of University and Research.

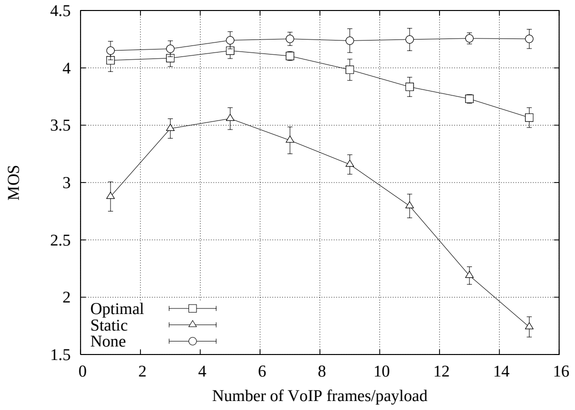

Figure 8. MOS vs. aggregation level, average network delay = 30 ms, PLR = 0.01

7. REFERENCES

[1] G. Athanasiou, T. Korakis, O. Ercetin, and L. Tassiulas. Dynamic cross-layer association in 802.11-based mesh networks. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[2] L. Atzori, and M. L. Lobina. Playout buffering in ip telephony: a survey discussing problems and approaches. *IEEE Commun. Surveys Tut.*, vol. 8, no. 3, 3rd Qtr. 2006, pp. 36–46.

[3] H. V. Balan, L. Eggert, S. Niccolini, and M. Brunner. An experimental evaluation of voice quality over the Datagram Congestion Control Protocol. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[4] R. Birke, M. Mellia, and M. Petracca. Understanding VoIP from backbone measurements. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[5] P.T. Brady. A model for generating on-off speech patterns in two-way conversation. *Bell System Technical J.*, vol. 48, Sep. 1969, pp. 2445–2472.

[6] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying Skype User Satisfaction. Proc. *ACM SIGCOMM 2006*, Pisa, Italy, Sep. 11–15, 2006, pp. 399–410.

[7] C.-N. Chuah, and R. H. Katz. Characterizing packet audio streams from Internet multimedia applications. Proc. *IEEE ICC 2002*, New York, USA, Apr. 28–May 2, 2002, pp. 1199–2203.

[8] C. Cicconetti, M. L. Garcia-Osma, X. Masip, J. Sá Silva, G. Santoro, G. Stea, and H. Tarasiuk. Simulation model for end-to-end QoS across heterogeneous networks. Proc. *IPS-MoMe 2005*, Warsaw, Poland, Mar. 14–15, 2005.

[9] C. Cicconetti, E. Mingozzi, and G. Stea. An integrated framework for enabling effective data collection and statistical analysis with ns2. Proc. *WNS2 2006*, 10 Oct. 2006, Pisa, Italy.

[10] R. G. Cole, and J. H. Rosenbluth. Voice over IP performance monitoring. *ACM CCR*, vol. 31, no. 2, 2001, pp. 9–24.

[11] http://www.doxygen.org/

[12] E. Gündüzhan, and K. Momtahan. A linear prediction based packet loss concealment algorithm for PCM coded speech, *IEEE Trans. Speech Audio Process.*, vol. 9, no. 8, Nov. 2001.

[13] F. Hammer, P. Reichl, and A. Raake. The well tempered conversation: interactivity, delay and perceptual VoIP quality. *Proc. ICC 2005*, Seoul, Korea, May 16–20, 2005, pp.244–249.

[14] http://www.ieee-INFOCOM.org/2007/

[15] ITU-T Recommendation G.107. The Emodel, a computational model for use in transmission planning. Dec. 1998.

[16] ITU-T Recommendation G.108. Application of the Emodel: A planning guide. Sep. 1998.

[17] ITU-T Recommendation G.113. Transmission impairments due to speech processing. Feb. 2001.

[18] ITU-T recommendation P.59, Telephone Transmission Quality Objective Measuring Apparatus – Artificial Conversational Speech, Mar. 1993.

[19] A. Kashyap, S. Ganguly, S. R. Das, and S. Banerjee. VoIP on wireless meshes: Models, algorithms and evaluation. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[20] S. Kurkowski, T. Camp, and M. Colagrosso. MANET simulation studies: The incredibles. Proc. *ACM MC^{2}R*, vol. 9, no. 4, Oct. 2005, pp. 50–61.

[21] A. M. Law, and W. D. Kelton. Simulation modeling and analysis. Third edition, *McGraw-Hill*, 2000.

[22] Y. Li, M. Chiang, A. R. Calderbank, and S. N. Diggavi. Optimal rate-reliability-delay tradeoff in networks with composite links. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[23] A.P. Markopoulou, F. A. Tobagi, and M. J. Karam. Assessment of VoIP quality over Internet backbones. Proc. *IEEE INFOCOM* 2002, New York, USA, June 23–27, pp. 150–159.

[24] S. B. Moon, J. Kurose, and D. Towsley. Packet audio playout delay adjustment: Performance bounds and algorithms. *Multimedia Systems*, vol. 6, no. 1, 1998, pp. 17–28.

[25] http://nsnam.isi.edu/nsnam/

[26] http://www.nsnam.org/

[27] Z. Qiao, L. Sun, N. Heilemann, and E. Ifeachor. A new method for VoIP quality of service control use combined adaptive sender rate and priority marking. *IEEE ICC 2004*, Paris, France, June 20–24, pp. 1473–1477.

[28] R. Ramjee, J. F. Kurose, D. F. Towsley, and H. Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. Proc. *IEEE INFOCOM 1994*, Toronto, Canada, pp. 680–688.

[29] A. Rix, J. Beerends, M. Holler, and A. Hekstra. Perceptual Evaluation of speech quality (PESQ) – a new method for speech quality assessment of telephone networks and codecs. Proc. *IEEE ICASSP 2001*, May 7–11, 2001, pp. 73–76.

[30] S. Shin, and H. Schulzrinne. Experimental measurement of the capacity for VoIP traffic in IEEE 802.11 WLANs. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[31] Skype, http://www.skype.com, continuously updated.

[32] H. Wu, K. Tan, Y. Zhang, and Q. Zhang. Proactive scan: Fast handoff with smart triggers for 802.11 wireless LAN. Proc. *IEEE INFOCOM 2007*, Anchorage, USA, May 6–12.

[33] http://info.iet.unipi.it/~cng/ns2voip/.