Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel

Version 2.2, September 2010

S. Salsano[(1)], F. Ludovici[(1)], A. Ordine[(1))]

[(1)]University of Rome “Tor Vergata”

Available at: ***http://netgroup.uniroma2.it/NetemCLG***

Contact: stefano.salsano@uniroma2.it

TABLE OF CONTENTS

1 Introduction

netem is a module of Linux kernel that provides Network Emulation functionality for testing protocols by emulating the properties of wide area networks. The current version emulates variable delay, loss, duplication and re-ordering.

It has already been pointed out (see for example [9]) that the generation of loss with "correlation" is not working properly in the current version of Netem. Starting from this consideration, in this work we have:

- provided evidence of the weakness of current implementation
- introduced a new loss model, which integrates and complements several loss models available in the literaure
- implemented the loss generation according to the proposed model within a new module called sch_netem2
- added the sch_netem2 as a new "qdisc" within tc

As of now, we have realized a Netem2 system that can completely replace the current Netem, and we provide the relevant patches to be applied to linux kernel 2.6.29. It may be better to simply add the proposed enhancements to the Netem, by releasing a new version of Netem. Therefore, we have submitted our work to the Netem mailing list for discussion.

2 The problem: weakness of current Netem loss generator

In this Section we present a few examples of how the loss generator implemented in Netem works, to prove the noticeable diseases that have driven us to rewrite it.

In the first test, we introduce a 2% loss in packets coming out from *eth0* interface, through the command:

```
root@darkstar:/# tc qdisc add dev eth0 root netem loss 2
```

Trying a ping on the local network and transmitting 10000 packets, we see the following output. The packet loss, as we expected, is about 2%.

```
root@darkstar:/# ping -i 0.0001 -c 10000 -q 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
---192.168.1.1 ping statistics ---
10000 packets transmitted, 9798 received, 2% packet loss, time 20754ms
rtt min/avg/max/mdev = 1.353/1.802/5.967/0.471 ms, ipg/ewma 2.075/1.665 ms
```

Now we introduce a 10% correlation with still the same loss probability. In this case we have a very bad performance: the loss probability of the generated sequence is quite null.

```
root@darkstar:/# tc qdisc change dev wlan0 root netem loss 2 10
```

```
root@darkstar:/# ping #i 0.0001 #c 10000 #q 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
### 192.168.1.1 ping statistics ###
10000 packets transmitted, 9984 received, 0% packet loss, time 18916ms
rtt min/avg/max/mdev = 1.366/1.850/46.033/0.878 ms, pipe 4, ipg/ewma 1.891
```

If we use a higher value of correlation, the result are still worse with only 6 packet lost in a sequence of 10000.

```
root@darkstar:/# tc qdisc change dev wlan0 root netem loss 2 30

root@darkstar:/# ping #i 0.0001 #c 10000 #q 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
### 192.168.1.1 ping statistics ###
10000 packets transmitted, 9994 received, 0% packet loss, time 18430ms
rtt min/avg/max/mdev = 1.363/1.804/6.726/0.481 ms, ipg/ewma 1.843/1.769 ms
```

The decay of packet losses with correlation seems to be lower if we consider higher value of loss in input, for example 10%. Without correlation, the performances are still good:

```
root@darkstar:/# tc qdisc add dev wlan0 root netem loss 10

root@darkstar:/# ping #i 0.0001 #c 10000 #q 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
### 192.168.1.1 ping statistics ###
10000 packets transmitted, 8980 received, 10% packet loss, time 30197ms
rtt min/avg/max/mdev = 1.356/1.828/8.308/0.509 ms, ipg/ewma 3.020/1.813 ms
```

With a 10% correlation, the losses in the generated sequence are about half of what we have asked:

```
root@darkstar:/# tc qdisc change dev wlan0 root netem loss 10 10

root@darkstar:/# ping #i 0.0001 #c 10000 #q 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
### 192.168.1.1 ping statistics ###
10000 packets transmitted, 9451 received, 5% packet loss, time 25209ms
rtt min/avg/max/mdev = 1.347/1.840/21.326/0.524 ms, pipe 2, ipg/ewma 2.521
```

Increasing the correlation value, the dynamics is similar of what we have just seen, with a loss of only 1% in the generated sequence.

```
root@darkstar:/# tc qdisc change dev wlan0 root netem loss 10 30

root@darkstar:/# ping #i 0.0001 #c 10000 #q 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
### 192.168.1.1 ping statistics ###
10000 packets transmitted, 9868 received, 1% packet loss, time 20697ms
rtt min/avg/max/mdev = 1.364/1.885/18.923/0.691 ms, pipe 2, ipg/ewma 2.069
```

Similar problems were evidenced in this discussion: (https://lists.linux-foundation.org/pipermail/netem/2007-September/001156.html), where the author explains them with theoretical considerations about the algorithm used by Netem to decide if a packet has to be lost, that is based on a correlated random number generator. These results drove us to develop a new loss model to regard burst losses in a better way. Considering the theoretical problems of the

correlated generator, we decided to use another approach, chosing a Markov chain model.he proposed loss model.

3 Existing loss models in the literature

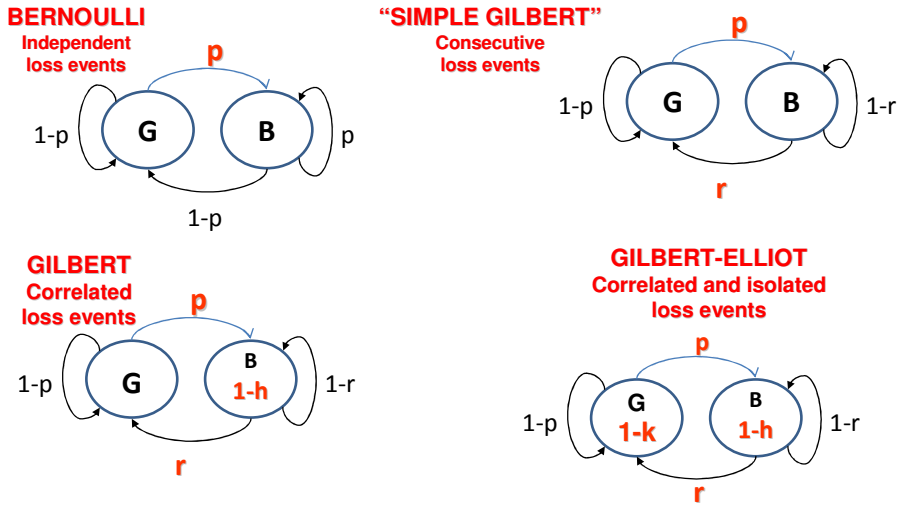

Figure 1: 4 loss models existing in the literature

The Bernoulli loss model has one state and one parameter, the loss probability p. It is only able to model uncorrelated loss events.

The "Simple Gilbert" model has two states and two independent parameters. It is able to model a system with "consecutive loss events", which can be characterized by a "loss probability" and a "burst duration". The two input parameters (p and r) can be tuned in order to characterize the system.

The Gilbert model has two states (Good and Bad) and three independent parameters. Within the Good state packets are never lost. Within the Bad state there is a probability h that a packet is transmitted. This is able to model a system in which the loss events appear in burst, but within the bursts there are some transmitted packets and some lost packets. A good way to characterize this system is to consider a "loss probability", a "burst duration" and a "loss density" within the burst (loss density = 1-h). If the loss density is 100% (i.e. h = 0) the model becomes equivalent to the "Simple Gilbert".

The Gilbert-Elliot model has two states (Good and Bad) and four independent parameters. In this case is it possible to have loss events also in the Good state, with probability 1-k. Therefore k is the probability that the packet is transmitted while the system is in Good state. If k=1 the model becomes equivalent to the Gilbert model. The Gilbert-Elliot model can be used to represent a system in which there is a Good state with a relatively low loss probability (e.g. up to a few percent) and in which the loss events appear as "isolated" and independent each other and a Bad state in which you have a relatively high loss rate (e.g. from 30-40% up to 100%) so that it is possible to detect a "burst" of loss event corresponding to the permanence of the system in the Bad state.

We note that a drawback of Gilbert and Gilbert-Elliot model is that the length of a "Bad" period when h > 0 does not exactly correspond to the length of a burst of loss events measured from the first loss event to the last loss event. In fact, when the system enters in the Bad state, it does not have to generate a loss event immediately, but there is a loss event with probability 1-h. Likewise, just before going back to the "Good" state, the system can generate a set of "transmission" events still being in the "Bad" state. Therefore the length of the "Bad" period according to the model is greater than the length of the burst measured from the first to last loss event. For example, assuming a loss density of 50% (therefore h=0.5) you may have something similar to the pattern shown in Figure 2.

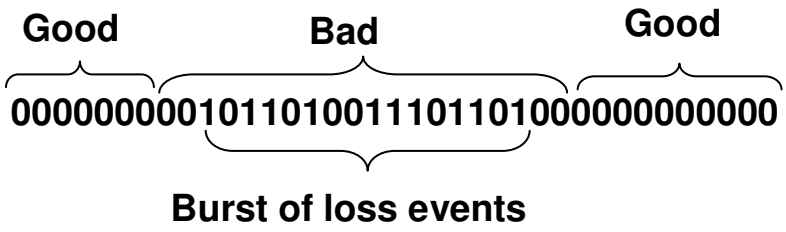

Figure 2: Example of a loss pattern generate by a Gilbert (or Gilbert-Elliot) model

4 The proposed loss model

4.1 The 4-state Markov model

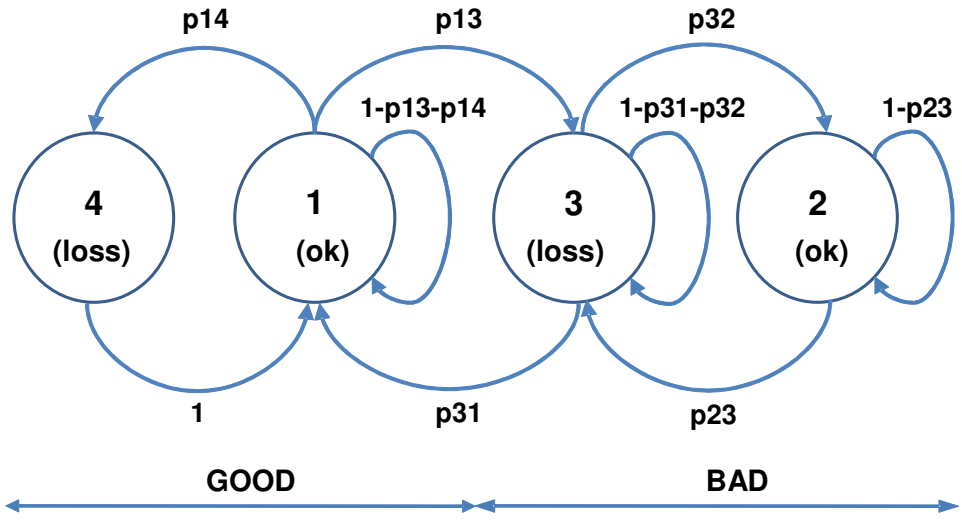

Figure 3: 4-state Markov model

This 4-state Markov is the combination of two 2-state Markov sub-models that represents two situations: *burst* periods, during which packets are received and lost according to a first 2-state model and *gap* periods during which packets are received and lost according to a second 2-state model. The same 4-state model has already been proposed in [6] and was used in [3].

The states have the following definition:

- State 1 – packet received successfully
- State 2 – packet received within a burst
- State 3 – packet lost within a burst
- State 4 – isolated packet lost within a gap

From the definition of State 4, we have p_{41}=1. At this point there are five independent transition probabilities. We will consider $p_{13}, p_{31}, p_{23}, p_{32}, p_{14}$ while:

$$\begin{cases} p_{11} = 1 - p_{13} - p_{14} \\ p_{22} = 1 - p_{23} \\ p_{33} = 1 - p_{31} - p_{32} \\ p_{44} = 0 \end{cases}$$

The Markov chain can be described by the following set of balance equations. Solving the system, we find the expression of state probabilities π_i, i=1...4 as a function of the five independent transition probabilities p_{ij}:

$$\begin{cases} \pi_1(p_{13} + p_{14}) = \pi_3 p_{31} + \pi_4 p_{41} \\ \pi_3 p_{32} = \pi_2 p_{23} \\ \pi_1 p_{14} = \pi_4 \ p_{41} \\ \pi_1 + \pi_2 + \pi_3 + \pi_4 = 1 \end{cases}$$

State 4 is associated to isolated packet losses, after that the system must come back to State 1, so we can assume p_{41}=1. Solving the system, we get the states probabilities:

$$\begin{cases} \pi_1(p_{13} + p_{14}) = \pi_3 p_{31} + \pi_4 \\ \pi_3 p_{32} = \pi_2 p_{23} \\ \pi_1 p_{14} = \pi_4 \\ \pi_1 + \pi_2 + \pi_3 + \pi_4 = 1 \end{cases} \Rightarrow \begin{cases} \pi_1 = \dfrac{p_{23} p_{31}}{p_{13} p_{23} + p_{23} p_{31} + p_{14} p_{23} p_{31} + p_{13} p_{32}} \\ \pi_2 = \dfrac{p_{13} p_{32}}{p_{13} p_{23} + p_{23} p_{31} + p_{14} p_{23} p_{31} + p_{13} p_{32}} \\ \pi_3 = \dfrac{p_{13} p_{23}}{p_{13} p_{23} + p_{23} p_{31} + p_{14} p_{23} p_{31} + p_{13} p_{32}} \\ \pi_4 = \dfrac{p_{14} p_{23} p_{31}}{p_{13} p_{23} + p_{23} p_{31} + p_{14} p_{23} p_{31} + p_{13} p_{32}} \end{cases}$$

4.2 The GI (General and Intuitive) model

The above described 4-state model uses transition probabilities between states as parameters to characterize the packet loss process. These parameters are hardly related to quantities that have an understandable meaning for an "end-user" of the model. So we aim to define a set of parameters that are more intuitive. User, in fact, would like to handle parameters clearly connected to the loss probability, the length of a burst or similar concepts. We will refer to this characterization of the loss process and to the underlying 4-state markovian model as "GI loss model", where GI stands for "General and Intuitive". We will show that the GI set of parameters is equivalent to the set of state transition probabilities, by providing the corresponding expressions. Therefore the GI approach can

be used to fully characterize the 4-state model. The 5 quantities chosen as GI parameters are the following:

- Loss probability P_{Loss};
- Mean burst length $E(B)$;
- Loss density within the burst ρ;
- Isolated loss probability P_{ISOL};
- Mean good burst length $E(GB)$;

In the next sub-section we will evaluate the state transition probabilities for the case in which all the 5 parameters are used to characterize the loss model. Then, we will show how it is possible to characterize a loss model with reduced sets of parameters (respectively 4, 3, 2 and 1 parameter).

4.2.1 GI model with 5 parameters

The expressions for P_{Loss}, ρ, P_{ISOL} and $E(GB)$ can be simply evaluated as follows:

$$P_{Loss} = \pi_3 + \pi_4 \qquad (1)$$

$$\rho = \frac{\pi_3}{\pi_3 + \pi_2} \qquad (2)$$

$$P_{ISOL} = \frac{\pi_4}{\pi_4 + \pi_1} \qquad (3)$$

$$E(GB) = \frac{1}{p_{23}} \qquad (4)$$

In order to evaluate $E(B)$, we introduce two auxiliary variables E_2 and E_3 that will be useful to calculate the average burst length.

System stays in State 2 for an average time E_2 that is equal to the average duration of a "good" sub-burst within a burst and in State 3 for an average time E_3 that is the average duration of a loss burst.

$$E_2 = \frac{1}{p_{23}}$$

$$E_3 = \frac{1}{p_{31} + p_{32}}$$

The average length of the burst is:

$$E(B) = \sum_{n=0}^{\infty}(E_3 + n(E_2 + E_3))\frac{p_{31}}{p_{31} + p_{32}}\left(\frac{p_{32}}{p_{31} + p_{32}}\right)^n = \frac{E_2 p_{32} + E_3(p_{31} + p_{32})}{p_{31}} = \frac{p_{32} + p_{23}}{p_{23}p_{31}} \qquad (5)$$

Solving the system we obtain the transition probabilities p_{ij} as:

$$\begin{cases} P_{Loss} = \pi_3 + \pi_4 = \dfrac{p_{13}p_{23} + p_{14}p_{23}p_{31}}{p_{13}p_{23} + p_{23}p_{31} + p_{14}p_{23}p_{31} + p_{13}p_{32}} \\ P_{ISOL} = \dfrac{\pi_4}{\pi_4 + \pi_1} = \dfrac{p_{14}p_{23}p_{31}}{p_{14}p_{23}p_{31} + p_{23}p_{31}} \\ \rho = \dfrac{\pi_3}{\pi_3 + \pi_2} = \dfrac{p_{13}p_{23}}{p_{13}p_{23} + p_{13}p_{32}} \\ E(B) = \dfrac{p_{32} + p_{23}}{p_{23}p_{31}} \\ E(GB) = \dfrac{1}{p_{23}} \end{cases} \Rightarrow \begin{cases} p_{31} = \dfrac{1}{E(B)\cdot\rho} \\ p_{13} = \dfrac{P_{LOSS} - P_{ISOL}}{E(B)(1-P_{ISOL})(\rho - P_{Loss})} \\ p_{23} = \dfrac{1}{E(GB)} \\ p_{32} = \dfrac{1-\rho}{\rho\cdot E(GB)} \\ p_{14} = \dfrac{P_{ISOL}}{1-P_{ISOL}} \end{cases}$$

where we substitute π_1, π_2, π_3, π_4 with the expressions in function of the transition probabilities found above. The validity of this model is subject to the following constraints:

$E(B) \geq 1/\rho$

$P_{ISOL} \leq P_{LOSS}$, $P_{LOSS} < \rho$

$P_{LOSS} - P_{ISOL} \leq E(B)\,(1- P_{ISOL})\,(\rho- P_{LOSS})$

$E(GB) \geq 1$

$E(GB) \geq (1- \rho)/ \rho$

$P_{ISOL} \leq 1/2$

4.2.2 Independent loss events within the bursts: GI model with 4 parameters

If there is no correlation between losses within the burst, we can state the following equation between transition probabilities:

$$p_{23} = 1 - \frac{p_{32}}{1 - p_{31}}$$

This equation states that the probability of losing a packet after a successful transmission within the burst (first member) equals the probability of losing a packet after another packet has just been lost. Note that $1- p_{31}$ is the probability of remaining in the burst or "Bad state" after you have had a loss event in the burst. We call this condition "independent loss events during the burst".

Under this condition, we need only 4 parameters to characterize the system, and we cannot assign an arbitrary value to E(GB).

Removing $E(GB)$ and replacing its definition with the equation: $p_{23} = 1 - \dfrac{p_{32}}{1 - p_{31}}$ we obtain the following expressions for the transition probabilities:

$$\begin{cases} P_{Loss} = \pi_3 + \pi_4 \\ P_{ISOL} = \dfrac{\pi_4}{\pi_4 + \pi_1} \\ \rho = \dfrac{\pi_3}{\pi_3 + \pi_2} \\ E(B) = \dfrac{p_{32} + p_{23}}{p_{23} p_{31}} \\ p_{23} = 1 - \dfrac{p_{32}}{1 - p_{31}} \end{cases} \Rightarrow \begin{cases} p_{31} = \dfrac{1}{E(B) \cdot \rho} \\ p_{13} = \dfrac{P_{LOSS} - P_{ISOL}}{E(B)(1 - P_{ISOL})(\rho - P_{Loss})} \\ p_{32} = \dfrac{(1-\rho) \cdot [E(B) \cdot \rho - 1]}{\rho \cdot [E(B) - 1]} \\ p_{23} = \dfrac{E(B) \cdot \rho - 1}{[E(B) - 1]} \\ p_{14} = \dfrac{P_{ISOL}}{1 - P_{ISOL}} \end{cases}$$

The following constraints apply here:

$E(B) \geq 1/\rho$

$P_{ISOL} \leq P_{LOSS}$, $P_{LOSS} < \rho$

$P_{LOSS} - P_{ISOL} \leq E(B)\,(1 - P_{ISOL})\,(\rho - P_{LOSS})$

$(1-\rho)[\rho\,E(B)-1] < \rho\,[E(B)-1] \Rightarrow E(B) > (2\rho - 1)/\rho^2$

$P_{ISOL} \leq 1/2$

4.2.3 Removing isolated loss events: GI model with 3 parameters

Now we consider a 3-state model obtained from the above shown 4-state setting p_{14}=0. This means we will not have isolated loss events and will never enter in State 4. Since P_{ISOL}=0 the system can be described through three effective GI parameters:

- Loss probability P_{Loss};
- Loss density within the burst ρ;
- Mean burst length $E(B)$;

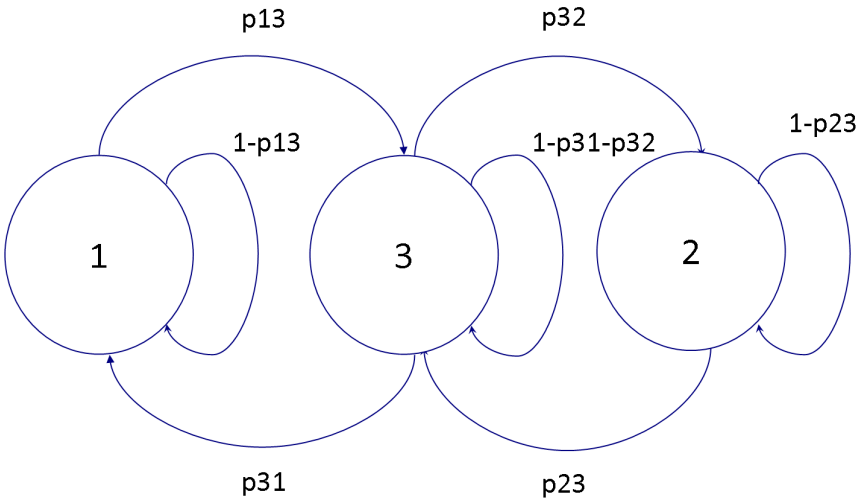

Figure 4: 3-state model

The transition probabilities become:

$$\begin{cases} P_{Loss} = \pi_3 \\ \rho = \dfrac{\pi_3}{\pi_3 + \pi_2} \\ E(B) = \dfrac{p_{32} + p_{23}}{p_{23} p_{31}} \\ p_{23} = 1 - \dfrac{p_{32}}{1 - p_{31}} \end{cases} \Rightarrow \begin{cases} p_{13} = -\dfrac{P_{Loss}}{E(B) \cdot (P_{Loss} - \rho)} \\ p_{31} = \dfrac{1}{E(B) \cdot \rho} \\ p_{32} = \dfrac{(1-\rho) \cdot [E(B) \cdot \rho - 1]}{\rho \cdot [E(B) - 1]} \\ p_{23} = \dfrac{E(B) \cdot \rho - 1}{E(B) - 1} \end{cases}$$

The model is valid under the following constraints:

$E(B) \geq 1/\rho$

$P_{LOSS} < \rho$

$P_{LOSS} - P_{ISOL} \leq E(B)\,(1 - P_{ISOL})\,(\rho - P_{LOSS})$

$(1-\rho)[\rho\,E(B)-1] < \rho\,[E(B)-1] \Rightarrow E(B) > (2\rho-1)/\rho^2$

4.2.4 Consecutive losses: GI model with 2 parameters

Now we remove the State 2 relative to packets received during burst periods. Therefore, we can model loss bursts made of consecutive loss events. We set p_{32}=0 and p_{23}=1 for consistency reasons, obtaining a 2-state *Markov* chain:

$$\begin{cases} \pi_1(p_{13}) = \pi_3 p_{31} \\ \pi_1 + \pi_3 = 1 \end{cases} \Rightarrow \begin{cases} \pi_1 = \dfrac{p_{31}}{p_{13} + p_{31}} \\ \pi_3 = \dfrac{p_{13}}{p_{13} + p_{31}} \end{cases}$$

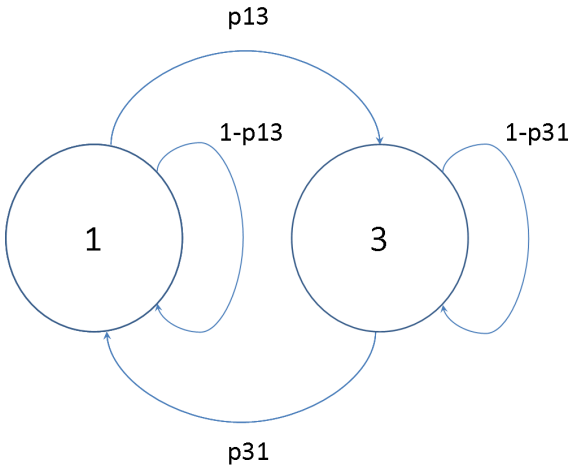

Figure 5: The 2-state Markov chain

Since we have P_{ISOL}=0 and ρ=1 the GI parameters involved are only two: P_{LOSS} and $E(B)$. Mean length of burst corresponds to the mean time the system is in State 3, so $E(B)=\frac{1}{p_{31}}$.Then the transition probabilities are:

$$\begin{cases} P_{Loss}=\pi_3 \\ E(B)=\dfrac{1}{p_{31}} \end{cases} \Rightarrow \begin{cases} p_{31}=\dfrac{1}{E(B)} \\ p_{13}=\dfrac{P_{Loss}}{E(B)\cdot(1-P_{Loss})} \end{cases}$$

4.2.5 Bernoulli model: GI model with 1 parameter

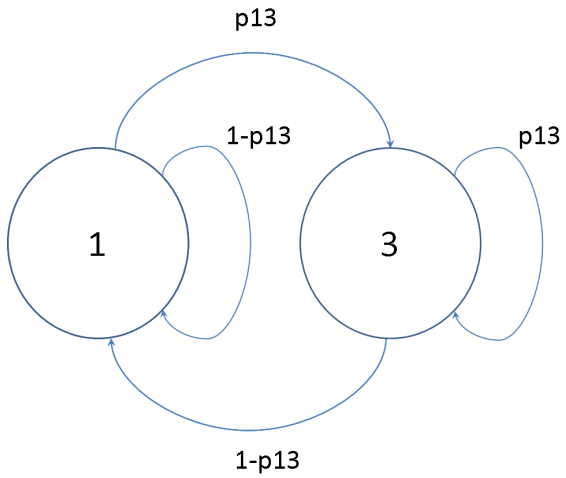

Figure 6: The Bernoulli model as a Markov chain

If we consider the special case of a 2-state *Markov* where the *r* parameter depends on *p* by the relation *r*=1-*p*, we obtain the *Bernoulli* model. At this point, the *Bad* state collapses on the *Good* one, so we can see this model as a 1-state *Markov* chain with an only independent parameter.

The Bernoulli model is the simplest case of loss model we consider. It has only one parameter and can be see as a special case of the Simple Gilbert model where *r*=1-*p*. So the only significant GI parameter is the loss probability P_{LOSS}=*p*. So we have:

$$\begin{cases} p_{31}=P_{Loss} \\ p_{13}=1-P_{Loss} \end{cases}$$

4.3 Relations between the models in the literature and the 4-state Markov model

In this section we want to relate the 4 loss models existing in the literature and described in section 3 with the 4-State Markov model and its GI characterization that we have proposed.

Our goal is to show that:

- the 4 state model is able to produce loss patterns practically equivalent to those generated by the existing models

- to evaluate the input parameters to be provided to the 4 state model to "simulate" the behaviour of a given set of parameters for one of the existing models.

We used an analytical approach which allowed us to evaluate the GI parameters (like burst length, burst density ecc.) for the existing models starting from the native parameters of the model.

Then we can use these evaluated GI parameters as input to our 4 state model.

A simulation campaing has been used to validate this approach, as described in section 7. We compared the loss patterns produced by the simulators of the existing models and of the 4 state models with the evaluated GI parameters. We found that the parameters evaluated by generated loss patterns are in accordance with our analytical models.

4.3.1 Revisiting the Gilbert-Elliot model

In order to relate the classical Gilber-Elliot model and the 4-state model, we put ourselves under the condition of "independent loss events during the burst". Roughly speaking, states 1 and 4 correspond to the *Good* state in Gilbert-Elliot, States 3 and 2 correspond to the *Bad* state. Note that the *Good* state is more properly a "low loss" state [6]. The main difference is that transition from Bad to Good can happen in the Gilber-Elliot after either a loss o a transmission event, while in 4-state model the transition can only happen after a loss event, and this is needed in order to properly evaluate a burst length, i.e. starting from a loss event and ending with a loss event. The second difference is that in the Gilbert Elliot model it is possible to have two consecutive "isolated" loss events, while in the the 4-state model an isolated loss is always followed by a transmission.

Our goal in this section is to define a special case of the GI model whose behavior is very similar to the Gilbert-Elliot one, and to evaluate its parameters as a function of the Gilbert Elliot parameters. We refer to it as the "Gilbert-Elliot-4s" model.

Let us consider the loss patterns produced by the Gilbert Elliot model. If we consider a burst length ignoring that there can be leading and trailing transmission events in a burst, the burst loss, burst length and the length in the bad period can be evaluated in the following way:

$$P_{LOSS} = \frac{r}{p+r}(1-k) + \frac{p}{p+r}(1-h)$$

$$E(\overline{B}) = \frac{1}{r}$$

$$\overline{\rho} = 1-h$$

$$P_{ISOL} = 1-k$$

However, as shown in section 3, the loss sequences generated using the Gilbert-Elliot model could have an "extended" burst length and so a "reduced" density due to the possible transmission events at the start and at the end of the burst. Therefore we have denoted this "extended" burst length and this reduced density as $E(\overline{B})$ and $\overline{\rho}$. Since the GI model states explicitely that a burst starts and ends with a loss event (according to the definition given in the RFC3611 [7]), we need to fix the relations with some correction factors to take into account this fact. Then we will find the value of

GI paramters that makes the GI model's behavior the more possible similar to the Gilbert-Elliot's one, obtaining the special case of the GI model that we denote as Gilbert-Elliot-4s.

If the system is in the state 1 (call it Good) and there is a loss event (according to an "independent" loss probability 1-*k*), without entering in the *burst* condition, the system goes into state 4 (call it "Bad into good"). At the next step, it deterministically comes back to state 1. This is what we call an "*isolated loss*". Instead, if the system is in the state 3 (call it "Bad") and there is a loss event (according to an independent loss probability 1-*h*) it remains in the same state. Otherwise, if a packet is correctly transmitted without coming out from the *burst* condition, it goes to state 2 (call it "Good into Bad"). The difference respect to the Gilbert-Elliot model is that the transitions from state 1 (Good) to state 3 (Bad) only happen after a transmission event and will be followed by a loss event. Likewise the transitions from state 3 (Bad) to state 1 (Good) only happen after that a loss event is verified and will be followed with a "transmission" event. We believe this is a much "cleaner" modeling as a transition from Good to Bad in the model corresponds to a transmission event followed by a loss event and viceversa a transition from Bad to Good in the model corresponed to a loss event followed by a transmission event.

In order to map the Gilbert-Elliot model into our 4-state model, we will evaluate the following GI parameters:

- Loss probability P_{Loss};
- Isolated loss probability P_{ISOL};
- Mean burst length $E(B)$;
- Loss density within the burst ρ;

Then we will derive the transition probabilities that correspond to the evaluated GI parameters.

$$P_{LOSS} = \frac{r}{p+r}(1-k) + \frac{p}{p+r}(1-h)$$

$$P_{ISOL} = 1-k$$

As already noted, under the Gilbert Elliot model the loss events in the Good state (i.e. corresponding to the P_{ISOL}) can also be consecutive, while in the 4 state model they will always be followed by a transmission event. The average number of these loss events will in any case be the same, the "patterns" can be slightly different. The difference will be higher for increasing value of P_{ISOL}, as the probability of having consecutive "isolated" loss events increases with P_{ISOL}.

In order to evaluate the burst length we have to consider that there is an "extended" burst length that includes the initial and final "transmission" events with the burst. This is easily evaluated as:

$$E(\overline{B}) = \frac{1}{r}$$

We want to evaluate the "corrected" burst length $E(B)$, therefore we will consider all the possible extended burst lengths, and for each of them we will evaluate the probability of the corrected burst length.

The probability that the extended burst length equals i can be evaluated as follows:

$$P\{\overline{B} = i\} = r(1-r)^{i-1}$$

Let as denote a burst as a sequence of “L” for loss and “T” for transmissions, for example “TLLL” denotes a bursts of one transmission and three losses (the extended length is 4 and the corrected length is 3, as there is an initial transmission.

If the extended burst length is 1 (i = 1) the corrected burst length can be:

1 (sequence: “L”) with probability (1-h)
0 (sequence: “T”), with probability h

If i = 2 the corrected burst length can be:

2 (LL) with probability $(1-h)^2$
1 (TL or LT) with overall probability $2h(1-h)$
0 (TT) with probability h^2

If i = 3 the corrected burst length can be:

3 (LxL) with probability $(1-h)^2$
2 (TLL or LLT) with probability $2h(1-h)^2$
1 (TTL, LTT, TLT) with probability $3h^2(1-h)$
0 (TTT) with probability h^3

If i = 4 the corrected burst length can be:

4 (LxxL) with probability $(1-h)^2$
3 (TLxL or LxLT) with probability $2h(1-h)^2$
2 (TTLL, TLLT, LLTT) with probability $3h^2(1-h)^2$
1 (TTTL, TTLT, TLTT, LTTT) with probability $4h^3(1-h)$
0 (TTTT) with probability h^4

If i = 5 the corrected burst length can be:

5 (LxxxL) with probability $(1-h)^2$
4 (TLxxL, LxxLT) with probability $2h(1-h)^2$
3 (TTLxL, LxLTT, LTxTL) with probability $3h^2(1-h)^2$
2 (TTTLL, TTLLT, TLLTT, LLTTT) with probability $4h^3(1-h)^2$
1 (TTTTL, TTTLT, TTLTT, TLTTT, LTTTT) with probability $5h^4(1-h)$
0 (TTTTT) with probability h^5

If i = 6 the corrected burst length can be:

6 (LxxxxL) with probability $(1-h)^2$
5 (TLxxxL, LxxxLT) with probability $2h(1-h)^2$
4 (TTLxxL, LxxLTT, LTxxTL) with probability $3h^2(1-h)^2$
3 (TTTLxL, TTLxLT, TLxLTT, LxLTTT) with probability $4h^3(1-h)^2$
2 (TTTTLL, TTTLLT, TTLLTT, TLLTTT, LLTTTT) with probability $5h^4(1-h)^2$
1 (TTTTTL, TTTTLT, TTTLTT, TTLTTT, TLTTTT, LTTTTT) with probability $6h^5(1-h)$
0 (TTTTTT) with probability h^6

i	$P\{\overline{B} = i\}$	6	5	4	3	2	1	0
1	r						(1-h)	h

2	r(1-r)					$(1-h)^2$	$2h(1-h)$	h^2
3	$r(1-r)^2$				$(1-h)^2$	$2h(1-h)^2$	$3h^2(1-h)$	h^3
4	$r(1-r)^3$			$(1-h)^2$	$2h(1-h)^2$	$3h^2(1-h)^2$	$4h^3(1-h)$	h^4
5	$r(1-r)^4$		$(1-h)^2$	$2h(1-h)^2$	$3h^2(1-h)^2$	$4h^3(1-h)^2$	$5h^4(1-h)$	h^5
6	$r(1-r)^5$	$(1-h)^2$	$2h(1-h)^2$	$3h^2(1-h)^2$	$4h^3(1-h)^2$	$5h^4(1-h)^2$	$6h^5(1-h)$	h^6
…								

The probability of having a burst with corrected length 0, i.e. with all transmission events is:

$$P(B=0)=P\{\overline{B}=1\}h+P\{\overline{B}=2\}h^2+P\{\overline{B}=3\}h^3+...=$$
$$=rh+r(1-r)h^2+r(1-r)^2h^3+...=$$
$$=r\sum_{i=0}^{\infty}(1-r)^i h^{i+1}=rh\sum_{i=0}^{\infty}[(1-r)h]^i=rh\left[\frac{1}{1-(1-r)h}\right]=\frac{rh}{1-h+rh}$$

The probability or having a burst with corrected length 1 is:

$$P(B=1)=P\{\overline{B}=1\}(1-h)+P\{\overline{B}=2\}2h(1-h)+P\{\overline{B}=3\}3h^2(1-h)+...=$$
$$=r(1-h)+r(1-r)2h(1-h)+r(1-r)^2 3h^2(1-h)+...=$$
$$=r(1-h)\sum_{i=0}^{\infty}(i+1)(1-r)^i h^i=r(1-h)\sum_{i=0}^{\infty}(i+1)[(1-r)h]^i=$$
$$=r(1-h)\left\{\sum_{i=0}^{\infty}i[(1-r)h]^i+\sum_{i=0}^{\infty}[(1-r)h]^i\right\}=$$
$$=r(1-h)\left\{\frac{(1-r)h}{[1-(1-r)h]^2}+\frac{1}{1-(1-r)h}\right\}=$$
$$=r(1-h)\left\{\frac{h-rh}{[1-h+rh]^2}+\frac{1}{1-h+rh}\right\}=r(1-h)\left\{\frac{h-rh+1-h+rh}{(1-h+rh)^2}\right\}=$$
$$=r(1-h)\left\{\frac{1}{(1-h+rh)^2}\right\}=\frac{r(1-h)}{(1-h+rh)^2}$$

For $j \geq 2$, the probability of having a burst with corrected length j is

$$P(B=j)=P\{\overline{B}=j\}(1-h)^2+P\{\overline{B}=j+1\}2h(1-h)^2+P\{\overline{B}=j+2\}3h^2(1-h)^2+\ldots=$$
$$=r(1-r)^{j-1}(1-h)^2+r(1-r)^j 2h(1-h)^2+r(1-r)^{j+1}3h^2(1-h)^2+\ldots=$$
$$=r(1-h)^2(1-r)^{j-1}\sum_{i=0}^{\infty}(i+1)(1-r)^i h^i=r(1-h)^2(1-r)^{j-1}\sum_{i=0}^{\infty}(i+1)[(1-r)h]^i=$$
$$=r(1-h)^2(1-r)^{j-1}\frac{1}{(1-h+rh)^2}=(1-r)^{j-1}\frac{r(1-h)^2}{(1-h+rh)^2}$$

The average corrected length, for bursts with corrected length > 0 is:

$$E(B)=1\cdot\frac{P\{B=1\}}{P\{B>0\}}+\sum_{j=2}^{\infty}j\frac{P\{B=j\}}{P\{B>0\}}$$

Where $P\{B>0\}=1-P\{B=0\}=1-\frac{rh}{1-h+rh}=\frac{1-h}{1-h+rh}$, therefore:

$$E(B)=\frac{1-h+rh}{1-h}\left(\frac{r(1-h)}{(1-h+rh)^2}+\frac{r(1-h)^2}{(1-h+rh)^2}\sum_{j=2}^{\infty}j(1-r)^{j-1}\right)=$$
$$=\frac{r}{(1-h+rh)}\left(1+(1-h)\sum_{j=1}^{\infty}(j+1)(1-r)^j\right)=$$
$$=\frac{r}{(1-h+rh)}\left(1+(1-h)\left(\sum_{j=1}^{\infty}j(1-r)^j+\sum_{j=1}^{\infty}(1-r)^j\right)\right)=$$
$$=\frac{r}{(1-h+rh)}\left(1+(1-h)\left(\frac{(1-r)}{r^2}-1+\sum_{j=0}^{\infty}(1-r)^j\right)\right)=$$
$$=\frac{r}{(1-h+rh)}\left(1+(1-h)\left(\frac{(1-r)}{r^2}-1+\frac{1}{r}\right)\right)=\frac{r(2-h)}{(1-h+rh)}\left(1+(1-h)\left(\frac{1-r-r^2+r}{r^2}\right)\right)=$$
$$=\frac{r}{(1-h+rh)}\left(1+(1-h)\left(\frac{1-r^2}{r^2}\right)\right)=\frac{r}{(1-h+rh)}\left(\frac{r^2+(1-h)(1-r^2)}{r^2}\right)=$$
$$=\frac{(r^2+1-r^2-h+r^2h)}{r(1-h+rh)}=\frac{(1-h+r^2h)}{r(1-h+rh)}$$

As for the loss density within the burst, we have to take into account that the density increases as all the loss events are concentrated in the burst that have at least one loss and that the length of these "corrected" bursts $E(B)$ is shorter than the "default" burst length in the Gilbert Elliot model $E(\overline{B})$ As we have done for the burst length, we call $\overline{\rho}$ the "default" burst density that corresponds to the loss probability in the "Bad" state. Let us define $\overline{N}$ the number of Gilbert Elliot burst in an arbitrary interval, and N the number of "corrected" burst in the same interval. The following equation holds

$\overline{\rho}E(\overline{B})\overline{N}=\rho E(B)N$, therefore we can evaluate $\rho=\frac{\overline{\rho}E(\overline{B})\overline{N}}{E(B)N}$

We can now evaluate the ratio $\frac{\overline{N}}{N}$ considering the probability that a Gilbert Elliot burst is also a "corrected" burst:

$$\frac{\overline{N}}{N}=\frac{1}{P\{B>0\}}=\frac{1}{1-P\{B=0\}}=\frac{1}{\frac{1-h}{1-h+rh}}=\frac{1-h+rh}{1-h}$$

Therefore the density is:

$$\rho=\frac{\overline{\rho}E(\overline{B})\overline{N}}{E(B)N}=(1-h)\frac{\frac{1}{r}}{\frac{(1-h+r^2h)}{r(1-h+rh)}}\frac{1-h+rh}{1-h}=\frac{(1-h+rh)^2}{(1-h+r^2h)}$$

We can also find the relations between the transition probabilities and the Gilbert-Elliot-4s parameters.

$$\begin{cases} P_{LOSS}=\frac{r}{p+r}(1-k)+\frac{p}{p+r}(1-h)=\frac{p_{13}p_{23}+p_{14}p_{23}p_{31}}{p_{13}p_{23}+p_{23}p_{31}+p_{14}p_{23}p_{31}+p_{13}p_{32}} \\ P_{ISOL}=1-k=\frac{p_{14}p_{23}p_{31}}{p_{14}p_{23}p_{31}+p_{23}p_{31}} \\ \rho=\frac{(1-h+rh)^2}{(1-h+r^2h)}=\frac{p_{13}p_{23}}{p_{13}p_{23}+p_{13}p_{32}} \\ E(B)=\frac{(1-h+r^2h)}{r(1-h+rh)}=\frac{p_{32}+p_{23}}{p_{23}p_{31}} \\ p_{23}=1-\frac{p_{32}}{1-p_{31}} \end{cases}$$

$$\Rightarrow \begin{cases} p_{13}=\frac{(h-k)p\{[1+h(-1+r)]r\}}{kr\{k+h^2(-1+r)(-1+2p+r)-h[k+p(-2+r)+(-1+r)^2-kr^2]\}} \\ p_{31}=\frac{[1+h\,(-1+r)]\,r}{[1+h\,(-1+r)]^2} \\ p_{32}=-\frac{(-1+h)h(-1+r)^2[(1-h+hr)^2-r(1-h+hr)]}{(1-h+hr)^2(1-h+hr^2)-r(1-h+hr)} \\ p_{23}=\frac{(1-h+hr)^2-r(1-h+hr)}{1+h(-1+r^2)-r(1-h+hr)} \\ p_{14}=-1+\frac{1}{k} \end{cases}$$

4.3.2 Revisiting the Gilbert model

We have already shown in section 4.3.1 a special case of the GI model which has a behavior very similar to the Gilbert-Elliot model, and called it Gilbert-Elliot-4s. The Gilbert model is a special

case of the Gilbert-Elliot where we set 1-*k*=0 removing the *Bad into Good* state. So the *Gilbert* model can be seen as a simplified 4-state model with only 3 states, that we will call Gilbert-4s. If we consider the Gilbert-4s model we have the following relation between the Gilbert parameters and the transition probabilities, simplifying the Gilbert-Elliot-4s case with k=1:

$$\begin{cases} P_{LOSS} = \dfrac{p}{p+r}(1-h) = \dfrac{p_{13}p_{23}}{p_{13}p_{23} + p_{23}p_{31} + p_{13}p_{32}} \\ P_{ISOL} = 0 \\ \rho = \dfrac{(1-h+rh)^2}{(1-h+r^2h)} = \dfrac{p_{13}p_{23}}{p_{13}p_{23} + p_{13}p_{32}} \\ E(B) = \dfrac{(1-h+r^2h)}{r(1-h+rh)} = \dfrac{p_{32}+p_{23}}{p_{23}p_{31}} \\ p_{23} = 1 - \dfrac{p_{32}}{1-p_{31}} \end{cases}$$

$$\Rightarrow \begin{cases} p_{13} = \dfrac{(h-1)p\{[1+h(-1+r)]r\}}{r\{1+h^2(-1+r)(-1+2p+r) - h[1+p(-2+r)+(-1+r)^2 - r^2]\}} \\ p_{31} = \dfrac{[1+h(-1+r)]r}{[1+h(-1+r)]^2} \\ p_{32} = -\dfrac{(-1+h)h(-1+r)^2[(1-h+hr)^2 - r(1-h+hr)]}{(1-h+hr)^2(1-h+hr^2) - r(1-h+hr)} \\ p_{23} = \dfrac{(1-h+hr)^2 - r(1-h+hr)}{1+h(-1+r^2) - r(1-h+hr)} \\ p_{14} = 0 \end{cases}$$

4.3.3 Revisiting the "Simple Gilbert" model

The *Gilbert* model can be simplified setting *h*=0. In this way, the *Gilbert* model becomes the same of the 2-state *Markov* model and is able to represent only consecutive burst losses. This is also called "Simple Gilbert" model [11]. Note that in this case there are no ambiguities with the burst definition because when the system is in the burst condition we will have only losses and no transmissions. So in this case it is not needed to define special cases to make the model equivalent to the GI.

The Simple Gilbert Elliot seen introduced in Section 4.3.3 is equivalent to the 2-state Markov. So we have the following relation:

$$\begin{cases} p = p_{13} \\ r = p_{31} \end{cases}$$

4.4 Relations between models

As we have widely explained, the proposed GI (General and Intuive) model and the underlying 4-state Markov model include all the others described model as particular cases. The following table shows the features of all the models in terms of what type of losses they can describe:

MODEL	GI PARAMETERS	STATES	NUMBER OF PARAMETERS	BURST LENGTH	BURST DENSITY	ISOLATED LOSSES	BURST LOSSES CORRELATION
4-state Markov	P_{LOSS},E(B),ρ P_{ISOL},E(GB)	4	5	Yes	Yes	Yes	Yes
Gilbert-Elliot-4s	P_{LOSS}, E(B),ρ, P_{ISOL}	4	4	Yes	Yes	Yes	No
Gilbert-4s	P_{LOSS}, E(B), ρ	3	3	Yes	Yes	No	No
2-state Markov	P_{LOSS}, E(B)	2	2	Yes	No	No	No
Simple Gilbert	P_{LOSS}, E(B)	2	2	Yes	No	No	No
Bernoulli	P_{LOSS}	2	1	No	No	No	No

Table 1: Features of GI loss models and its special cases

5 Definition and identification of a burst

Intuitivelly, a “Bad” burst is a period during which a high proportion of packets lost. When a “Bad” burst can included transmitted packets, there is the problem of delimiting the burst. In fact it is possible that a single burst contains a sequence of transmitted packets in the middle, so that it can be interpreted as two different bursts. On the other hand, it is possible that two different bursts are separated by only one or two transmitted packets so they can be “merged” in a single “Bad” burst. In [7] a formal definition of a burst is given. A burst is defined, in terms of a value g_{min}, as the longest sequence that starts with a lost or discarded packet, does not contain any occurrences of g_{min} or more consecutively received (and not discarded) packets, and ends with a lost or discarded packet. Therefore the delimitation of bursts on an observed sequence of received/lost packets depends on the choice of the g_{min} value.

This means that when we generate a received/loss sequence using the GI model, the average length of the burst E(B), evaluated with a given g_{min} value will not exactly match the input E(B) parameter.

In particular if g_{min} is small (and the loss density in the burst is not high) there will be a higher probability of splitting a burst into smaller bursts => E(B) tends to be smaller than the input. If g_{min} is large there is an higher probability of joining two bursts into a longer burst (and this will me more likely if the average “inter-arrival” of bursts is relatively small) => E(B) tends to be larger than the input.

6 NetemCLG Implementation

6.1 NetemCLG and tc-netem patches

This section describes how to install the patches that adds the new correlated loss models to the *sch_netem* module of the Linux kernel and to the *tc* utility. These instructions are included in file *INSTALL.TXT* into the patch package.

To install the NetemCLG kernel patch do:

```
cd <your linux source tree>

patch -p1 <../netem_clg_patch
```

substitute *../netem_clg_patch* with the path where you have unpacked the archive. Then enable as module "*Network Emulator (NETEM)*" in the kernel configuration under *Networking*, *Networking Options*, *QoS and/or fair queuing*. The command to load netem is "*modprobe sch_netem*" while the command to unload is "*modprobe -r sch_netem*".

To install the tc patch do:

```
cd <your iproute2 source tree>

patch -p1 <../tc_patch
```

substitute *../tc_patch* with the path where you have unpacked the *iproute2* sources. Then build *iproute2*.

The patches were tested with Linux kernel version 2.6.32.2 and iproute2-2.6.31.tar.bz2 package.

6.2 tc-netem user manual (for new options)

The modified version of *tc* allows the user to add loss features to the outgoing packets according to several models, according to the following syntax:

Adding a loss pattern according to a model

tc qdisc add/change/del dev eth0 root netem loss_GI ploss [burst_length [density [pisol [good_burst_length]]]]

tc qdisc add/change/del dev eth0 root netem loss_4state p13 [p31 [p32 [p23 [p14]]]]

tc qdisc add/change/del dev eth0 root netem loss_gilb_ell p [r [1-h [1-k]]]

tc qdisc add/change/del dev eth0 root netem loss_gilb_ell_4s p [r [1-h [1-k]]]

Querying information from a loss model

tc qdisc add dev eth0 root netem2 query loss_GI ploss [burst_length [density [pisol [good_burst_length]]]]

tc qdisc add dev eth0 root netem2 query loss_4state p13 p31 [p32 p23 [p14]]

tc qdisc add dev eth0 root netem2 query loss_gilb_ell_4s p [r [1-h [1-k]]]

Adding a deterministic loss pattern

tc qdisc add dev eth0 root netem2 query loss_pattern filename [repetitions]

Enabling the logging mode

tc qdisc add dev eth0 root netem2 logging 1 .....

Where the correspondence between options and model are:

OPTION	MODEL
Loss_GI	GI (General and Intuitive)
Loss_4state	4-state Markov. (with transition probabilities as input parameters) Allows special cases: 3-state, 2-state, Bernoulli
Loss_bern	Bernoulli
Loss_gilb_ell_4s	Gilbert-Elliot-4s (allows Gilbert-4s as special case)

Table 2: Options and models in tc-netem2

If the selected option is *loss_gilb_ell* the Gilbert-Elliot generation algorithm will be used. If the selected option is one of *loss_GI*, *loss_4_state* or *loss_gilb_ell_4s* the 4-state generation algorithm will be used. In the case of *loss_4state* the transition probabilities specified as input are directly passed to the algorithm while for the other options the input parameters are mapped to the 4-state transition probabilities and passed to *netem* thought the *netlink* socket. The conversion formulas are shown in the following table:

Loss_4state	Loss_GI
p_{13}	$\frac{P_{Loss} - P_{ISOL}}{E(B)(1-P_{ISOL})(\rho - P_{Loss})}$
p_{31}	$\frac{1}{E(B)\cdot\rho}$
p_{23}	$\frac{E(B)\cdot\rho - 1}{E(B)-1}$

p_{32}	$\frac{(\rho-1)\cdot(E(B)\cdot\rho-1)}{\rho\cdot(1-E(B))}$
p_{14}	$\frac{P_{ISOL}}{1-P_{ISOL}}$

Table 3: Relations between options and models in tc-netem2

For the *Gilbert-Elliot-4s* case, the conversion is made in two steps: firstly, the GI parameters that correspond to the input parameters are calculated. Then, the GI parameters are used to calculate the 4-state transition probabilities. The relations used are the following:

$$\begin{cases} P_{LOSS} = \frac{r}{p+r}(1-k)+\frac{p}{p+r}(1-h) = \frac{p_{13}p_{23}+p_{14}p_{23}p_{31}}{p_{13}p_{23}+p_{23}p_{31}+p_{14}p_{23}p_{31}+p_{13}p_{32}} \\ P_{ISOL} = 1-k = \frac{p_{14}p_{23}p_{31}}{p_{14}p_{23}p_{31}+p_{13}p_{32}} \\ \rho = \frac{(1-h+rh)^2}{r(1-h+rh)} = \frac{p_{13}p_{23}}{p_{13}p_{23}+p_{13}p_{32}} \\ E(B) = \frac{1-h+r^2h}{r(1-h+rh)} = \frac{p_{32}+p_{23}}{p_{23}p_{31}} \\ E(GB) = \frac{1}{p_{23}} \end{cases}$$

Only one of the three different labels *add*, *change* and *del* has to be chosen: *add* is used to initialize a new option, *change* to set new parameter(s) value(s) and *del* to remove already set parameters with a previous command. Moreover, the label *eth0* identifies the chosen network interface, so it is needed to replace it with the correct value according to the interface chosen for the simulation. The parameters values are always expressed as percentage, except for burst length that is integer.

Using the query mode, the transition probabilities and the GI (*General and Intuitive*) parameters that corresponds to the specified input parameters are calculated and printed to screen, but no operation is done on the *qdisc* so the user can understand what he would obtain with certain input parameters without necessarily make changes to the system configuration

6.2.1 Query mode examples

Now we present a few examples of how the query mode works. In particular we will analyze one of the most interesting things, that is the possibility of finding the length of the good burst changing the burst density or setting it manually and seeing how the transition probabilities change. The first of them shows the transition probabilites and all the GI (*General and Intuitive*) parameters that correspond to a sequence with a 2% loss probability and a mean burst length of 10. While p_{13} and p_{31} depend on the two input parameters, the others are set to the default values.

```
$ tc qdisc add dev eth0 root netem query  loss_GI 2 10
```

```
Transition probabilities are:
------------------------------
p13 is 0.204%
p31 is 10.000%
p32 is 0.000% (using statistical independence hypothesis)
p23 is 100.000% (using statistical independence hypothesis)
p14 is 0.000%

GI (General and Intuitive) parameters would be:
------------------------------
ploss is 2.000%
burst duration is 10.000
burst density is 100.000%
isolated ploss is 0.000%
good burst duration is 1.000 (using statistical independence hypothesis)
```

Now we reduce the burst density to 80% leaving the other parameters to old values. This choice affects p32 and p23, so the good burst became longer since we have less packet loss during the burst, which length is the same of the previous case.

```
$ tc qdisc add dev eth0 root netemquery loss_GI 2 10 80

The transition probabilities are:
------------------------------
p13 is 0.256%
p31 is 12.500%
p32 is 19.444% (using statistical independence hypothesis)
p23 is 77.778% (using statistical independence hypothesis)
p14 is 0.000%

The GI (General and Intuitive) parameters would be:
------------------------------
ploss is 2.000%
burst duration is 10.000
burst density is 80.000%
isolated ploss is 0.000%
good burst duration is 1.286 (using statistical independence hypothesis)
```

Note that in the examples shown until now we are calculating the good burst length still using the statistical independence hyphotesis. Now we try to set manually a higher value of good burst duration. Comparing this to the previous case, p_{32} increases. This is reasonable because the burst length and the burst density are unchanged, so we have a higher number of "good" bursts and we pass from state 3 to state 2 more frequenty. Moreover, the "good" burst length is higher, so it is less probable that we return to State 3, so p_{23} decreases.

```
$ tc qdisc add dev eth0 root netem loss_GI_query 2 10 80 0 2

The Transition probabilities are:
------------------------------
p13 is 0.256%
```

```
p31 is 12.500%
p32 is 12.500%
p23 is 50.000%
p14 is 0.000%

The GI (General and Intuitive) parameters would be:
-------------------------------
ploss is 2.000%
burst duration is 10.000
burst density is 80.000%
isolated ploss is 0.000%
good burst duration is 2.000
```

The last example shows that burst density value has a heavy influence on the good burst duration: reducing density to 50% increases it more than twofold.

```
$ tc qdisc add dev eth0 root netem loss_GI_query 2 10 50

The transition probabilities are:
-------------------------------
p13 is 0.417%
p31 is 20.000%
p32 is 44.444% (using statistical independence hypothesis)
p23 is 44.444% (using statistical independence hypothesis)
p14 is 0.000%

The GI (General and Intuitive) parameters would be:
-------------------------------
ploss is 2.000%
burst duration is 10.000
burst density is 50.000%
isolated ploss is 0.000%
good burst duration is 2.250 (using statistical independence hypothesis)
```

6.2.2 Deterministic pattern

The *loss_pattern* option allows the user to add a deterministic loss pattern to the packets outgoing from the selected network interface. The loss pattern is read from a text file (which name is specified as an input parameter) as a sequence of “0” and “1” where “0” represents a transmission event and “1” a loss event. This function can be useful to see how a system responds to a particular pattern and, for example, to test the efficiency of FEC algorithms.

6.2.3 Logging mode

The *logging* option sets a logging level. Actually it works for *loss_GI*, *loss_4state*, *loss_gilb_ell*, *loss_gilb_ell_4s*, *loss_pattern* options. If not specified, level 0 will be used so no data will be logged. If logging level is 1 for each loss events the kernel logs will include a line like "netem2 loss event algorithm [type] x RFPLE y" where RFPLE stands for "Received From Previous Loss Event" and it counts the number y of good packets received between two loss events while x is the number

of all lost packets and algorithm refers to the selected loss generation algorithm (GI, gilb_ell or deterministic). The *type* label applies only to the GI algorithm and can be *burst* or *isolated*.

6.3 NetemCLG: code enhancements

6.3.1 NetemCLG

This section describes the differences between *Netem* and *NetemCLG*, or rather the pieces of code added to *Netem* to implement the new features described in Section 6.2. The modified files are *pkt_sched.h* in *<kernel_source_path>/include/linux* and *sch_netem* in *<kernel_source_path> /net/sched*. Note that *<kernel_source_path>* stands for the path where the kernel sources have been unpacked.

Both in *pkt_sched.h* were added:

- the constant *NETEM_CLG_GILB_ELL* (set to 0), *NETEM_CLG_4_STATES* (set to 1) and *NETEM_CLG_DETERMIN* (set to 2) to switch between the three correlated loss models.

```
#define NETEM_CLG_GILB_ELL  0
#define NETEM_CLG_4_STATES  1
#define NETEM_CLG_DETERMIN2
```

- the label *TCA_NETEM_LOSS_CLG*
- the structure *tc_netem_loss_clg* containing the parameters relative to the correlated loss models: *model* to select a model, *a1*, *a2*, *a3*, *a4*, *a5* that contains the paramters relative to *GI*, *4-state* or *Gilbert-Elliot* models, *logging_level*. The deterministic loss pattern is stored in the array *sequence*, which has length equal to the value of the variable *pattern_length*. The current position in the array is the value of the variable *pattern_index* while the number of repetitions is stored in *pattern_redo*.

```
struct tc_netem_loss_clg /* Correlated Loss Generator parameters */
{
        /* which correlated loss model is used */
        __u8 model;
        /* GI and Gilbert-Elliot models */
        __u32 a1;       /* p13 for GI or p for Gilbert-Elliot */
        __u32 a2;       /* p31 for GI or r for Gilbert-Elliot */
        __u32 a3;       /* p32 for GI or h for Gilbert-Elliot */
        __u32 a4;       /* p14 for GI or 1-k for Gilbert-Elliot */
        __u32 a5;       /* p23 used only in GI */
        /* Deterministic loss generator */
        /* loss sequence: 1 is loss event, 0 is transmission */
        __u32 *sequence;
        /* length of the sequence */
        __u32 pattern_length;
        /* number of repetitions of the sequence */
        __u32 pattern_redo;
        /* points to the current value of the sequence */
        __u32 pattern_index;
        /* Logging parameters *
        /* logging level - set to 1 to enable */
        __u8 logging_level;
};
```

Only in *sch_netem.c* were added:

- the following variables into the struct *netem_sched_*data:

```
/* data for Correlated Loss Generation models */
        u8 model;       /* which correlated loss model is used */
        /* 4-states and Gilbert-Elliot models */
        u32 clg_a1;     /* p13 for 4-states or p for Gilbert-Elliot */
        u32 clg_a2;     /* p31 for 4-states or r for Gilbert-Elliot */
        u32 clg_a3;     /* p32 for 4-states or h for Gilbert-Elliot */
        u32 clg_a4;     /* p14 for 4-states or 1-k for Gilbert-Elliot */
        u32 clg_a5;     /* p23 used only in 4-states */
        /* Deterministic loss generator */
        /* loss sequence: 1 is loss event, 0 is transmission */
        u32 *sequence;
        /* length of the sequence */
        u32 pattern_length;
        /* number of repetitions of the sequence */
        u32 pattern_redo;
        /* points to the current value of the sequence */
        u32 pattern_index;
        /* Logging parameters */
        /* logging level - set to 1 to enable */
        u8 logging_level;
        /* State variables */
        /* state of the Markov chain */
        u8 chain_state;
        /* number of transmissions after the last loss event */
        u32 transmissions;
```

- the deterministic loss generator function: it extracts an element (1 means loss event, 0 means transmission) from the current loss pattern:

```
static int get_loss_pattern_element(struct netem_sched_data *q)
{
   int element=0;
   element=q->sequence[q->pattern_index];
   if (q->pattern_index == (q->pattern_length-1)) {
        if (q->pattern_redo!=0) q->pattern_redo--;
        q->pattern_index=0;
   }
   else
        q->pattern_index++;
   return element;
}
```

- the *loss_4state* generator function: it generates losses according to the 4-state Markov chain adopted in the GI (*General and Intuitive*) loss model. If it returns 1 the next packet will be lost, if it returns 0 it will be transmitted. The function makes a comparision between random_4state and the transition probabilities outgoing from the current state, then decides the next state and if the next packet has to be transmitted or lost. The four states correspond to: successfully transmitted packets within a gap period (State 1), isolated losses within a gap period (State 4), lost packets within a burst period (State 3), successfully transmitted packets within a burst period (State 2).

```
static int get_loss_4state_element(struct netem_sched_data *q) {

   int element = 0; /* 0 means transmission, 1 loss event */
   u32 random_4state = net_random(); /* extracts a random number */
   switch(q->chain_state) {
```

```
    case 1:
        if(random_4state<q->clg_a4) {
            q->chain_state=4;
            element=1;
            break;
        } else if ((q->clg_a4<random_4state)
                &&(random_4state<q->clg_a1)) {
            q->chain_state=3;
            element=1;
            break;
        } else if(q->clg_a1<random_4state) {
            q->chain_state=1;
            element=0;
            break;
        }
    case 2:
        if(random_4state<q->clg_a5) {
            q->chain_state=3;
            element=1;
            break;
        } else {
            q->chain_state=2;
            element=0;
            break;
        }
    case 3:
        if(random_4state<q->clg_a3) {
            q->chain_state=2;
            element=0;
            break;
        } else if((q->clg_a3<random_4state)
                && (random_4state<(q->clg_a2+q->clg_a3))) {
            q->chain_state=1;
            element=1;
            break;
        } else if ((q->clg_a2+q->clg_a3)<random_4state) {
            q->chain_state=3;
            element=1;
            break;
        }
    case 4:
        q->chain_state=1;
        element=0;
        break;
    }
    return element;
}
```

- the *loss_gilb_ell* generator function: it generates losses according to the the *Gilbert-Elliot* loss model or its special cases (*Gilbert* or *Simple Gilbert*). The function makes a comparision between *random_gilb_ell* and the transition probabilities outgoing from the current state, then decides the next state. A second random number is extracted and the comparision with the loss probability of the current state decides if the next packet will be transmitted or lost.

```
static int get_loss_gilb_ell_element(struct netem_sched_data *q) {

    int element = 0; /* 0 means transmission, 1 loss event */
```

```
u32 random_gilb_ell; /* extracts a random number */
switch(q->chain_state) {
case 1:
        random_gilb_ell=net_random();
        if(random_gilb_ell<q->clg_a1)
                q->chain_state=2;
        random_gilb_ell=net_random();
        if(random_gilb_ell<q->clg_a4) {
                element=1;
                break;
        } else {
                element=0;
                break;
        }
case 2:
        random_gilb_ell=net_random();
        if(random_gilb_ell<q->clg_a2)
                q->chain_state=1;
        random_gilb_ell=net_random();
        if(q->clg_a3>random_gilb_ell) {
                element=1;
                break;
        } else {
                element=0;
                break;
        }
}
return element;
```

- the following code to the *netem_enqueue* function. It calls, depending on what model is selected, one of the funtions described above and, if the returned value is 1, drops the packet. If logging is enabled, prints a line in the kernel logs:

```
if ((q->model == NETEM_CLG_DETERMIN) && (q->pattern_redo>1 ||
                                                q->pattern_redo==0)) {
        loss_element=get_loss_pattern_element(q);
        if (loss_element==1) {
                --count;
                if (q->logging_level==1)
                        printk("netem determin loss %d RFPLE %d \n",
                                sch->qstats.drops++,
                                q->transmissions);
                q->transmissions=0;
        } else
                q->transmissions++;
}
if (q->model==NETEM_CLG_4_STATES) {
        /* 4state loss model algorithm (used also for GI model)
        * Extracts a value from the 4-states loss generator,
        * if it is 1 drops a packet and eventually writes the event in
        * the kernel logs
        */
        loss_element=get_loss_4state_element(q);
        if (loss_element==1) {
                --count;
                if (q->logging_level == 1) {
                        if(q->chain_state == 4)
                                printk("netem 4_stat isolated loss %d"
                                        " RFPLE %d \n",
                                        sch->qstats.drops++,
```

```
                                        q->transmissions);

                        else
                                printk("netem 4_stat burst loss %d"
                                        " RFPLE %d \n",
                                        sch->qstats.drops++,
                                        q->transmissions);
                }
                q->transmissions=0;
        } else
                q->transmissions++;

    } else if (q->model==NETEM_CLG_GILB_ELL) {
        /* Gilbert-Elliot loss model algorithm
        * Extracts a value from the Gilbert-Elliot loss generator,
        * if it is 1 drops a packet and eventually writes the event in
        * the kernel logs
        */
        loss_element=get_loss_gilb_ell_element(q);
        if (loss_element==1) {
                --count;
                if (q->logging_level==1)
                        printk("\nnetem gilb_ell loss %d RFPLE %d",
                        sch->qstats.drops+1, q->transmissions);
                q->transmissions=0;
        } else
                q->transmissions++;
    }
```

The function *get_loss_clg* to read the parameters from the userspace:

```
static void get_loss_clg(struct Qdisc *sch, const struct nlattr *attr)
{
        struct netem_sched_data *q = qdisc_priv(sch);
        const struct tc_netem_loss_clg *l = nla_data(attr);

        q->model = l->model;
        q->logging_level = l->logging_level;
        q->transmissions = 0;
        if ((l->model == NETEM_CLG_GILB_ELL)
                || (l->model == NETEM_CLG_4_STATES)) {
        q->clg_a1 = l->a1;
        q->clg_a2 = l->a2;
        q->clg_a3 = l->a3;
        q->clg_a4 = l->a4;
        q->clg_a5 = l->a5;
        q->chain_state = 1;
        }

        if (l->model==NETEM_CLG_DETERMIN) {
        int i = 0;
        q->pattern_length = l->pattern_length;
        q->pattern_redo = l->pattern_redo;
        q->sequence = kmalloc(l->pattern_length*sizeof(size_t),
                GFP_KERNEL);

        for (i=0; i<q->pattern_length; i++) {
```

```
                q->sequence[i] = l->sequence[i];
        }
        q->pattern_index = 0;
        }

}
```

The functions *netem_change* and *netem_dump* were modified to handle the new parameters, calling the function *get_loss_clg* when needed.

```
static int netem_change(struct Qdisc *sch, struct nlattr *opt)
{
…
if (tb[TCA_NETEM_LOSS_CLG])
        get_loss_clg(sch, tb[TCA_NETEM_LOSS_CLG]);
…
}
static int netem_dump(struct Qdisc *sch, struct sk_buff *skb)
{
…
  struct tc_netem_loss_clg clg;
…
        clg.model = q->model;
  clg.a1 = q->clg_a1;
  clg.a2 = q->clg_a2;
  clg.a3 = q->clg_a3;
  clg.a4 = q->clg_a4;
  clg.a5 = q->clg_a5;
  clg.pattern_length = q->pattern_length;
  clg.pattern_redo = q->pattern_redo;
  clg.logging_level = q->logging_level;
  NLA_PUT(skb, TCA_NETEM_LOSS_CLG, sizeof(clg), &clg);
}
```

6.3.2 tc-netem

This Section describes the pieces of code added to *iproute2* package. The modified files are: *pkt_sched.h* in <iproute2_source_path>/include/linux, and *q_netem.c* in <iproute2_source_path >/net/sched . A new file, *tc-netem.8* (which contains the Netem manpage) was added in <iproute2_source_path>/man/man8. Note that <iproute2_source_path> stands for the path where the *iproute2* sources have been unpacked.

The pieces of codes added to implement the new features to *tc* are now shown, explaining them step by step. In file *pktsched.h* were made the same changes seen for the kernel file with the same name. In the file *q_netem.c* the following code was added to the function *netem_parse_opt*:

- logging

```
    } if (matches(*argv, "logging") == 0) {
                        NEXT_ARG();
                        ++present[TCA_NETEM_LOSS_CLG];
                        if (get_size(&clg.logging_level, *argv)) {
                                explain1("logging");
```

```
                return -1;
        }
```

- *query mode*

```
    } if (matches(*argv, "query") == 0) {
        query=1;
```

- *loss_GI*: if the chosen option is loss_GI ("General and Intuitive") the variables relatives to the transition probabilities and the GI parameters are initially set to the default values. Then the mandatory ploss parameters and, if specified, the optional parameters, are read from keyboard and stored to the appropriate variabiles. If the query mode is the 4-state transition probabilities are calculated and printed to the screen together with the correspondent GI parameters. If the query mode is disabled the transition probabilities are passed to the kernel *loss_clg* structure.

```
} else if (matches(*argv, "loss_GI") == 0) {
        double p13=0;
        double p31=1;
        double p32=0;
        double p23=1;
        double p14=0;
        float ploss=0;
        float burst_length=1;
        float rho=100;
        float pisol=0;
        float good_burst_length=1;
        NEXT_ARG();
        ploss=strtod(*argv,(char **)NULL);
        if (NEXT_IS_NUMBER()) {
                NEXT_ARG();
                burst_length=strtod(*argv,(char **)NULL);
                if (NEXT_IS_NUMBER()) {
                        NEXT_ARG();
                        rho=strtod(*argv,(char **)NULL);
                        if (ploss>rho) {
                                printf("\nError: ploss>density");
                                break;
                        }
                        if(NEXT_IS_NUMBER()) {
                                NEXT_ARG();
                                pisol=strtod(*argv,(char **)NULL);
                                if (pisol>ploss) {
                                        printf("\nError: pisol>ploss");
                                        break;
                                }
                                if (ploss>(rho-pisol*(rho-100))) {
                                        printf("\nError: ploss>"
                                                "density-pisol(density-1)");
                                        break;
                                }
                                if(NEXT_IS_NUMBER()) {
                                        NEXT_ARG();
                                        good_burst_length=strtod(*argv,(char
                                                **)NULL);
                                        if(good_burst_length>burst_length) {
                                                printf("\nError:"
                                                        "good burst length"
```

```
                                                        ">burst length");
                                                break;
                                        }
                                }
                        }
                }
        }
        if(burst_length == 0) {
                p13=ploss/100;
                p31=(100-ploss)/100;
        } else if((burst_length!=0) && (rho == 1) && (pisol == 0)) {
                p13=ploss/(burst_length*(100-ploss));
                p31=1/burst_length;
        } else if ((burst_length!=0) && (rho!=1) && (pisol == 0)) {
                p13=(-ploss)/(burst_length*ploss-burst_length*rho);
                p31=100/(burst_length*rho);
        } else if ((burst_length!=0) && (rho!=1) && (pisol!=0)) {
                p13=(100*(pisol-ploss))/
                        (burst_length*(pisol-100)*(rho-ploss));
                p31=100/(burst_length*rho);
                p14=pisol/(100-pisol);
        }
        if((burst_length!=0) && (rho!=100)) {
                if(good_burst_length == 1) {
                        p23=(burst_length*(rho/100)-1)
                                / (burst_length-1);
                        p32=(1+(rho/100)*(rho/100)*burst_length
                                -(rho/100)-burst_length*(rho/100))
                                / ((rho/100)-(rho/100)*burst_length);
                } else {
                        p23=1/good_burst_length;
                        p32=(100-rho) / (rho*good_burst_length);
                }
        }
        if (query == 1) {
                printf("\nThe transition probabilities are:");
                printf("\n--------------------------------");
                printf("\np13 is %.3f%% ", 100*p13);
                printf("\np31 is %.3f%% ", 100*p31);
                printf("\np32 is %.3f%% ", 100*p32);
                printf("\np23 is %.3f%% ", 100*p23);
                printf("\np14 is %.3f%%\n ", 100*p14);
                if (burst_length == 1)
                        burst_length=1/p31;
                printf("\nThe GI (General and Intuitive) parameters will be:");
                printf("\n--------------------------------");
                printf("\nploss is %.3f%% ", ploss);
                printf("\nburst length is %.3f", burst_length);
                printf("\nburst density is %.3f%% ", rho);
                printf("\nisolated ploss is %.3f%% ", pisol);
                printf("\ngood burst length is %.3f\n ", good_burst_length);
                return -1;
        }
        else {
                present[TCA_NETEM_LOSS_CLG] = 1;
                clg.model=1;
                clg.a1=ONE*p13;
                clg.a2=ONE*p31;
```

```
                clg.a3=ONE*p32;
                clg.a4=ONE*p14;
                clg.a5=ONE*p23;
        }
```

- loss_4state:if the selection options is *loss_4_state* the previous considerations stands, but there is no need for the conversion and if the query mode is disabled the input parameters are directly passed to the qdisc,. If the query mode is enabled, the correspondent GI parameters are calculated and printed to screen.

```
} else if (matches(*argv, "loss_4state") == 0){
        double p13=0;
        double p31=100;
        double p32=0;
        double p23=100;
        double p14=0;
        double ploss=0;
        double burst_length=0;
        double rho=100;
        double pisol=0;
        double good_burst_length=1;
        NEXT_ARG();
        p13=strtod(*argv,(char **)NULL);
        if (NEXT_IS_NUMBER()) {
                NEXT_ARG();
                p31=strtod(*argv,(char **)NULL);
                if (NEXT_IS_NUMBER()) {
                        NEXT_ARG();
                        p32=strtod(*argv,(char **)NULL);
                                if (NEXT_IS_NUMBER()) {
                                NEXT_ARG();
                                p23=strtod(*argv,(char **)NULL);
                                        if (NEXT_IS_NUMBER()) {
                                                NEXT_ARG();
                                                p14=strtod(*argv,(char **)NULL);
                                        }
                                }
                        }
        } else
                p31=100-p13;
        if (query == 1) {
                ploss=100*(p13*p23+p14*p23*p31)
                        /(p13*p23+p23*p31+p14*p23*p31+p13*p32);
                burst_length=100*(p32+p23)/(p23*p31);
                rho=100*(p13*p23)/(p13*p23+p13*p32);
                pisol=100*(p14*p23*p31)/(p14*p23*p31+p23*p31);
                good_burst_length=100/p23;
                printf("\nThe transition probabilities are:");
                printf("\n--------------------------------");
                printf("\np13 is %.3f%% ", p13);
                printf("\np31 is %.3f%% ", p31);
                printf("\np32 is %.3f%% ", p32);
                printf("\np23 is %.3f%% ", p23);
                printf("\np14 is %.3f%%\n ", p14);
                printf("\nThe GI (General and Intuitive) "
                        "parameters will be:");
                printf("\n--------------------------------");
                printf("\nploss is %.3f%%", ploss);
```

```
                printf("\nburst length is %.3f", burst_length);
                printf("\nburst density is %.3f%% ", rho);
                printf("\nisolated ploss is %.3f%% ", pisol);
                printf("\ngood burst length is %.3f", good_burst_length);
                printf("\n");
                return -1;
        }
        else {
                present[TCA_NETEM_LOSS_CLG] = 1;
                clg.model=1;
                clg.a1=ONE*(p13/100);
                clg.a2=ONE*(p31/100);
                clg.a3=ONE*(p32/100);
                clg.a4=ONE*(p14/100);
                clg.a5=ONE*(p23/100);
        }
```

- loss_gilb_ell: the Gilbert-Elliot model, or one of its special cases (Gilbert, Simple Gilbert, Bernoulli) will be used In this case there is no need to conversions and no information will be printed.

```
} else if (matches(*argv, "loss_gilb_ell") == 0) {
        NEXT_ARG();
        if(get_percent(&clg.a1, *argv)) {
                explain1("clg.a1");
                return -1;
        }
        if (NEXT_IS_NUMBER()) {
                NEXT_ARG();
                if(get_percent(&clg.a2, *argv)) {
                        explain1("clg.a2");
                        return -1;
                }
                if (NEXT_IS_NUMBER()) {
                        NEXT_ARG();
                        if(get_percent(&clg.a3, *argv)) {
                                explain1("clg.a3");
                                return -1;
                                }
                        if (NEXT_IS_NUMBER()) {
                                NEXT_ARG();
                                if(get_percent(&clg.a4, *argv)) {
                                        explain1("clg.a4");
                                        return -1;
                                }
                        }
                        else
                                clg.a4=0;
                }
                else {
                        clg.a3=ONE;
                        clg.a4=0;
                }
        }
        else {
                clg.a2=ONE-clg.a1;
                clg.a3=ONE;
                clg.a4=0;
```

```
                }
                present[TCA_NETEM_LOSS_CLG] = 1;
                clg.model=0;
```

- loss_gilb_ell_4s: the option is similar to the Gilbert-Elliot one but uses the 4-state generator. To do this, the correspondent transition probabilities and intuitive parameters are calculated. If the query mode is enabled, they are only printed to screen.

```
} else if (matches(*argv, "loss_gilb_ell_4s") == 0) {
                NEXT_ARG();
                if(get_percent(&clg.a1, *argv)) {
                        explain1("clg.a1");
                        return -1;
                }
                if (NEXT_IS_NUMBER()) {
                        NEXT_ARG();
                        if(get_percent(&clg.a2, *argv)) {
                                explain1("clg.a2");
                                return -1;
                        }
                        if (NEXT_IS_NUMBER()) {
                                NEXT_ARG();
                                if(get_percent(&clg.a3, *argv)) {
                                        explain1("clg.a3");
                                        return -1;
                                        }
                                if (NEXT_IS_NUMBER()) {
                                        NEXT_ARG();
                                        if(get_percent(&clg.a4, *argv)) {
                                                explain1("clg.a4");
                                                return -1;
                                        }
                                }
                                else
                                        clg.a4=0;
                        }
                        else {
                                clg.a3=ONE;
                                clg.a4=0;
                        }
                }
                else {
                        clg.a2=ONE-clg.a1;
                        clg.a3=ONE;
                        clg.a4=0;
                }
                ploss=(r-k*r+p-h*p)/(p+r);
                burst_length=(1/(r-r*h))*((h*h*(1-h+2*r*h*h-6*r*h+2*r))+(2*h*(1-h)*(1-
h+r*h*h-3*h*r+r))+((1-h)*(1-h)*(1-h)));
                if(h!=0)
                        rho=(1-h)/(r*burst_length);
                else
                        rho=1;
                if(k!=1)
                        pisol=1-k;
                else
                        pisol=0;
                p13=(pisol-ploss)/(burst_length*(pisol*(rho-1)-rho+ploss));
```

```
                p31=1/(burst_length*rho);
                if (rho<1) {
                        p32=(1+rho*rho*burst_length-rho-burst_length*rho)
                        / (rho-rho*burst_length);
                        p23=(burst_length*rho-1) / (burst_length-1);
                }
                else {
                        p32=0;
                        p23=1;
                }
                if (pisol!=0)
                        p14=(pisol) / (1-pisol);
                else
                        p14=0;
                if (query == 1) {
                        printf("\nThe transition probabilities are:");
                        printf("\n--------------------------------");
                        printf("\np13 is %.3f%% ", 100*p13);
                        printf("\np31 is %.3f%% ", 100*p31);
                        printf("\np32 is %.3f%% ", 100*p32);
                        printf("\np23 is %.3f%% ", 100*p23);
                        printf("\np14 is %.3f%%\n ", 100*p14);
                        if (burst_length == 1)
                                burst_length=1/p31;
                        printf("\nThe GI (General and Intuitive) parameters will be:");
                        printf("\n--------------------------------");
                        printf("\nploss is %.3f%% ", ploss);
                        printf("\nburst length is %.3f", burst_length);
                        printf("\nburst density is %.3f%% ", rho);
                        printf("\nisolated ploss is %.3f%% ", pisol);
                        printf("\ngood burst length is %.3f\n ", good_burst_length);
                        return -1;
                }
                else {
                        present[TCA_NETEM_LOSS_CLG] = 1;
                        clg.model=1;
                        clg.a1=ONE*p13;
                        clg.a2=ONE*p31;
                        clg.a3=ONE*p32;
                        clg.a4=ONE*p14;
                        clg.a5=ONE*p23;
                }
```

- loss_pattern: the textfile with the filename specified as imput will be read and its content (that must be a sequence of “0” and “1”) will be stored in the pattern array.

```
} else if (matches(*argv, "loss_pattern") == 0){
                NEXT_ARG();
                int i;
                FILE *sequence;
                sequence=fopen(*argv, "r");
                if (sequence == NULL) {
                        printf("Could not open the file %s \n", *argv);
                        exit(1);
                }
                fseek(sequence, 0, SEEK_END);
                fgetpos(sequence, &clg.pattern_length);
                rewind(sequence);
                clg.sequence=malloc(clg.pattern_length*sizeof(int));
```

```
                        for(i=1; i<clg.pattern_length; i++) {
                                if(fgetc(sequence)=='1')
                                        clg.sequence[i-1]=1;
                                else
                                        clg.sequence[i-1]=0; }
                        fclose(sequence);
                        if (NEXT_IS_NUMBER()) {
                                NEXT_ARG();
                                if(get_size(&clg.pattern_redo, *argv)) {
                                        explain1("pattern_redo");
                                        return -1;
                                }
                                clg.pattern_redo++;
                        } else
                                clg.pattern_redo=0;
                        present[TCA_NETEM_LOSS_CLG] = 1;
                        clg.model=2;
```

Another pieces of code were added to *netem_print_opt()* function and netem_dump() functions:

```
if (present[TCA_NETEM_LOSS_CLG] &&

     addattr_l(n, 1024, TCA_NETEM_LOSS_CLG, &clg, sizeof(clg)) < 0)

  return -1;


static int netem_print_opt(struct qdisc_util *qu, FILE *f, struct rtattr *opt)
{
…
const struct tc_netem_loss_clg *clg = NULL;
…
if (tb[TCA_NETEM_LOSS_CLG]) {
                        if (RTA_PAYLOAD(tb[TCA_NETEM_LOSS_CLG]) < sizeof(*clg))
                                return -1;
                        clg = RTA_DATA(tb[TCA_NETEM_LOSS_CLG]);
            }
..
if (clg && clg->model==NETEM_CLG_GILB_ELL) {
            fprintf(f, " p %s", sprint_percent(clg->a1, b1));
            fprintf(f, " r %s", sprint_percent(clg->a2, b1));
            fprintf(f, " 1-h %s",
                        sprint_percent(clg->a3, b1));
            fprintf(f, " 1-k %s",
                        sprint_percent(clg->a4, b1));
  } else if (clg && clg->model==NETEM_CLG_4_STATES) {
            fprintf(f, " p13 %s", sprint_percent(clg->a1, b1));
            fprintf(f, " p31 %s", sprint_percent(clg->a2, b1));
            fprintf(f, " p32 %s", sprint_percent(clg->a3, b1));
            fprintf(f, " p23 %s", sprint_percent(clg->a5, b1));
            fprintf(f, " p14 %s",
                        sprint_percent(clg->a4, b1));
  } else if (clg && clg->model==NETEM_CLG_DETERMIN) {
            fprintf(f, " pattern_length %d", clg->pattern_length);
            if(clg->pattern_redo>1)
                        fprintf(f, " pattern_redo %d", (clg->pattern_redo-1));
  }
```

7 Testing the proposed loss model

7.1 Sqngen: loss sequence generator

7.1.1 Overview

Sqngen is a loss sequence generator, developed to evaluate the suitability of the various loss models before implementing them in the new *Netem* version inside the Linux kernel. The program is written in C language and works from line of command. We define a "*loss sequence*" as a sequence of "0" and "1" where "1" represents a loss event and "0" represents a successful transmission event. S*qngen*, on the basis of the input parameters, generates a certain number of sequences (*num_of_samples*), prints them on the screen and makes two checks on each sequence that re-calculates the GI parameters analyzing the generated sequences : an *internal check* that identifies the beginning and the end of the bursts using the tags introduced in the sequences during the generation, , and a *"blind external check"* that analyses the sequence without any additional information. After all, a simple statistical analysis is made an all the generated sequences calculating mean, variance and variation coefficient for the GI parameters. Then, all the relevant data (input parameters, GI parameters calculated by internal and external check, statistical data) are saved to text files.

7.2 Building sqngen

To build *sqngen* it is required to have GNU Scientific Library 1.12 ((http://freshmeat.net/redir/gnuscientificlibrary/3556/url_tgz/gsl-1.12.tar.gz) installed. This is needed because *sqngen* uses Tausworthe random generator contained in the GSL libraries. The code is the same used in *net_random()* or *random32()* function of Linux kernel, that is also used by *Netem*. Once installed GSL, it is possible to compile *sqngen* through *gcc* with the command:

```
g++ -Wall sqngen.c -lgsl -lgslcblas -o sqngen
```

The building was tested with GCC 4.2.4 on Slackware Linux 12.2.

7.2.1 User manual

The syntax for *sqngen* is:

```
$ sqngen num_of_samples length gmin_max model_type model_params
```

where *model_type* can be:

- loss_GI
- loss_4state
- loss_bern
- loss_gilb_4s
- loss_gilb_ell_4s
- loss_gilb

- loss_gilb_ell

while *model_params* is a set of model parameters dependent on the *model_type*, as in the following examples, *num_of_samples* represents the number of sequences to generate, *length* represents the length of each sequence while g_{min_max} is the maximum value of *gmin* used in the external check. To skip the external check, g_{min_max} value has to been set to 0.

```
$ sqngen num_of_samples length gmin_max loss_GI ploss [burst_length [density [pisol [good_
burst_length]]]]
```

this syntax selects GI (*General and* Intuitive) as model and uses its own parameter as input. The *burst_length* and *good_ burst_length* parameters are expressed as integer values while *ploss*, *density* and *pisol* as percentages. The only mandatory parameter is *ploss* that corresponds, alone, to the *Bernoulli* model. If *burst_length* is also specified, it corresponds to the 2-state *Markov o*r to the *Simple Gilbert* model. The *density* parameter extends the model to the 3-state (or the *Gilbert* one) and *pisol* to the 4-state model (or to the *Gilbert-Elliot*). If *good_ burst_length* is not specified, the hypothesis of statistical independence between burst losses will be used. Otherwise, the transition probability between good sub-bursts and loss bursts will be fixed by the user. This is equivalent to specify manually all the five transition probabilities of 4-state model.

```
$ sqngen num_of_samples length gmin_max loss_4state p13 p31 [p32 p23 [p14]]
```

this one selects GI ("General and Intuitive") model using the transition probabilities (expressed as percentages) as input parameters. The two transition probabilities p13 and p31 are mandatory and, if specified alone, corresponds to a 2-state Markov chain. The other parameters are optional and allow the user to consider the 3-state (p32 and p23) or the complete 4-state models (p14).

```
$ sqngen num_of_samples length gmin_max loss_bern p
```

this one selects the *Bernoulli* model. It is needed to specify only the loss probability, expressed as percentage.

```
$ sqngen num_of_samples length gmin_max loss_gilb p r [1-h]

$ sqngen num_of_samples length gmin_max loss_gilb_4s p r [1-h]
```

while loss_gilb option selects the loss generation algorithm based on the Gilbert model, loss_gilb_4s selects the Gilbert-4s model that is a special case of the GI model which behaviour is very similar to the Gilbert's. It is needed to specify two mandatory parameters, expressed as percentages, that represent the transition probabilities between the "Good" and the "Bad" states. The third parameter is the loss probability in the "Bad" state. If *1-h* is not specified, the *Simple Gilbert* model will be used.

```
$ sqngen num_of_samples length gmin_max loss_gilb_ell p r [1-h 1-k]

$sqngen num_of_samples length gmin_max loss_gilb_ell_4s p r [1-h 1-k]
```

the difference between loss_gilb_ell and loss_gilb_ell_4s is the same described above for loss_gilb and loss_gilb_4s. In this case the options select, respectively, the Gilbert-Elliot or the Gilbert-Elliot-4s models. The parameters are the same of the *Gilbert* model adding *1-k* that is the loss probability

when the system is in the *Good* state. If *1-k* is not specified, the *Gilbert* model will be used. If *1-h* too is not specified, the *Simple Gilbert Model* will be used.

The relations between options and model are shown in the following table:

OPTION	MODEL	MANDATORY PARAMETERS	OPTIONAL PARAMETERS
loss_GI	GI (General and Intuitive)	P_{LOSS}, *E(B)*	ρ, P_{ISOL}, *E(GB)*
loss_4state	GI (General and Intuitive) using 4-state transition probabilities (with special cases: 3-state, 2-state)	p_{13}, p_{31}	[p_{23}, p_{32}], p_{14}
loss_bern	Bernoulli	*p*	N/A
loss_gilb	Gilbert (with Simple Gilbert as special case)	*p*, *r*	*1-h*
loss_gilb_ell	Gilbert-Elliot (with special cases: Gilbert, Simple Gilbert)	*p*, *r*	*1-h*, *1-k*
loss_gilb_4s	Gilbert-4s (with Simple Gilbert as special case)	*p*, *r*	*1-h*
loss_gilb_ell_4s	Gilbert-Elliot-4s (with special cases: Gilbert, Simple Gilbert)	*p*, *r*	*1-h*, *1-k*

Table 4: Relations between options and models in sqngen

7.2.2 How it works: code description

7.2.2.1 sqngen.c structure

The program is composed of six functions:

- *sqn_gen*: is the GI sequence generator. It returns a sequence of "0" and "1" according to the transition probabilities.
- *gilb_ell_gen*: is the Gilbert-Elliot sequence generator.On the basis of the p,r, 1-h and 1-k parameters, returns a sequence of "0" and "1".
- *internal_check*: estimates the five GI (General and Intuitive) analyzing the generated sequence and using the "S" and "E" tags that defines the beginning and the end of the bursts.

- *external_blind_check*: estimates the five GI (General and Intuitive) analyzing the generated sequence without using any extra information. The check is repeated using different values of g_{min}, which is the maximum number of ones between two zeros that identifies a burst.
- *statistics*: calculates mean, variance and coefficient of variation for the five GI (General and Intuitive) parameters.
- *arg_error*: manages the errors concerning argument number or wrong format.
- *main*: is the main function, that is executed when the program is launched. It manages the input parameters parsing and calls the other functions.

The functions will be now described one by one.

7.2.2.2 Sqn_gen function (GI sequence generator)

*void sqn_gen(gsl_rng * q, char* sequence, int length, double p13, double p31, double p32, double p23, double p14)*

This function is a sequence generator based on the GI (*General and Intuitive*) loss model. It is implemented through a *for* loop repeated for a number of times equal to the length of the sequence. The first step is the extraction of a pseudorandom number with uniform distribution. This is made through the *gsl_rng_get* function of the GNU Scientific Library 1.5 that implements a *Tausworthe* generator. The generated number is stored in the variable *GI_random*. The algorithm enters in a *switch* construct controlled by the variable *state* that assumes values between 1 and 4 depending on the state where the system is.

The *switch* construct uses the following series of variables:

- *state*: identifies the current state. Its default value is "1". At each step it is updated according to the result of the comparision between the last random number extracted and the transition probabilities to the other states ("2","3" or "4").
- *tags: counts the number of 'S' and 'E' tags that defines the beginning and the end of each burst.The number of iterations of the loop is length+tags.*
- *i: counts the iterations*
- *length: length of the sequence. It doesn't include the tags.*

In particular:

- If the system is in State 1: a '0' is appended to the sequence and printed to the screen. The variable *GI_random* is compared with p_{13}. If *var_random*<p_{13} there is a transition to State 3. In this way we have a loss event and we are entering in a new burst, so an 'S' tag is inserted before the first '1' and the variable *tags* is incremented. The "*i*" counter is also incremented to avoid to overwrite it at the next step. If *var_random*>p_{13} but *var_random*>1-p_{14} the system goes to State 4 and we have an isolated loss event If none of the two conditions is verified, system stays in State 1.
- If the system is in State 2: a '1' is appended to the sequence and printed to the screen . The variable *GI_random* is compared with p_{23}. If *var_random*<p_{23} the system goes to State 3.
- If the system is in State 3: a '0' is appended to the sequence and printed to the screen . The

variable *GI_random* is compared with p_{23}. If *var_random*<p_{32} the system goes to State 2 .If *var_random*<p_{31} the system comes back to State 1 and a 'E' tag is appended to the sequence to mark the end of the burst. The "*i*" counter is also incremented to avoid to overwrite it at the next step. If no one of the two condition is satisfied, so *var_random*>(p_{31}+p_{32}), the system stays in State 3 and we have a further loss event.

- If the system is in State 4: a '0' is appended to the sequence and printed to the screen .Since this is an isolated loss event, the system goes to State 1.

The following figure shows summarizes how the algorithm works:

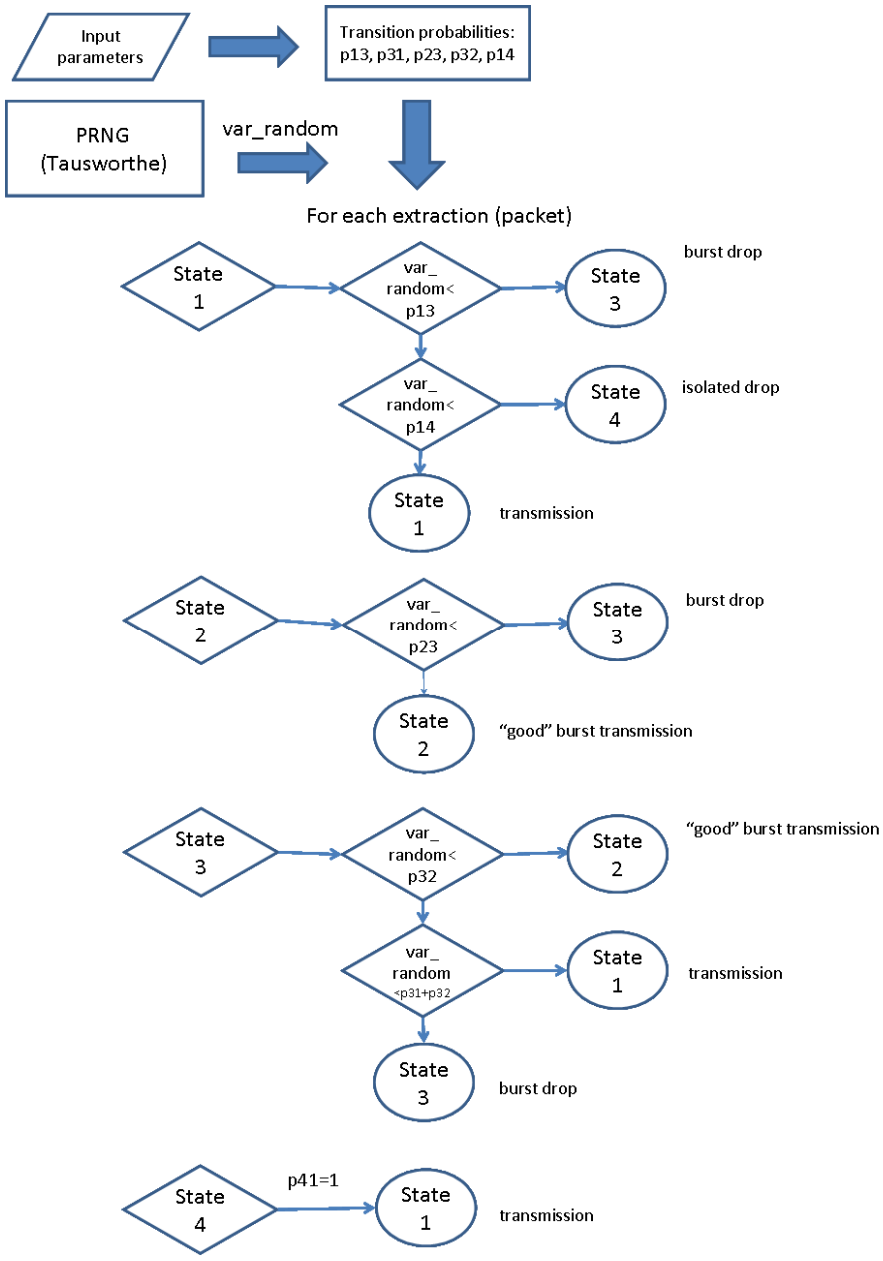

Figure 7: GI loss generator algorithm

At this point, two different checks are performed on the parameters of the generated sequence.

7.2.3 Gilb_ell_gen_function (Gilbert-Elliot sequence generator)

*void gilb_ell_gen(gsl_rng * q, char* sequence, int length, double p, double r, double k, double h)*

- This function is a sequence generator based on the *Gilbert-Elliot* loss model (including as special cases the *Gilbert* and the *Simple Gilbert* models). As seen for the *sqn_gen* function, it is implemented through a *for* loop, repeated for a number of times equal to the length of the sequence. A pseudorandom number is extracted and stored in the *gilb_ell_random* variable. The *switch* construct is still controlled by the variable *state* which values can be 1 or 2.

At each extraction, *gilb_ell_random* is compared with the loss probability in the current state (that is *1-k* for state 1 and *1-h* for state 2). Then there is a second comparision with the probability of going out from the current state (that is *p* for state 1 and *r* for state 2).

7.2.3.1 Internal_check function

*void internal_check(int length, char sequence[], int sample, double *ploss_bis, double *mean_length_of_burst_bis, double *density_bis, double *pisolated_loss_bis, double *mean_length_of_good_burst_bis)*

This function performs a first check on the generated sequence, calculating the GI parameters using the variables increased during the generating process described in the previous section. The function gets in input the length of the generated sequence (*length)*, the sequence identification number (*sample*) within the series *num_of_samples* and the pointers to the arrays where the respective parameters will be stored. The sequence is analyzed looking through each element one by one. The following variables are used:

- gilb_ell_burst: *boolean variabile. If set to "true" we are in a burst condition according to Gilbert-Elliot.This corresponds to the "Bad" state of Gilbert-Elliot. If the burst doesn't contain any loss event it is not a burst according to RFC3611.*
- real_burst: boolean variable. *if set to "true" we are in a burst condition after we have had a loss event after entering in a Gilbert-Elliot burst.*
- good_burst: Boolean_variable. *if set to "true" we are in a "good sub-burst" so we have had some transmission in a real burst.*
- num_of_bursts_bis: *number of bursts.*
- length_of_all_bursts_bis: *sum of the length of all the bursts we have had.*
- num_of_drops_bis: *total number of loss events.*
- num_of_burst_drops_bis: *number of burst loss events.*
- num_of_isolated_drops_bis: *number of isolated loss events.*
- initial_zeros: *counts the zeros at the beginning of the current Gilbert-Elliot burst*
- final_zeros: *counts the zeros at the end of current Gilbert-Elliot burst*
- ones:*counts the ones (loss events) in the current burst*
- tags: *counts the "S" and "E" tags that marks the beginning and the end of the bursts*

if we find an 'S' tag we are at the beginning of a burst according to Gilbert-Elliot, so we set gilb_ell_burst=true and the value of num_of_bursts is incremented. If we find 'E' we are at the end

of the burst so we set to false all the boolean variables relative to being into a burst: gilb_ell_burst, real_burst and good_burst. Moreover we remove from the length of the bursts the number of initial and final zeros. If we are in a Gilbert-Elliot burst without having had any loss event and we find a '1', then we enter in a real burst and set real_burst=true. If we are in a real burst and find a '1' or '0', length_of_all_bursts is incremented. If we found a '0' and we still haven't had losses, this is an initial zeros. Otherwise, it could be a final zero or an element of a "good sub-burst", so both final_zeros and length_of_all_good_bursts are incremented. IF we found a '1' we are sure that we are in a real burst, so we set real_burst=true and good_burst=false because we are also sure of being out of any good bursts. In this case the previous zeros were not final zeros but elements of a good_bursts, so we set final_zeros=0.

At this point we use the following formulas:

- $\text{ploss} = 100\dfrac{\text{num\_of\_drops\_bis}}{\text{length}}$
- $\text{mean\_length\_of\_burst} = \dfrac{\text{length\_of\_all\_bursts\_bis}}{\text{num\_of\_bursts\_bis}}$
- $\text{density} = 100\dfrac{\text{num\_of\_burst\_drops\_bis}}{\text{length\_of\_all\_bursts\_bis}}$
- $\text{pisol} = 100\dfrac{\text{num\_of\_isolated\_drops\_bis}}{\text{length}}$
- $\text{mean\_length\_of\_good\_burst} = \dfrac{\text{length\_of\_all\_good \_bursts\_bis}}{\text{num\_of\_good\_bursts\_bis}}$

where *ploss* is the ratio between the number of losses and the length of the sequence, *pisol* is the ratio between the isolated losses and the length of the sequence, *mean_length_of_burst* is the ratio between the number of losses within the *bursts* and the sum of the length of all *bursts* in which the system have been during the sequence generation process. The m*ean_length_of_good_burst* formula is analogue to the previous one. Once have calculated the parameters, they will printer to screen, stored in the apposite arrays in sample position, and save to text files, as we will seen ahead.

7.2.3.2 External blind check function

*void external_blind_check (int length, int gmin_max, int sequence[], int sample, float *ploss_ter, float *mean_length_of_burst_ter, float *density_ter, float *pisolated_loss_ter, float *mean_length_of_good_burst_ter)*

In this second check the GI parameters are calculated performing an analysis of the sequence, looking through the array where it was stored. We call this “external” because the check is done outside of the generation algorithm, and also “blind” because it leaves aside from any information different from the sequence itself. The function gets in input the length of the sequence (*length)*, the maximum value of g_{min} to use in the check (*gmin_max*), the *array* containing the sequence

(*sequence*[]), the pointers to the arrays where, similarly to the case of internal check, the estimated values will be stored.

The *ploss* estimation is made counting the number of "1" in the sequence. This is trivial, while for the other parameter the question is very tricky because they involve the concept of *burst*. Since we haven't any extra information except the sequence itself, it is needed to use a strict and not ambiguous definition of what is a *burst*. We will use the definition presented in RFC3611, that is:

A burst is defined, in terms of a value g_{min}*, as the longest sequence that starts with a lost or discarded packet, does not contain any occurrences of* g_{min} *or more consecutively received (and not discarded) packets, and ends with a lost or discarded packet.*

Then, estimated values are different choosing different values of $g_{min.}$ We will now describe the various step of the algorithm.

A new *boolean* variable, named *this_is_burst*, is introduced. Its values are *true* when we are into a *burst* and false when we are outside. The check is repeated through a *for* loop using different values of g_{min}, obtained using a base value (2) and doubling it at each instance of the loop, until the maximum value $g_{min\ \_max}$ is reached. There is a second *for* loop where the analysis is performed. Two situations are distinguished, depending on the values of *this_is_burst variable*. In both cases the sequence analysis is done comparing two successive values (kth and k+1th) at each time. The following figure shows how the algorithm works:

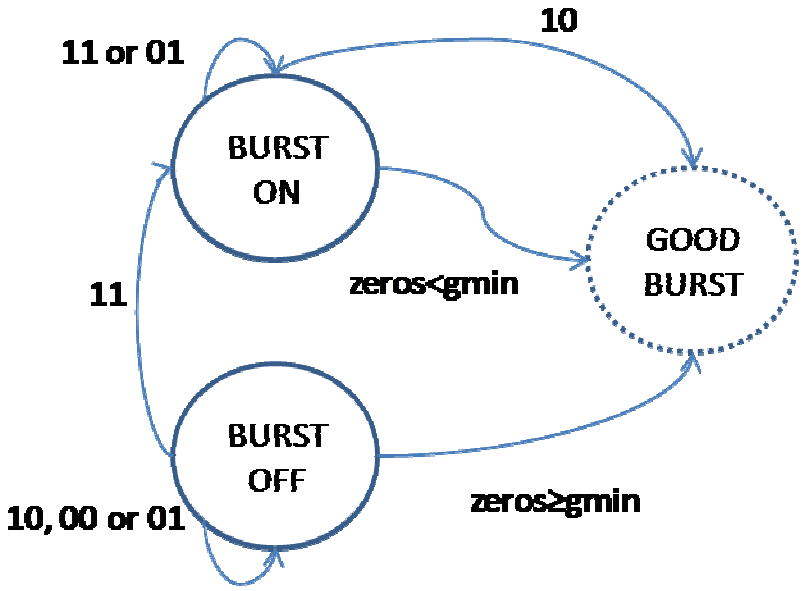

Figure 8: "Blind" external check algorithm

- If *this_is_burst=true*, so we are into the *burst*, we can have three different situations:
  - If k-th value is 0 we stays into the *burst* without a loss event, so only the variable *length_of_all_bursts* is increased;
  - If k-th value is 1 and the following is still 1 we have a loss event within the *burst* so the variables *num_of_drops*, *num_of_burst_drops* and *length_of_all_bursts* are increased;
  - If k-th value is 1 and the following is 0 (and the sequence contains at least other g_{min} values) there is another loss within the *burst,* so the variables *num_of_drops*,

num_of_burst_drops, *length_of_all_bursts* are increased. At this point, according to the definition, the 1 could be the end of the *burst*. To check if this is true, a further *for* loop analyzes the following *gmin* values and if they all assume the value 0 the 1 is really the end of the burst. In this case the variable *this_is_burst* is set to *false.* Note that this check is performed only if the sequence contains at least g_{min} values following the 1 because it would be senseless to check the presence of a certain number of 0 in the sequence when arriving to the end of the sequence itself.

- If *this_is_burst=false*, so we are outside the *burst*:
  - If k-th value is the last of the sequence and is a 1 we have an isolated loss, so the variables *num_of_drops* and *num_of_isolated_drops* are increased.
  - If k-th value is 1and the following is still 1, according to the definition, we are entering in a new *burst* and we have its first loss event. The values of the variable I *this_is_burst* is set to *true*, and the variable *num_of_drops*, *num_of_burst_drops, length_of_all_bursts, num_of_bursts are increased*.
  - If the k-th value is 1 and the following is 0 we have a isolated loss event, so the variables *num_of_drops* and *num_of_isolated_drops* are increased.

At this stage the GI parameters are calculated through the formulas seen for the internal check's case. Then they are printed to screen, stored in the *arrays* and saved to text files. Inside the *arrays* every value is put at distance *sample*j+i-1* from the other, where $i = \log_2(\frac{gmin}{2}) + 1$ and $j = \log_2(\frac{gmin\_max}{2}) + 1$. In this way we use unique one-dimensional array instead of a two-dimensional *array* in which the index *i,j* would represent the values of *sample* and *gmin*.

7.2.3.3 Statistics function

void statistics(int num_of_samples, float ploss[], float mean_length_of_burst[], float density[], float pisolated_loss[], float mean_length_of_good_burst[], int gmin, int gmin_max)

The *statistics* function characterizes the generated sequences in statistic terms, calculating the following estimators:

Sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

Sample variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

Standard deviation (from samples) $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

Variation coefficient $c_x = \frac{s}{|\bar{x}|}$

The function gets in input: *num_of_samples*, the arrays with the GI parameters calculated by one of the two checking methods, the g_{min_max} that it the maximum value of g_{min} considered in external check and the g_{min} value for which the statistics have to been calculated. If we want to calculate the statistics for internal check, the function has to been called fixing at 2 both the field relative to g_{min} and g_{min_max}. Finally, the statistics are printed to screen and saved in *statistics.dat*.

7.2.3.4 Main function

The *main* function, as first stage, saves the input parameters in the apposite variables. If the Gilbert or the Gilbert-Elliot models are selected, the *gilb_ell_gen* function will be used. Otherwise (*loss_4state, loss_GI, loss_bern, loss_gilb_4s, loss_gilb_ell_4s* options) *sqn_gen* will be used. In this case, if the input values are the transition probabilities, this is made in directly while if the user specifies the GI parameters or the *Bernoulli*, *Gilbert-4s*, *Gilbert-Elliot-4s* parameters they are mapped to the transition probabilities. At this stage, after having defined the needed variables and *arrays*, the pseudo-random number generator is initialized by the function *gsl_rng_set(r, time(NULL))*. Then we enter a *for* loop that is repeated for a number of times equal to the number of sequences to generate. Inside the loop the input parameters are saved to the file *data.dat*, and then the functions *sqn_gen*, *internal_check and external_check* are called. At this point, through the *statistics* function mean, variance and variation coefficient are calculated. The function is called many times in another for loop using different values of g_{min} until g_{min_max}.

7.2.3.5 Output

The program prints on the screen all the generated sequences, together with the parameters calculated with the two checks, as in the following example:

```
bash-3.1$ ./sqngen 2 1000 4 loss_GI 2 10 80 2

Transition probabilities are:
 p13=0.000000%
 p31=12.500000%
 p32=19.444445%
 p23=77.777778%
 p14=2.040816%

The sequence is:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010001000000000010000000
00000000000000000000000000000000000000000000000000000000000000000000000000000001010000001000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000010000000000001000000000000000000000000000000000000000000000000010000000000000000000000
00000000000000000000000000000000000000000001000000000001000000000000000000000000000000000000000000000000000100000000000000100000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

The sequence has been generated using the loss_GI option with the following input parameters:
P_loss 2.000% Mean_length_of_burst 10 Density 80.000% P_isol 2.000% Mean_length_of_good_burst 0

The generated sequence has the following characteristics (from the generator):
P_loss 1.400% Mean_length_of_burst 0.000 Density nan% P_isol 1.400% Mean_length_of_good_burst 0.000

For gmin=2 the generated sequence has the following characteristics (from sequence analysis):
P_loss 1.400% Mean_length_of_burst 0.000 Density nan% P_isol 1.400% Mean_length_of_good_burst 0.000

For gmin=4 the generated sequence has the following characteristics (from sequence analysis):
P_loss 1.400% Mean_length_of_burst 0.000 Density nan% P_isol 1.400% Mean_length_of_good_burst 0.000

The sequence is:
11111111111111111111111111111111111111111111111111111111111111111101111111111111111111111111111111111111111111111111111111111111111111110111111111111
11111111111111111101111111111111111111111111111111111111111111111110111111111111111111111111111111111111111111111111111111111111101111111111111
11110111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111101111111111111111111111111111111111111111101111111111111111101111111111111111111111111111111111111111110111111111111111111111111
11111111111111111111111111111111111110111111111111111111111111111111111111011111111111111111111111111111011111011111111111111111111111111111111
11111111111111111111101111111111111111011011111111111111111111111101111111111111111111111111111111111111111111111111111111101111111111111111111111
1111111101111111111111111111111111111111111111111111111111111111111110111111111111101111111111111111111111111111111111111111111111111101111111

The sequence has been generated using the loss_GI option with the following input parameters:
P_loss 2.000% Mean_length_of_burst 10 Density 80.000% P_isol 2.000% Mean_length_of_good_burst 0
```

```
The generated sequence has the following characteristics (from the generator):
P_loss 2.300% Mean_length_of_burst 0.000 Density nan% P_isol 2.300% Mean_length_of_good_burst 0.000

For gmin=2 the generated sequence has the following characteristics (from sequence analysis):
P_loss 2.300% Mean_length_of_burst 0.000 Density nan% P_isol 2.300% Mean_length_of_good_burst 0.000

For gmin=4 the generated sequence has the following characteristics (from sequence analysis):
P_loss 2.300% Mean_length_of_burst 0.000 Density nan% P_isol 2.300% Mean_length_of_good_burst 0.000

INTERNAL CHECK STATISTICS

These are the mean values:
ploss 1.850% mean_length_of_burst 0.000 density nan% pisol 1.850% mean_length_of_good_burst 0.000

These are the variance values:
ploss 0.405 mean_length_of_burst 0.000 density nan pisol 0.405 mean_length_of_good_burst 0.000

These are the standard deviation values:
ploss 0.636% mean_length_of_burst 0.000 density nan% pisol 0.636% mean_length_of_good_burst 0.000

These are the variation coefficients:
ploss 0.344 mean_length_of_burst nan density nan pisol 0.344 mean_length_of_good_burst nan

EXTERNAL CHECK STATISTICS

For gmin=2
These are the mean values:
ploss 1.850% mean_length_of_burst 0.000 density nan% pisol 1.850% mean_length_of_good_burst 0.000

These are the variance values:
ploss 0.405 mean_length_of_burst 0.000 density nan pisol 0.405 mean_length_of_good_burst 0.000

These are the standard deviation values:
ploss 0.636% mean_length_of_burst 0.000 density nan% pisol 0.636% mean_length_of_good_burst 0.000

These are the variation coefficients:
ploss 0.344 mean_length_of_burst nan density nan pisol 0.344 mean_length_of_good_burst nan

For gmin=4
These are the mean values:
ploss 1.850% mean_length_of_burst 0.000 density nan% pisol 1.850% mean_length_of_good_burst 0.000

These are the variance values:
ploss 0.405 mean_length_of_burst 0.000 density nan pisol 0.405 mean_length_of_good_burst 0.000

These are the standard deviation values:
ploss 0.636% mean_length_of_burst 0.000 density nan% pisol 0.636% mean_length_of_good_burst 0.000

These are the variation coefficients:
ploss 0.344 mean_length_of_burst nan density nan pisol 0.344 mean_length_of_good_burst na
```

Data are also saved in two text files: *data.dat* e *statistics.dat*. Every line of *data.dat* has the following format:

- If the input parameters are the transition probabilities:
  sample;num_of_samples;length;gmin_max;loss_markov;p31;p31;p32;p23;p14;
  ploss_bis;duration_bis;density_bis;
  gmin_1;pisol_bis;ploss_ter;duration_ter;density_ter;pisol_ter;
  gmin_2;pisol_bis;ploss_ter;duration_ter;density_ter;pisol_ter...
- If the input parameters are the GI parameters:
  sample;num_of_samples;length;gmin_max;loss_phys;ploss;duration;density;pisol;
  good_duration;ploss_bis;duration_bis;density_bis;pisol_bis;good_duration_bis;
  gmin_1;ploss_ter;duration_ter;density_ter;pisol_ter;good_duration_ter;
  gmin_2;ploss_ter;duration_ter;density_ter;pisol_ter;good_duration_ter;...
- If the input parameters are the parameters of Bernoulli, Gilbert or Gilbert-Elliot model, we will have, compared to the previous case, *loss_bern, loss_gilb* or *loss_gilb_ell* instead of loss_phys and the opportune parameters *p*, *p* and *r*, *p r 1-h 1-k* instead of *ploss, duration, density, pisol, good_duration*.

The variables have the same meaning we have just explained, the -bis suffix are relative to the parameters calculated by the internal check while the -ter suffix are relative to the parameters calculated by the external check. Each one of the various g_{min}, g_{min2} is relative to a different value of g_{min} used in external check. This is needed because the external check is repeated for different

values of gmin. *Sample* is the sequence which the values are relative (for example *sample*=1 and *num_of_samples*=10 means that we are considering the first of ten generated sequence).

The lines of *statistics.dat* have the following format:

- internal;N/A;ploss_mean;mean_length_of_burst_mean;density_mean;pisol_mean; mean_length_of_good_burst_mean;ploss_variance;mean_length_of_burst_mean; density_mean;pisol_variance; mean_length_of_good_burst_variance; ploss_stddev;mean_length_of_burst_stddev;density_stddev;pisol_stddev; mean_length_of_good_burst_mean_stddev;ploss_cx;mean_length_of_burst_cx; density_cx;pisol_cx; mean_length_of_good_burst_cx
- external;gmin;ploss_mean;mean_length_of_burst_mean;density_mean;pisol_mean; mean_length_of_good_burst_mean;ploss_variance;mean_length_of_burst_mean; density_mean;pisol_variance;mean_length_of_good_burst_variance; ploss_stddev;mean_length_of_burst_stddev;density_stddev;pisol_stddev; ploss_cx;mean_length_of_burst_cx;density_cx;pisol_cx; mean_length_of_good_burst_cx

where "internal" and "external" identifies the check method. Note that *gmin* is not used in internal check that is unique. However, to maintain the same format for all the lines, we will show N/A in g_{min} fields relative to internal check.

The example in Section 7.2.3.5 leads up to the following lines of *data.dat* and *statistics.dat:*

1;2;1000;4;loss_phys;2;10;80;2;N/A;2.000000;0.000000;nan;2.000000;0.000000;2;2.000000;0.000000;nan;2.000000;0.000000;4;2.000000;0.000000;nan; 2.000000; 0.000000;

2;2;1000;4;loss_phys;2;10;80;2;N/A;2.400000;0.000000;nan;2.400000;0.000000;2;2.400000;0.000000;nan;2.400000;0.000000;4;2.400000;0.000000;nan; 2.400000; 0.000000;

intenal;N/A;2;1000;loss_phys;2.000000;10.000000;80.000000;2.000000;N/A;2.200000; 0.000000;nan;2.200000;0.000000;0.080000;0.000000;nan;0.080000;0.000000;0.282843;0.000000; nan;0.282843;0.000000;0.128565;nan;nan;0.128565;nan;

external;2;2;1000;loss_phys;2.000000;10.000000;80.000000;2.000000;N/A;2.200000; 0.000000;nan;2.200000;0.000000;0.080000;0.000000;nan;0.080000;0.000000;0.282843;0.000000; nan;0.282843;0.000000;0.128565;nan;nan;0.128565;nan;

external;4;2;1000;loss_phys;2.000000;10.000000;80.000000;2.000000;N/A;2.200000; 0.000000;nan;2.200000;0.000000;0.080000;0.000000;nan;0.080000;0.000000;0.282843;0.000000; nan;0.282843;0.000000;0.128565;nan;nan;0.128565;nan;

Note that "*nan"* means "not applicable".

7.3 Statistical analysis of generated loss sequences

In this section we evaluate the performance of our loss generator considering the size of the confidence interval where the generated P_{LOSS} values fall. We will compare the statistical values (mean, variance and coefficient of variation) obtained from theoretical consideration with experimental values, calculated using *sqngen*'s internal check after having generated a series of

sequences with certain input parameters. The aim of this part of the job is to estimate if and how the generator can be considered well enough for our purposes. The computation of the standard deviation (and so, of the variation coefficient) depends on the considered the loss model. For the Bernoulli model, we will use the mean and variance tests to validate our algorithm.

7.3.1 Mean and variance tests

The mean test (with unknown variance), as explained in [13], is based on the parameter $Q_\mu = \frac{\overline{X} - \eta_0}{S/\sqrt{n}}$ which is distributed as a *Student* variable. So, chosen a level of significance α, the hypothesis H_0 which states sample mean is η_0 is not rejected if Q falls into the interval between the two percentiles $t^2_{\alpha/2}(n-1)$ and $t^2_{1-\alpha/2}(n-1)$ where (n-1) are the degrees of freedom. Since variance value is unknown, we are estimating it from samples. The variance test (with unknown mean) is based on the parameter $Q_\sigma = \sum_{i=1}^{n} \frac{(X_i - \overline{X})^2}{\sigma_0^2}$ which is distributed as Chi-Square. So, chosen a level of significance α, the hypothesis H_0 which states sample variance is σ_0 is not rejected if Q falls into the interval between the two percentiles $\chi^2_{\alpha/2}(n-1)$ and $\chi^2_{1-\alpha/2}(n-1)$ where (n-1) are the degrees of freedom. Since mean value is unknown, we are estimating it from samples.

7.3.2 Statistics for the Bernoulli model

First of all, we consider the Bernoulli model. In this situation we extract *n* values which represent *n* packets: part of them is lost with probability P_{Loss}. This can be described by a binomial random variable where the parameters are $p = P_{Loss}$ and *n*=length of the sequence. As known, binomial random variable has mean $E[x] = np$ and variance $Var[x] = npq$. Considering a series of sequences, we can estimate mean, variance and coefficient of variation of loss events. In Table 5 we show the results of a simulation performed through our *sqngen* generator using *loss_bern* option, and so specifying only the P_{LOSS} parameter. The simulation consists in the generation of 50 sequences of length 200000 with P_{LOSS} =2%. The first line of the table shows the input parameters and the statistics (mean and variance) obtained both from the theory and from internal check. The second and the third line, respectively, shows the parameters used in mean and variance test. As we can see, both the test are satisfied with a significance level α=5%.

SAMPLES	LENGTH	PLOSS	BINOMIAL MEAN	BINOMIAL VARIANCE	$\overline{X}$	$\overline{X}_{\text{variance}}$
	n	p	np	npq		
40	200000	2%	4000	3920	4005,65	3062,6
α	α/2	1-(α/2)	n-1	$\chi^2_{\alpha/2}$	$\chi^2_{1-\alpha/2}$	Q_μ
0.05	0.025	0.975	39	23.654325	58.120060	0,645

α	α/2	1-(α/2)	n-1	$t_{\alpha/2}$	$t_{1-\alpha/2}$	Q_σ
0.05	0.025	0.975	39	23.654325	58.120060	30,470

Table 5: Mean and variance test for Bernoulli model

7.3.3 Mean, variance and coefficient of variation for the Gilbert-Elliot model

We calculate the coefficient of variation (CV) using the equations presented in [12]. In this job the authors adopt a *Gilbert-Elliot* loss model which formally is a 2-state model, respectively named *Bad* (B) and *Good* (G). We can have loss events in both the states with probability *1-k* and *1-h* respectively. If we set *h*=0 and *k*=1 we obtain a simple 2-state *Markov* chain where B state represents only correctly received packets and G state represents only loss packets. The formula is obtained starting from the moment generating function $G_N(z)$ and $B_N(z)$ where N is the length of the sequence. While in steady state we have $G_0(z)=p_1$ and $G_1(z)=p_3$ where p_1 and p_3 are the state probabilities, $G_{N+1}(z)$ can be derived with a iterative procedure taking into account the state transition.

$$G_{N+1}(z) = (1 - p_{13})zG_n(z) + p_{13}zB_n(z)$$

and similarly for $B_{N+1}(z)$. The *k*-moments are derived by the relation $E\left[X^K\right] = \frac{\partial}{\partial z} X(z)\Big|_{z=1}$. So it is possible to obtain an explicit solution for mean, variance and coefficient of variation of loss events. In particular the expression for the coefficient of variation is very simple:

$$c_v(N) = \frac{1}{\sqrt{N}} \sqrt{\frac{hp + kr}{(1-h)p + (1-k)r} + \frac{2pr(1-p-r)(h-k)^2}{[(1-h)p + (1-k)r]^2 (p+r)} \left[1 - \frac{1-(1-p-r)^N}{N(p+r)}\right]}$$

Equation 1: Coefficient of variation for Gilbert-Elliot model

Since mean is $\mu_N = N \cdot P_{LOSS} = N[(1-k)\frac{r}{p+r} + (1-h)\frac{p}{p+r}]$ we can find the expression for the variance:

$$\sigma^2{}_N = [c_v \cdot \mu_N(N)] =$$

$$N[(1-k)\frac{r}{p+r} + (1-h)\frac{p}{p+r}]^2 \left\{ \frac{hp+kr}{(1-h)p+(1-k)r} + \frac{2pr(1-p-r)(h-k)^2}{[(1-h)p+(1-k)r]^2 (p+r)} \left[1 - \frac{1-(1-p-r)^N}{N(p+r)}\right] \right\}$$

For the "Simple Gilbert" model with the choices: $p=p_{13}$, $r=p_{31}$, *k*=1 and *h*=0 we get the simplified expressions for the mean, cV and variance:

$$\mu_N = N \cdot P_{LOSS} = N(\frac{p}{p+r})$$

$$c_v(N) = \frac{1}{\sqrt{N}} \sqrt{\frac{p_{31}}{p_{13}} + \frac{2p_{13}p_{31}(1-p_{13}p_{31})}{{p_{13}}^2(p_{13}p_{31})} \left[1 - \frac{1-(1-p_{13}-p_{31})^N}{N(p_{13}+p_{31})} \right]}$$

$$\sigma^2{}_N = [c_v \cdot \mu_N(N)] = N(\frac{p}{p+r})^2 \left\{ \frac{r}{p} + \frac{2pr(1-p-r)}{p^2(p+r)} \left[1 - \frac{1-(1-p-r)^N}{N(p+r)} \right] \right\}$$

Note that, as explained in Section 4.4 of [11], with *h*=0 and *k*=0 the *Markov* chain collapses to a single state. That is equivalent to a *Bernoulli* model.

7.3.4 Statistics for the GI model using 2-states

Now we show data from a simulation performed using the 2-state model, using the *loss_GI* option of *sqngen* and setting the two parameters P_{LOSS} and E(B). This is equivalent to consider a "Simple Gilbert" or a 2-state Markov model. We can't describe any longer this situation through a binomial random variable, in fact we don't have independent losses but we are expressly stating that the losses are consecutive within a *burst* period which length is 10. So we will calculate the theoretical values for mean, variance and cV using the formulas shown in the previous section. The following table shows a comparison between theoretical and experimental parameters, for different values of *n*, P_{LOSS} and E(B). The coefficients of variation of the generated sequences are always similar to the theoretical ones, but the accuracy is worst if we consider smaller values of loss probability. Changing E(B) value doesn't effect the results heavily. The simulation is made of 50 sequences which length is 200000.

PLOSS	E(B)	p	r	LOSS MEAN (theory)	LOSS VARIANCE (theory)	LOSS cV (theory)	LOSS MEAN (generated)	LOSS VARIANCE (generated)	cV (generated)
2.00%	5	0.00408	0.2	4000,	34495,251	0.04643	1,9929	0,00792	0,044663
2.00%	10	0,00204	0.1	4000	72908.619	0.06750	1.9905	0.02015	0.071317
2.00%	20	0,00102	0.05	4000	149729.709	0.09673	1.9832	0.03879	0.099307
1.00%	5	0.00202	0.2	2000	17621.613	0.06637	0.9896	0.00419	0.065373
1.00%	10	0.00101	0.1	2000	37222.255	0.09646	0.9809	0.00922	0.09789
1.00%	20	0.00051	0.05	2000	76420.630	0.13822	1.0154	0.01433	0.117897
0.50%	5	0.00101	0.2	1000	8905.053	0.09436	0.486	0.00284	0.10967
0.50%	10	0.00051	0.1	1000	18804.614	0.13712	0.4998	0.00589	0.153571
0.50%	20	0.00025	0.05	1000	38602.258	0.19647	0,.5078	0.01159	0.211986

Table 6: Statistics for the GI model using 2-states

Increasing the length of the sequence, the variation is lesser, both in theoretical and experimental values.

SAMPLES	LENGTH	PLOSS	E(B)	PLOSS	PLOSS	PLOSS
				μ (%)	σ	cV
40	10000	2%	10	1.809%	0.5447	0.3011
40	20000	2%	10	1.889%	0.3602	0.1907
40	100000	2%	10	2.004%	0.1741	0.0868
40	200000	2%	10	2.007%	0.1187	0.0591
40	1000000	2%	10	1.984%	0.0705	0.0355
40	2000000	2%	10	1.994%	0.0426	0.0213

Table 7: Dependence of mean, variance and coefficient of variation on the sequence length for the GI model using 2-states

The number of samples considered doesn't seem to help in reducing variation, as we see in the following table. We can assume that 40 samples are enough to estimate variance

SAMPLES	PLOSS	E(B)	PLOSS	PLOSS	PLOSS
			μ	σ^2	cV
25	2%	10	1.996	0.10395	0.0520
40	2%	10	1.954	0.12200	0.0668
50	2%	10	1.962	0.13561	0.06904
100	2%	10	2.015	0.13344	0.06624
200	2%	10	1.993	0.13845	0.06942
250	2%	10	1.997	0.12861	0.06433
500	2%	10	1.994	0.13522	0.06784
1000	2%	10	1.997	0.13492	0.06751
2000	2%	10	1.997	0.13575	0.06794
2500	2%	10	2.000	0.13671	0.06839

5000	2%	10	1.999	0.13170	0.06583

Table 8: Dependence of mean, variance and coefficient of variation on the number of samples for the GI model using 2-states

7.3.5 Statistics for Gilbert-Elliot and Gilbert-Elliot-4s models

In this section we report the statistics of a simulation relative to the *Gilbert* model.performed specifying the three parameters p, r and 1-h. We will calculate the theoretical values for mean, variance and cV using the formulas seen for the *Gilbert-Elliot* model choosing 1-*k*=0. The number of samples is 50, each sequence has length 200000.We repeated the simulations using both *loss_gilb_ell* and *loss_gilb_ell_4s* to show through experimental data the equivalence of our Gilbert-Elliot-4s model with the original Gilbert-Elliot.

model	input				theoretical						experimental					
	p	r	1-h	1-k	ploss	ploss	ploss	E(B)	density	pisol	ploss	ploss	ploss	E(B)	density	pisol
					mean	variance	cV	mean	mean	mean	mean	variance	cV	mean	mean	mean
gilb_ell	0.00204	0.01	1.00	0	1.900	0.018	0.0691	10.00	1	0	1,990	0.018	0.069	9,926	100	0
gilb_ell_4s	0.00204	0.01	1.00	0	1.900	0.021	0.0746	10.00	1	0	1,975	0.021	0.074	0.44	100	0
gilb_ell	0.00204	0.01	0.01	0	1.500	0.011	0.0689	9.780	0.834	0	1,603	0.011	0.068	9,768	83,817	0
gilb_ell_4s	0.00204	0.01	0.01	0	1.500	0.011	0.0664	9.780	0.834	0	1,602	0.011	0.066	9,791	83,764	0
gilb_ell	0.00204	0.01	0.00	0	0.690	0.004	0.0669	9.181	0.599	0	0.690	0.004	0.066	9,130	60,092	0
gilb_ell_4s	0.00204	0.01	0.00	0	0.990	0.004	0.0658	9.181	0.599	0	1,002	0.004	0.065	9,129	59,992	0
gilb_ell	0.00204	0.01	0.02	0	0.400	0.001	0.0821	7.923	0.41	0	0.351	0.001	0.082	7,942	41,163	0
gilb_ell_4s	0.00204	0.01	0.02	0	0.400	0.001	0.0786	7.923	0.41	0	0.338	0.001	0.078	7,893	41,136	0
gilb_ell	0.00204	0.01	0.00	0.001	1.000	0.004	0.0654	9.181	0.599	9.8E-05	1,002	0.004	0.065	9,142	59,925	0.009
gilb_ell_4s	0.00204	0.01	0.00	0.001	1.000	0.004	0.0672	9.181	0.599	9.8E-05	1,014	0.004	0.067	9,092	60,044	0.009

Table 9: Statistics for Gilbert-Elliot and Gilbert-Elliot-4s models

7.3.6 Statistics for the GI model using 3 and 4 states

In this section we reconsider the full GI model using 3 and 4 states through the *loss_GI* option and the parameters P_{LOSS}, $E(B)$, ρ, P_{ISOL}. all the cases of the previous section adding a isolated loss probability of 0.1%. So we are now using the full *Gilbert-Elliot* model. The number of samples and the length of each sequence are still 50 and 200000.

7	E(B)	ρ	PISOL	PLOSS	PLOSS	PLOSS	E(B)	ρ	PISOL	E(GB)
				μ	σ2	cV	μ	μ	μ	μ
2	5	100	0	1,992	0,0079	0,044	4,967	100	0	0
2	5	80	0	1,996	0,0080	0,044	4,998	80,022	0	1,335
2	5	50	0	1,994	0,0031	0,028	4,967	50,118	0	2,662
2	10	100	0	1,990	0,0201	0,071	10,02	100	0	0
2	10	80	0	2,008	0,0136	0,058	10,023	79,963	0	1,287
2	10	50	0	2,006	0,0070	0,041	10,028	50,050	0	2,245
2	20	100	0	1,983	0,0387	0,099	19,964	100	0	0

2	20	80	0	2,009	0,0263	0,080	19,889	79,966	0	1,266
2	20	50	0	2,013	0,0231	0,075	20,122	49,983	0	2,113
1	5	100	0	0,989	0,0041	0,065	4,968	100	0	0
1	5	80	0	0,987	0,0034	0,059	4,958	80,193	0	1,329
1	5	50	0	1,005	0,0019	0,043	5,007	50,041	0	2,662
1	10	100	0	0,980	0,0092	0,097	9,796	100	0	0
1	10	50	0	0,990	0,0043	0,066	9,893	50,081	0	2,250
1	20	100	0	1,015	0,0143	0,117	20,28	100	0	0
1	20	80	0	1,024	0,0175	0,129	20,18	79,784	0	1,273
1	20	50	0	1,002	0,0127	0,112	20,27	49,983	0	2,109
0,5	5	100	0	0,486	0,0028	0,109	4,849	100	0	0
0,5	5	80	0	0,497	0,0016	0,082	4,940	80,194	0	1,335
0,5	5	50	0	0,501	0,0008	0,059	5,050	49,882	0	2,679
0,5	10	100	0	0,499	0,0058	0,153	9,958	100	0	0
0,5	10	80	0	0,511	0,0032	0,112	10,133	79,752	0	1,281
0,5	10	50	0	0,508	0,0029	0,106	10,353	49,650	0	2,258
0,5	20	100	0	0,507	0,0115	0,211	19,772	100	0	0
0,5	20	80	0	0,487	0,0086	0,190	19,252	79,935	0	1,273
0,5	20	50	0	0,517	0,0050	0,137	20,121	50,079	0	2,108
2	10	100	0,01	1,995	0,01346	0,058	10,037	100	0,010	0
2	10	80	0,01	1,979	0,0121	0,055	9,937	80,004	0,009	1,280
2	10	50	0,01	2,005	0,0070	0,041	10,022	50,073	0,0097	2,243

Table 10: Statistics for the GI model using 3 and 4 states

7.4 Generated loss probability plots

We will now show a series of plots to evaluate the loss generator accuracy, using the same simulation data of Section 7.3. Each plot, for a fixed input loss probability and density, shows a line joining the mean loss probability values that correspond to different input values of burst length. The error bars represent the standard deviation

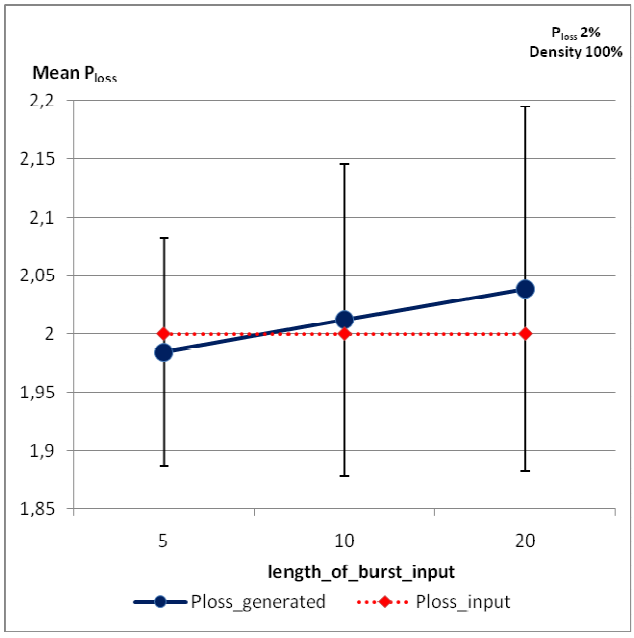

Figure 9: Generated loss probability (2%) versus burst length, Density 100

The first plot was made using a 2-state model specifying the two GI parameters P_{LOSS} and E(B). We see that the distance between input and generated values is less than 3%. The experimental values, as seen in the previous sections, are consistent with the theoretical ones. The size of the confidence interval depends on the chosen burst length. In fact the standard deviation becomes higher if we consider longer bursts, resulting in a stronger variation of P_{LOSS}.

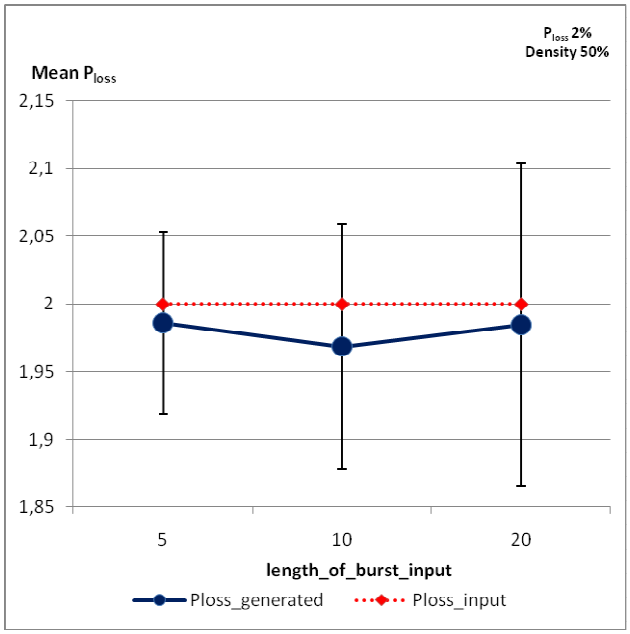

Figure 10: Generated loss probability (2%) versus burst length, Density 50%

In this second plot we change the burst density value to the half, setting the third GI parameter ρ to 50%. This corresponds to a 3-state Markov model. The generated values are still close to the input ones. However, there is a fluctuation of P_{LOSS} with the burst length value. Moreover, the standard deviation still increments with higher values of *E(B)* but with a smaller confidence interval.

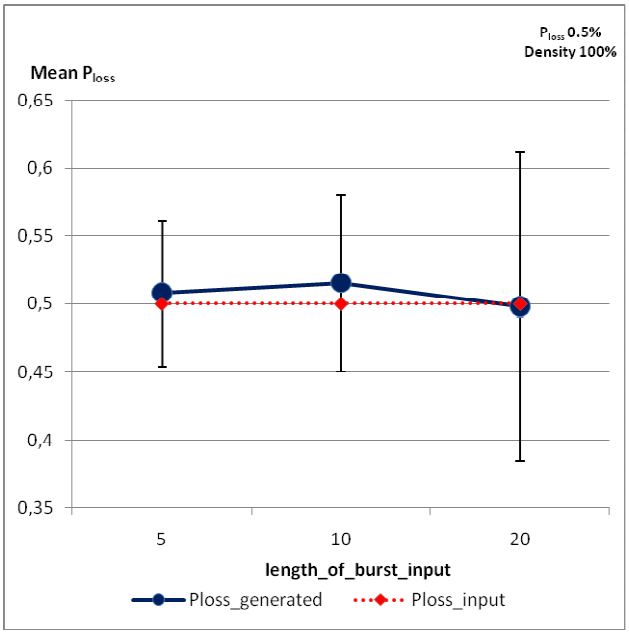

Figure 11: Generated loss probability (0.5%) versus burst length, Density 100%

Coming back to the 2-state model (or, equivalently, to ρ=100%) and considering smaller values of P_{LOSS}, for example 0.5%, while the mean values are still close to the input ones, the coefficients of variation are higher. So the generator's behavior is less predictable using small loss probabilities.

References

[1] Gilbert E. N. “Capacity of a Burst Noise Channel”, BSTJ September 1960

[2] Elliot E.O., "Estimates of Error Rates for Codecs on Burst-Noise Channels", Bell System Technical Journal 42 (1963) 1977-1997

[3] Clark A., “Modeling the Effects of Burst Packet Loss and Recency on Subjective Voice Quality”, IPtel 2001 Workshop

[4] Yajnik, M. Sue Moon, Kurose, J. Towsley, D. "Measurement and modelling of the temporal dependence in packet loss", INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE.

[5] ITU-T SG12 D.139: “Study of the relationship between instantaneous and overall subjective speech quality for time-varying quality speech sequences”, France Telecom

[6] VoIP Troubleshooter http://www.voiptroubleshooter.com/indepth/burstloss.html

[7] T. Friedman, R. Caceres, A. Clark (Eds) “RTP Control Protocol Extended Reports (RTCP XR)”, IETF RFC 3611

[8] Hemminger S. "Network Emulation with NetEm", Open Source Development Lab, April 2005 (http://devresources.linux-foundation.org/shemminger/netem/LCA2005_paper.pdf)

[9] Netem Packet loss + correlation, from Netem Mailing list (https://lists.linux-foundation.org/pipermail/netem/2007-September/001156.html)

[10] Generic Netlink HOWTO – The Linux Foundation, http://www.linuxfoundation.org/en/Net:Generic_Netlink_HOWTO

[11] G.Hassingler, O.Hohlfeld, “The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet”, in 14. GI/ITG Konferenz MMB 2008, Dortmund, Germany

[12] Hohlfeld, "Stochastic Packet Loss Model to Evaluate QoE Impairments", PIK - Praxis der Informationsverarbeitung und Kommunikation journal

[13] G. Galati, G. Pavan, “Teoria dei fenomeni aleatori”, Texmat pp.358-359