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

Version 3.1, August 2012

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

[(1)]University of Rome “Tor Vergata”

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

Contact: stefano.salsano@uniroma2.it

TABLE OF CONTENTS

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 correlated loss 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 the module sch_netem
- added new option in tc-netem

We first realized a patch that introduced new correlated loss models to netem. The patch applied to Linux kernel 2.6.32 and iproute2-2.6.32 package. On 18 Dec 2009 we have submitted a patch to the netdev@vger.kernel.org mailing list in order to have NetemCLG included in the netem kernel module and in the iproute2 utilities.

The patch has been reworked and applied to kernel 2.6.39, while the needed changes to iproute2 have been added to iproute2-3.2.0. The version that has been included in kernel 2.6.39 and iproute2-3.2.0 includes the GE (Gilbert-Elliot) model and the 4 states model using transition probability as input according to the follow syntax:

```
[ loss state P13 [P31 [P32 [P23 P14]]]
```

```
[ loss gemodel PERCENT [R [1-H [1-K]]]
```

We found two problems in the kernel module that implements the 4 states and GE loss generators. Therefore we propose a patch to sch_netem.c to fix these problems. (We have spotted the problems using the test tools developed to measure loss and loss correlation generated by netem kernel modules: trace2seq.c, sqnanalyzer.c).

Moreover we propose a further patch to tc-netem that allows input of parameters using the GI (General and Intuitive model) according to the following syntax

```
[ loss gimodel P [ BURST [DENSITY [PISOL [GOOD_BURST_LENGTH]]]]]
```

1 The original problem: weakness of Netem correlated 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, choosing a Markov chain model in the proposed loss model.

2 Existing loss models in the literature

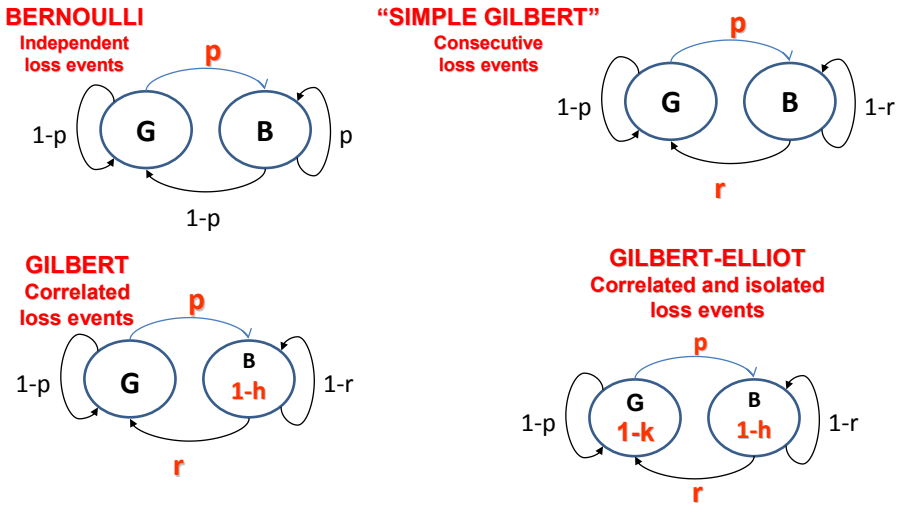


Figure 1: 4 loss models existing in the literature

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.

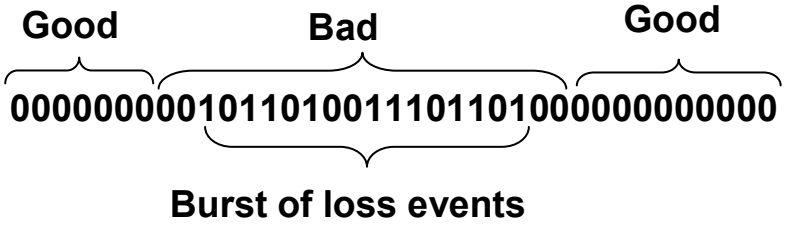


Figure 2: Example of a loss pattern generate by a Gilbert (or Gilbert-Elliot) model

3 The proposed loss model

3.1 The 4-state Markov model

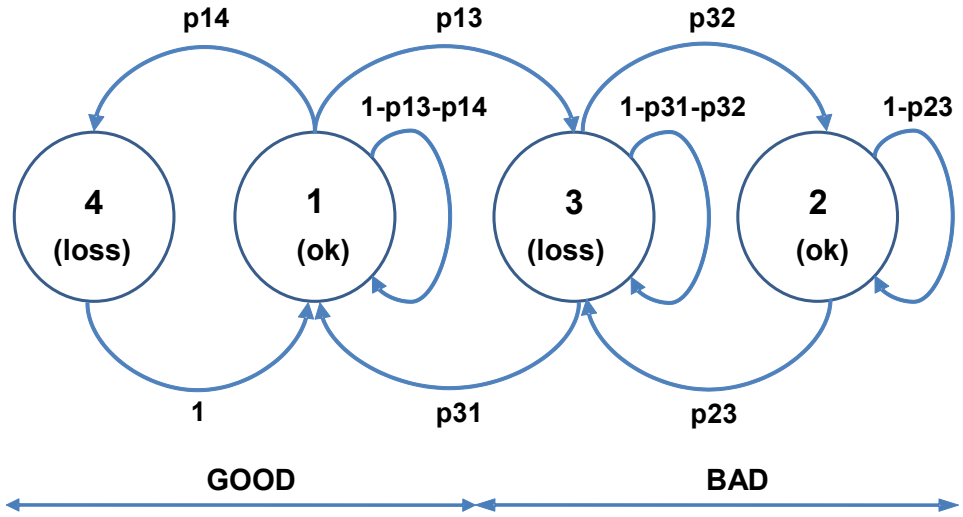


Figure 3: 4-state Markov model

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:

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

3.2 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;
- 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).

3.2.1 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:

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

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

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

$E(GB) \geq 1$

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

$P_{ISOL} \leq 1/2$

3.2.2 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: $p_{23} = 1 - \frac{p_{32}}{1 - p_{31}}$ we obtain the following expressions for the transition probabilities:

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

The following constraints apply here:

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

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

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

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

$P_{ISOL} \leq 1/2$

3.2.3 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 $E(B)$;

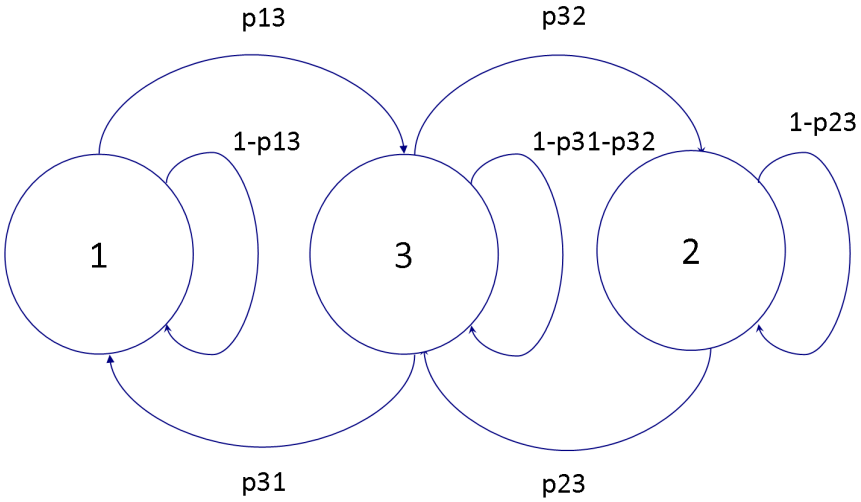


Figure 4: 3-state model

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:

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

$P_{LOSS} < \rho$

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

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

3.2.4 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:

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

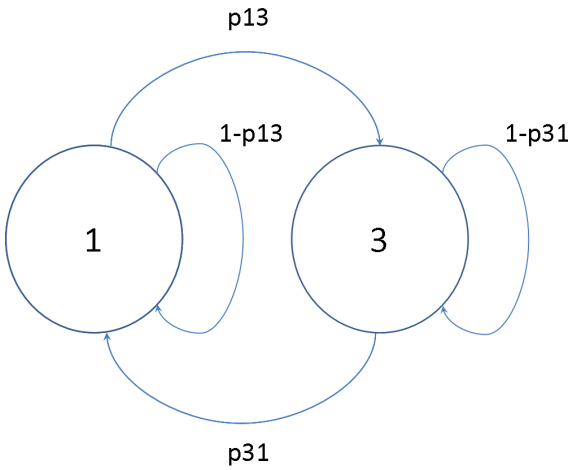


Figure 5: The 2-state Markov chain

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 $E(B) = \dfrac{1}{p_{31}}$.Then the transition probabilities are:

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

3.2.5 Bernoulli model: GI model with 1 parameter

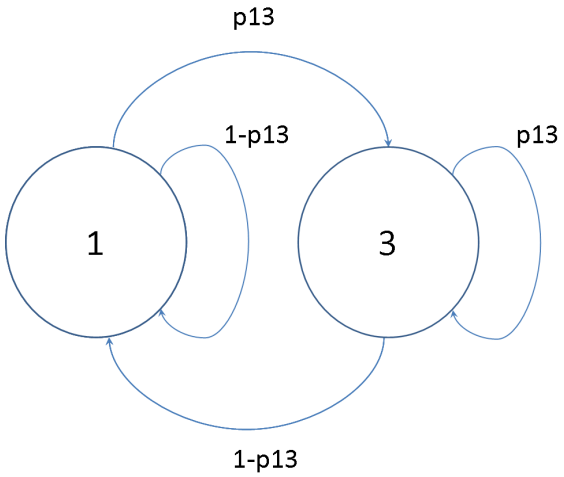


Figure 6: The Bernoulli model as a Markov chain

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.

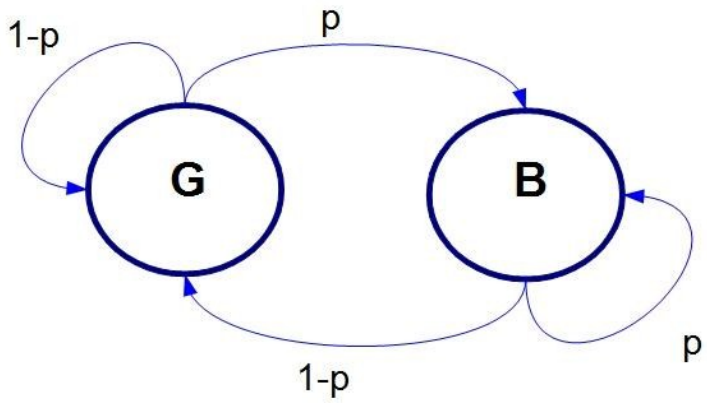
The Bernoulli model is the simplest case of loss model we consider. It has only one parameter and can be see as a special case of the Simple Gilbert model where *r*=1-*p*. So the only significant GI parameter is the loss probability P_{LOSS}=*p*. So we have:

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

3.3 Gilbert – Elliot Model

3.3.1 Bernoulli model

The Bernoulli loss model has two states and only one parameter, the loss probabilities p. This model is only able to model uncorrelated loss events.

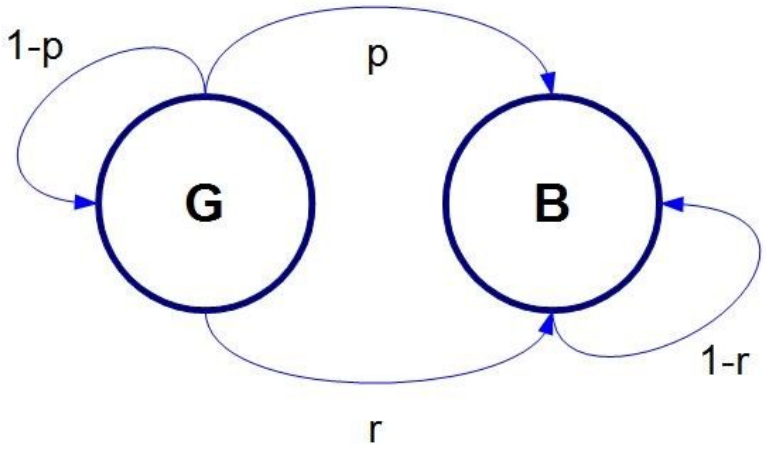


Since p is the only parameter, the system is described as follows:

$$\begin{cases} r = 1 - p \\ h = 0 \\ 1 - k = 0 \end{cases}$$

3.3.2 Simple-Gilbert model

The "Simple Gilbert" model has two states and two independent parameters, p and r. It is able to model a system with consecutive loss events, which can be characterized by a *loss probability* and a *burst duration*.

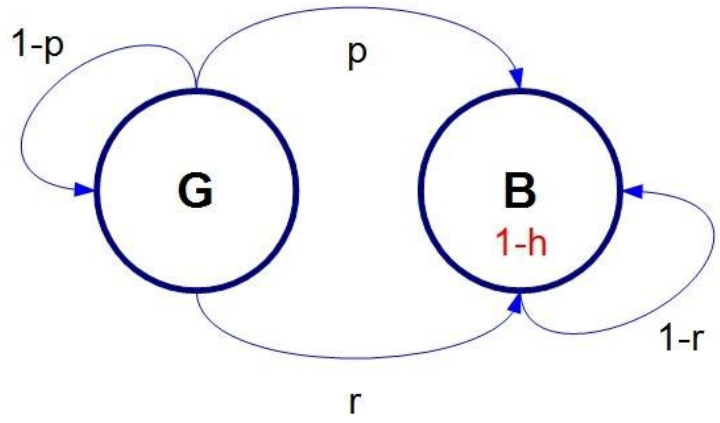


The other two parameters are:

$$\begin{cases} h = 0 \\ 1 - k = 0 \end{cases}$$

3.3.3 Gilbert model

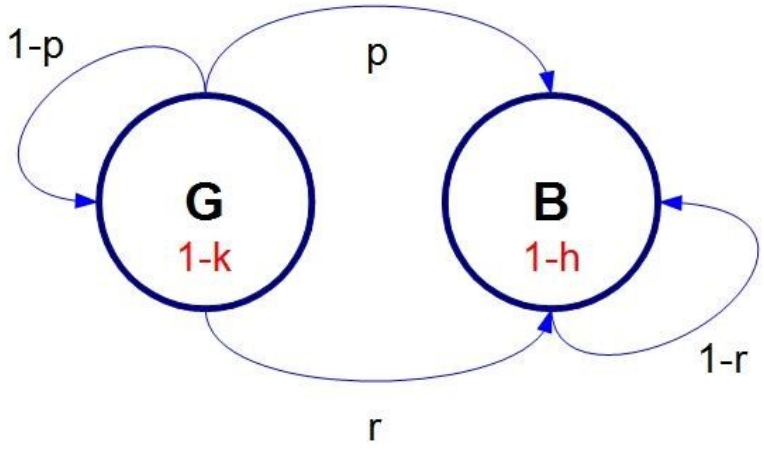
The "Gilbert" model has two states (Good and Bad) and three independent parameters. Within the Good state packet are never lost, while within the Bad state there is a probability h that the packet is transmitted. This is able to model the system in which the loss events appear in burst, but within the bursts there are some transmitted packets and some loss 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% (h=0) the model becomes equivalent to Simple-Gilbert. The only other parameter is 1- k = 0.

3.3.4 Gilbert-Elliot model

The "Gilbert-Elliot" model has two states (Good and Bad) and four independent parameters. In this case it is 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 system in which there is a Good state with a relatively low loss probability 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 so that it is possible to detect a "burst" of loss event corresponding to the permanence of the system in the Bad state.

4 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 2 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 campaign 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.

4.1.1 Revisiting the Gilbert-Elliot model

In order to relate the classical Gilbert-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 Gilbert-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 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 2, 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 explicitly 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 parameters 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 correspond 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 PISOL) 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 PISOL, as the probability of having consecutive "isolated" loss events increases with PISOL.

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	$P\{B=i\}$	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	$r(1-r)^3$			$(1-h)^2$	$2h(1-h)^2$	$3h^2(1-$	$4h^3(1-h)$	h^4

						$h)^2$		
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:

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

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

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

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:

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

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

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

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

$$\frac{r}{(1-h+rh)}\left(1 + (1-h)\left(\frac{(1-r)}{r^2} - 1 + \frac{1}{r} \right) \right) = \frac{r(2-h)}{(1-h+rh)}\left(1 + (1-h)\left(\frac{1-r-r^2+r}{r^2} \right) \right) =$$

$$\frac{r}{(1-h+rh)}\left(1 + (1-h)\left(\frac{1-r^2}{r^2} \right) \right) = \frac{r}{(1-h+rh)}\left(\frac{r^2 + (1-h)(1-r^2)}{r^2} \right) =$$

$$\frac{\left(r^2 + 1 - r^2 - h + r^2 h\right)}{r(1-h+rh)} = \frac{(1-h+r^2 h)}{r(1-h+rh)}$$

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:

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

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

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

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

4.1.2 Revisiting the Gilbert model

We have already shown in section 4.1.1 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:

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

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

4.1.3 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 4.1.3 is equivalent to the 2-state Markov. So we have the following relation:

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

4.2 Relations between models

As we have widely explained, the proposed GI (General and Intuitive) 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
GI model (4-state Markov)	P_{LOSS},E(B),ρ P_{ISOL},E(GB)	4	5	Yes	Yes	Yes	Yes
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

Intuitively, 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 to Linux Kernel 3.0.16 and iproute2-3.2.0

These patches allow to upgrade the Linux Kernel 3.0.16 and iproute2-3.2.0 with the latest change. The kernel patch allows to correct the errors found in the file sch_netem.c, and in particular in the function of the generators of losses of various model, while the other patch is applied to the iproute2 and it is necessary to add the GI model to the application.

6.1.1 How to apply patches

This section describes how to install the patches that adds the new change of the 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 <../Kernel_3.0.16_patch
```

substitute *../Kernel_3.0.16_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 <../iproute2-3.2.0_patch
```

substitute *../iproute2-3.2.0_patch* with the path where you have unpacked the archive. Then build *iproute2*.

6.2 tc-netem user manual (for the latest version)

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 gimodel P [BURST [DENSITY [PISOL [GOOD_BURST_LENGTH]]]]

tc qdisc add/change/del dev eth0 root netem loss gemodel PERCENT [R [1-H [1-K]]]

tc qdisc add/change/del dev eth0 root netem loss state P13 [P31[P32[P23 14]]]

Querying information from a loss model

tc qdisc add dev eth0 root netem query loss gimodel P [BURST [DENSITY [PISOL [GOOD_BURST_LENGTH]]]]

Where the correspondence between options and model are:

OPTION	MODEL
Loss gimodel	GI (General and Intuitive) Allows special case: 4, 3, 2 and 1 parameters
Loss gemodel	Bernoulli, Simple Gilbert, Gilbert, Gilbert-Elliot
Loss state	Loss 4-state with the transition probabilities: p13, p31 ,p31, p23, p14

Table 2: Options and models in tc-netem

If the selected option is loss *gemodel* the Gilbert-Elliot generation algorithm will be used. If the selected option is one of loss *gimodel* 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:

Transition probability	Loss_gimodel
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

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 probabilities 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 gimodel 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 netem query loss gimodel 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 hypothesis. 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 frequently. 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 query  loss gimodel 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 query loss gimodel 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)
```

6.3 Original NetemCLG and tc-netem patches to Linux Kernel 2.6.39.1 and iproute2-2.6.39

6.3.1 How to apply patch to kernel before it was accepted the scheduler in the kernel

This section describes how to install the patches that adds the new change of the 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 <../Kernel_2.6.39.1_patch
```

substitute *../Kernel_2.6.39.1_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 <../iproute2-2.6.39_patch
```

substitute *../iproute2-2.6.39_patch* with the path where you have unpacked the *iproute2* sources. Then build *iproute2*.

6.4 tc-netem user manual for the originally proposed patch (historical)

The first modified version of *tc* allowed the user to add loss features to the outgoing packets according to the following syntax. This is reported mainly for historical reasons as the use of the old version is deprecated.

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 netem2 loss_bern p

tc qdisc add/change/del dev eth0 root netem2 loss_gilb p r [1-h]

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 netem2 loss_gilb_4s p r [1-h]

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_bern p

tc qdisc add dev eth0 root netem2 query loss_gilb_4s p r [1-h]

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	Gilbert and Simple Gilbert as special case
Loss_gilb_ell	Gilbert-Elliot and special cases: Gilbert, Simple Gilbert
Loss_gilb_4s	Gilbert-4s
Loss_gilb_ell_4s	Gilbert-Elliot-4s (allows Gilbert-4s as special case)

Table 4: Options and models in tc-netem2

If the selected option is *loss_gilb* or *loss_gilb_ell* the Gilbert-Elliot generation algorithm will be used. If the selected option is one of *loss_GI*, *loss_4_state,loss_bern,loss_gilb_4s* 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*2 thought the *netlink* socket. The conversion formulas are shown in the following table:

Loss_4state	Loss_GI	Loss_bern
p_{13}	$\frac{P_{Loss} - P_{ISOL}}{E(B)(1 - P_{ISOL})(\rho - P_{Loss})}$	P_{Loss}
p_{31}	$\frac{1}{E(B) \cdot \rho}$	$1 - P_{Loss}$
p_{23}	$\frac{E(B) \cdot \rho - 1}{E(B) - 1}$	1
p_{32}	$\frac{(\rho - 1) \cdot (E(B) \cdot \rho - 1)}{\rho \cdot (1 - E(B))}$	0
p_{14}	$\frac{P_{ISOL}}{1 - P_{ISOL}}$	0

Table 5: Relations between options and models in tc-netem2

For the *Gilbert-4s* and *Gilbert-Elliot-4s* cases, 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:

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

Note that 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.

6.5 NetemCLG: code enhancements

6.5.1 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.c* 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 structure *tc_netem_gimodel* 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* models.

```
struct tc_netem_gimodel /* State transition probabilities for 4 state model*/
{
        __u32 p13;
        __u32 p31;
        __u32 p32;
        __u32 p14;
        __u32 p23;
};


struct tc_netem_gemodel  /* Gilbert-Elliot model*/
{
        __u32 p;
        __u32 r;
        __u32 h;
        __u32 k1;
};
```

Only in *sch_netem.c* were added:

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

```
enum{
        CLG_RANDOM,
        CLG_4_STATES,
        CLG_GILB_ELL,
} loss_model;

/* Correlated Loss Generation models */
struct clgstate{
        u8 state;           /* State of the Markov chain */
        /* 4-states and Gilbert-Elliot models */
        u32 a1;             /* p13 for 4-states or p for Gilbert-Elliot */
        u32 a2;             /* p31 for 4-states or r for Gilbert-Elliot */
        u32 a3;             /* p32 for 4-states or h for Gilbert-Elliot */
```

```
        u32 a4;         /* p14 for 4-states or 1-k for Gilbert-Elliot */
        u32 a5;         /* p23 used only in 4-states */
}clg;
```

- the *loss_4state* generator function: it generates losses according to the 4-state Markov chain adopted in the GI (*General and Intuitive*) loss model. The function makes a comparison between the return value of the net_random() function 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 bool  loss_4state (struct netem_sched_data *q) {

  struct clgstate *clg=&q ->clg;
  u32 rnd = net_random(); /* extracts a random number */
  switch(clg ->state) {
  case 1:
        if(rnd<clg ->a4) {
                clg ->state=4;
                return true;

        } else if ((clg->a4<rnd && rnd<(clg->a1 + clg->a4)){
                clg->state=3;
                return true;
        } else if((clg->a1 + clg->a4)<rnd)
                clg->state=1;
        break;
  case 2:
        if(rnd<clg->a5) {
                clg->state=3;
                return true;
        } else {
                clg->state=2;
        break;
  case 3:
        if(rnd<clg->a3)
                clg->state=2;
        else if((clg->a3<rnd)
                        && (rnd<(clg->a2+clg->a3))) {
                clg->state=1;
        } else if ((clg->a2+clg->a3)<rnd) {
                clg->state=3;
                return true;
        }
        break;
  case 4:
        clg->state=1;

  break;
  }
  return false;
}
```

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

```
static bool loss_gilb_ell (struct netem_sched_data *q) {

   struct clgstate *clg=&q->clg;

   switch(clg->state) {
   case 1:
         if(net_random()<clg->a1)
               clg->state=2;
         if(net_random()<clg->a4) {
               return true;
         break;
   case 2:
         if(net_random()<clg->a2)
               clg->state=1;
         if(clg->a3>net_random()) {
               return true;
         break;
   }
   return false;
}
```

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

```
static bool loss_event (struct netem_sched_data *q)
{
         switch(q->loss_model){
         case CLG_RANDOM:
               return q->loss && q->loss >= get_crandom(&q->loss_cor);
         case CLG_4_STATES:
               return loss_4state(q);
         case CLG_GILB_ELL:
               return loss_gilb_ell(q);
         }
         return false;
}
```

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 nlattr *la;
         int rem;

         nla_for_each_nested(la, attr, rem){
               u16 type=nla_type(la);
```

```
                switch(type){
                case NETEM_LOSS_GI:{
                        const struct tc_netem_gimodel *gi =nla_data(la);

                        if(nla_len(la) != sizeof(struct tc_netem_gimodel)){
                                pr_info("netem: incorrect gi model size\n");
                                return -EINVAL;
                        }
                        q->loss_model= CLG_4_STATES;

                        q->clg.state = 1;
                        q->clg.a1= gi->p13;
                        q->clg.a2 = gi->p31;
                        q->clg.a3 = gi->p32;
                        q->clg.a4 = gi->p14;
                        q->clg.a5 = gi->p23;
                        break;
                }
                case NETEM_LOSS_GE:{
                        const struct tc_netem_gemodel *ge =nla_data(la);

                        if(nla_len(la) != sizeof(struct tc_netem_gemodel)){
                                pr_info("netem: incorrect ge model size\n");
                                return -EINVAL;
                        }
                        q->loss_model= CLG_GILB_ELL;
                        q->clg.state = 1;
                        q->clg.a1= ge->p;
                        q->clg.a2 = ge->r;
                        q->clg.a3 = ge->h;
                        q->clg.a4 = ge->k1;
                        break;
                }
                }
        }
        return 0;
  }
```

Add the function *dump_loss_model* :

```
static int dump_loss_model (struct netem_sched_data *q, struct sk_buff *skb)
{
  switch(q->loss_model){
        case CLG_RANDOM:
                .....................
        case CLG_4_STATES:{
                struct tc_netem_gimodel = {
                        .p13 = q->clg.a1,
                        .p31 = q->clg.a2,
                        .p32 = q->clg.a3,
                        .p14 = q->clg.a4,
                        .p23 = q->clg.a5,
                }
                NLA_PUT(skb, NETEM_LOSS_GI, sizeof(gi), &gi);
                break;
```

```
            }
            case CLG_GILB_ELL:{
                    struct tc_netem_gemodel = {
                            .p->clg.a1,
                            .r->clg.a2,
                            .h->clg.a3,
                            .k1->clg.a4,
                    }
                    NLA_PUT(skb, NETEM_LOSS_GE, sizeof(ge), &ge);
                    break;
            }
  }
   .....
}
```

6.5.2 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 > /tc. 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*:

- *query mode*

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

- *loss state*: if the selection options is *loss state* there is no need for the conversion and the input parameters (transition probabilities: p13, p31, p32, p23, p14) are directly passed to the qdisc.

```
} else if (!strcmp(*argv, "state")) {
            double p13;

            NEXT_ARG();
            if (parse_percent(&p13, *argv)) {
              explain1("loss p13");
              return -1;
            }

            /* set defaults */
            set_percent(&gimodel.p13, p13);
            set_percent(&gimodel.p31, 1. - p13);
            set_percent(&gimodel.p32, 0);
            set_percent(&gimodel.p23, 1.);
            set_percent(&gimodel.p14, 0);
            loss_type = NETEM_LOSS_GI;

            if (!NEXT_IS_NUMBER())
              continue;
            NEXT_ARG();
```

```
        if (get_percent(&gimodel.p31, *argv)) {
                explain1("loss p31");
          return -1;
 }

        if (!NEXT_IS_NUMBER())
          continue;
        NEXT_ARG();
        if (get_percent(&gimodel.p32, *argv)) {
          explain1("loss p32");
          return -1;
        }

        if (!NEXT_IS_NUMBER())
          continue;
        NEXT_ARG();
        if (get_percent(&gimodel.p23, *argv)) {
          explain1("loss p23");
          return -1;
        }
        if (!NEXT_IS_NUMBER())
          continue;
        NEXT_ARG();
        if (get_percent(&gimodel.p14, *argv)) {
          explain1("loss p14");
          return -1;
        }
}
```

- *loss_gimodel*: if the chosen option is *loss gimodel* ("General and Intuitive") the variables relatives to the transition probabilities and the GI parameters are initially set to the default values. Then the mandatory p parameters and, if specified, the optional parameters, are read from keyboard and stored to the appropriate variables. 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 (!strcmp(*argv, "gimodel")) {
                NEXT_ARG();
                 double p13=0;
                double p31=1;
                double p32=0;
                double p23=1;
                double p14=0;
                double p=0;
                double burst=0;
                double density=1;
                double pisol=0;
                double good_burst_length=1;

                if (parse_percent(&p, *argv)) {
                                explain1("loss gimodel p");
                                return -1;
                }
```

```
                int input_params = 1; /* counts the number of input parameters */
```

```
        if (NEXT_IS_NUMBER()) {
                NEXT_ARG();
                input_params ++;
                burst=strtod(*argv,(char **)NULL);
                  if (NEXT_IS_NUMBER()) {
                        NEXT_ARG();
                        input_params ++;
        if (parse_percent(&density, *argv)) {
                explain1("loss gimodel density");
                return -1;
        }

                        if (p >= density){
                                printf("\nError: p >= density\n");
                                break;
                        }
                        if ((burst * density) < 1){
                                printf("\nError: (burst * density) < 1\n");
                                break;
                        }


                        if(NEXT_IS_NUMBER()) {
                                NEXT_ARG();
                                input_params ++;
                                if (parse_percent(&pisol, *argv)) {
                                        explain1("loss gimodel pisol");
                                        return -1;
                                }

                                if (pisol > p){
                                        printf("\nError: pisol > p\n");
                                        break;
                                }
                                if ((p-pisol) > (burst*(1-pisol)*(density-p))){
                                        printf("\nError: (p-pisol) > burst*(1-pisol)(density-
p)\n ");
                                        break;
                                }
                                if (pisol > 0.5){
                                        printf("\nError: pisol > 0.5\n");
                                        break;
                                }
                                if(NEXT_IS_NUMBER()) {
                                        NEXT_ARG();
                                        input_params ++;
                                        good_burst_length=strtod(*argv,(char**)NULL);
                                        if (good_burst_length < 1){
                                                printf("\nError: good_burst_lenght < 1
\n");
                                                break;
                                        }
                                        if (good_burst_length < ((1-density)/density)){
                                                printf("\nError: good_burst_lenght <
(1-density)/density\n");
                                                break;
                                        }
                                }
```

```
                        }
                }
                }
                if (input_params>=2) {
                if (input_params>=3) {
                        if (input_params>=4) {
                                if (input_params>=5) {
                                /*input_params == 5 */
                                p23=1/good_burst_length;
                                p32=(1-density) / (density*good_burst_length);
                                p14=pisol/(1-pisol);
                                p13=(pisol-p)/(burst*(pisol-1)*(density-p));
                                p31=1/(burst*density);
                        } else {
                                /*input_params == 4 */
                                p23=(burst * density - 1)/(burst - 1);
                                p32=(density-1)*(burst * density - 1) / (density*(1 - burst));
                                p14=pisol/(1-pisol);
                                p13=(pisol-p)/(burst*(pisol-1)*(density-p));
                                p31=1/(burst*density);
                                good_burst_length = 1/p23;
                                }
                        } else {
                        /*input_params == 3 */
                        p23=(burst * density - 1)/(burst - 1);
                        p32=(density-1)*(burst * density - 1) / (density*(1 - burst));
                        p14 = 0;
                        /*              pisol = 0;
                         *              p13=(pisol-p)/(burst*(pisol-1)*(density-p));*/
                        p13=(-p)/(burst*(-1)*(density-p));
                        p31=1/(burst*density);
                        good_burst_length = 1/p23;
                        }
                } else {
                        /*input_params == 2 */
                        /*p14 = 0;
                         p32 = 0;
                         p23 = 1;
                          pisol = 0;
                         density = 1;*/
                        p13=(-p)/(burst*(-1)*(1-p));
                        p31=1/burst;
                }
        } else {
                /*input_params == 1 */
        /       *p14 = 0:
                p32 = 0;
                p23 = 1;
                pisol = 0;
                density = 1;*/
                p13=p;
                p31=1-p;
                burst = 1 / p31;
        }
        if(query == 1){
                printf("\nTransition probabilities are:\n ");
                printf("--------------------------------");
                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);
                printf("\nGI (General and Intuitive) parameters will be: \n");
                printf("--------------------------------");
                printf("\np is %.3f%% ", 100*p);
                printf("\nburst is %.3f", burst);
                printf("\ndensity is %.3f%% ", 100*density);
                printf("\nisolated ploss is %.3f%% ", 100*pisol);
                printf("\ngood burst length is %.3f\n", good_burst_length);
        } else {
                loss_type = NETEM_LOSS_GI;
                set_percent(&gimodel.p13,p13);
                set_percent(&gimodel.p31,p31);
                set_percent(&gimodel.p32,p32);
                set_percent(&gimodel.p23,p23);
                set_percent(&gimodel.p14,p14);
        }
```

- *loss gemodel*: 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 (!strcmp(*argv, "gemodel")) {
                double p;
                double h;

                NEXT_ARG();
                if (parse_percent(&p, *argv)) {
                        explain1("loss gemodel p");
                        return -1;
                }

                /* set defaults */
                set_percent(&gemodel.p,p);
                set_percent(&gemodel.r, 1. - p);
                set_percent(&gemodel.h, 0);
                set_percent(&gemodel.k1, 0);
                loss_type = NETEM_LOSS_GE;

                if (!NEXT_IS_NUMBER())
                        continue;
                NEXT_ARG();
                if (get_percent(&gemodel.r, *argv)) {
                        explain1("loss gemodel r");
                        return -1;
                }

                if (!NEXT_IS_NUMBER())
                        continue;
                NEXT_ARG();

                if (parse_percent(&h, *argv)) {
                        explain1("loss gemodel h");
                        return -1;
                }
```

```
                        /*set parameter h*/
                        set_percent(&gemodel.h,1. -h);

                        if (!NEXT_IS_NUMBER())
                                continue;
                        NEXT_ARG();
                        if (get_percent(&gemodel.k1, *argv)) {
                                explain1("loss gemodel k");
                                return -1;
                        }
                } else {
                        fprintf(stderr, "Unknown loss parameter: %s\n",
                                *argv);
                        return -1;
                }
}
```

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

```
if (loss_type != NETEM_LOSS_UNSPEC) {
        struct rtattr *start;

        start = addattr_nest(n, 1024, TCA_NETEM_LOSS |
  NLA_F_NESTED);
        if (loss_type == NETEM_LOSS_GI) {
                if (addattr_l(n, 1024, NETEM_LOSS_GI,
                        &gimodel, sizeof(gimodel)) < 0)
                   return -1;
        } else if (loss_type == NETEM_LOSS_GE) {
                if (addattr_l(n, 1024, NETEM_LOSS_GE,
                        &gemodel, sizeof(gemodel)) < 0)
                   return -1;
        } else {
                fprintf(stderr, "loss in the weeds!\n");
                return -1;
        }

        addattr_nest_end(n, start);
  }
```

```
static int netem_print_opt(struct qdisc_util *qu, FILE *f, struct rtattr *opt)
{
        ...
        const struct tc_netem_gemodel *gemodel = NULL;
        const struct tc_netem_gimodel *gimodel = NULL;
        ...
        Struct rtattr *lb[NETEM_LOSS_MAX];

        if (tb[TCA_NETEM_LOSS]) {
                        struct rtattr *lb[NETEM_LOSS_MAX + 1];
                parse_rtattr_nested(lb, NETEM_LOSS_MAX, tb[TCA_NETEM_LOSS]);
                if (lb[NETEM_LOSS_GI])
```

```
                        gimodel = RTA_DATA(lb[NETEM_LOSS_GI]);
                if (lb[NETEM_LOSS_GE])
                        gemodel = RTA_DATA(lb[NETEM_LOSS_GE]);
        }
        ......

        if (gimodel) {
                fprintf(f, " loss state p13 %s", sprint_percent(gimodel->p13, b1));
                fprintf(f, " p31 %s", sprint_percent(gimodel->p31, b1));
                fprintf(f, " p32 %s", sprint_percent(gimodel->p32, b1));
                fprintf(f, " p23 %s", sprint_percent(gimodel->p23, b1));
                fprintf(f, " p14 %s", sprint_percent(gimodel->p14, b1));
          }
        if (gemodel) {
                fprintf(f, " loss gemodel p %s",sprint_percent(gemodel->p, b1));
                fprintf(f, " r %s", sprint_percent(gemodel->r, b1));
                fprintf(f, " h %s", sprint_percent(gemodel->h, b1));
                fprintf(f, " 1-k %s", sprint_percent(gemodel->k1, b1));
          }
}
```

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 behavior 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 command line. 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, the *length* of the sequence specifies the number of events. 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, re-calculating the GI parameters in two ways from 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 on 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_4state
- loss_GI
- 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.

```
$ 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_gilb_ell p r [1-h 1-k]
```

This syntax selects the Gilbert-Elliot model. If we specify only the loss probability *p*, the *Bernoulli* model is selected. Otherwise if we specify the parameters *p* and *r*, the *Simple-Gilbert* model is selected and if we specify the parameters *p*, *r* and *1-h* we select the *Gilbert* model. Finally, if we define the parameters *p*, *r*, *1-h* and *1-k*, the Gilbert-Elliot model will be used. The parameters p and r represent the transition probabilities between the "Good" and the "Bad" states. The third parameter is the loss probability in the "Bad" state while the fourth parameter, 1-k , is the loss probability

when the system is in the *Good* state. The parameters must be expressed as percentage and it is needed to specify the mandatory parameters *p*. If *1-k* is not specified, the *Gilbert* model will be used. If also *1-h* 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	p_{13}, p_{31}	p_{23}, p_{32}, p_{14}
loss_gilb_ell	Gilbert-Elliot (with special cases: Gilbert, Simple Gilbert, Bernoulli)	p	r, $1-h$, $1-k$

Table 6: 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 zeros between two ones 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, unsigned int p13, unsigned int p31, unsigned int p32, unsigned int p23, unsigned int 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 *net_random(gsl_rng *q)* function that generates the number through the *gsl_rng_get* function of the GNU Scientific Library 1.5 that implements a *Tausworthe* generator. 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 result of function *net_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 result of function *net_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 result of function *net_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:

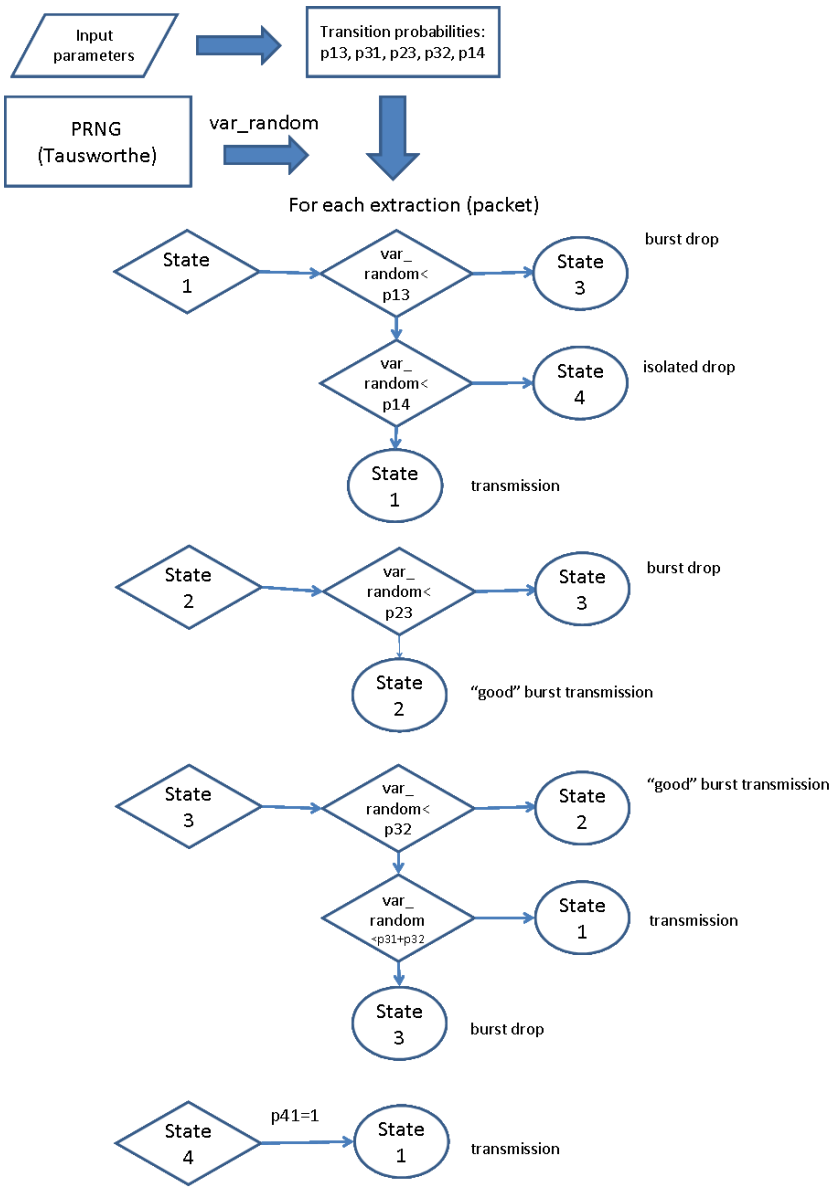


Figure 7: GI loss generator algorithm

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, unsigned int p, unsigned int r, unsigned int k, unsigned int 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 of the *net_random()* function. The *switch* construct is still

controlled by the variable *state* which values can be 1 or 2.

At each extraction, the value of return of the *net_random()* function 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 comparison 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*.
- num_of_good_bursts_bis: *number of good bursts.*
- length_of_all_bursts_bis: *sum of the length of all the bursts we have had*.
- length_of_all_good_bursts_bis: *sum of the length of all the good 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, and for the length of the good burst only the number of the 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

do not have 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 (k-th and k+1-th) at each time. The following figure shows how the algorithm works:

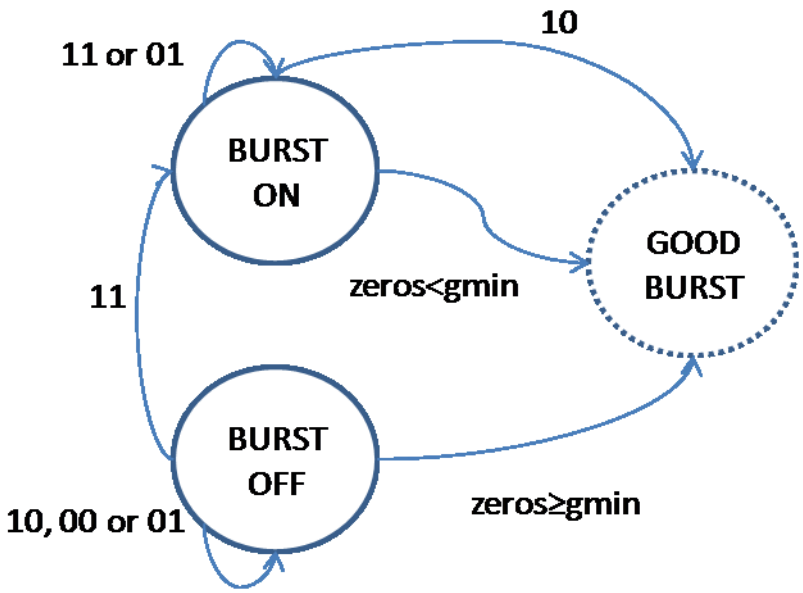


Figure 8: "Blind" external check algorithm

- 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 appropriate variables. If the Gilbert or the Gilbert-Elliot models are selected, the *gilb_ell_gen* function will be used. Otherwise (*loss_GI* , *loss_4state* 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 that 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
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000100000000010000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000101000000100000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000100000000000100000000000000000000000000000000000000000000000000001000000000000000000000
00000000000000000000000000000000000000000001000000000001000000000000000000000000000000000000000000000000000010000000000000010000000000000000000
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:
11111111111111111111111111111111111111111111111111111111111111111011111111111111111111111111111111111111111111111111111111111111111111011111111111
11111111111111111101111111111111111111111111111111111111111111111101111111111111111111111111111111111111111111111111111111111111101111111111111
11110111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111101111111111111111111111111111111111111111011111111111111111101111111111111111111111111111111111111111111011111111111111111111111
11111111111111111111111111111111111110111111111111111111111111111111111110111111111111111111111111111011111011111111111111111111111111111111111
11111111111111111111101111111111111110110111111111111111111111011111111111111111111111111111111111111111111111111111111011111111111111111111111
1111111101111111111111111111111111111111111111111111111111111111111101111111111111011111111111111111111111111111111111111111111111111101111111

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 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 G state represents only correctly received packets and B 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 $E[X^K] = \frac{\partial}{\partial z} X(z)|_{z=1}$. 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.

7.3.3 Statistics for the GI model using 2 parameters

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 does not affect the results heavily. The simulation is made of 50 sequences whose length is 200000.

PLOSS	E(B)	LOSS MEAN	LOSS VARIANCE	LOSS CV	LOSS MEAN	LOSS VARIANCE	CV (generated)

		(theory)	(theory)	(theory)	(generated)	(generated)	
2.00%	5	4000,	34495,251	0.04643	2,0007	0,00862	0,04641
2.00%	10	4000	72908.619	0.06750	1.98268	0.01766	0.06702
2.00%	20	4000	149729.709	0.09673	1.96733	0.03362	0.0932
1.00%	5	2000	17621.613	0.06637	1.00476	0.00532	0.07258
1.00%	10	2000	37222.255	0.09646	0.99797	0.01091	0.10467
1.00%	20	2000	76420.630	0.13822	1.01728	0.02061	0.14113
0.50%	5	1000	8905.053	0.09436	0.51034	0.00217	0.09135
0.50%	10	1000	18804.614	0.13712	0.50688	0.00411	0.12646
0.50%	20	1000	38602.258	0.19647	0.48995	0.00796	0.18208

Table 7: Statistics for the GI model using 2-states

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

SAMPLES	LENGTH	PLOSS	E(B)	PLOSS	PLOSS	PLOSS
				μ (%)	Σ	cV
40	10000	2%	10	1.946%	0.24543	0.25458
40	20000	2%	10	1.996%	0.18467	0.2152
40	100000	2%	10	2.009%	0.04589	0.10663
40	200000	2%	10	1.986%	0.0162	0.06424
40	1000000	2%	10	2.014%	0.00376	0.03044
40	2000000	2%	10	2.008%	0.00181	0.02120

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

The number of samples considered does not 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	2.013	0.02552	0.07936
40	2%	10	1.997	0.01208	0.05503
50	2%	10	1.988	0.01598	0.06356
100	2%	10	2.005	0.02036	0.07115
200	2%	10	2.008	0.02023	0.07083
250	2%	10	2.001	0.02092	0.07228
500	2%	10	2.002	0.0172	0.06549
1000	2%	10	2.000	0.01744	0.06602
2000	2%	10	2.003	0.01858	0.06805

Table 9: Dependence of mean, variance and coefficient of variation on the number of samples for the GI model using 2-states

7.3.4 Statistics for Gilbert-Elliot 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 *loss_gilb_ell* model.

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	0.204	1	100	0	16.94	1.075	0.069	100	100	0	16.84	1.081	0.061	99.65	100	0
gilb_ell	0.204	1	1	0	0.169	0.00015	0.072	50.74	3.921	0	0.169	0.00018	0.078	50.83	3.963	0
gilb_ell	0.204	1	2	0	0.338	0.0005	0.082	67.44	4.418	0	0.333	0.0006	0.077	66.88	4.448	0
gilb_ell	0.204	1	2	0.1	0.421	0.0069	0.065	67.44	4.418	0.1	0.420	0.0074	0.064	67.41	4.418	0.0842

Table 10: Statistics for Gilbert-Elliot model

7.3.5 Statistics for the GI model using 3 and 4 parameters

In this section we reconsider the full GI model using 3 and 4 parameters 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 experiments is 50 and the length of each sequence is 200000.

PLOSS	E(B)	ρ	PISOL	PLOSS	PLOSS	PLOSS	E(B)	ρ	PISOL	E(GB)
				μ	σ2	cV	μ	μ	μ	μ
2	5	100	0	1,995	0,01195	0,05479	4,97762	100	0	0
2	5	80	0	2,000	0,0072	0,042	5,023	79,889	0	1,336
2	5	50	0	2,009	0,0023	0,024	5,001	50,106	0	2,656
2	10	100	0	1,984	0,0103	0,051	9,926	100	0	0
2	10	80	0	1,989	0,0111	0,053	9,972	80,068	0	1,284
2	10	50	0	2,008	0,0100	0,049	9,977	50,007	0	2,254
2	20	100	0	1,982	0,037	0,097	20,104	100	0	0
2	20	80	0	2,026	0,0224	0,073	20,213	79,955	0	1,266
2	20	50	0	1,998	0,018	0,067	20,034	50,073	0	2,108
1	5	100	0	0,978	0,0034	0,059	4,939	100	0	0
1	5	80	0	0,984	0,0036	0,061	4,970	80,045	0	1,334
1	5	50	0	0,993	0,0023	0,048	4,956	50,163	0	2,666
1	10	100	0	1,006	0,0085	0,091	10,007	100	0	0
1	10	50	0	0,993	0,0050	0,071	10,09	49,951	0	2,247
1	20	100	0	0,962	0,0196	0,145	19,50	100	0	0
1	20	80	0	1,019	0,0140	0,116	20,37	80,049	0	1,264
1	20	50	0	0,993	0,0066	0,082	19,91	50,047	0	2,104
0,5	5	100	0	0,497	0,0019	0,088	5,026	100	0	0
0,5	5	80	0	0,505	0,0018	0,084	5,029	80,066	0	1,329
0,5	5	50	0	0,502	0,0007	0,054	5,035	49,890	0	2,679
0,5	10	100	0	0,490	0,0032	0,118	9,938	100	0	0
0,5	10	80	0	0,502	0,0030	0,109	10,069	79,853	0	1,289
0,5	10	50	0	0,489	0,0031	0,114	9,7887	50,211	0	2,252
0,5	20	100	0	0,515	0,0136	0,226	20,256	100	0	0
0,5	20	80	0	0,515	0,0097	0,191	19,985	79,815	0	1,264
0,5	20	50	0	0,504	0,0040	0,126	20,148	50,093	0	2,095
2	10	100	0,01	1,999	0,01597	0,063	10,019	100	0,0092	0
2	10	80	0,01	1,987	0,01118	0,053	9,864	80,0142	0,0096	1,284
2	10	50	0,01	2,011	0,0059	0,038	10,018	49,997	0,00907	2,252

Table 11: 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

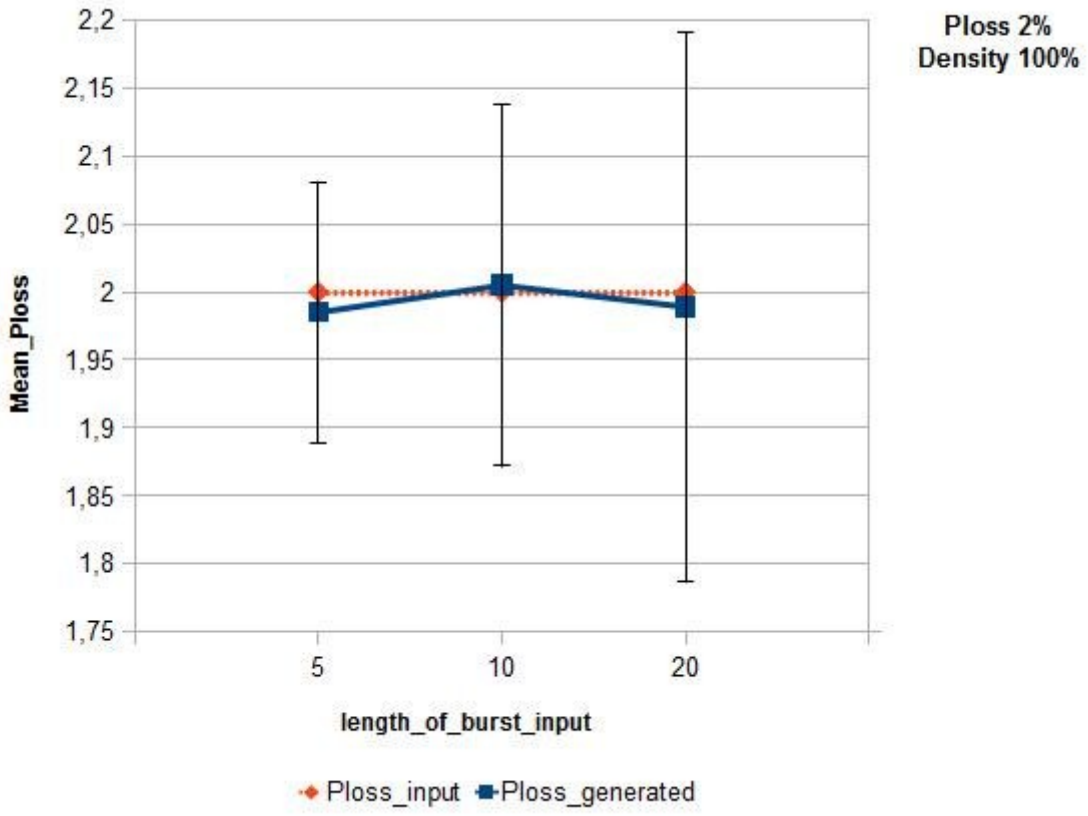


Figure 9: Generated loss probability (2%) versus burst length, Density 100

The first plot was made using a 2-state model specifying the two GI parameters P_{LOSS} and E(B). 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}.

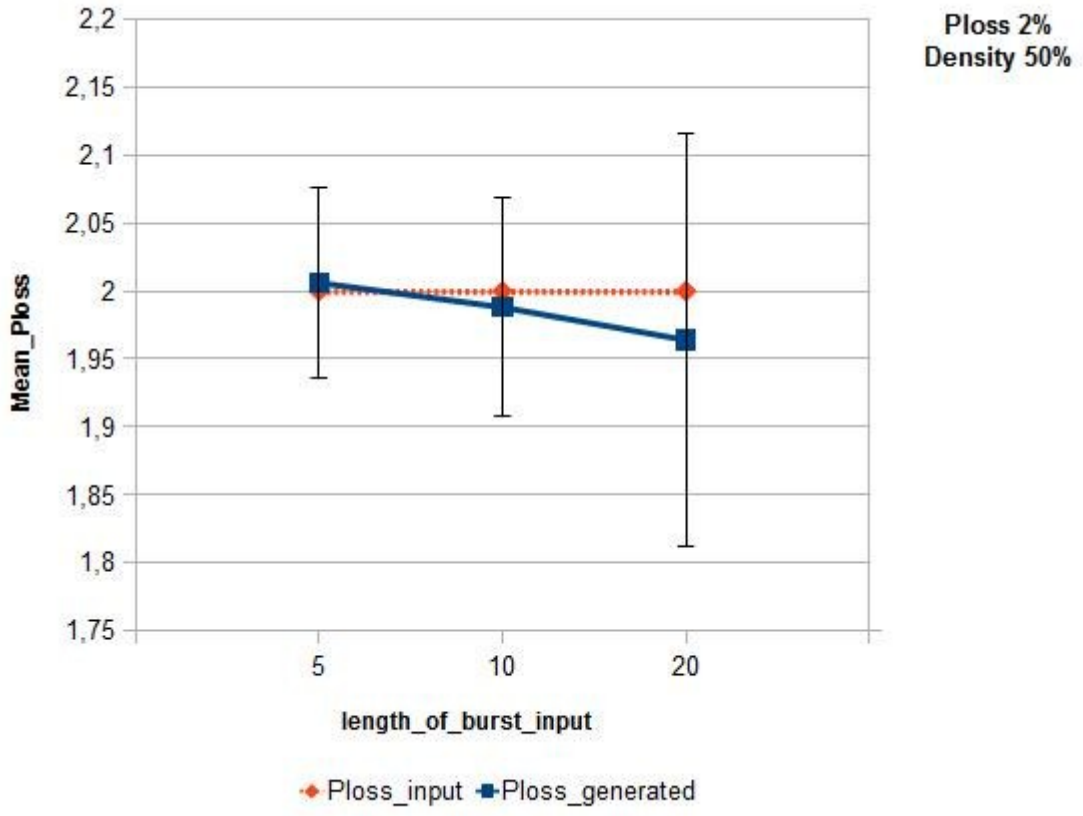


Figure 10: Generated loss probability (2%) versus burst length, Density 50%

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.

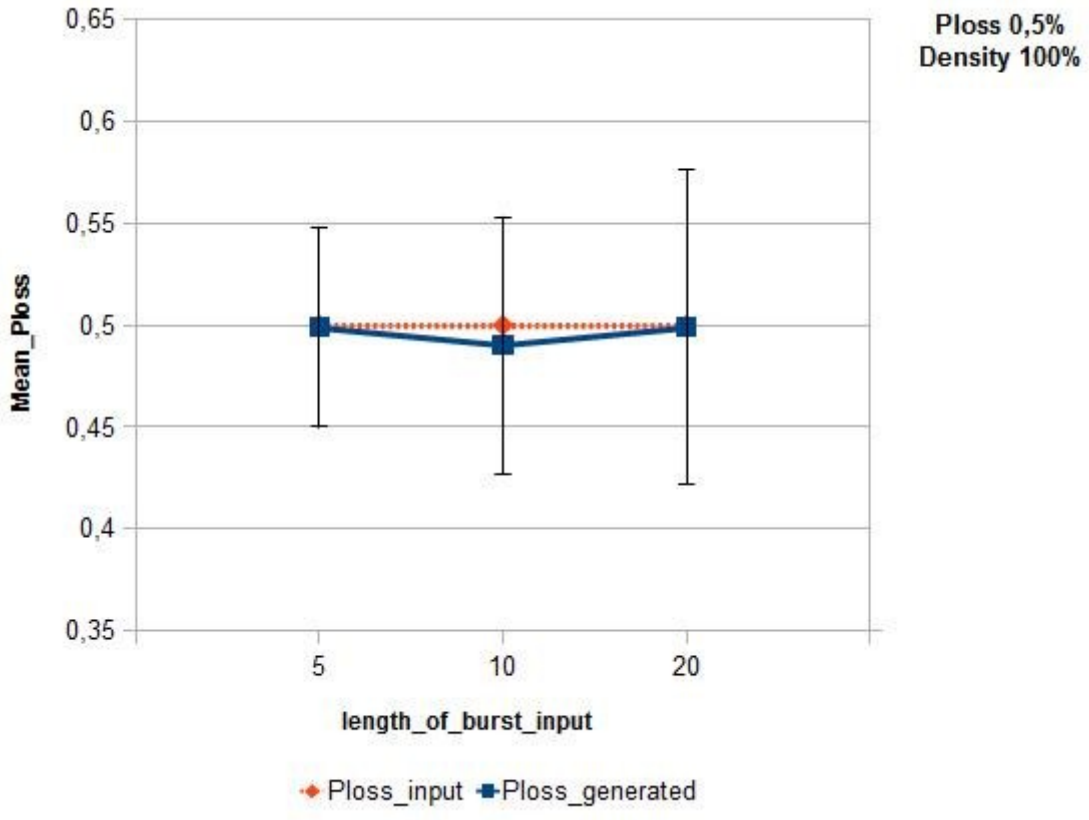


Figure 11: Generated loss probability (0.5%) versus burst length, Density 100%

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.

8 Testing the Kernel Space implementation

8.1 Trace2seq: generating loss sequences from traces

Trace2seq generates a loss sequence file suitable to be processed by sqnanalyzer starting from traces captured with tcpdump. *Trace2seq* is developed to evaluate the correct behavior of the various loss models in the new *Netem* version inside the Linux kernel. The program is written in C language and works from command line and receives as input parameter the number of packets transmitted. 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. *Trace2seq* generates a sequence, taking from the file prova.txt the ICMP packets stored in the file using TCPDUMP. Then delete packets of type "request" and finally checks the sequence number of ICMP packets by setting the value '0' for sequence number in the file, and the value '1' to the sequence number that have been lost. The sequence generated is saved in the file *sequence.txt*. To check the GI parameters analyze the sequence generated by the program *sqnanalyzer.c* that runs a *"blind external check"*, as in the case of sqngen.

8.2 Compile trace2seq and sqnanalyzer

To compile *trace2seq* and *sqnanalyzer* through *gcc* use the command:

```
g++ trace2seq.c  -o trace2seq

g++ sqnanalyzer.c  -o sqnanalyzer
```

8.3 How to use TCPDUMP

Once you have chosen the loss model to apply to the Kernel, we can save data traffic in the file prova.txt.

We open two terminals. In the first we type the command:

```
tcpdump  -i wlan0 -n icmp -w prova_2.txt
```

In this way tcpdump is listening, that is, expected packets are transmitted. So to transmit the packets in the second terminal type the command:

```
sudo ping -i 0.1 –c 200  192.168.2.1
```

When the packet transmission is completed, we block tcpdump by typing Ctrl+C, and then to convert the file prova_2.txt into a file readable in a text editor using the command:

```
tcpdump  -r prova_2.txt > prova.txt
```

8.4 Sqnanalyzer: Sequence analyzer

Sqnanalyzer is a program written reusing part of the code of *sqngen* to analyze a known loss sequence made of "0" and "1" stored in a text file. The syntax is the following:

```
sqnanalyzer sequence.txt
```

where *sequence.txt* stands for the name of the file that contains the sequence.
Once the sequence has been read, it is analyzed by the algorithm used by *sqngen*'s external check that calculates the GI parameters, repeating the analysis starting from g_{min}=2 until g_{min}=128 doubling its value at each step. The following figure shows an example with a sequence generated from GI model with parameter Ploss = 20% and Burst = 5.

```
daniele@daniele-Satellite-L300:~$ ./sqnanalyzer sequence.txt

The sequence is:
00000000000000000000000000000000011000000000000000000000000010000000000000000000000000000
00000000000000000000111000011000000000011111111100011111000110000000000111110000000000000000
00100000001100000000000000000000000000000000000000000001110000001000000000000000001111111111111
11111100000000000000000000000000111100000000000000000000000000000000000000000000000000000000000
00000000011111111111111100000000111000000000000000010011110000000000000000000000000000000000
10000011111111100000000000001111011111111000000000000000000000000010000000000001000000000000
00000000000000111111110000000000000000000000000000111100000000000000000011111100000000000000
00000000000000000000000000000000000000000000000111100000000000000000000000000000000000111111
11111100000000000000000000000011000000000011110000000000000000000000000000000011110000000000
00100000000000000111000000000000011111110000000000000000000000000000000000000000001000000
00000000000000000000000000000000000000111100000000010000000111100000000000000000000000000
00000000000000000000000000000000000100000000000000000000000111111100000000000000000000000000
00000001110000011111000001000000000001111111111111011000000111110000000111010000011111111111
11111111111111000000000000011111000000000111100000000000000000011111111111100000000000111111
11111100110000000000000011110001110000000011111110000000000000000000000000000000000000000000
00000000000000000000000000011000000000000000000000000000111111111000000000000000001111000000
00000000000000000000011100000000000000000000000000000000000000000000000000000000000000000000
00000011000000011111111111000000000000000000000000000011000000000000000000000000000000111111
11100001000000000000000000000000000000000000010000000000000000000000000000000000000000000000
00000000000000000100001000000000000000000000000000000000000000000000000000000000000010000000000
00000000001000000000000011000000000000000000000000000000000000000000000000000000000000000000000
00011

Length is: 2000


For GMIN=2 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 6.113 Density 99.074% P_isol 0.900% Mean_length_of_good_burst 1.000

For GMIN=4 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 6.837 Density 95.821% P_isol 0.900% Mean_length_of_good_burst 2.000

For GMIN=8 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 9.023 Density 83.505% P_isol 0.750% Mean_length_of_good_burst 4.000

For GMIN=16 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 19.690 Density 57.093% P_isol 0.650% Mean_length_of_good_burst 7.656
```

For GMIN=32 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 59.625 Density 34.696% P_isol 0.400% Mean_length_of_good_burst 12.460

For GMIN=64 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 202.857 Density 23.521% P_isol 0.250% Mean_length_of_good_burst 17.516

For GMIN=128 the generated sequence has the following characteristics (from sequence analysis):
P_loss 16.950% Mean_length_of_burst 1966.000 Density 17.243% P_isol 0.000% Mean_length_of_good_burst 22.288

Now we show another example another example of the analysis through the *sqnanalyzer* and *trace2seq* program. The following figure shows an example with a sequence generated from Gilbert-Elliot model with parameter p = 20% and r = 30%.

$ sudo tc qdisc add dev wlan0 root netem loss gemodel 20 30

daniele@daniele-Satellite-L300:~$ ./trace2seq 2000
La sequenza è stata generata.

daniele@daniele-Satellite-L300:~$ ./sqnanalyzer sequence.txt

The sequence is:
000111100000000000011110010001000000000000000000000000000011111000010000000011110000000001101100000
100001101000000000000000111110000110000000000000000000011110000010000111111111111111100000000000000
0110000000000011111111111000000000101110000000000000000011111100001111111000000111110000000000010111
111100011100000000000111111000000000000111000000000000001111111100001111111111100000001111111110000
100110000000011100100011111111111101111000111111100000000111011110000000111000000000000000011101100
111100000000100000011011111111110000000100000011001001100000111000011001111100000000000000000000000000
110000000000111000110000111000011111111111000011111101110000001110000000000000000000000000000000000
000101000000111000111100000000011111001000011111111000000000000000000000100010111100101111111110000
001100010001000010000100110001111101100001001110110111111101111111110000000000101111100000001000
000011100001111000001100110100000000000000001111100001011110111100000011010011000111000000011000011
1111110000000110000111011111110000000000000000000001111111111010000110000000100000000100000000001111
100000100000001000000000001000001000011110001100010111111110000101100011001100001100111110000010 0
001111100001111111101111111111100001100010011001100110111001110000001111111111100000000111111011
111111100110000001101000000000000100110111111111011011100011110000011000000000000000000001000000000 0
1111100001111100010011111000000000001111000111100000001111000000000011111111111111000000000000000
011100100111111111110000100000000000000001100111000110000000000000000001001100000000000001111000101100
0000000011000111100010111111111100001111111000011111000000011110000000000000000000000111111001111111000
1111000000000000000000001100010000110000000000001110001100010111000000000110110000000100011110000000
00000000000000001100011001000000000100010001111000000000011001001111111011100000001111000000000010
000110001101111111110000000000011110001111001000000111111100000000110000000000000000001100111111000
0111110100000111011110111100010011100000111000000000000110000000000000100000000000000000101000000011
10111

Length is: 2000

For GMIN=2 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 5.257 Density 96.037% P_isol 2.850% Mean_length_of_good_burst 1.000

For GMIN=4 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 9.069 Density 80.757% P_isol 1.850% Mean_length_of_good_burst 1.935

For GMIN=8 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 26.833 Density 59.394% P_isol 0.950% Mean_length_of_good_burst 3.189

For GMIN=16 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 112.267 Density 46.081% P_isol 0.400% Mean_length_of_good_burst 4.365

For GMIN=32 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 977.500 Density 40.000% P_isol 0.100% Mean_length_of_good_burst 5.167

For GMIN=64 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 1997.000 Density 39.259% P_isol 0.000% Mean_length_of_good_burst 5.274

For GMIN=128 the generated sequence has the following characteristics (from sequence analysis):
P_loss 39.200% Mean_length_of_burst 1997.000 Density 39.259% P_isol 0.000% Mean_length_of_good_burst 5.274

Finally the following figure show the last example with a sequence generated from 4-state model with parameter p13 = 5% and p31= 20%.

$ sudo tc qdisc add dev wlan0 root netem loss state 5 20

daniele@daniele-Satellite-L300:~$ ./trace2seq 2000
La sequenza è stata generata.

daniele@daniele-Satellite-L300:~$ ./sqnanalyzer sequence.txt

The sequence is:
00011111100000000000000000000000000000000000000011111111111000000000111000000000000000000000
00000000000000000000000000000000000000000000100000000000000000000000001100000000000000000000
00000000000000001011111111111111000000001111111000000000011111111111000000000000000000000000
00000000111111111000011110000000000000000000000000000000000001111110000011111111100000000011
10000000000000000000000000000000000001000000011111111000000000000000000000000000000000000000
00000000000000111100000000000000000010000000000001000000000000000000110000000000000000000000
00000000000000000000000000000000000000000000011100000000000000000000000011110000000000000000
00000000000000000000000000000011111000000000001100001000000001111110100000000000000000000000
00000000000111100000110011111000000000000000000000011110000000000000011111111000000000011000
00111111111111010000011111100000000110000000000000000000011000000000000010000000000000000000
00000000000000000000000000000000000000000100000000000000001000000000001100000000000000000011
10000100000000000000000000000000001111111100000000000000000000000000000100011110000000001
01000000000000000000000000000011111000000000000000000000010000000000000000000000000000000000
00011111111111111110000000000000000000011111100000000000000000000000000000000000000011111111
11111111100000000000000000000000000000000000000000011111111111111110000000000000000011110000
00000011000000000000000000011111100000111110000000000011000000000011111110000000000000000000
00000111111111111100000011110011101111110000000000000000000000000000000000000111100000000110
00111111111111100000000011100000000000000000000000000000000000000000000000011100000000000000
00000000000010000000000000000000000000011000000000000000000000000000000000000000000001111
11101011111111000001111111111000000000111100000000000000000000000000000000000000000000000000
00011111111111100000000000000000011111110000000000001111000111111110000000011000000000000010
00000

Length is: 2000

For GMIN=2 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 6.306 Density 98.721% P_isol 0.750% Mean_length_of_good_burst 1.000

For GMIN=4 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 6.746 Density 96.985% P_isol 0.750% Mean_length_of_good_burst 1.500

For GMIN=8 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 9.120 Density 85.088% P_isol 0.650% Mean_length_of_good_burst 3.579

For GMIN=16 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 19.471 Density 59.063% P_isol 0.500% Mean_length_of_good_burst 6.949

For GMIN=32 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 52.750 Density 37.441% P_isol 0.300% Mean_length_of_good_burst 11.579

For GMIN=64 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 612.667 Density 21.763% P_isol 0.050% Mean_length_of_good_burst 18.203

For GMIN=128 the generated sequence has the following characteristics (from sequence analysis):
P_loss 20.050% Mean_length_of_burst 1997.000 Density 20.080% P_isol 0.000% Mean_length_of_good_burst 19.463

8.5 Comparison of statistical analysis between User and Kernel space

8.5.1 Statistics for the GI model using 2, 3 and 4 parameters

In this section we reconsider the full GI model using 2, 3 and 4 parameters P_{LOSS}, *E(B), Density,* P_{ISOL}. We will calculate the user space value through the *sqngen* program and the value of kernel space using *sqnanalyzer*. Both methods calculate the results for different value of the gmin. The number of experiments is 1 and the length of the sequence is 20000.

	PARAMETERS				USER SPACE				KERNEL SPACE			
	PLOSS	E(B)	DENSITY	PISOL	PLOSS (%)	E(B)	DENSITY (%)	PISOL (%)	PLOSS (%)	E(B)	DENSITY (%)	PISOL (%)
GMIN = 2	20	5	100	0	19,335	6,212	99,095	0,715	20,310	6,341	99,090	0,705
	20	5	80	0	20,350	5,323	88,226	3,040	19,530	5,119	88,736	3,065
	20	5	80	5	20,125	5,353	88,672	6,075	20,010	5,489	87,780	6,110
GMIN = 4	20	5	100	0	19,335	7,058	95,542	0,690	20,310	7,119	95,536	0,620
	20	5	80	0	20,350	7,883	77,123	2,415	19,530	7,437	78,047	2,465
	20	5	80	5	20,125	8,043	76,030	5,265	20,010	7,905	76,506	5,405
GMIN = 8	20	5	100	0	19,335	9,720	83,009	0,575	20,310	10,146	82,434	0,530
	20	5	80	0	20,350	12,051	64,583	1,905	19,530	11,010	65,379	1,930
	20	5	80	5	20,125	13,223	60,250	4,430	20,010	13,751	58,460	4,375
GMIN =16	20	5	100	0	19,335	20,557	57,932	0,400	20,310	19,697	60.048	0,380
	20	5	80	0	20,350	27,595	45,595	1,225	19,530	27,270	43,368	1,140
	20	5	80	5	20,125	45,830	35,292	2,495	20,010	43,401	35,597	2,475

Table 10: Statistics for the GI model using 2, 3 and 4 parameters

As we can see from the above table, the resulting values of user space and kernel space are very similar.

8.5.2 Statistics for the Gilbert-Elliot model using 2, 3 and 4 parameters

In this section we report the statistics of a simulation relative to the *Gilbert-Elliot* model performed specifying all parameters p, r, 1-h and 1-k. Also in this case we will calculate the user space and kernel space values as in the section. The number of experiments is 1, and the length of the sequence is 20000.

	PARAMETERS				USER SPACE				KERNEL SPACE			
	P	R	1-H	1-K	PLOSS (%)	E(B)	DENSITY (%)	PISOL (%)	PLOSS (%)	E(B)	DENSITY (%)	PISOL (%)
GMIN = 2	20	30	100	0	39,645	5,262	95,521	3,180	39,650	5,430	95,273	2,970
	20	30	40	0	16,410	2,962	92,640	8,040	16,195	3,140	91,189	7,605
	20	30	40	10	21,810	3,242	89,032	10,770	22,285	3,149	89,917	10,960
GMIN = 4	20	30	100	0	39,645	8,911	80,760	2,225	39,650	9,464	80,537	2,035
	20	30	40	0	16,410	4,873	71,429	6,960	16,195	4,676	74,297	6,815
	20	30	40	10	21,810	5,641	67,917	9,225	22,285	5,767	67,236	9,285
GMIN = 8	20	30	100	0	39,645	29,614	57,625	0,980	39,650	26,696	60,044	1,180
	20	30	40	0	16,410	11,345	45,440	5,275	16,195	10,787	47,583	5,365
	20	30	40	10	21,810	19,007	40,206	6,220	22,285	20,142	39,754	5,630
GMIN=16	20	30	100	0	39,645	247,311	43,112	0,195	39,650	216,869	43,366	0,150
	20	30	40	0	16,410	47,295	26.218	2,770	16,195	47,718	26,072	2,945
	20	30	40	10	21,810	163,626	25,045	1,525	22,285	151,796	25,595	1,305

Table 11: Statistics for the Gilbert-Elliot model using 2, 3 and 4 parameters

As above, the test result shows that the value of user and kernel space are very similar.

8.6 Generated loss probability plots

8.6.1 Mean burst length plots

The plots concerning length of burst show the variation of E(B) parameter with the value of gmin chosen for the external check algorithm.

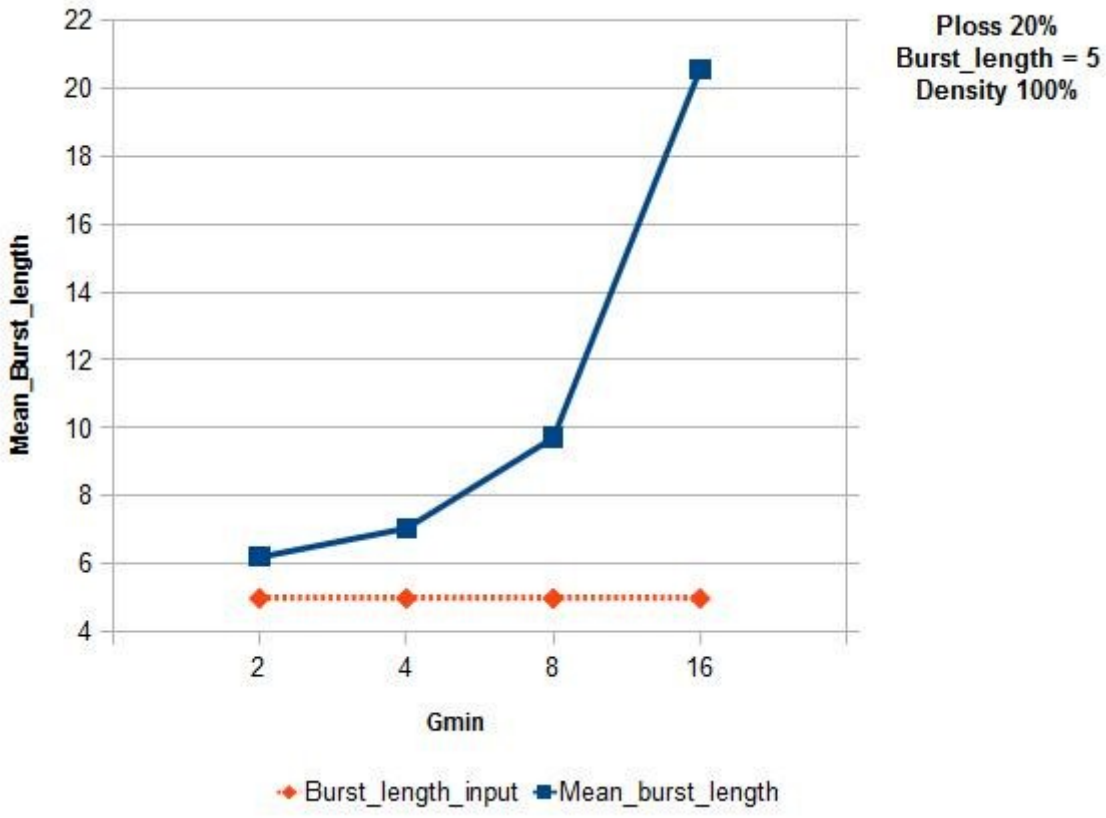


Figure 12: Burst length (5) versus gmin, Ploss 20%, Density 100%

In this case the burst length is fixed to 5,the ploss to 20% and density to 100%. The most interesting fact is the dependence from g_{min} of the value estimated though external check. The value is close for small values of g_{min} and is farther if we consider higher values of g_{min}. This is connected to the chosen input value for burst length.

8.6.2 Mean density plots

The plots concerning density, as for burst length, show the variation of the parameter with the value of g_{min} chosen for the external check algorithm.

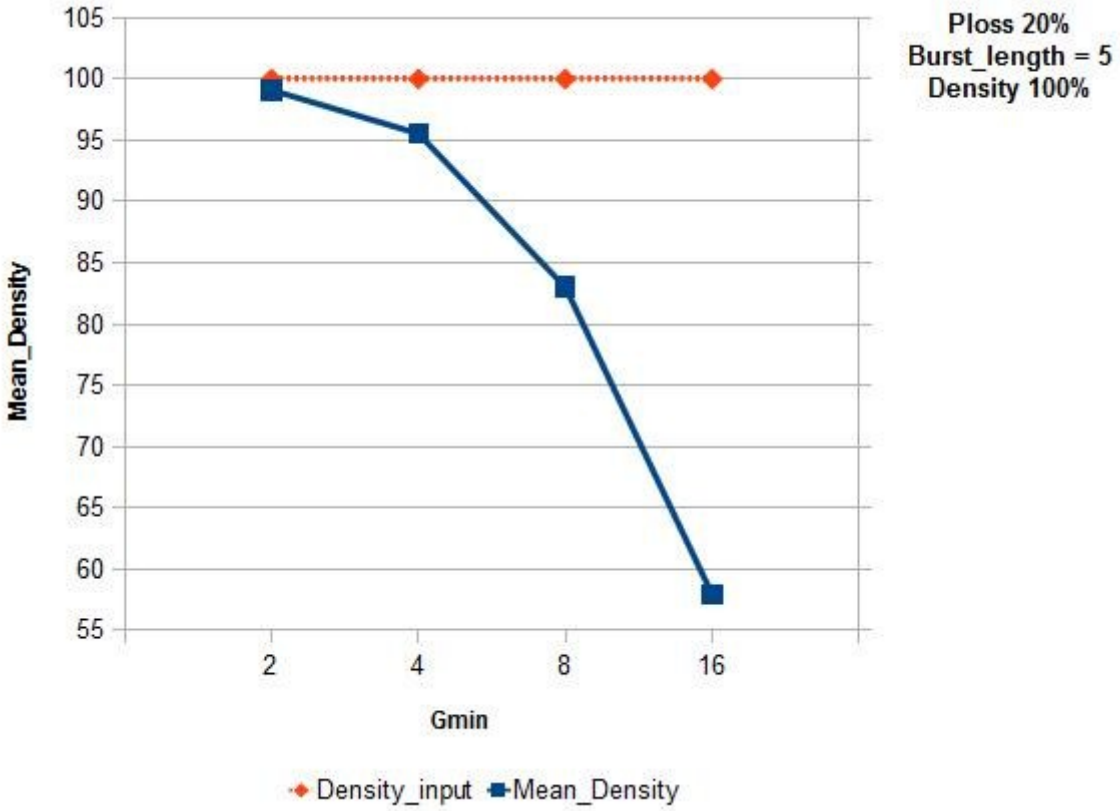


Figure 13: Density (100%) versus gmin, Ploss 20%, Burst length 5

In this case the estimated value is good only for small values of g_{min}. The question is mirror to the burst length case: if we choose to generate a small value of burst length, we get a sequence with short bursts so the estimation will be good only for comparable values of g_{min}.

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 Techical 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