FREIE UNIVERSITÄT BERLIN

NetemCG – IP Packet-Loss Injection using a Continuous-Time Gilbert Model

Philipp Reinecke, Matthias Dräger, and Katinka Wolter

TR-B-11-05
September 2011

FACHBEREICH MATHEMATIK UND INFORMATIK
SERIE B • INFORMATIK

NetemCG – IP Packet-Loss Injection using a Continuous-Time Gilbert Model

Philipp Reinecke, Matthias Dräger, and Katinka Wolter

Freie Universität Berlin
Institut für Informatik
Takustraße 9
14195 Berlin, Germany
{philipp.reinecke, matthias.draeger, katinka.wolter}@fu-berlin.de

Abstract. Injection of IP packet loss is a versatile method for emulating real-world network conditions in performance studies. In order to reproduce realistic packet-loss patterns, stochastic fault-models are used. In this report we desribe our implementation of a Linux kernel module using a Continuous-Time Gilbert Model for packet-loss injection.

1 Introduction

Testbed-driven performance and dependability evaluation of distributed systems requires the emulation of a realistic networking environment. Common disturbances such as packet loss may have an adverse effect on both dependability and performance of the network. As illustrated in e.g. [1, 2], packet loss may affect dependability of higher networking and system layers even with the reliable TCP protocol, which guarantees reliable data transmission. Consequently, methods for reproducing disturbances are required.

Injection of IP packet loss is an indispensible tool when evaluating fault-tolerant network protocols. Fault injection at the IP level also provides a convenient way for assessing performance and reliability of distributed systems in which the network stack is considered just an off-the-shelf component. As the injected faults force the network stack to apply the same fault-handling procedures that are executed under real-world operating conditions (such as e.g. packet retransmissions in TCP), the same operational patterns of the network stack that are present in a real network can be expected to emerge in the testbed.

IP packet loss patterns have been the focus of intense study in the past decades (e.g. [3–5]). While in the simplest case packet loss may be described by a Bernoulli model, packet loss is often comprised of bursts of elevated loss probability, which can be modelled more closely with Gilbert-Elliot (GE) models [3–5]. A Gilbert-Elliot model describes the loss process as a Markov Chain with different loss probabilities for each state.

There exist a number of fault injection modules for the Linux kernel. The Bernoulli loss model is implemented by the default Linux kernel module Netem.[1]

[1] Note that Netem supports the injection of other disturbances as well as a simple correlated loss model, but these are out of scope for this report.

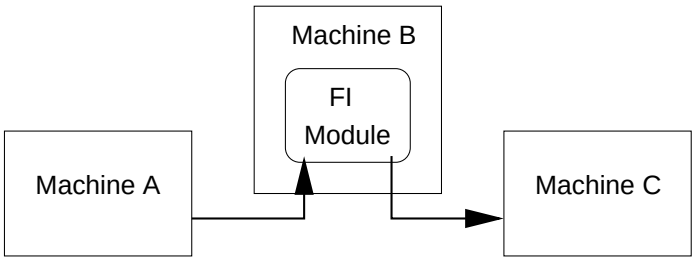

Fig. 1. General approach to packet-level fault-injection.

In [6] the default module was modified such that it allows replaying packet-loss traces. The Netem-CLG (Correlated Loss Generator) module [7] extends the default Netem implementation by a discrete-time extended Gilbert-Elliot model. In this report we describe our implementation of a continuous-time Gilbert-Elliot model for packet-loss generation in the Netem module. We start with the theoretical background in Section 2. We then describe our modifications to the default module in Section 3 before illustrating installation and use of the module in Section 5.

2 Background

The most straightforward way of injecting packet loss according to a loss process described by a loss model is depicted in Figure 1: One computer is set up to act as a router between two or more hosts or networks. On this machine, the TCP/IP stack is augmented by a fault-injection (FI) module. For each arriving outside packet the FI module determines whether to forward the packet to its destination or to discard it, according to the loss model. With a Bernoulli loss model, for each packet one trial is performed and the packet is dropped according to the outcome of this trial. With a Markovian (or Gilbert) model, the dropping probability for the Bernoulli trial is selected according to the current state of the model when the packet arrives. Markovian models have the advantage that they enable accurate modelling of elevated loss models or loss bursts.

Markovian loss models, often called Gilbert models are typically defined as Discrete-Time Markov Chains (DTMCs) (e.g. [5]). The left part of Figure 2 shows an example of a two-state discrete-time Markovian loss model with loss probabilities l_1 and l_2 and state transition probabilities p_{12} and p_{21}. In a DTMC loss model, state changes occur at discrete time intervals. Certainly, this view is appropriate for modelling the loss process as described by a packet trace: Each packet constitutes a time instant at which a state-change may occur. Implementation of fault-injection using a discrete-time model is straightforward as well: With each arriving packet the next state is selected randomly from the possible successor states (including the same state) of the current state.

In the NetemCG module we take an alternative view, in that we assume a Continous-Time Markov Chain (CTMC), instead of a DTMC. Here, state

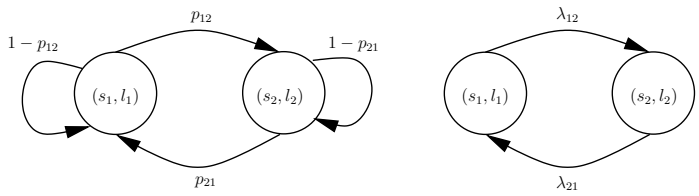

Fig. 2. DTMC and CTMC models of the packet-loss process.

sojourn times are described by exponentially distributed random variables. The right-hand side of Figure 2 illustrates a two-state continuous-time Markovian loss model, again with loss rates l_1, l_2. Transition rates are λ_{12} and λ_{21}. Packet-loss injection according to a continuous-time Gilbert model is implemented by playing the CTMC, as follows: Upon entering a state, the state sojourn time is drawn from an exponential distribution with rate equal to the sum of outgoing rates. When this time has passed, the next state is chosen based on the embedded Markov chain.

The main difference between the DTMC and CTMC models is that the former only perform state-changes if and when a packet arrives, whereas state-changes in the latter occur independently of the arrival process. That is, the former actually relies on a DTMC embedded in the arrival process, whereas the latter is independent of the arrival process. While both models are equally capable of describing a packet-loss trace, they have different effects on packet streams whose arrival rate changes in response to packet loss, as illustrated in [8].

3 Implementation

We implemented support for the continuous-time Gilbert model as a modification of the Netem module. The model is specified in a text file and sent to the Netem module via the `tc` command from the `iproute2` package [9]. Some hands-on examples for using the module are shown in Section 5. In the following subsections we describe a few of the implementation challenges, and our approaches to solving them.

3.1 Parameter specification

A loss model in NetemCG consists of at least one node, and an arbitrary (including zero) number transitions. It is possible to create a loss model with an arbitrary number of nodes and transitions. Both the default Netem module and the NetemCLG module only allow for small models with fixed size and thus require only a few parameters for specifying the fault process. These parameters can easily be specified on the command-line of the `tc` tool. However, this simplicity comes at the cost of allowing only a limited number of states. For the NetemCG module, where we wanted to be able to specify an arbitrary number

```
ModelDescription = { '\#' { Comment } '\n'
        | { '' '' '\n' }
        | Node '\n'};
Comment = { Any character };
Node = NodeNumber LossProbability {[ TransitionList ]} '\n';
NodeNumber = Any natural number;
LossProbability = Any real number in [0,1];
TransitionList = { Transition };
Transition = (NodeNumber ',' TransitionRate, ',' Scale);
TransitionRate = Any natural number;
Scale = Any integer number;
```

Fig. 3. NetemCG Input File Syntax

of states, parameter specification on the command line would have been impractical. Therefore, we augmented the `tc` tool such that it loads model descriptions from a text file and transmits them to the kernel module. This was made difficult by the fact that the routines used for communication with the module are limited with respect to the size of the data to be transmitted. Note that a similar problem was encountered by the authors of the trace-driven approach [6], where the trace had to be loaded into the kernel module. In [6], an additional problem was the size of the trace, which could be larger than the amount of memory available to the module. The solution applied by [6] was to load the trace sequentially, sending a new chunk on-demand. In NetemCG, we need the complete model inside the module at any time, and cannot load it on-demand.

We implemented loading the model into the kernel as follows: First, the CTMC model described in the input file is parsed and the Embedded Markov Chain (EMC) as well as the mean sojourn times for each state are computed. We then transmit the number of nodes, the list of nodes (specified by their sojourn times) and finally the EMC to the kernel, using the Netlink interface.

The syntax of the model file is straightforward: Each line can contain either whitespace, a comment, or the description of a node. Comments are specified by starting the line with a `#`. Nodes are specified by the number of the node, followed by the loss probability (in $[0, 1]$) and the list of transitions in brackets. Each transition is a tuple of the number of the target state, the transition rate, and the exponent of the scale parameter (more on that in the next section). Figure 3.1 gives the EBNF description of the input file syntax.

3.2 State Transitions

The loss process is generated in the kernel module as follows: First, each time a packet arrives a Bernoulli experiment is performed using the current loss probability. Second, packet loss probabilities change according to the CTMC loss model, i.e. every time the CTMC enters a new state, that state's loss probability becomes the current loss probability. The CTMC is played based on the

EMC and the state sojourn rates. Upon entering a state, the module draws an exponentially-distributed random number with parameter equal to that state's sojourn rate, and sets a kernel timer using this value. When the timer expires, the module chooses the next state based on the outgoing probabilities in the EMC, and enters the next state. Initially, the module is in the first state given in the input file.

3.3 Kernel Limitations

Since the module runs within the Linux kernel, a few limitations do apply. First, the kernel does not support floating-point numbers. This requires mapping of random numbers in the range $[0, 1]$ to the range of long integers ($[0, 2^{32} - 1]$). Second, in order to be able to specify loss period lengths in different ranges, we introduced a scale parameter s. The scale parameter is a natural number in the interval $[0, 9]$ and is used as the negative exponent to base 10 when scaling the rate. Third, in order to generate random variates with exponential distribution efficiently, we applied the inversion method. Unfortunately, the kernel does not support the logarithm operation. We remedied this by implementing the logarithm using a table of pre-computed logarithm values, applying binary search to identify the required value.

Note that some practical limits are imposed by both the kernel and the range of values for the variables: First, the resolution of the kernel timer limits the minimum sojourn time for a state. With the default resolution of 250Hz, the minimum sojourn time is 4 ms. The maximum sojourn time is bounded by the `MAX_INT` value, which is typically $2^{32} - 1$. Second, the size of data structures in the `tc` command limits the maximum mean sojourn time that can be specified to 10^9 ms. Third, the sum of all outgoing rates for a node must be less than 2^{32} to avoid overflows.

4 Compilation and Installation of NetemCG

We provide the source code as patches to the Linux kernel source tree and the `iproute2` package. The patches are available at http://cst.mi.fu-berlin.de/staff/reinecke.html.

To patch the Netem module a Linux kernel version 2.6.35 and newer is required. The patch can be applied as follows:

```
cd <Linux kernel source>
patch -p1 < netemcg-kernel.patch
```

The new kernel version can then be configured, compiled, and installed in the usual way. Note that the Netem module must be selected and that kernel modules must be compiled as well.

Likewise, the `iproute2` package needs to be patched by

```
cd <iproute2 source directory>
patch -p1 < netemcg-iproute2.patch
```

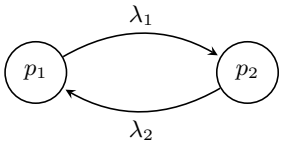

Fig. 4. A simple generic model.

Compilation then consists of running `make`. The new version may be installed using `make install`, although `tc` can also be run directly from the source directory.

5 Examples

We start with a discussion of the model file syntax. Consider the model given in Figure 4. The model consists of two states with loss probability p_1 and p_2 and transition rates λ_1 and λ_2. In order to specify this model for use by NetemCG, we assign numbers to the states and then give the transitions for each node. In this example, the model can be described by the following lines:

1 p_1 `[(2,` λ_1 `,0)]`
2 p_2 `[(1,` λ_2 `,0)]`

Note the trailing zero in each transition specification. This is the value of the scale parameter s_i for the transition rate. The scale parameter allows the specification of transition rates less than 1, since the effective transition rate is computed as

$$\lambda_i' = \lambda_i \cdot 10^{-s_i}.$$

To illustrate this further, let us now assume that $\lambda_1 = 0.05, \lambda_2 = 0.95, p_1 = 0$, and $p_2 = 1.0$. The resulting model file *example1.model* is

```
1 0.0 [(2,5,2)]
2 1.0 [(1,95,2)]
```

If the model is saved in the file `example1`, it can be loaded into NetemCG with the following command

```
tc qdisc add dev eth0 root netem model example1.model
```

Finally, let us consider a larger loss model, as shown in Figure 5. This model contains multiple states and multiple transitions. Note that state 2 has two outgoing transitions. We can specify the model as

```
1 0.0 [(2,2,0)]
2 1.0 [(1,5,1)(3,5,1)]
3 0.5 [(1,3,0)]
```

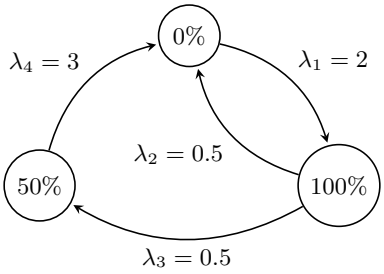

Fig. 5. Example 2: Loss model with multiple nodes und multiple transitions.

6 Conclusion

In this report we gave a description of the NetemCG module for injecting packet loss using a continuous-time Gilbert-model with arbitrary structure and size.

References

1. Reinecke, P., van Moorsel, A.P.A., Wolter, K.: The Fast and the Fair: A Fault-Injection-Driven Comparison of Restart Oracles for Reliable Web Services. In: QEST '06: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, Washington, DC, USA, IEEE Computer Society (2006) 375–384
2. Reinecke, P., Wolter, K.: Phase-Type Approximations for Message Transmission Times in Web Services Reliable Messaging. In Kounev, S., Gorton, I., Sachs, K., eds.: Performance Evaluation – Metrics, Models and Benchmarks. Volume 5119 of Lecture Notes in Computer Science., Springer (June 2008) 191–207
3. Zhang, Y., Paxson, V., Shenker, S.: The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. ACIRI Technical Report (2000)
4. Zhang, Y., Duffield, N., Paxson, V., Shenker, S.: On the Constancy of Internet Path Properties. In: IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, New York, NY, USA, ACM (2001) 197–211
5. Hohlfeld, O., Geib, R., Haß linger, G.: Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments. In: Proc. of the 16th International Workshop on Quality of Service (IWQoS). (June 2008) 239–248
6. Keller, A., Baumann, R., Fiedler, U.: TCN Trace Control for Netem. http://tcn.hypert.net/ (2009) (last seen 6 June 2011).
7. Salsano, S., Ludovici, F., Ordine, A.: Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel. Technical report, University of Rome "Tor Vergata"
8. Reinecke, P., Wolter, K.: On Stochastic Fault-Injection for IP-Packet Loss Emulation. In: Proceedings of the 8th European Performance Engineering Workshop – EPEW 2011. (2011) (to appear).
9. Hubert, B., Graf, T., Maxwell, G., van Mook, R., van Oosterhoutand P. Schroeder, M., Spaans, J., Larroy, P.: Linux Advanced Routing and Traffic Control. http://lartc.org Last visited August 18th, 2011.