

KauNet: Improving Reproducibility for Wireless and Mobile Research

Johan Garcia
Karlstad University / ENSICA
Universitetsgatan 2
Karlstad, Sweden
johan.garcia@kau.se

Emmanuel Conchon
LAAS-CNRS / ENSICA
7 avenue de Colonel Roche
Toulouse, France
econchon@ensica.fr

Tanguy Pérennou
LAAS-CNRS / ENSICA
7 avenue de Colonel Roche
Toulouse, France
perennou@ensica.fr

Anna Brunstrom
Karlstad University
Universitetsgatan 2
Karlstad, Sweden
anna.brunstrom@kau.se

ABSTRACT

This paper presents the KauNet emulation system that provides pattern-based emulation. KauNet enables bit precise placement of bit-errors, exact and repeatable packet losses, delays and bandwidth variations. The design and performance of KauNet is discussed. An example is also provided of how it can be integrated in a specific emulation framework to enhance emulation for mobile and wireless systems.

Categories and Subject Descriptors

C.2.m [Computer-Communication Networks]: Miscellaneous—*Emulation*; C.4 [Computer Systems Organization]: Performance of Systems

General Terms

Experimentation, Performance, Verification

Keywords

Network emulation, Pattern files, W-NINE

1. INTRODUCTION

Wireless and mobile networks provide their users with new levels of convenience, flexibility and freedom. New technologies and continuous research promise more connectivity and higher bandwidths. When developing and researching new technologies and architectures, means to assess their performance is paramount. However, evaluating the performance of wireless and mobile systems is challenging due to the complexities and number of variables involved. Performance can be evaluated by several metrics and at several levels of abstraction ranging from analytical evaluation, via

simulation or experiments in an emulated environment, up to full scale live experiments. Each of these approaches provides different trade offs with regards to the amount of detail considered, model validation requirements, degree of experimental control, reproducibility, and so on. In many cases the approaches are complementary and can provide different insights about the topic under study. In this paper we focus on the emulation approach and present the KauNet emulation system that enables a new degree of control and reproducibility, and thus provides a slightly different trade off compared to most other emulation systems. By employing fine-grained pattern-based control over the emulated behavior, KauNet enables emulation-based experiments with a high degree of control and reproducibility of the emulated conditions. The system is flexible with regards to the origin of the patterns which can be created from collected traces, previous simulations or analytical expressions.

The design of KauNet is centered around a number of pattern-handling extensions to the well known Dummynet emulator [13], together with user-space programs for pattern creation and management. The use of Dummynet as a starting point provides a stable code-base that has been in wide-spread use for several years, as well as the integration with the ipfw program for emulation setup and management. Dummynet has the ability to loose packets and to apply bandwidth restrictions and delays to the packets, thus emulating the desired link or network conditions. KauNet extends these abilities by also including the ability to introduce bit-errors. Furthermore, KauNet allows deterministic losses in addition to the probabilistic losses provided by Dummynet. In fact, KauNet allows bit-errors, packet losses, delay and bandwidth changes to be exactly and reproducibly controlled on a per packet or per-millisecond basis with the use of patterns.

With regards to emulating multiple connections or links, KauNet inherits the Dummynet firewall rule capabilities that are specified with ipfw. These capabilities make it easy to create multiple emulated flows emulating for example the effective bandwidths and delays for a set of nodes in a mobile network. When the nodes move around their effective conditions change, and the conditions in the emulated environment should change over time to reflect this. While Dummynet allows emulated conditions to be changed dur-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MobiEval'07, June 11, 2007, San Juan, Puerto Rico, USA.

Copyright 2007 ACM 978-1-59593-762-9/07/0006 ...\$5.00.

ing the run of the emulation, these changes must be done using command line tools (typically under the control of a script). Using regular Dummynet, there would be a need to precisely synchronize the invocation and execution of the command line tools in order to try and create an exact reproduction from one experimental run to the other. Instead of using command line tools to change the emulated conditions, KauNet uses patterns that describe the desired changes and also allow precise control over packet loss and bit-errors. These patterns are inserted into the KauNet kernel space at the start of the emulation, and allows a much more exact control of when the conditions change. Thus the fine-grained control over the evolution is under the control of the patterns, but higher-level dynamic events can still be incorporated by using the command line tools to dynamically switch between multiple patterns.

While this paper focuses on KauNet in the context of evaluation of mobile and wireless systems, it should be noted that KauNet is more generally applicable. It is also useful as a help for protocol implementation verification, and for general transport layer and application layer performance evaluations. KauNet has for example been used to examine web service response times, see [6].

The rest of the paper is structured as follows. Section 2 provides some background and discusses related work. Section 3 presents a more detailed description of the design and capabilities of the KauNet system. Section 4 gives an example of how integration between KauNet and a pattern-generation and emulation control system can be performed. Finally, some conclusions are provided.

2. BACKGROUND AND RELATED WORK

When evaluating performance there are four common approaches; analytical modeling, simulation, emulation and live network experiments.

Analytical modeling tries to describe the essential behavior of an entity (such as a network) with a mathematical expression that given some input parameters, provides some metric of interest. When a suitable expression has been derived, it can then easily be used to predict the performance of the entity under a range of conditions. However, in order to create a tractable formula the expression must often be simplified which introduces inaccuracies. This highlights the importance of model verification to ensure that the model does not deviate too much from the actual behavior. Nevertheless, modeling can be useful to get approximate behavior that can be mathematically analyzed.

The simulation approach also uses abstract representations of the entity under study, but in this case the abstractions are much more detailed and can include much more of the functionality. Also with simulations, there is a need to verify that the abstractions used in the simulation are correct and representative. The simulation approach is very flexible and can handle both small and large network topologies. Simulation allows a large parameter space to be explored and can provide considerable detail in the output.

In contrast to analytical modeling and simulation, the emulation approach uses a mixture of real entities and abstractions. By attaching real entities to an emulator, all aspects of the real entities are naturally captured. The emulator contains an abstraction of some underlying entity, and mimics the behavior of this emulated entity in real time. The emulation approach can provide insights into various envi-

ronmental factors for the real entities such as possible interaction effects with the operating system, device drivers and communications hardware. However, emulation is typically less scalable in topology size and parameter space than simulation.

The live network approach entails performing experiments on a running communications network. This naturally includes all aspects, both at the endpoints and at the network level. However, live experiments are hard to control and repeat. Getting access to live networks to the extent necessary may also be problematic in some instances. If the experiments study a behavior that occurs only for specific network conditions, this requires that these conditions can be introduced in the live network, which may not always be the case.

Considering network emulation, it has been used for quite some time in the networking community, and there exists a number of emulation packages. Dummynet [13] is a network emulator originally designed for testing networking protocols. It enforces queue and bandwidth limitations, delays, and packet losses. It is implemented in the FreeBSD kernel, and can be used on users workstations, or on dedicated PCs acting as routers or bridges. Since Dummynet is integrated with the firewall functionality, packets can easily be classified according to IP address, port number, type of service, and other fields. Matching packets can then be put into different pipes, where different pipes can have their own parameters for bandwidth limitations, delays and loss probability. A similar functionality is provided by NIST Net [3], which allows a Linux PC to emulate a wide variety of network conditions. These include tunable packet delay distributions, bandwidth limitations, packet reordering and packet duplication. Like in Dummynet, NIST Net induces non-congestion related losses using a random process. Neither emulator can introduce bit errors. Seawind [7] is designed to study data transfer for a single user, mainly in a cellular environment. Packet loss in Seawind can be based upon various random distributions, but may also be based upon a sequence in an external file.

Trace-driven approaches such as described in [9, 10, 11] measure real traffic and then use the information in the emulated scenario. In [9], probe packets are actively sent to another host, and the delay and loss are measured. In [10, 11], traffic is passively observed and distilled traces can then be used for example in the PaM [10] network emulator. PaM can then drop, corrupt or delay intercepted packets, using the distilled trace as a model representation of the time-varying characteristics of the network.

Other related tools include EMPOWER [17] which is a distributed emulation system capable of emulating large wireline and wireless networks, and it includes the EMWin [16] emulator that is specifically designed to handle mobility issues. NCTUns [15] is a hybrid emulator and simulator capable of emulating a wide range of network devices and protocols. Ns-2 [2] is a popular simulator that has been extended with limited emulation capability [5]. However it primarily remains a discrete event simulation tool.

A differentiating factor between previous trace-based emulators and KauNet is the level of detail that the traces represent. While most other trace-driven emulators only use traces to calculate parameters for stochastic generation of delays and losses, KauNet traces can be used to control the behavior of each individual packet with regards to bit-errors,

packet loss, delay, and effective bandwidth. The ability to precisely control the conditions for each individual packet, efficiently and for large numbers of packets, is the major novelty of KauNet.

3. KAUNET DESIGN

KauNet is composed of two major parts. Extensions to the dummynet code inside the FreeBSD kernel, and tools to create and manage pattern files.

3.1 Overview

KauNet extends the Dummynet functionality with the ability to precisely position losses, introduce bit-errors, control delays and bandwidth changes. Using patterns, this new functionality can be precisely controlled on a per packet basis (data-driven mode), or a per millisecond basis (time-driven mode). Compressed patterns are created ahead-of-time and then inserted into kernel space under the control of KauNet. During emulation these patterns are then “played” to control the emulated behavior. During emulation setup the `ipfw` command can be used to create multiple rules that specify how different flows should be sent to different pipes, which in turn have different patterns. This allows the emulation of multiple hops using many different patterns.

3.2 Usage

This section provides a glimpse of how KauNet is controlled and managed by the user. In order to control the kernel part of KauNet, the `ipfw pipe config` command has been extended with the following commands that applies to a specified pipe:

```
pattern <filename>
```

This specifies the pattern file that should be used for setting up a pattern.

```
pindex <pipe number> ber|pkt|del|bw
```

This specifies that a pipe should use the same pattern file as used by another pipe for the specified pattern type. Note that if the specified pipe number itself is redirected, the pattern redirected to is used, i.e. redirections are transitive. Each pipe has its own independent pointer into its current position in the shared pattern.

```
appendpattern <filename>
```

This specifies a pattern that should be used once the currently running pattern is exhausted. When the end of the current pattern is reached, the default behavior is to wrap-around and start from the beginning of the same pattern. The `appendpattern` command allows a new pattern to be specified that will be seamlessly switched over to when the end of the current pattern is reached.

```
[timedr|datadr]
```

These optional keywords explicitly specify which mode to use for advancing the index of the pattern file preceding the keyword in the command line. Normally the mode specified in the pattern header is used, and this parameter is left out. In some instances it may be desirable to override the mode specified in the pattern file, which is possible with these keywords. The time-driven mode advances the index once per millisecond regardless of whether or not any data is transferred. For time-driven bit-errors the amount of movement by the index is coupled to the bandwidth restriction used

in the same pipe. This restriction indicates how many bits to process per millisecond. Other kinds of time-driven patterns do not require a bandwidth restriction to be set since they work only on a millisecond level and individual bits do not need to be accounted for. For data-driven patterns the index move forward one step for each packet, except for bit-error patterns where the index is increased according to the number of bits in the packet.

```
[nosync]
```

This optional keyword specifies that forwarding of the index coupled to the time-driven pattern file should start immediately, and not wait for the first packet to be transferred. The default behavior is to wait until the first packet is received, and then start the clock that controls the forwarding of time-driven patterns.

3.3 Pattern Generation Tools

In addition to the kernel pattern management that is handled by `ipfw` as described above, a command line tool has been developed to create and manage patterns. The `patt_gen` tool can generate patterns according to several parametrized distributions. It is also capable of importing uncompressed pattern descriptions from simple text files. These text-files can be written manually, or generated by arbitrarily complex models, off-line simulators or trace collection equipment. A detailed description of the `patt_gen` tool can not be provided due to space restrictions. To com-

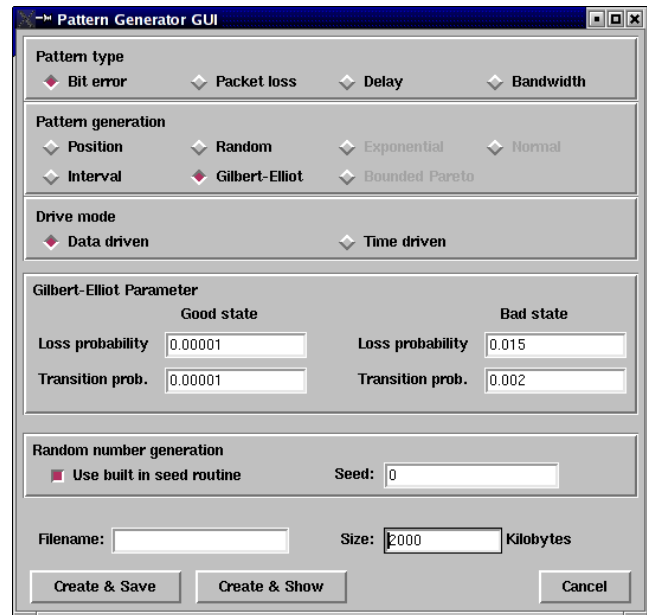


Figure 1: The `pg_gui` pattern creation GUI

plement the `patt_gen` tool a GUI called `pg_gui` was developed. The appearance of `pg_gui` is shown in Fig 1, where the panels for bit-error pattern generation are shown. From Fig 1 it can be seen that there are several possible generation functions, but only some of them are active for bit-errors. Using the GUI it is also possible to visually view the generated patterns and interactively explore various pattern generation parameters. Fig 2 shows an example of the pattern generated by the settings shown in Fig 1. The figure shows

the distribution of bit-errors for the 2 Mbyte pattern, and the vertical error streaks visualizes the bit-error aggregation caused by the Gilbert-Elliott parameters. A considerable part of the power and flexibility of the pattern approach, however, lies in its ability to import patterns from other sources via the command line tool as discussed in section 4.

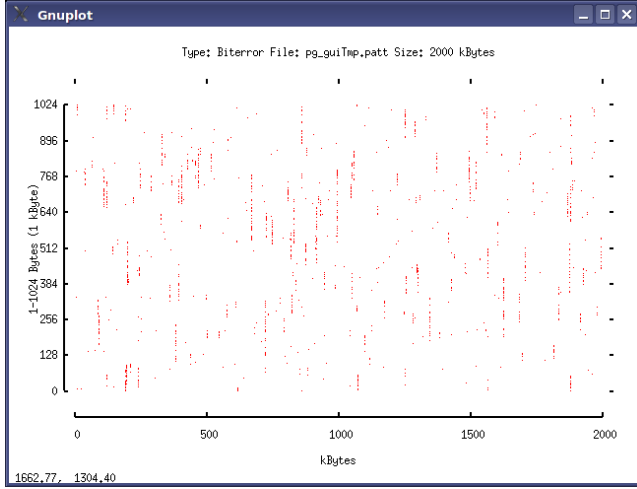


Figure 2: A visualized Gilbert-Elliott bit-error pattern

3.4 Pattern and Scenario Files

The pattern files are stored and imported into the kernel in a compressed format. This compressed format has several components including:

- Pattern type. This byte indicate if the pattern was produced to be used in time-driven or data-driven mode and what pattern type it is.
- Pattern size. This unsigned integer specifies the length of the pattern. The unit of the size is either kilobytes, packets or milliseconds dependent on the type of pattern in the pattern file.
- Indicator array. This array of shorts contains indicator values. Each bit in the indicator array indicates if the short at the corresponding position in the data array contains a run-length value or a data value.
- Data array. This array contains shorts that are either a run-length number indicating the distance to skip (i.e run-length value) until the next short is evaluated, or a data value whose semantics are dependent on the pattern type.

The `patt_gen` utility natively generates this compressed format, but it can also import uncompressed numerical lists or binary files and generate compressed pattern files from these.

One pattern is necessary for each of the controllable aspects, i.e. bit-errors, packet losses, bandwidth changes and delay changes. Since many experiments require simultaneous control of several aspects, multiple patterns need to be managed. Consider for example the case of a handover, where the worsening link conditions as a node moves away

from an access point might increase both delays and losses, and possibly also induce bit-errors depending on the specific emulated technology. To simplify the management of pattern files and to allow simple packaging of several pattern files, a scenario file format has been defined. A scenario file is a concatenation of several pattern files with an additional header. The scenario file header includes a scenario ID (SID) and a free text field that contains a textual description of the scenario. Scenario files are also created using the `patt_gen` utility, and the user specifies the pattern files, the SID and the free text. The SIDs are of special interest as the program is designed to only accept SIDs which include a correct checksum digit. The idea is that the SIDs should be used by users who wish to make their scenario files publicly available in order to share scenarios or simplify replication of their experiments. We intend to distribute series of globally unique SIDs to any interested researcher, and set up a repository of scenario files and related scenario meta-information where users can share their scenario files.

3.5 Emulation Performance and Scalability

Due to very efficient pattern handling inside the kernel, the performance impact of the KauNet extensions are minimal. Initial experiments performed on a standard desktop computer (2.4 GHz Dell OptiPlex, 512 Mb RAM) acting as a gigabit Ethernet router shows that the maximum achievable throughput goes down from approximately 368 Mbps (circa 31 kpps¹) when Dummynet/KauNet is not used, to around 352 Mbps (circa 29 kpps) with KauNet enabled and a bit-error pattern with a BER of 10^{-5} . This indicates that moving forward in the pattern and applying the bit-errors² require very little overhead.

With regards to the memory requirements needed to keep the patterns inside the kernel, it should be noted that the patterns are stored using a run-length compressed format, and that decompression occurs stepwise as the pattern information is consumed. As an example, take the storage requirements needed for the 2 Mb bit-error pattern shown in Fig 2. That specific pattern contains 1483 bit errors resulting in a BER of 9.1×10^{-5} . As an uncompressed bit-pattern this pattern would need 2 Mb, expressing the loss positions as a textual list requires 12285 bytes, and as a compressed pattern the space required is 5234 bytes. The actual storage requirement for a pattern is dependent on the type of pattern and the amount of bit-errors, losses, etc. in the pattern. We have successfully imported compressed patterns over 20000 times the size of the example compressed pattern described earlier (i.e. > 100Mb). This fact, in conjunction with the ability to seamlessly integrate a new pattern when the current pattern is ended, minimizes the risk for scalability problems due to the pattern data requirements of KauNet.

Considering that the FreeBSD kernel is not a real-time kernel, the exact timing of imposed packet delays cannot be guaranteed if the system is overloaded. In the experimental work performed so far with KauNet, this has not been a noticeable problem. However, most of this work has not been performed at heavy loads. Further examination of KauNet when emulating high bandwidths and at extreme loads are part of our current work.

¹kpps=kilo packets per second

²on average, there is one bit-error in every packet for this experiment

4. PATTERN CREATION FOR MOBILITY

As previously mentioned, the ability of KauNet to precisely introduce delays, losses and bandwidth restrictions based on pattern files greatly improves the reproducibility of an experiment. The main difficulty is to obtain patterns reflecting realistic conditions in a complex situation, e.g. multiple mobile nodes communicating via a fading channel. While writing such patterns from scratch is impossible, generating them using analytical models, simulations and/or previously collected traces is possible.

For example, an analytical model was used to create the patterns needed by KauNet to emulate a Land Mobile Satellite (LMS) channel in the context of satellite IP video broadcasting. Nine packet loss patterns corresponding to 3 different mobile speeds (1.5, 15 or 30 m/s) and 3 different environments (suburban, light or heavy tree shadow) were used [1]. These patterns were dynamically loaded into KauNet according to the position and speed of an interactively guided virtual mobile. This setup allowed to qualitatively demonstrate the quality of the decoded H264 stream (300 kbps).

A more general approach can also be used to generate patterns for mobile and wireless emulation. The remainder of this section presents how such an approach was developed by integrating KauNet in the emulation core of the W-NINE wireless emulation platform.

4.1 W-NINE Overview

W-NINE (Wireless NINE, NINE standing for NINE Is a Network Emulator [4, 12]), depicted in Figure 3, is an IP-level wireless network emulation platform developed by LAAS-CNRS and ENSICA. W-NINE aims at using accurate and time-consuming models with a small time granularity while respecting real-time constraints. For this purpose, W-NINE proposes the use of a simulation stage prior to emulation to reduce the number of events computed in real time.

The *simulation stage* uses SWINE (Simulator for Wireless Network Emulation), a dedicated wireless network simulator, which computes all of the mobility and propagation aspects of a wireless network based on a high level description of the experiment. In addition to mobility and propagation, it also computes the quality of service that will be encountered at the IP-level according to the radio communication technology used in the wireless network (e.g. 802.11b/g). This evolution of quality of service over time is synthesized in an emulation scenario that will then be read in real time during the emulation.

The *emulation stage* relies on NINE, a fully centralized emulation platform where experimentation nodes are connected via a router/emulator with a dedicated Ethernet network. An emulation manager periodically sends update commands to the emulator to refresh the QoS according to the simulated emulation scenario. All of these commands are sent on an administration network so that this traffic will not collide with the experimental traffic.

4.2 SWINE Architecture

To generate an emulation scenario SWINE uses a high level description file provided by the end user. In this file, the user specifies the number of nodes, the way they will move, the obstacles and the models that represent the propagation of the radio signal in the network. Finally, the end user specifies the technology model and the communication-

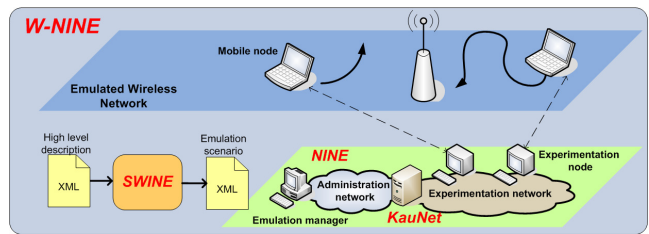


Figure 3: The W-NINE platform

level behavior such as hidden terminals or handover mechanisms that have to be considered for the network to emulate.

To compute the resulting emulation scenario, SWINE uses three steps: a mobility step, a propagation step and a communication step. At the end of each step, traces are generated and can then be reused in any other simulation.

The *mobility step* computes all the node positions at every time step of the experiment. At the current time, SWINE is able to deal with classical mobility models such as the random waypoint model or group mobility models (e.g. pursue mobility model). Other mechanisms such as obstacle avoidance have been implemented to deal with the environment. In addition, a predefined path mobility model has been implemented allowing every kind of mobility traces to be fed into SWINE (e.g. traces produced by a dedicated mobility simulator such as VanetMobiSim [8], an extension of CanuMobisim [14] for vehicular mobility, or even the *setdest* mobility tool used in Ns-2 [2]).

Based on the positions of the nodes, the *propagation step* computes the radio signal strength for every pair of nodes of the network at every time step of the experiment. Several models have been considered to compute such a radio signal. The large scale propagation model, such as the pathloss exponent model, mainly deals with the impact of the distance on a radio communication. To manage the absorption of the signal caused by the obstacles, some shadowing models such as the lognormal shadowing model have been considered. Finally, fading models (e.g. Rayleigh) can be used to deal with multi-path radio signal propagation effects.

During the *communication step* SWINE computes the IP-level QoS conditions that will be encountered by terminals in the emulated wireless network. Based on the radio level computed during the propagation step and according to vendor specifications, it is possible to determine the appropriate transmission rate at the physical level. With this transmission rate, it is then possible to compute the maximum available throughput at the IP-level, taking into account MAC parameters such as backoff times and inter-frame spacings for 802.11b/g networks. In addition to the technology, the communication step also simulates some traffic dependent phenomena such as the effect of hidden terminals that can occur in ad-hoc networks.

More details about the models implemented in SWINE can be found in [12]. Due to its open architecture, SWINE also allows the introduction of new models without any recompilation of the simulator's core. For example, it is possible to add a ray tracing model to the propagation step to more accurately compute the radio signal power that is received by a mobile node. Such a model is very time-consuming, but can safely be used during the simulation stage which is not submitted to real-time constraints.

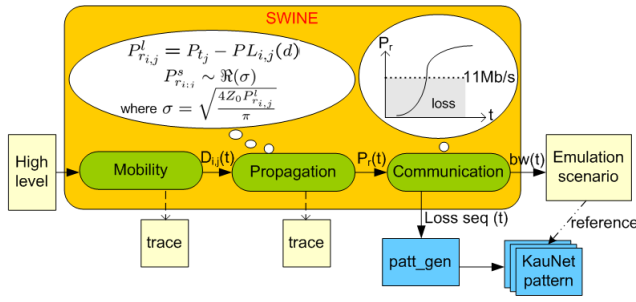


Figure 4: Pattern generation process with SWINE using a Rayleigh fading model

4.3 Generating Patterns with SWINE

W-NINE and KauNet have been integrated to provide a realistic and reproducible emulation platform. For example, the ability of KauNet to precisely introduce losses in a communication has shown its usefulness to reproduce conditions that can be encountered during a multicast communication or when the 802.11 auto rate fallback (ARF) mechanism is deactivated [4]. For this purpose, SWINE now generates loss patterns during the communication step of the simulation stage according to the computed radio signal strengths. Figure 4 presents such a computation of the radio signal using a pathloss exponent model to deal with the large scale effects ($P_{r_{i,j}}^l$) and using a rayleigh fading model to compute the small scale effects ($P_{r_{i,j}}^s$). In this example, the sequence of losses is computed based on a communication table where a single transmission rate is specified assuming the ARF mechanism is deactivated (i.e. if the received radio signal is below the reception threshold for the current transmission rate, a loss is introduced in the communication).

To prevent any loss of precision between the simulation phase and the emulation phase, losses are computed every millisecond, which corresponds to a good KauNet granularity. This loss sequence is then sent by SWINE to the `patt_gen` tool to generate the corresponding loss pattern file.

More complex scenarios can also be modeled. The 802.11 ARF algorithm introduces changes in the transmission rate when a certain amount of packets is lost or received in sequence. At the IP level, this leads to both a throughput change and a delay change, which can be modeled by KauNet scenarios and their ability to link a throughput variation with an introduction of delay. Currently, SWINE determines the moments of change of the physical transmission rate. These moments can be used to produce the adequate changes in a KauNet scenario.

5. CONCLUSIONS AND FUTURE WORK

The KauNet trace-based emulation system has been developed to allow fine-grained and repeatable control of bit-errors, packet losses, bandwidth and delay changes. The use of scenario files and a central scenario repository simplifies the reproduction of experiments and encourages sharing. The integration of KauNet and W-NINE was presented as one example of integration and how complex mobility and propagation models can be used to create patterns used by KauNet. Current work includes further refinement of KauNet as well as more radical extensions. These extensions will allow run-time loading of finite-state-machines and con-

strained programmability. The aim is to further increase the dynamicity of the emulation system while retaining the good performance of the pattern-based foundation.

6. ACKNOWLEDGMENTS

The authors wish to acknowledge the funding by the EU NEWCOM network of excellence and thank all who have contributed to the KauNet and W-NINE effort.

7. REFERENCES

- [1] A. Bouabdallah, M. Kieffer, J. Lacan, G. Sabeva, G. Scot, C. Bazile, and P. Duhamel. Evaluation of cross-layer reliability mechanisms for satellite digital multimedia broadcast. *IEEE Transactions on Broadcasting*, 53(1):391–404, 2007.
- [2] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in Network Simulation. *IEEE Computer*, 33(5):59–67, May 2000.
- [3] M. Carson and D. Santay. NIST Net: A linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.
- [4] E. Conchon, J. Garcia, T. Pérennou, and M. Diaz. Improved IP-level Emulation for Mobile and Wireless Systems. In *Proceedings of the IEEE Wireless Communications & Networking Conference (IEEE WCNC)*, March 2007.
- [5] K. Fall. Network emulation in the VINT/NS simulator. *Proceedings of the fourth IEEE Symposium on Computers and Communications*, July 1999.
- [6] J. Garcia, P. Hurtig, and A. Brunstrom. The effect of packet loss on the response times of web services. In *Proceedings 3rd International Conference on Web Information Systems and Technologies (WebIST2007)*, Barcelona, Spain, March 2007.
- [7] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen. Seawind: A wireless network emulator. In *Proceedings of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, September 2001.
- [8] M. Fiore and J. Haerri and F. Filali and C. Bonnet. Vehicular Mobility Simulation for VANETs. In *Proceedings of the 40th IEEE Annual Simulation Symposium (ANSS)*, March 2007.
- [9] B. Melander and M. Björkman. Trace-driven network path emulation. Technical report 2002-037, Department of Information Technology, Uppsala University, Sweden, 2002.
- [10] B. Noble, G. Nguyen, M. Satyanarayanan, and R. Katz. RFC 2041: Mobile network tracing, October 1996.
- [11] B. Noble, M. Satyanarayanan, G. Nguyen, and R. Katz. Trace-based mobile network emulation. In *Proceedings of ACM SIGCOMM '97*, September 1997.
- [12] T. Pérennou, E. Conchon, L. Dairaine, and M. Diaz. Two-stage Wireless Network Emulation. In *Proceedings of the 2004 Workshop on Challenges of Mobility (WCM 2004)*, August 2004.
- [13] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, January 1997.
- [14] I. Stepanov, J. Hähner, C. Becker, J. Tian, and K. Rothermel. A Meta-Model and Framework for User Mobility in Mobile Networks. In *Proceedings of the 11th International Conference on Networking 2003 (ICON 2003)*, September 2003.
- [15] S. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, C. Chiou, and C. Lin. The Design and Implementation of the NCTUns 1.0 Network Simulator. *Computer Networks*, 42(2):175–197, 2003.
- [16] P. Zheng and L. Ni. EMWin: Emulating a Mobile Wireless Network using a Wired Network. In *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, 2002.
- [17] P. Zheng and L. Ni. Empower: A network emulator for wireline and wireless networks. In *Proceedings of IEEE InfoCom*. IEEE Computer and Communications Societies, March 2003.