EUROPEAN COOPERATION
IN THE FIELD OF SCIENTIFIC
AND TECHNICAL RESEARCH

EURO-COST

COST IC1004
TD(11)01038
Lund, Sweden
June 20-21, 2011

SOURCE: Institute of Communication Networks
and Computer Engineering
University of Stuttgart

Approaches for evaluating the application performance of future mobile networks

Thomas Werthmann, Matthias Kaschub, Christian Blankenhorn,
and Christian M. Mueller
Pfaffenwaldring 47
D 70569 Stuttgart
Germany
Phone: +49 711 685 69015
Fax: +49 711 685 67983
Email: thomas.werthmann@ikr.uni-stuttgart.de

Approaches for evaluating the application performance of future mobile networks

Thomas Werthmann, Matthias Kaschub,
Christian Blankenhorn, and Christian M. Mueller

June 12, 2011

Abstract

In order to achieve high data rates, future mobile networks employ complex algorithms in the physical and MAC layers. Previous studies have shown relevant interactions between these algorithms and transport and application layer mechanisms. An efficient design of lower layers requires consideration of higher layers, given that the service quality experienced by the user is the ultimately relevant performance metric.

In this paper, we give a brief overview of different approaches that allow for the joint evaluation of PHY, MAC, transport, and application layer. We then focus on the integration of virtual machines into our system level simulation tool. This approach enables us to evaluate the application performance even if the entire setup was too slow to run in real-time. With this approach, evaluating the application performance can be done by both deriving objective performance metrics and playing back traces (e.g. video) later on.

1 Introduction

In the early stage of development of new PHY or MAC layer algorithms, we usually determine the system performance by mathematical analysis, often under largely simplifying assumptions. As more and more real world constraints are included, our models become more and more complex and mathematical analysis quickly becomes infeasible. We usually resort to simulations, where we start with an abstract model that is close to the mathematical analysis and subsequently refine this model to add more and more detail. Once our new PHY or MAC layer algorithms become more mature, we are interested in their real-world performance. As wireless network engineers, our simulators already use detailed channel, PHY and MAC layer models, but we generally face two problems: How to model the behavior of network, transport and application layer and the strong interaction between them? How to measure application performance?

While the data plane of the network layer in an IP network is quite simple and easy to model, the processes on the transport layer are very complex. The Transport Control Protocol (TCP), continuously adapts to current network conditions and reacts upon packet loss and changes in bottleneck link capacity or transmission times. This implicit interaction with lower layers can lead to undesired effects, such as bad link utilization or excessive transmission delays, as it was observed in GPRS networks [2]. The simulation of TCP requires TCP models that quickly become complex themselves and bear a high risk for implementation errors. Another drawback is that the gap between the behavior of a TCP model and the behavior of the real TCP implementation is difficult to determine. A solution to these problems is to extract the code of the TCP stack of open source operating system kernels and to integrate this code in the simulation. This approach was chosen in the “Network Simulation Cradle” [5]. Since the extracted code is widely used production code, its behavior matches the reality well. The approach also covers all current TCP flavors.

To assess application performance, an analysis of network and transport layer metrics is not sufficient, given that the link between these metrics and the application layer performance is not obvious. Depending on the size and number of the application layer data units that are sent over the network, an application might be more constrained by the round-trip-time or by the capacity of the channel between source and destination. For the example of web surfing, a higher PHY layer bit rate not necessarily results in a faster display of a web page. If the higher rate comes with larger delays, the opposite could be the case.

The construction of application models usually requires to run extensive measurements to extract its statistical properties which are then transformed into behavioral models. Given that today’s applications exhibit very diverse communication patterns that might even change from one version of an application to another, this is a tedious task. As a consequence, the application models we use in simulations often do not keep up with application development. An example is the web model described [3], which is still widely used for performance evaluation in wireless networks [10]. The model is based on traffic measurements in the 1990s and only comprises HTTP version 1.0, which leads to a behavior that is very different from one of modern web browsers.

In this paper, we propose a way to include real implementations of TCP/IP stacks and applications in a wireless network simulation. In contrast to other emulation frameworks, we still run event-driven simulations and do not require a continuous time wall clock. This is achieved by using virtual machines that are controlled by the event calendar of the network simulation process. This allows for simulations that run slower or faster than real-time, depending on the computational complexity of our PHY layer model. In section 2, we give a brief overview of different methods to jointly evaluate PHY/MAC layer, transport layer, and applications. Section 3 describes our approach and section 4 discusses ways to determine application performance using our simulation framework.

2 Methods for cross-layer evaluation

In the following paragraphs, we present a selection of evaluation methods, that allow for the consideration of the PHY, MAC, and transport layer as well as the applications.

The most simple way of regarding both network and application effects is to connect real hosts via a *network emulator* [6, 7, 9]. Applications and the network stack run on the real hosts while the PHY/MAC layer is implemented in the network emulator. With this method, performance metrics like e.g. the link utilization or queue fill levels can be measured in the network emulator. Alternatively, performance metrics may be gained on the hosts itself by modifying or observing the applications. In order to evaluate large scenarios with many hosts, *host virtualization* can be used to instantiate more hosts with a limited number of real hosts [4]. In case that multiple virtual hosts share the same real network interface, attaching the network emulator may require using virtual connection like e.g. VLANs. Deriving performance values can be done in the same way as with non-virtualized hosts.

The PHY/MAC models of today’s and future wireless networks are too complex to be computed in real-time, which is necessary for network emulation. To allow the emulation of such networks, a virtual time for the host virtualization and network emulation can be used [8]. The virtual time can advance more slowly, giving the network emulator more time. In order to avoid extensive synchronization, the ratio of virtual time and real-time must be the same on all hosts.

Having a fixed ratio of virtual time and real-time is inefficient if the processing effort is unevenly

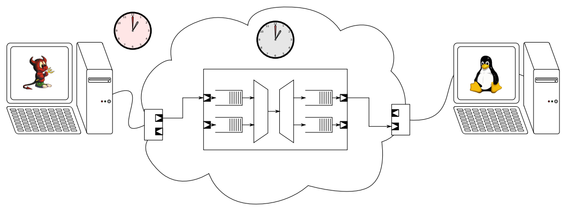

Figure 1: emulation in a lab setup

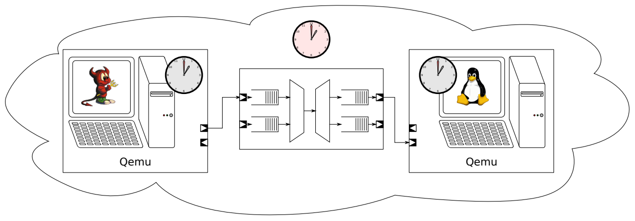

Figure 2: virtual nodes as entities of the simulation

distributed over time. Thus, our approach is to use event-driven simulation for both the network and the applications. This way, the time jumps from one event to the next one avoiding unnecessary waiting.

3 Our Implementation

In our approach, the network simulation controls virtualized computers (see Figure 2). On these virtualized computers an unmodified operating system and unmodified applications are running and creating network traffic. To model the interactions between the virtualized hosts and the network simulation, both need to share the same time source.

Since the network simulation performs all actions based on events, the simulated time follows from these events and is not related to the real-time. The virtualized computers need to adapt to this event-based time. Therefore the simulation has to control the time of these virtualized computers. In this section we explain our architecture together with alternatives and its properties and characteristics.

3.1 Design choices

There are a lot of programs available to execute a virtualized computer and its applications. We chose *QEMU*[1] based on the following criteria:

- Since none of the available virtualization solution directly supports being controlled by a simulation program, it needs to be open-source, so we can modify its source code accordingly.
- It should impose as few restrictions as possible on the host computer that runs the simulation. In particular, it should not require special hardware, special kernel features or super-user privileges. *QEMU* runs in user space on a wide variety of host CPUs and operation systems.

[1] `http://www.qemu.org/`

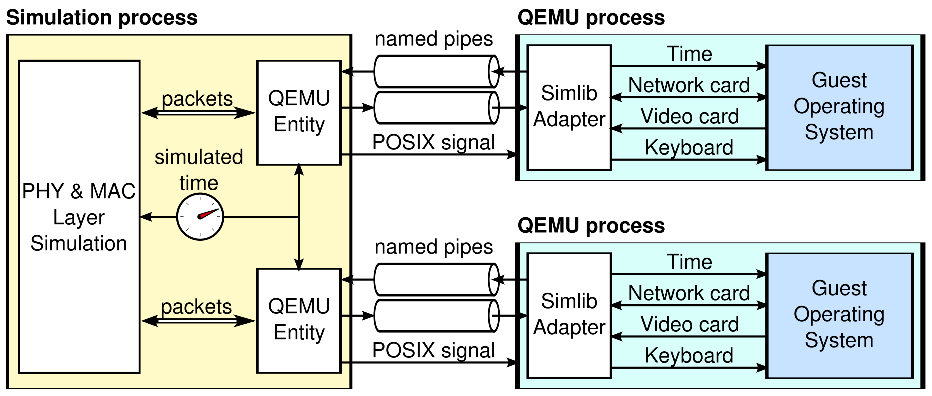

Figure 3: architecture of the communication between simulation process and virtualized computers

- We do not need exact per-clock-cycle emulation of the virtual CPU, as e.g. *bochs*[2] is doing it. This feature would come at the cost of high computational complexity.
- An optional — but in the context of mobile devices prominent — feature is to emulate CPUs that are common in mobile devices (e.g. *ARM*) while running the simulation on normal x86 hosts CPUs.

Further, QEMU is very closely related to *KVM*[3], which gives us the opportunity to use the same architecture with *KVM*. On hosts that support hardware virtualization, *KVM* will reduce the computation-overhead by an order of magnitude.

We modified version 0.13 of *QEMU* to communicate with our network simulation library *IKR-SimLib*[4].

3.2 Architecture

As explained above, the intention of our modifications is to make the clock of the virtual computer to adapt to the clock of the network simulation. Since the simulation does not run in real-time, the virtual computers do also not run in real-time. Therefore all communication of the virtual computers with the outside world must only happen to and from the network simulation program. Figure 3 shows the architecture of the simulation program including virtual computers.

Since *QEMU* emulates exactly one virtual computer per *QEMU*-process, we start multiple *QEMU* processes. Currently, the virtualized computers use a simple clock with a 1 kHz tick. The ticks of all virtualized computers are represented as discrete events in the calendar of our simulation program. Hereby, the simulation program ensures consistence with the time of other events, e.g. from the network simulation or from other instances of *QEMU*.

Communication between the processes uses two unidirectional pipes between the simulation program and each *QEMU* process. A simple binary TLV based protocol is used to exchange message over these named pipes. In addition, the simulation program can send a POSIX signal to the *QEMU* process to interrupt its current execution. In addition to the already mentioned tick-events, the simulation program and *QEMU* exchange messages containing network packets, keystrokes, serial console, and graphical screen shots.

[2]`http://bochs.sourceforge.net/`

[3]Kernel Virtual Machine, `http://www.linux-kvm.org/`

[4]`http://www.ikr.uni-stuttgart.de/IKRSimLib/`

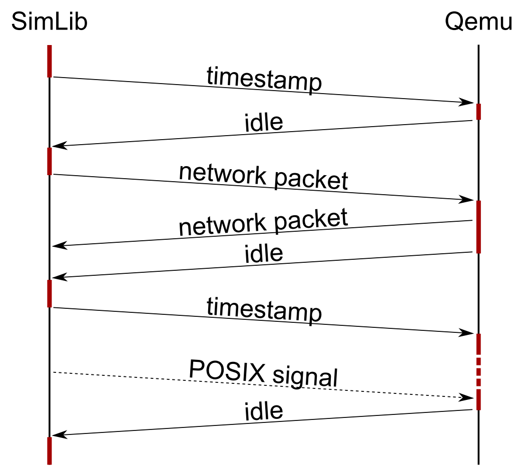

Figure 4: signaling between simulation and virtualization

We designed the interaction of the simulation program with the *QEMU* process to be as simple and as reproducible as possible. When the simulation program processes a tick-event in its calendar, it will perform the following steps:

- The simulation program sends the time of this tick-event to the *QEMU* process.
- The *QEMU* process updates its clock time and executes the virtual CPU until the virtual CPU becomes idle and all requests to the hard disk are completed.
- The *QEMU* process sends a message to the simulation program to confirm that it finished computing. All events inside *QEMU* are blocked from that time on, to make sure that it will not resume computation before the next tick arrives.
- In case the virtual CPU does not become idle within a specified period in time (e.g. because it is running a busy loop waiting for the time to advance), the simulation sends a POSIX signal to interrupt the execution and return the control flow to the simulation program.

Figure 4 shows an exemplary course of events for this signalization. Thus, all computation of a *QEMU* process is caused by an event in the simulation program. While this event is carried out, the simulation time is not updated. Only one *QEMU* process is allowed to compute at any time.

Since the virtual CPU generally executes until it becomes idle, and the time is not advancing while it is executing, this models unlimited processing power. However actions that perform a busy loop — such as *Linux* measuring the *bogo-mips* — are interrupted by a POSIX signal, where the exact time the signal arrives depends on the scheduler of the host system. Therefore, multiple runs of the same simulation behave slightly different, which prohibits bit precise reproducibility.

3.3 Scalability

The number of virtual computers in one simulation is limited by memory and processing power.

With *QEMU* version 0.13 the minimum amount of memory for a virtual computer is 8 MB. This is sufficient for a small kernel and a simple web server. However, to execute a graphical desktop environment with a recent web browser, at least 512 MB are required. The *QEMU* process itself requires about 5 MB to 10 MB resident memory. When simulating a high number of virtual computers, most of the pages will be equal in all these computers. Although we have not investigated this yet, we suppose that Linux *KSM* (Kernel Samepage Merging [1]) can drastically reduce the memory requirements.

Due to the strictly sequential procedure explained above, the whole simulation including all *QEMU* processes is occupying only one CPU of the host at a time. This limits the processing speed of the simulation. When the virtual computers are under high load (e.g. booting their operating system or loading an application), the costly CPU emulation performed by *QEMU* becomes the limiting factor. While this is not the case when the virtual computers are idle, the ticks have to be signalled to all virtualized computers at a 1 ms interval. In our experiments, a simulation with 10 virtual computers pinging each other executes about as fast as real-time on a host with an Intel Core-2 CPU. However, in the simulation of wireless networks the processing speed becomes less important, since the computation of wireless channel models and complex transmission schemes is typically more expensive than the virtualization.

4 Usage models and performance evaluation

We presented a framework which allows incorporating virtual computers into our simulation library. Inside these virtual computers, we can run standard operating systems and use standard applications. However, we can not use these virtual computers interactively, because they do not execute in real-time. Therefore, we have to model the interactions of the human users and we have to measure the application performance.

To emulate the users' interactions with the virtual computers, we implemented an interface to the virtual computers keyboard. We can schedule events in our simulation libraries calendar which send keystrokes to the virtual computers. Hereby we can, for example, load different web pages in predefined intervals of time.

To evaluate the network performance, we can use either subjective or objective measures. As a subjective metric, we record screen casts from the virtual computers. These can be played back in real-time and allow the observer to judge the networks performance. While this can be used for demonstration, it is not feasible to analyze videos to deduce an objective measure (e.g. web page loading times). Instead, it is easier to enhance the applications in the virtual computer to report measures of the performance. Although this requires modifications of the applications, which we tried to avoid, these modifications can be small and should not influence the applications behavior. For some applications, there are already extensions available which support performance evaluation.[5] The output of these tools can be feed back to the simulation program via the serial console. There, it can be aggregated in statistics and be used for objective evaluations.

5 Conclusion

For a sound evaluation of mobile communication networks, behavior of transport protocols and applications has to be taken into account. We gave an overview of different approaches to achieve this. We then explained our approach, which allows us to integrate virtualized computers into our simulation program. Although this cannot replace the other approaches, it complements them well. The particular advantage is that new applications and protocols can be integrated into simulations without exhaustive traffic measurement and modeling. Compared to approaches like the "Network Simulation Cradle", our implementation can be easily updated to new versions of the operating systems kernel.

[5]See for example Watir (`http://watir.com/`), which allows to remote control most common web browsers.

We have implemented the described architecture and thereby proved its feasibility. The major drawback is the resource utilization of our approach. To mitigate the synchronization overhead of an idle system, we plan to extend our implementation to support *HPET* (High Precision Event Timers). This would allow to use a tickless linux kernel in the virtual computers, which does not need ticks at a regular interval. In addition, we plan to make use of the inherent parallelization of our architecture. For that purpose, we are going to model a short link delay between the virtual computers and the remaining system, which we can then utilize to decouple the processes.

References

[1] Andrea Arcangeli, Izik Eidus, and Chris Wright. Increasing memory density by using ksm. In *Proceedings of the Linux Symposium*, Montreal, Quebec, Canada, July 2009. Red Hat.

[2] R. Chakravorty, S. Katti, I. Pratt, and J. Crowcroft. Using tcp flow-aggregation to enhance data experience of cellular wireless users. *Selected Areas in Communications, IEEE Journal on DOI - 10.1109/JSAC.2005.845628*, 23(6):1190–1204, 2005.

[3] Hyoung-Kee Choi and John O. Limb. A behavioral model of web traffic. In *Proceedings of the Seventh Annual International Conference on Network Protocols*, ICNP '99, pages 327–, Washington, DC, USA, 1999. IEEE Computer Society.

[4] Andreas Grau, Steffen Maier, Klaus Herrmann, and Kurt Rothermel. Time jails: A hybrid approach to scalable network emulation. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, PADS '08, pages 7–14, Washington, DC, USA, 2008. IEEE Computer Society.

[5] S. Jansen and A. McGregor. Simulation with real world network stacks. In *Simulation Conference, 2005 Proceedings of the Winter*, page 10 pp., dec. 2005.

[6] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns-2. In *Distributed Simulation and Real-Time Applications, 2004. DS-RT 2004. Eighth IEEE International Symposium on*, pages 29 – 36, oct. 2004.

[7] D. Mahrenholz and S. Ivanov. Adjusting the ns-2 emulation mode to a live network. In *Kommunikation in Verteilten Systemen (KiVS)*, pages 205–217. Springer, 2005.

[8] Steffen Dirk Maier. *Scalable computer network emulation using node virtualization and resource monitoring*. PhD thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2011.

[9] M.C. Necker, C.M. Gauger, S. Kiesel, and U. Reiser. IKREmuLib: A Library for Seamless Integration of Simulation and Emulation. In *Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2006)*, March 2006.

[10] NGMN Alliance. Radio access performance evaluation methodology. available online at http://www.ngmn.org/, June 2007.