Cost Effective Cross-layer Protocol Testing: A Case Study

Marcelo Odin Guirado*, José Pablo Escobedo†, Ana Cavalli†, Stéphane Maag† and Ariel Sabiguero Yawelak*
*Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Montevideo, Uruguay
e-mail: {modin|asabigue}@fing.edu.uy
† Département Logiciels-Réseaux
TELECOM SudParis (ex INT), Évry, France
email: {jose.escobedo|ana.cavalli|stephane.maag}@it-sudparis.eu

***Abstract*—Model-based testing has been successfully applied to conformance testing of reactive systems such as network protocols, both on general purpose and embedded systems. However, doubts persist about the overhead on time and human resources required. This work addresses the empirical quantification of model-based validation costs for medium and small-sized projects. In particular, it asseses the cost of model-based testing for cross-layer protocols. This paper presents how, in spite of some initial constraints in budget, human resources, time frame and available tools, it was possible to produce a formal specification or model, implement a fully functional prototype and test it for conformance with test cases derived from the model.**

I. Introduction

Mobile devices support for wireless technologies and TCP/IP connectivity allows the access to services through a variety of networking infrastructures. However, TCP/IP was developed for wired networks and devices with sufficient resources (bandwidth, energy, etc) such as servers or desktop computers. A popular approach to improve the performance of TCP/IP over wireless mediums are cross-layer protocol optimizations.

As a part of the SCAN [1] project a system was required to add cross-layer protocol optimizations to devices with constrained resources, such as embedded devices, while requiring minimum or no modification to the network protocol stack. Due to the intricacy of network protocols and the particular challenges of cross-layer design (high coupling, interaction among optimizations, etc), cost effective validation techniques and tools were required.

Model-based testing [2], [3] has been successfully applied to conformance testing of reactive systems such as network protocols. However, time and human resources required may turn this approach inefficient. In this paper is described how model-based testing was applied to cross-layer protocol optimization in order to determine if it is a suitable and cost effective methodology for the problem domain.

A prototype with a simple optimization protocol [4] was implemented. To address modularity and non intrusiveness the ECLAIR [5] architecture was followed. Simulations and testing techniques were conducted to validate the system. Its specification was modeled in SDL [6] and test cases were semi-automatically derived from the model. An *ad-hoc* testing automation tool was built to run the test suite against the prototype.

A full iteration -from concept phase to execution and validation- was accomplished within a tight schedule and with scarce resources. The time devoted to this iteration comprised the time to study tools and modeling language, to model the specification of the system, to build an implementation and to test it.

The rest of this paper is organized as follows: in section II the methodology is presented. Cross-layer design and architectures were studied, as well as suitable means for validation such as verification and testing. In part III the cross-layer optimization protocol and its implementation are presented and discussed. In part IV the model of the system specification is briefly described, an explanation on how it was used to validate and test the prototype is given, and the results in terms of cost are summarized. In part V conclusions are presented.

II. Methodology

Layered architecture, such the ISO OSI [7] model, divides networking software in layers which are collections of protocols with a specific function (such as binary transmission, physical addressing, logical addressing, and so on). Each layer provides services to the layer above and consumes services from the layer below through well defined interfaces. The layered architecture promotes the isolation among layers, and this modularity makes possible to replace implementations at each layer while maintaining the interfaces. TCP/IP protocol stack follows a similar layered architecture.

However, for embedded devices and wireless communications resources such as bandwidth, energy, etc. become scarce. In order to optimize the utilization of these resources, higher layers may benefit from information from lower layers and vice versa.

Methodological grounds for this work are sketched in the rest of this section. Cross-layer design and architectures were

The work presented in this paper is the result of a French-Uruguayan cooperation, which was partially funded by the SCAN Project.

studied, as well as suitable means for validation like verification and testing.

A. Cross-layer design and architecture

Cross-layer design consists in violating layer isolation in order to offer feedback between a layer with information and another layer that can use this information to operate more efficiently. An example of cross-layer optimization is flow control among MAC and transport layers [8].

Although cross-layer design have proven a popular approach to improve performance on wireless networks, there is a trade off between modular architecture and optimizations. There are various cross-layer architectures proposed in the literature that could help alleviate these problems, such as PMI [9], ICMP based [10], CLASS [11], CrossTalk [12], MobileMan [13], ECLAIR [5]. Each architecture proposes a different approach for signaling among layers, such as adding extra information at PDU, direct interaction among layers through function calls, a common network parameters repository, etc.

The ECLAIR [5] architecture was chosen because it allowed non intrusive introduction of cross-layer optimizations (namely, it does not require modifications on the network protocol stack), minimum stack overhead, extensibility, reversibility, portability and efficiency. Furthermore, details were publicly available. ECLAIR separates the protocol implementation, residing in an optimization subsystem, from the interaction with the protocol stack, residing in a tuning subsystem. This allows any changes taking effect in the protocol stack to be isolated from the actual protocol.

Inter layer feedback is achieved through notifications of changes in protocol stack data structures. The optimization of the protocol stack behavior is achieved through changes of the network parameters through an API presented by the tuning subsystem, plus the algorithms in the optimization modules.

There are some limitations to the scope of this architecture. Only asynchronous event detection is addressed through either notifier chains or by polling the desired data structure with a pre-established frequency. Explicit synchronization of the network protocol stack and the optimizations protocols behavior is not addressed, which can lead to loops. Integrity of the data structures when concurrently accessed by the protocol stack and the optimization protocol is not addressed.

B. Validation and Model-Based Testing

As the terminology on validation, verification and testing is not always consistent in the literature, the meaning of these concepts in the context of this paper are presented here. To validate a system is to provide information that increases the confidence in the hypothesis that the system was built correctly. The most popular approaches for validation are verification and testing. Verification is to interact with the model of a system (using the model for simulations or mathematically prove properties of the model), and testing is to perform experiments on an implementation of the system trying to find defects.

It has been stated [14] that 50% of the defects are introduced in the coding phase of a software developing process, and only 15% are detected in design phase. Most defects are found during testing. The cost of correcting a defect is higher the later it is found. A consequence of this is that while it is important to use techniques to detect defects as early as possible (such as model checking or any verification techniques), it is on the testing phase of the actual implementations that most defects are found so both approaches are convenient and complementary.

For this reason model-based testing paradigm was chosen. The goal of model-based testing is to reduce costs while developing quality software by automating the testing process as much as possible, in particular automating the test case generation. In order to do so, a formal model of the system specification is built and used to derive the test cases. This formal model is developed in parallel to the implementation of the system, thus the model could be verified and test cases generated while the system is still being implemented.

The ioco [15] (input output conformance testing) theory is behind well known automated test case generation tools such as TGV [16], [17] and TorX [18]. The theory states that it is possible to establish the existence of a mathematical relation (namely, ioco) between a formal model (the model of the specification) and an implicit model corresponding to an actual implementation by probing the implementation. In practice, to prove conformance that way is not possible because any non trivial system would require a test suite too huge to make testing impractical. A compromise is achieved by restricting test case generation to certain parts/behaviors of the model specified as test purposes.

The formal model is built in any of multiple formal specification languages such as SDL, LOTOS, IF, IOLTS, etc. In general the model used for test case generation differs from the model used for designing the system in that the first is used to capture the behavior of the system while the later is used to capture the structure of the system. It also differs in the abstraction level, since the model used for test generation can ommit many details. Therefore, the model and the system can be built independently and simultaneously, and test cases can be obtained even before the system had been finished, shortening the time required for testing.

Maturity in modeling for design is not required, actually the implementation could be developed from an informal specification of the requirements. However, it makes more sense to use model-based testing when the software development process has already automated the test execution.

SDL [6] was chosen as the modeling language for the model of the system specification. SDL is known for its success in the telecommunications and protocol design domain [19] [20], as well as for the availability of tools to build models in that language.

III. Prototype

In this section the cross-layer optimization protocol and its implementation are presented and discussed.

A. Use Case

The optimization protocol that was chosen to be implemented [4] was presented along with the ECLAIR architecture proposal. Although a detailed description is beyond the scope of this work, a brief explanation follows.

The selected use case consists in setting priorities for TCP connections that in turn determine the size of the receiver announced window in each connection, thus accelerating or slowing the sender, redistributing the throughput according to the priorities. It is based on the following relation: $Throughput = \frac{RCV\_Window}{RTT}$. The receiver window is a parameter included in the TCP header which is adjusted depending the use of the receptor buffer to prevent the loss of packets.

While in theory this approach may seem valid, in practice there are some flaws. The size of the receiver announced windows represents the memory available for that connection, but memory is not taken into account when setting the size of the new announced window.

When the size of the received segments is too small (for instance, interactive sessions) it is not a good idea to rise the size of the announced window because it would be filling the buffers mainly with control information.

If the size of the receiver window is raised but the application cannot consume the received packets, the buffers will be filled for that connection and further packets will be discarded. This may even be seen from the sender as a congestion, furthering slowing the transmission rate (congestion control).

The model $Throughput = \frac{RCV\_Window}{RTT}$ does not contemplate the probability of packets loss, which is not negligible in wireless networks, in particular for video transmissions or real time traffic.

It could be argued that cross-layer feedback is not present, since the priority of the connections are set by a user. However, in the architecture the user is modeled as part of a *user layer* and its feedback is valuable as context information. While in this case priorities are set by a person, they could also be set automatically according to access policies for each user and/or application. The user layer and the interaction with the transport layer simplifies the complexity of proper synchronization and monitoring of the network protocol data structures.

B. Implementation

An implementation of this use case is reported to exist for Linux 2.4.x, however it was not available and was not used in this work. Furthermore, there are many differences from GNU/Linux Kernels 2.4.x to 2.6.30 which was the latest Kernel available at the time of the implementation.

The system was implemented in two parts. On the one hand, there is a program running in user space (a command line tool) to set the priorities for the connections. On the other hand, there is a Kernel module (implemented as a device driver), which can directly access and modify the Kernel data structure. It allows *real time*, user driven recalculation of the window size and modification the rcv_ssthresh and window_clamp parameters for the connections. Note that per flow optimization did not allowed the use of /proc/net without modifying the Kernel, and such modifications would affect the overall performance of the system.

The prototype was implemented on `archlinux` over VirtualBox, with GNU/Linux Kernel 2.6.30, and was compiled with GCC 4.4.5. `Cscope` was used to analyze the Kernel code. The implementation is generic enough so that it can be reused and expanded with other optimizations, and can easily be ported to embedded systems.

In the network protocol stack of the GNU/Linux Kernel there is a hash table accessible through the inet_lookup function. inet_lookup receives local and remote IP and port and returns a pointer to a sock, which is used to access the aforementioned parameters. This data structure and the functions in Kernel 2.4 (the Kernel version for which the use case was originally implemented) differs from the ones in GNU/Linux Kernel 2.6 (the Kernel version for which we developed our prototype). In Figure 1 is shown how the inet_lookup is used. While future releases of the operating system might result in modifications to the tunning layer, the optimization subsystem and the optimization protocol itself will remain unchanged. This is a strong point of the ECLAIR architecture.

```
static int set_receiver_window(u32 size,
                struct rwc_info_struct *conn_id)
{
        struct sock *sk;
        struct net *red;
        struct tcp_sock *tp;
        int tama;
        sk = inet_lookup (red, &tcp_hashinfo,
                          conn_id->ip_local,
                          conn_id->puerto_local,
                          conn_id->ip_remoto,
                          conn_id->puerto_remoto,
                          tama);
        if (sk!=0) {
                tp=tcp_sk(sk);
                lock_sock(sk);
                tp->rcv_ssthresh=size;
                tp->window_clamp=size;
                release_sock(sk);
                return(0);}
        else
                return(1);
}
```

Fig. 1. Setting the new receiver windows

While some experiments were conducted with the prototype, further experimentation is needed to establish its behavior in various real world scenarios. Since the use case was shown to have some flaws, other more realistic cross-layer optimization protocols should be implemented and tested.

Future work will include porting the system to other plaftorms such as OpenWRT and Android, and developing a SNMP interface to explore the possibilities of global cross-layer optimizations.

IV. Testing

In this section the model of the system specification is briefly described, an explanation on how it was used to validate and test the prototype is given, and the results in terms of cost are summarized.

A. Model of the system specification

The model of the use case was built with RTDS [21] by PragmaDev [22], following the ECLAIR architecture. It syntactically conforms the SDL-92 standard. A diagram in SDL can be seen in Figure 2

After the model was obtained, simulations were done in order to gain confidence in the model. Further simulations were conducted for the test purposes in order to generate test cases.

The system is composed of two blocks, a TL or Tuning Layer block for interacting with the protocol stack, and a OSS or Optimizing Subsystem that encompasses the PO or protocol optimizers. Inside the TL there are two process corresponding to the UTL or User Tuning Layer, and to the TCPTL or TCP Tuning Layer. Inside the OSS there is the RCWPO process, the receiver window protocol optimizer; here resides the optimization logic.

The UTL receives signals indicating priority change in connections and notifies the RCWPO. The RCWPO process receives signals that indicate a priority change has occurred, end emits signals for the TCPTL indicating that the window size must be changed and its new value. The TCPTL receives signals from the RCWPO and emits signals to the Kernel to change the appropriate data structures.

The nomenclature for the components was taken directly from the ECLAIR architecture papers.

B. Test case generation

The RTDS tool provided an environment for simulations. From the simulations, MSC [23] corresponding to the test cases were generated.

The MSC test cases were used as input to the tester and as part of the oracle to establish verdict of the tests. An example is shown in Figure 3

The input and outputs of the systems were modeled as signals as usual in SDL, and mapped to concrete inputs and outputs such as function invocations.This mapping was used to convert the MSC to input files to the tester.

C. Testing architecture

Testing architecture consists in the components to test, components of the tester, interaction points which are the interfaces of the IUT with the tester, which can be PO (Point of Observation, not to be confused with Protocol Optimizers from ECLAIR) or PCOs (Point of Control and Observation). In the SDL model, the tester 'exists' at the environment.

The tester has two parts: the Kernel (Kernel facilities are employed for registering the outputs) and the MTC which runs in user space, runs the test suites and emits the verdict. The IUT has two parts also, a Kernel module running the TCPTL and the RCWPO, and the UTL in user space. The IUT includes the Kernel module (comprising the RCWPO and TCPTL) and the user space application for setting priorities (namely, the UTL).

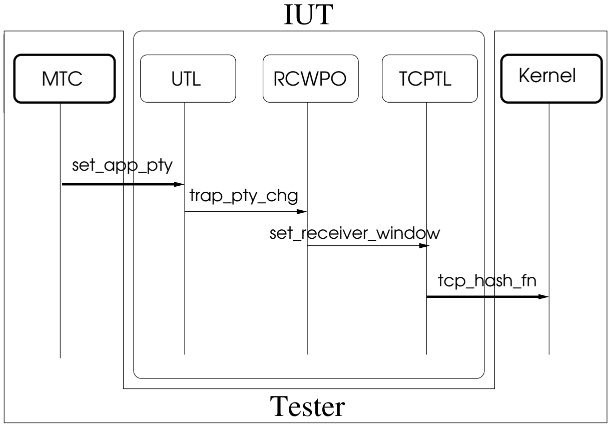

Fig. 3. Fragment of MSC test purpose

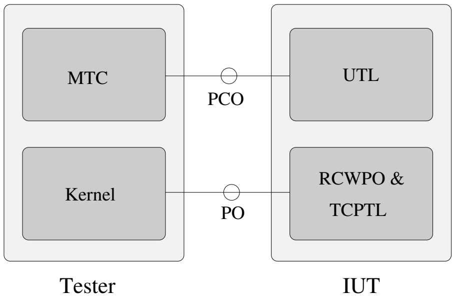

Fig. 4. Testing Architecture

```
set_app_pty FROM MTC, TO UTL
  (request the UTL to change the priority
  of a connection)
app_pty_ok FROM UTL TO MTC
  (returns the result of the priority change)
app_pty_err FROM UTL TO MTC
```

Fig. 5. Signals exchanged at the PCO

```
tcp_hash_fn FROM TCP_TL TO Kernel
  (request the sock corresponding to a connection)
tcp_hash_fn_value FROM Kernel TO TCP_TL
  (returns sock for the connection required)
read_rcv_wnd FROM TCP_TL TO Kernel
  (request parameters of a sock for an
   established connection)
read_rcv_wnd_value FROM Kernel TO TCP_TL
lock_socket FROM TCP_TL TO Kernel
set_wnd_clamp FROM TCP_TL TO Kernel
set_rcv_wnd FROM TCP_TL TO Kernel
release_socket FROM TCP_TL TO Kernel
```

Fig. 6. Signals exchanged at the PO

There are two interfaces between the tester and the IUT, one PCO used by the MTC to interact feed the input to the UTL,

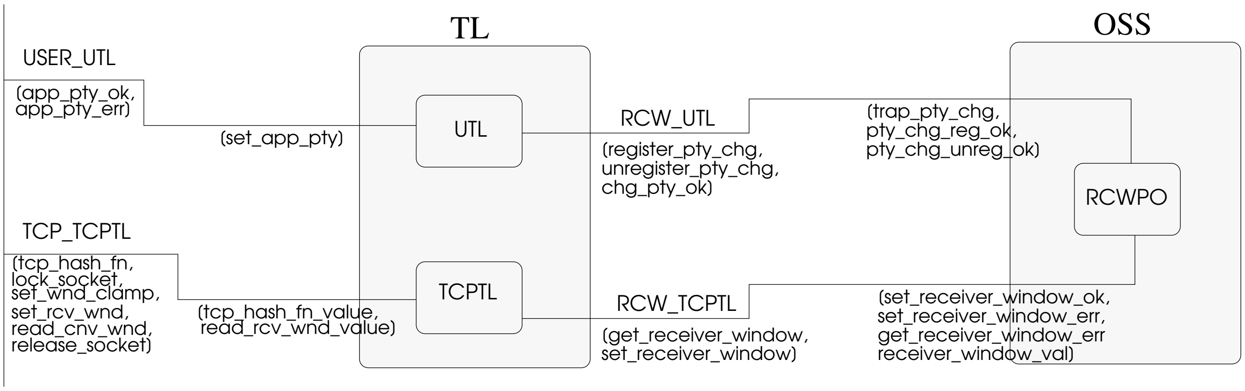

Fig. 2. System diagram in SDL

and one PO to capture the output of the Kernel module in Kernel space when values of rcv_ssthresh and window_clamp are set.

The signals exchanged at the PCO are listed in Figure 5, while the signals exchanged at the PO are presented in Figure 6.

These signals are used at the abstraction level of the model. For running the test suite, the signals are mapped to concrete operations, such as can be seen in the following examples:

The signal set_app_pty is mapped to

```
./userTL host_local port_local
   host_remote port_remote 2 2
```

userTL is a comand line utility to pass information to the UTL. The first numeric value identifies an operation (namely, set the pty) and the second the priority.

For tcp_hash_fn, the actual operation performed is an invocation of inet_lookup, a Kernel function. The most relevant parameters it receives are the local and remote IP and port, represented in network byte order by unsigned integers of 32 and 16 bits respectively, and a pointer to the 'hash_info' Kernel data structure

```
sock sk = inet_lookup(net, &hash_info,
  __be32 daddr, __be16 dest, __be32 saddr,
  __be16 source, int tama);
```

D. Test execution

An *ad-hoc* testing automation tool was built and used as the Main Test Component. It is a bash shell script that parses an input file and executes the input. When there are no more inputs to process, it parses an output file where the system traces were recorded in order to establish a verdict. At the time of the test, only the type of the signals and the data types (not values) were considered for verdict.

Note that the Kernel is part of the tester, for it receives the output of the system. This presents some challenges for capturing and logging the output.

Since the receiver window is included in the TCP header, a simple solution would have been to capture the TCP packets by means of Wireshark or a similar tool. However, this solution does not account for the window_clamp parameter. And, formally, the TCP packets are not the output of the modeled system.

In spite of the black box testing approach, the IUT was coded in a way that it wrote the outputs in the system log by means of the `printk()` function. A more appropriate solution would have been to intercept system calls and add a tracing facility before executing the actual code of the function.

Intercepting system calls would be an incomplete solution, since the prototype changes the value of variables explicitly, not by means of a system call but by an assignment statement. Therefore it might require memory inspection for the addresses where the data structure are stored.

These approaches have considerable potential but to keep on schedule were left for future work.

Something lacking in this work was the automatic conversion of MSC into input file for the tester, the fully automatic generation of test cases and a general framework for test execution.

For the general framework for test execution, TTCN-3 is the standard of the industry. TTCN-3[24][25] is the testing language promoted by the European Telecommunications Standards Institute (ETSI) which was designed for any kind of testing activity.

There are commercial tools that generate test cases from models in SDL and MSC which can be converted to ATS in TTCN-3. These tools can be used to fully automate the test case generation and execution process. We note that Telelogic [26] and Verilog [27] were the major SDL tool vendors. Telelogic complemented its TAU tool suite with Autolink for test case generation. Verilog extended OBJECTGeode with TestComposer. Verilog was acquired by Telelogic, which in turn was acquired by IBM [28].

E. Results

A two people team was assembled for the project, with no prior knowledge of modeling tools nor the modeling language and only minor knowledge on the problem domain. Thus the time devoted to learning is included in the cost of the project.

An iterative incremental software development process was followed. In week 1, the specification of the system was given. In weeks 2 and 3 the modeling language (SDL) for the formal specification was studied and the model of the system specification in SDL was produced. In week 3 the test cases were obtained from the model. In weeks 3 and 4 the prototype was implemented. In week 5 a test automation tool was implemented, and the prototype was tested. The project was evaluated.

The hours/person were approximately 320.

The UserTL module has 80 source lines of code and its compiled size was 2KB. The RWCPO plus the TCPTL module has 550 lines approximately and its compiled size was 9KB.

V. Conclusion

A cross-layer architecture was selected and studied thoroughly. As a result, a fully functional prototype that introduced cross-layer optimizations in devices with constrained resources was implemented. The chosen architecture was validated, as well as the model-based testing methodology, a formal model was obtained and used to derive test cases. The work started with the model, followed with the implementation and finished with an operational prototype that was tested.

Model-based testing approach, even without the fully automation of the test case generation proved a cost effective testing methodology even in a scarce resources situation, consisting of a two people team and a reduced time frame.

The applied methodology does not require sophisticated software developing processes, even though it can benefit from them in larger projects.

The need for an appropriate framework for test execution became evident. Even though the test automation tool developed was adequate, it lacked flexibility and portability and could not be used for a different project without some effort.

The prototype followed the use case specification, was built according to the ECLAIR architecture and was successfully tested and validated. Therefore, the methodology, tools and modeling techniques were appropriate to the problem domain.

To our knowledge the prototype was the first use of the use case and architecture for recent versions of the GNU/Linux Kernel. The work took place on a virtualized environment for practical reasons. Despite of that, is directly applicable to hardware implementations of corresponding embedded devices implementing GNU/Linux.

The model itself can be reused to include further optimizations or required refinements to the current, and to derive more test cases.

Acknowledgment

The authors would like to thank Eduardo Grampín who made this project possible. We would also like to thank Patrick Sénac for the inspiration taken from his work.

References

[1] (2010) SCAN website. [Online]. Available: https://www.niclabs.cl/scan

[2] Manfred Broy and Bengt Jonsson and Joost-Pieter Katoen and Martin Leucker and Alexander Pretschner (Eds.), *Model-Based Testing of Reactive Systems*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2005, no. 3472.

[3] Mark Utting and Bruno Legeard, *Model-Based Testing: A Tools Approach*. San Francisco, USA: Morgan Kaufmann, 2006.

[4] V. T. Raisinghani, A. K. Singh, and S. Iyer, "Improving TCP performance over mobile wireless environments using cross layer feedback," in *IEEE International Conference on Personal Wireless Communications (ICPWC-2002)*, Delhi, India, Dec. 2002, pp. 81–85.

[5] V. T. Raisinghani and S. Iyer, "Cross-layer feedback architecture for mobile device protocol stacks," *IEEE Commun. Mag.*, vol. 44, pp. 85–92, Jan. 2006.

[6] *Specification and Description Language (SDL)*, ITU-T Recommendation Z.100.

[7] *Information technology - OSI - Basic Reference Model*, ITU Recommendation X.200.

[8] L. Zhang, P. Sénac, E. Lochin, and M. Diaz, "Mobile TFRC: a congestion control for WLANs," in *The IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM 2008)*, Newport Beach CA, USA, Jun. 2008, pp. 23–26.

[9] J. Inouye, J. Binkley, and J. Walpole, "Dynamic network reconfiguration support for mobile computers," in *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, Budapest, Hungary, 1997, pp. 13–22.

[10] P. Sudame and B. R. Badrinath, "On providing support for protocol adaptation in mobile networks," *ACM Mobile Networks and Applications*, vol. 6, pp. 43–55, Jan. 2001.

[11] QiWang and M. Abu-Rgheff, "Cross-layer signalling for next-generation wireless systems," in *Wireless Communications and Networking (WCNC)*, New Orleans, USA, Mar. 2003, pp. 1084–1089.

[12] R. Winter, J. Schiller, N. Nikaein, and C. Bonnet, "Crosstalk: cross-layer decision support based on global knowledge," *IEEE Commun. Mag.*, vol. 44, pp. 93–99, Jan. 2006.

[13] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-layering in mobile ad hoc network design," *IEEE Computer*, vol. 37, pp. 48–51, Feb. 2004.

[14] Christel Baier and Joost-Pieter Katoen, *Principles of Model Checking*, ser. Lecture Notes in Computer Science. Cambridge, USA: The MIT Press, 2008, no. 3472.

[15] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," *Software-Concepts and Tools*, vol. 17, pp. 103–120, Mar. 1996.

[16] J. Fernandez, C. Jard, T. Jéeron, and C. Viho, "An experiment in automatic generation of test suites for protocols with verification technology," *Science of Computer Programming Special Issue on COST247, Verification and Validation Methods for Formal Descriptions*, vol. 29, pp. 123–146, Jul. 1997.

[17] C. Jard and T. Jéron, "TGV: theory, principles and algorithms, A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems," *International Journal on Software Tools for Technology Transfer*, vol. 7, pp. 297–315, Oct. 2004.

[18] J. Tretmans and H. Brinksma, "TorX: Automated model-based testing," in *First European Conference on Model-Driven Software Engineering*, Nuremberg, Germany, 2003, pp. 31–43.

[19] Ana Cavalli and Amardeo Sarma (Eds.), *SDL '97: Time for Testing*. Amsterdam, Netherlands: Elsevier, 1997.

[20] Laurent Doldi, *Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis*. Wiltshire, Great Britain: Wiley, 2003.

[21] (2010) Pragmadev RTDS. [Online]. Available: http://www.pragmadev.com/product/RTDSV4.pdf

[22] (2010) Pragmadev website. [Online]. Available: http://www.pragmadev.com/

[23] *Message Sequence Chart (MSC)*, ITU-T Recommendation Z.120.

[24] *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*, ETSI ETSI Standard 201 873-1 v2.2.1.

[25] Colin Willcock and Thomas Deiss and Stephan Tobies and Stefan Keil and Federico Engler and Stephan Schulz, *An Introduction to TTCN-3*. Wiltshire, Great Britain: Wiley, 2005.

[26] (2010) Telelogic AB. [Online]. Available: http://en.wikipedia.org/wiki/Telelogic

[27] (2010) Telelogic acquires VERILOG. [Online]. Available: http://www.highbeam.com/doc/1G1-58356443.html

[28] (2010) IBM acquires Telelogic. [Online]. Available: http://www-01.ibm.com/software/rational/welcome/telelogic/