Уральский государственный архитектурно-художественный университет
Национальный исследовательский Томский государственный университет
Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ИССЛЕДОВАНИИ СЛОЖНЫХ СТРУКТУР

МАТЕРИАЛЫ
ОДИННАДЦАТОЙ МЕЖДУНАРОДНОЙ КОНФЕРЕНЦИИ
6–10 июня 2016 г.

Томск
Издательский Дом Томского государственного университета
2016

augment the classical ones with the set of context variables, input and output parameter values as well as the set of predicates that 'show' which transition can be executed at a given point. Correspondingly, we discuss some heuristics for optimizing SDN controller components by extracting 'close to the best' implementation from the nondeterministic extended / timed machine.

As future work, we plan to implement the proposed solutions and perform the experimental evaluation with 'real-life' virtual networks for optimizing and testing SDN controllers and their modules.

References

1. *Jmila H., Zeghlache D.* An adaptive load balancing scheme for evolving virtual networks // Proceedings of the CCNC. 2015. P. 492–498.

2. *Doriguzzi C.R., Salvadori E., Aranda-Gutiérrez P.A., Stritzke C., Leckey A., Phemius K., Rojas E., Guerrero C.* NetIDE: Removing vendor lock-in in SDN // Proceedings of the NetSoft. 2015. P. 1–2.

3. *Gill, A.* State-identification experiments in finite automata // Information and Control. 1961. Vol. 4. P. 132–154.

4. *Villa T., Yevtushenko N., Brayton R.K., Mishchenko A., Petrenko A., Sangiovanni-Vincentelli A.* The Unknown Component Problem: Theory and Applications. Springer, 2012. 326 p.

SYNCHRONIZING SEQUENCES FOR EFFECTIVE NETWORK MONITORING: AN APPLICATION TO TCP

M. Suhopluev[1], *N. Schipachev*[1], *J. López*[2], *N. Kushik*[1, 2]

[1] Tomsk State University, Tomsk, Russia
[2] SAMOVAR, Télécom SudParis, CNRS, Université Paris-Saclay, Évry, France
{e12max, shchipachevnm}@gmail.com,{jorge.lopez, natalia.kushik}@télécom-sudparis.eu

The rapid development of information technologies requires that all network interactions are thoroughly checked. Traditional firewalls and intrusion detection systems (IDS) such as snort [1] inspect and filter network traffic based on the information contained on a single network packet (stateless inspection). In order to analyse complex network properties, the correlation between packets from the network traces, is now an emerging concept [2].

It should be mentioned that in passive testing or network monitoring, the implementation under test (IUT) cannot be interrupted or manipulated. Therefore, the current execution state of the IUT is considered to be a priori unknown. However, the number of properties to be verified at a given time instant can be significantly reduced if the IUT state is known and thus, only critical properties for this state are verified. Therefore, existing solutions for the so-called state identification problem [3] can be effectively used to determine the current IUT state. This approach have been proposed in [4] and in this presentation, we discuss how it can be applied for passive testing of the transmission control protocol (TCP) implementation [5].

The approach is based on the derivation of synchronizing sequences for a Finite State Machine (FSM) [6] that describes the TCP behavior. The FSM moves from one state to another when an input is applied; when the corresponding transition is executed the appropriate output is produced. The FSM can be brought to a known current state by the application of a synchronizing sequence, i.e. independently on the initial FSM state and the output reaction the current FSM state becomes known.

In order to apply this technique for the TCP passive testing, we adapted a state model for it, taken from [7]. The resulting FSM that describes the TCP behavior is complete and deterministic; it has nine states and ten inputs. We performed experimental evaluation to estimate how likely is that a synchronizing sequence can be observed during the monitoring process of the TCP implementation. The likelihood of a synchronizing sequence to appear among all network traces being observed we estimate as the probability of, where is the number of all the synchronizing sequences of length, and denotes all possible input sequences of length. Experimental results show that there do not exist synchronizing inputs for the TCP FSM. However, even when five input sequences of length two are synchronizing for this FSM. These sequences transfer the FSM from any of initial states to one three possible states, i.e. whenever of synchronizing pair of inputs is observed, only properties that are interesting for one of these three reachable states need to be checked. Moreover, if the length increases, the number of 'good' sequences of this length increases as well. In fact, if during the monitoring process a test engineer analyzes the traces of length five and more, then with a probability more than 30% a synchronizing sequence that uniquely identifies the current state of the TCP implementation can be observed. The later proves the applicability of the approach as it allows to reduce the number of properties to be checked during the network monitoring.

We mention that not all the input sequences can be observed when analyzing network traces. It can happen that among 30% of 'good' input sequences only 10% can be actually observed due to the IUT environment, observation

points' location, etc. Therefore, as future work, we plan to perform the experiments with real TCP implementations to estimate how often synchronizing sequences of interest appear during such monitoring. Testing against different test purposes such as, security, robustness, etc. is another direction of further TCP experimentation.

References

1. *Roesch M.* Snort: Lightweight intrusion detection for networks // 13th Systems Administration Conference (LISA). 1999. P. 229–238.

2. *Mallouli W., Wehbi B., Montes de Oca E., Bourdellès M.* Online network traffic security inspection using MMT tool // 24th International Conference on Software & Systems Engineering and their Applications (ICSSEA2012), 9th Workshop on System Testing and Validation (STV) Workshop. Paris, 2012.

3. *Lee D., Mihailis Y.* Testing finite-state machines: state identification and verification // IEEE Transactions on Computers. 1994. Vol. 43, is. 3. P. 306–320.

4. *Kushik N., López J., Cavalli A., Yevtushenko N.* Optimizing Protocol Passive Testing through 'Gedanken' Experiments with Finite State Machines // 2016 IEEE International Conference on Software Quality, Reliability & Security. 2016.

5. *Postel J.* Transmission Control Protocol (RFC 793) // Internet Engineering Task Force. 1981.

6. *Gill A.* State-identification experiments in finite automata // Information and Control. 1961. Vol. 4. P. 132–154.

7. *TCP* Finite State Machine from School of Computer Science University of St Andrews. URL: http://tcp.cs.st-andrews.ac.uk/index.shtml?page=tcp_fsm.

AUTOMATED CONSTRUCTION OF TEST PROGRAM GENERATORS FOR MICROPROCESSORS BASED ON FORMAL SPECIFICATIONS

A.D. Tatarnikov

Institute for System Programming of the Russian Academy of Sciences (ISP RAS), Moscow, Russia
andrewt@ispras.ru

Test program generation (TPG) and simulation is the most widely used approach for *functional verification* of microprocessors [1]. Growing complexity of modern hardware designs and shrinking time to market demand for highly efficient TPG tools. A common problem with many of such tools is that they are designed for testing a specific microprocessor design using a limited set of TPG methods. Consequently, when it is required to support new microprocessor designs or new TPG methods, a significant part of the tool has to be rewritten. This causes delays in the schedule and has a negative impact on the quality of testing, since the knowledge acquired in previous projects cannot be efficiently reused.

This paper presents a method to automatically construct a TPG tool for an arbitrary microprocessor design based on *formal specifications* of its *instruction set architecture (ISA)*. The method is implemented in MicroTESK [2], a tool being developed at ISP RAS. It is aimed to overcome the complexity and make it easy to apply the growing set of TPG methods to a wide range of microprocessor designs. The key idea of the method is that the tool is constructed from two main components: (1) an architecture-independent core that implements TPG methods applicable to all microprocessor designs and (2) an architecture model that holds design-specific knowledge. The core is implemented as a set of TPG engines and can easily be extended with new components. The model is constructed by processing formal specifications and consists of two parts: (1) *ISA simulator* that tracks the microprocessor state during generation and (2) *testing knowledge* that contains the description of situations to be covered by tests. The knowledge obtained from specifications is described using architecture-independent abstractions and can be used by a wide range of TPG engines. It is possible to extend the model with new types of knowledge by adding new components responsible for specification processing. The described approach helps minimize the efforts required to create a TPG tool by using simple formal specifications and taking maximum advantage of reusing existing components.

A constructed TPG tool generates test programs from *test templates* that describe verification tasks in terms of the model and the TPG methods implemented by the core. The TPG methods include methods of constructing instruction sequences and generating test data. They may be based on random, combinatorial, constraint-based, FSM-based and other techniques. The generation algorithm includes the following steps: (1) constructing an abstract instruction sequence that specifies input data in terms of test situations to be covered; (2) generating test data for these situations (e.g. by solving constraints); (3) executing instructions in the ISA simulator to track the microprocessor state and (4) printing the instruction sequence to a source code file in the assembly language. The information on the microprocessor state provided by the ISA simulator is used as a context for constraint solving and as a basis for creating tests with built-in checks. Also, simulation during generation helps assure that the generated test program is correct (i.e. it does not use illegal instruction arguments and does not create infinite loops).