16

Simulation-Based Verification of Network Protocols Performance

M. Baldi, F. Corno, M. Rebaudengo, P. Prinetto,
M. Sonza Reorda, G. Squillero
Politecnico di Torino – Dip. Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129 Torino, Italy
Tel. +39 11 564.7007, fax +39 11 564.7099
e-mail:{mbaldi,corno,reba,prinetto,sonza,squillero}@polito.it

Abstract

Formal verification techniques need to deal with the complexity of the systems being verified. Most often, this problem is solved by taking an abstract model of the system and aiming at a complete verification of an approximation of the system. This paper proposes a complementary verification approach, in which the approximation is introduced into the verification algorithm, instead of the system model: we achieve an approximate verification of a fairly complete and detailed system model. The proposed technique relies on coupling a Genetic Algorithm with a simulator of the system under verification, and is especially suited for verifying performance-related aspects. To prove the effectiveness of our approach, we applied it to the quantitative verification of a network protocol: the TCP protocol operating on a given network. A Genetic Algorithm is able to find a configuration of the traffic over the network that sensitizes a critical problem in the TCP protocol that would be difficult to find both with exact techniques and stochastic ones.

Keywords

Genetic Algorithms, Performance Verification, Approximate Verification, Network Protocols

1 INTRODUCTION[1]

Formal verification techniques collect growing interest in several areas, such as hardware design, protocol analysis, software testing. However, due to the complexity of most verification tasks, complete and exact verification is out of reach for techniques proposed so far. The problem is particularly evident when performance of the system is an essential part of its specification: most approaches concentrate on correctness verification, while abstracting from timing issues. Usually, complexity is tamed by dropping some requirements: to deliver exact results either we constrain the verification to small systems, or we neglect timing information, or we verify an abstract model of the system, or we consider just qualitative measures instead of quantitative performance figures, or we restrict to suitable subsets of description languages.

In this paper we explore a different approach to deal with the complexity problem in verification, namely, to sacrifice *exactness* in favor of a more complete system model, that includes timing and performance issues. In other words, we aim at developing an approximate verification algorithm that operates over a detailed description of the system being verified: although we cannot guarantee to find all violations of the specification, we are able to model with a greater detail the system behavior and therefore consider a larger set of potential problems.

In particular, we propose the adoption of *simulation-based* techniques that rely on a *Genetic Algorithm* (GA) (Holland, 1975) to explore the solution space and look for an inconsistency in the verification goal. Such techniques are gaining increasing acceptance in the field of test pattern generation for digital circuits, and we think that they can also be profitable applied to formal verification tasks. The paper aims at an experimental evaluation of the effectiveness of this approach, by applying Genetic Algorithms to a particular protocol-verification problem.

As a case study problem, we chose the performance verification of the Transmission Control Protocol (TCP) (Postel, 1981), a widely used computer network protocol.

We chose this problem because of the increasing demand for efficient and reliable communication protocols in the telecommunication industry. Computer networks, in fact, are gaining an increasing importance as they penetrate business and everyday-life (Stevens, 94). Technological evolution results in increased computational power and transmission capacity. These two phenomena open the way to the development and exploitation of new applications (e.g., video-conferencing) which require demanding services from the network. As a result

---

[1] This work is partially funded by EU-funded FOST and CHARME projects.

computer networks are in continuous evolution and the protocols regulating their operation must be assessed for their suitability to new technologies and ability to support the new applications. This assessment is particularly important since new technologies and applications often differ from the ones the protocol has been designed for.

The analysis of communication protocols is usually done through either formal modeling or verification, or simulation.

Formal verification techniques are based on modeling the protocol under study with an either general or special purpose description language, which is possibly coupled with automatic tools for property verification. LOTOS (ISO, 1988) is, for example, a formal language for specification of behaviour sequences and can be used for description of general distributed systems, among which protocols. The ESTELLE (Linn, 1986) protocol modeling language can be used to yield state oriented models. Petri nets (Peterson, 1981) are another widely used tool for the design, property verification, and performance evaluation of communication protocols because of their ability to model concurrency, nondeterminism, and communications. Moreover, Petri nets have been enhanced to model data manipulation and time dependency (Razouk and Phelps, 1985) in order to make them more suitable to communication protocol modeling. One of the advantages of exploiting Petri nets is the large availability of automatic tools for verification and performance evaluation of Petri net models. A possible approach is also the modeling of protocols with languages whose models are automatically translated into Petri nets for further analysis. One example is the Entity Behavioral Specification and Description Language (Suzuki et al., 1990) which is aimed only at protocol entity specifications and is coupled with an algorithmic procedure for the conversion of its models into Petri nets. These description languages are powerful enough to capture most of the protocol semantics, however, verification tools pose strong limits in the size and complexity of the descriptions they can deal with. Some important aspects of a network protocol that are difficult to verify are those related with different timeout counters, message queues, and transmission strategies based on message contents.

Simulation allows the protocol to be checked in its entire complex behavior and the dependencies among the various hardware and software components of a computer network to be explored. A typical simulator allows the actions taken by a protocol to be monitored while operating over a computer network with a specified topology and traffic load. The simulator models a number of sources that can be programmed to generate traffic according to statistical models defined after real sources. One or more sources behave according to the protocol under study; the others are considered background traffic generators. Poisson statistical processes (Hui, 1990) are often adopted as a model of background traffic.

We feel that neither approach is sufficient to give a reliable information about the protocol behavior. While the formal approach delivers a mathematically proven answer, it is forced to work on an over-simplified view of the protocol.

Simulation techniques, on the other hand, rely on a clever choice of the traffic pattern to yield useful results, and worst-case analysis is often performed by hand-designing the traffic.

In this work, we show how to exploit a GA to drive the generation of background traffic. The GA therefore aims at generating the worst-case traffic for the protocol under analysis, given some constraints on the traffic bandwidth and distribution. Errors and weaknesses in the network protocol can therefore be discovered via simulation of this worst-case traffic. The longer the GA runs without discovering any inconsistency, the higher the probability that the protocol is correctly designed and implemented. At the same time, we expect the GA to drive the network to conditions where the effective transfer rate is low, and we can study the behavior of the protocol under extreme load conditions. We expect a GA to perform significantly better than a pure stochastic approach, since the algorithm explicitly aims at finding an ill-behaved traffic.

In a previous work (Alba and Troya, 1996) GAs were applied to the analysis of communication protocols by checking the correctness of their finite state machine (FSM). Communicating protocol entities are expressed as a pair of communicating FSMs, a GA generates communication traces between the two protocol entities, and a FSM simulator executes them. The GA aims at detecting deadlocks and useless states.

By contrast, our approach, being based on a network simulator, is able to model most of the aspects of the protocol stack and the network on which it is employed: not just the state machine, but also counters, message queues, routing tables, different re-transmission speeds, and so on. Since the GA aims at generating the worst operating conditions of the protocol (by defining the load in the network) and finds the traffic configuration that minimizes its performance, the protocol is pushed to its limits. Some preliminary experimental results we gathered show that traffic generated by the GA is able to expose protocol shortcomings that were impossible to reveal with formal techniques, and that were not found by means of statistical simulation.

Section 2 describes the adopted methodology for approximate verification and outlines the tool architecture. Details about the GA are given in section 3, and section 4 reports some experimental results on the verification of the TCP protocol over a packet switched network. Some conclusions are drawn in section 5.

2 APPROXIMATE VERIFICATION METHODOLOGY

In this work we choose to examine the TCP protocol, whose characteristics and behavior are already well known to the computer network research community, in order to better focus on the verification methodology and exploitation of the GA, and not on the protocol itself. Hence, we do not expect to discover any new information about TCP behavior and performance, but to find in an automated

way well known data that, with traditional approaches, required skills and in-depth knowledge of the protocol.

We aim at studying the TCP protocol in real operating conditions. Thus, we set up an IP (Internet Protocol) network (Stevens, 1994) and a number of TCP sender-receiver pairs into it. The GA is not allowed to control such connections directly, but it monitors their activity, hence, we will refer to them as *probe connections*. The network is loaded with an additional *background traffic* generated by UDP (User Datagram Protocol, Postel, 1980) sender-receiver pairs controlled by the GA. We choose the UDP protocol for the sake of simplicity, but other kinds of background traffic sources can be modeled as well.

During the analysis process, the GA provides a *pattern* of the traffic generated by background sources and a network simulation is run on the given topology. During this simulation relevant data are gathered on the probe connections by the simulator program and provided to the GA, which uses them to estimate the "damage" that the background traffic did to the probe connections. Such information is then used to drive the generation of traffic patterns to be used in subsequent steps of the GA.

As for statistical methods, our analysis requires a large number of simulation runs to obtain a significant result, therefore the choice of the simulation program is critical in order to keep the computational time small. In our work, simulations were run using a publicly available simulator called *insane* (Internet Simulated ATM Networking Environment) (Mah, 1996). *Insane* adopts an event-driven mechanism for the simulation, therefore it is reasonably fast. Moreover, it is able to deal with an arbitrary network topology described as a TCL (Welch, 1995) data structure and many characteristics of network nodes and links (such as processing speed, buffering capacity, bandwidth and latency) can be controlled.

Figure 1 shows the architecture of the proposed verification environment, called Nepal, (Network Protocol Analysis Algorithm), which integrates the GA and the simulator. The approach is quite general and can be applied to different protocols using different simulators with limited effort.

3 GENETIC ALGORITHM DEFINITION

3.1 Introduction

GAs have been deeply investigated in the last two decades as a possible approach to solve many search and optimization problems.

GAs belong to the class of *evolutionary algorithms*, based on a mechanism which mimics the way nature follows to improve the characteristics of living beings. Each solution (*individual*) of the problem to be solved is represented as a string (*chromosome*) of elements (*genes*); each gene may have one of several values (*alleles*).

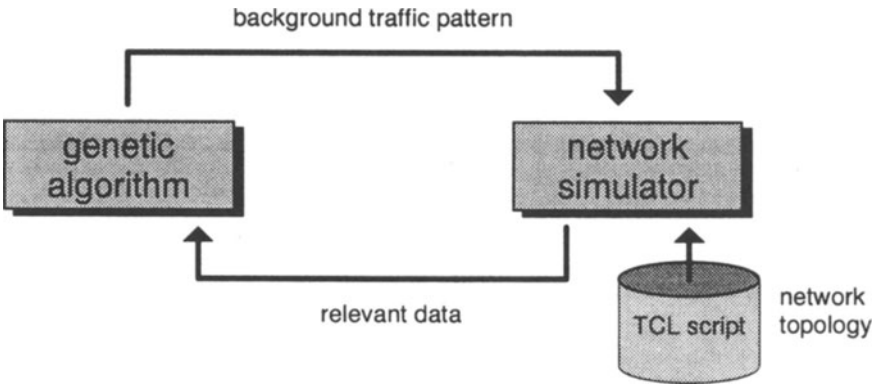

Figure 1 Architecture of the Nepal system.

A *fitness value* is associated with each individual, based on the value returned by an *evaluation function*. The fitness value measures how close the individual is to the optimum solution.

A set of individuals composes a *population*, which evolves from generation to generation through the creation of new individuals and the deletion of some old ones. The process starts with an initial population of individuals created in some way, e.g., randomly. New individuals are generated through *evolution*, which is based on two mechanisms:

- *cross-over*: two existing individuals are selected and their chromosomes are combined to obtain a new individual. The newly generated individuals are inserted in the population in place of some other individuals, e.g., the ones with the lowest fitness (*elitism*) (DeJong, 1975). Individual selection for the cross-over operator is generally made through techniques which guarantee that the selection probability of each individual is proportional to its fitness. The new individuals are thus likely to have a higher fitness than the old ones, and the process is oriented towards those sub-regions of the search space, where optimal solutions are supposed to exist.
- *mutation*: a gene of a selected individual is randomly changed. Mutation provides the method with additional chances of entering into unexplored sub-regions.

3.2 Nepal

The pseudo-code of the GA adopted within Nepal is shown in Figure 2. When the GA starts, N_p individuals are randomly generated. In our approach an individual represents a background traffic pattern. Due to the encoding of the delay values in genes, which will be explained in section 3.3, the initial background traffic is distributed according to the Poisson distribution.

At each generation the GA creates N_o new individuals by applying a crossover operators to two parents. Parents are randomly chosen with a probability linearly increasing from the worst individual (smallest fitness) to the best one (highest fitness). A mutation operator is applied over each new individual with a probability p_m.

At the end of each generation, the N_p individuals with higher fitness are selected for survival, and the worst N_o ones are deleted from the population. The fitness function measures the throughput of probe connections, i.e., the performance of the probe TCP connections perceived by end-users during the simulation experiment. All bytes successfully received at the TCP level, but not delivered to end-users, such as packets sent due to the retransmission mechanism of the protocol, are therefore not considered.

Since the fitness function should increase with the increasing goodness of a solution, and being in Nepal a solution good when the background traffic pattern is critical, the fitness function we defined is *inversely proportional* to the total number of bytes perceived by the end-users.

fitness = 1/(*total bytes delivered to the TCP users on all probe connection*)

This simple measure already delivers satisfactory results on simple network topologies. We are actually experimenting Nepal on more complex topologies, and we are considering the inclusion of additional parameters into the fitness function.

3.3 Encoding

Each individual encodes the full specification of the traffic generated by all the background connections for the whole duration of the simulation. A connection is specified by a UDP source and destination pair, while the route followed by packets depends on the network topology and on the routing mechanism. We assume that all packets sent on the network are of the same size, hence individuals do not need to encode this information for each packet.

Individuals are encoded as strings of N genes, where each gene represents a single background packet. Genes are composed of two parts: a TAG that indicates which background connection the packet will be sent on and a DELAY that represents how long the given source will wait before sending a new packet after sending the current one.

A model for background traffic commonly adopted in statistical network analysis is a Poisson process with negative exponentially distributed arrival times (Hui, 1990). In order to ensure that delays given by a randomly generated individual are exponentially distributed, the DELAY field is an index in an array of values exponentially distributed from 0 to K seconds with an average value of $0.4*K$. We make no assumptions on the number of genes with a given TAG,

however, N is usually large enough to make the initial random background traffic almost equally distributed between sources.

To ensure that a given background traffic is determined by an unique individual, genes are sorted within an individual to guarantee that the packet associated to gene i+1 is never sent before the packet associated with gene i. Otherwise, it would be possible to exchange adjacent genes with different TAGs without modifying the represented background traffic, and some crossover operators (explained in section 3.4) would not operate well. We say that genes in the individual are "sorted in time".

It is possible that the genes with a given TAG are not enough to generate traffic for the whole experiment, in this case the connection remains idle in the final part of the experiment. On the other hand, it is also possible that the simulation ends before all the packets are sent, and final genes are ignored.

Figure 2 shows, on the left hand side, a sample individual and, on the right hand side, the corresponding background traffic generated during the simulation experiment. The table mapping the DELAY field to exponential delays is reported in the middle of the figure.

In the individual shown in Figure 2, packet number 9 is not sent because the simulation reaches its end before the delay associated with gene number 6 expires. On the other hand, one gene with tag ⊠ is missing, because the delay associated with the last packet on that connection (packet number 7) expires before simulation's end.

```
P0 = random_population();
compute_fitness(P0);
i=0; /* current generation number */
while (i < max_generation)
{
     A = Pi;
     for j=0 to No
     { /* new element generation */
       s'= select_an_individual();
       s"= select_an_individual();
       sj = cross_over_operator(s', s");
       if(rand()≤pm)
         sj = mutation_operator(sj);
       A = A ∪  sj;
     }
     compute_fitness(A);
     i++;
     Pi = [ the Np best individuals ∈ A ];
}
return( best_individual(Pi) )
```

Figure 2 Pseudo-code of the genetic algorithm

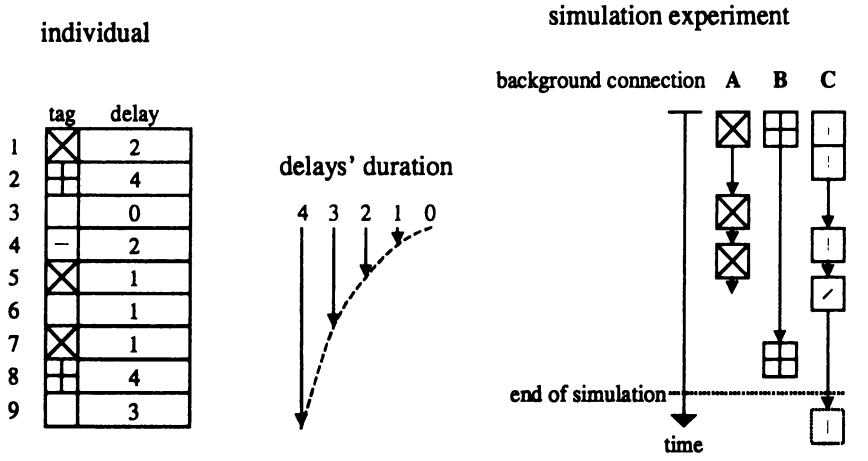

Figure 3 Individual encoding

3.4 Crossover Operators

Nepal implements 4 types of crossover: the standard *one-cut*, *two-cuts* and *uniform*, and the *mix-source* ones (Figure 4). After crossover, genes of the offspring need to be *sorted in time*.

In the **one-cut crossover** Nepal randomly selects a cut point c between 1 and N and generates a new individual copying the first c genes from the first parent and the subsequent N-c ones from the second parent.

The **two-cuts crossover** acts in a similar way, but it selects two cut points c_1, c_2 and copied genes are taken from the first parent at both the beginning and the end.

These two crossover operators generate a background traffic that is equal to the traffic generated by one parent for a given time and then suddenly changes and follows the traffic generated by the other parent (strictly speaking, it is possible that the cut points do not correspond exactly to the same time instants in the two parents, however, N is large enough for this to be neglected).

The **uniform crossover** generates an offspring taking each gene from the first or the second parent with the same probability.

In the **mix-source crossover**, Nepal first randomly associates each tag (i.e., each background source) to one of the two parents, then copies all the genes with the same tag from that parent. The new individual can be longer or shorter than N: in the former case the exceeding genes are dropped; in the latter one the individual is filled with random genes. The *mix-source* crossover aims at putting together good background sources from different individuals.

We can say that the *mix-source* crossover acts in the space (or source) domain while the *one-cut* and *two-cut* ones act in the time domain. The *uniform* crossover is hybrid and acts in both domains.

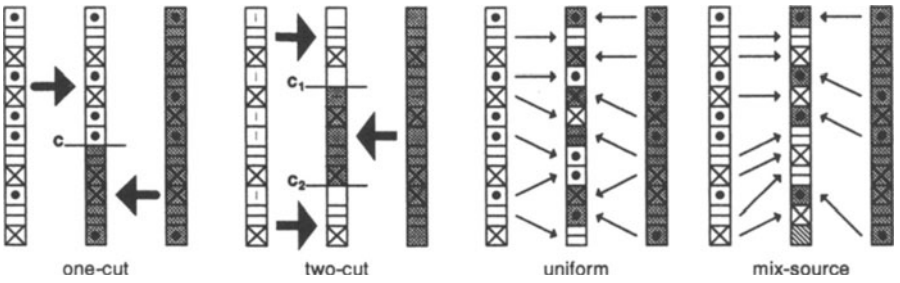

Figure 4 Crossover operators.

3.5 Mutations

Nepal implements 3 different mutation operators: *speed-up*, *slow-down* and *random*. Mutation applies on $N_m<N$ contiguous genes, where N_m is a constant. The first gene is randomly selected between 1 and N and the gene list is considered circular.

The **speed-up mutation** decreases the DELAY field by one without modifying the tag. DELAY values are considered circular.

The **slow-down mutation** increases the DELAY field by one without modifying the tag. DELAY values are considered circular.

The **random mutation** substitutes each gene with a new random one.

After each mutation, genes in the individual need to be sorted to fulfill the "sorted in time" property.

4 EXPERIMENTAL RESULTS

4.1 Prototypical implementation

We developed a prototypical implementation of the algorithm using the `perl 5` (Wall *et al.*, 1996) language. The implementation of the GA is less than 500 lines long and, with a limited effort, can be ported to different simulators and different platforms.

The source of the simulator consists of 38,370 lines of C++ code. We needed to modify *insane* to handle externally generated background traffic; modifications of the original code are limited to about 100 lines. Such modifications allow the user to define a traffic source taking the list of packets to be sent from a file or a Unix pipe.

4.2 Network topology

Figure 5 shows the topology of the IP (Internet Protocol) (Stevens, 1994) network exploited in the experiments. 3 TCP connections span from the transmitters TX*i*

to the receivers RX*i* through 3 IP routers. Each TCP connection performs long file transfers generating a series of 1024 byte messages at a maximum average rate of 1.33 Mb/s. Thus, the 3 TCP sources can generate an overall load of 4 Mb/s, if they are allowed by the TCP flow control mechanism. Acknowledgments from each TCP transmitter are carried by control messages flowing in the reverse direction. These 3 TCP connections represent the *probe connections* of the experiment.

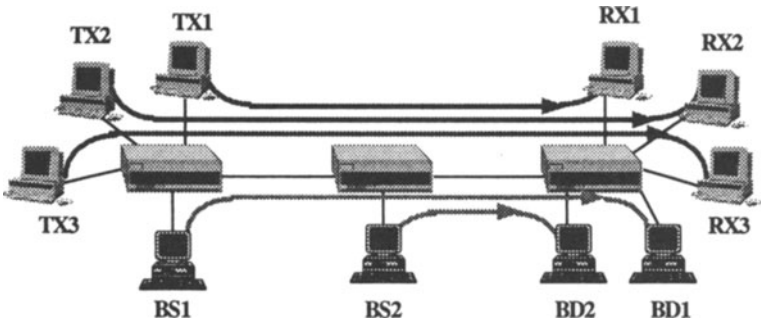

Figure 5 Topology of the network

Two sources (BS*i*) generate *background* UDP traffic directed to their respective destinations (BD*i*) over the same links traversed by the TCP connections. The GA, as described in section 3, controls the timing of background packets. The background traffic is generated only in the forward direction of the TCP connections and thus only data messages can be delayed and discarded due to network overload, not acknowledgments.

In this topology, the link between the rightmost two routers represents a bottleneck for the network since it is traversed by all the traffic generated by both the background sources and the TCP connections.

Each link shown in Figure 5 has a capacity of 10 Mb/s in each direction and introduces a fixed 10 μs delay. E.g., such links can be considered as dedicated 10 Mb/s Ethernet trunks, the 10 μs latency accounting for the propagation delay on the wires and the switching delay introduced by an Ethernet switch.

Routers introduce a fixed 0.1 ms delay, which accounts for the processing required on each packet and adds to the queuing delay. The *insane* simulator requires router output queues to be sized in terms of number of packets (we used 64 packet queues in our experiments). The buffering capacity of real routers is given in terms of byte volume, i.e., the number of packets that can be stored in a buffer depends on their size. We choose to deal with fixed size packets to work around this limitation of the simulator program.

4.3 Parameter values

For performing the experiments, two sets of parameters are defined. A first set of parameters, whose values are reported in Table 1, is used to describe the characteristics of the traffic. In the experiments we ran, they were set in order to model a 6 Mb/s background traffic level, because, in this situation, the probe connections traffic (4 Mb/s) plus the background traffic is nearly equal to the links bandwidth (10 Mb/s). With a smaller load, the background traffic would be unable to interfere significantly with probe connections; while with a larger one, the whole network would be congestioned and the TCP throughput would be very small even with a completely random traffic.

The remaining parameters are specifically related to the evolution mechanism of the GA, and are summarized in Table 2. The value of N mainly determined the duration of the experiments, while the values of the other parameters are quite standard for Genetic Algorithms and do not require any extensive tuning.

Table 1 Parameters influencing the traffic

TRAFFIC PARAMETER	*NAME*	*VALUE*
Simulation duration [s]	T	3
Total number of packets	N	4,500
Poisson distribution's parameter [ms]	K	6.67

Table 2 Parameters influencing the GA

TRAFFIC PARAMETER	*NAME*	*VALUE*
Population size	N_p	50
Reproduction rate	N_d/N_p	40%
Mutation probability	p_m	50%
Number of genes affected by a mutation	N_m	100

4.4 Results

We let the GA run for 500 generations on a Sun SPARC Station 5 with 64 MByte of memory. The GA required about 17,000 seconds of CPU time (mainly spent for sorting in time genes of newly generated individuals) and *insane* employed about 53,000 seconds of CPU time for running all the required simulations.

Figure 6 shows the decrease of the probe connections' throughput, i.e., the TCP bandwidth perceived by users. The X-axis reports the generation number of the GA. These values are compared to the ones of a standard statistical approach, where the background traffic is randomly generated according an equivalent Poisson process. For the statistical approach, we report the lowest throughput obtained after simulating a number of random patterns equal to the number of simulations required by Nepal until the given generation.

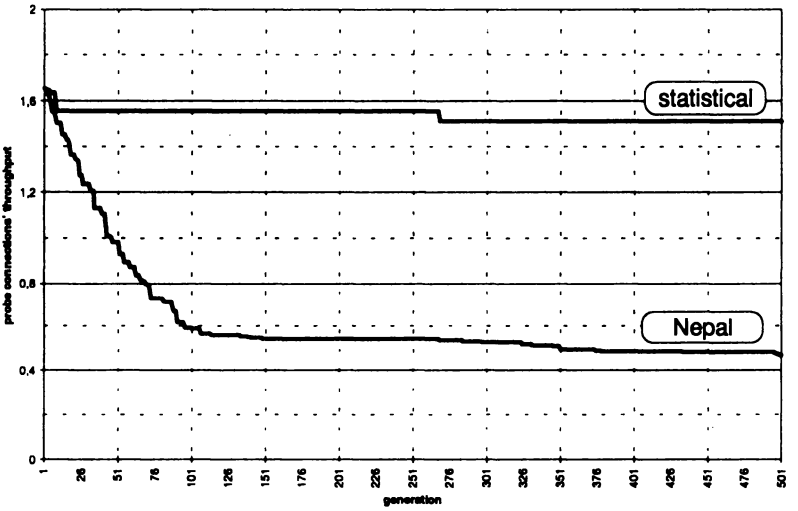

Figure 6 Throughput of probe connections

During the experiment, Nepal managed to dramatically degrade the probe connections' throughput from 1.66 Mb/s to 0.48 Mb/s with a small increment of the background traffic bandwidth. Thus, the genetic algorithm leads the background traffic sources to generating a traffic pattern that the TCP protocol cannot easily deal with.

Due to the fact that some genes may be discarded at the ends of the simulation, the bandwidth of a critical background traffic pattern generated by Nepal is slightly larger than the starting one. Thus, in order to eliminate this bias from the results, we defined a *disturbance efficacy* parameter at generation i as:

$$DE_i = \frac{T^* - T_i}{B_i}.$$

In this definition, T^* is the throughput of the TCP probe connections without the background noise traffic, T_i is the lowest throughput of the TCP probe connections archived until generation i and B_i is the corresponding background traffic bandwidth. In DE_i, the effects of the traffic are normalized with respect to the varying background traffic bandwidth B_i.

Figure 7 plots DE_i as a function of the GA generation i and clearly shows that the critical traffic pattern is identified by the genetic evolution, and cannot be easily found with standard statistical methods.

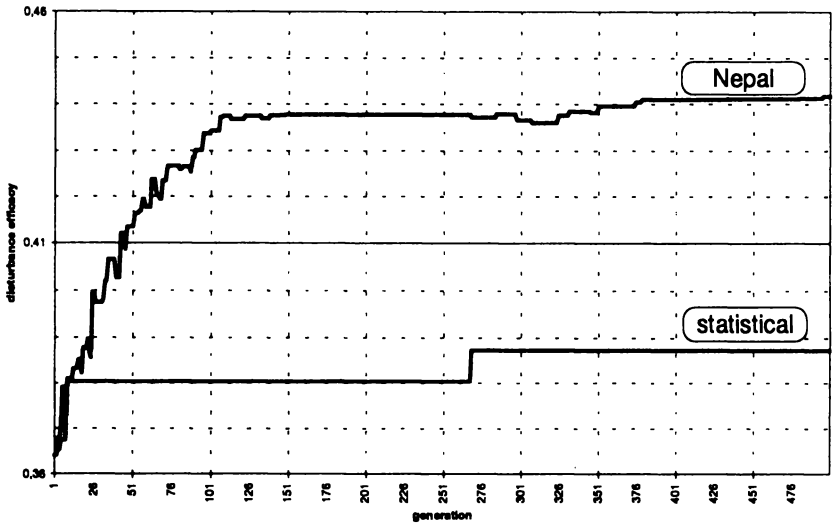

Figure 7 Disturbance efficacy

5 CONCLUSIONS

We presented a new heuristic methodology based on a Genetic Algorithm, for verifying the performance of a communication protocol in a realistic environment. We developed a prototypical implementation of the tool called Nepal and we ran some experiments, examining TCP connections in a simple IP network topology.

The results proved that, when the background traffic generation is driven by the genetic algorithm, the TCP performance is much lower than when traffic is generated by statistical methods: Nepal is able to identify protocol shortcomings that were not found by means of statistical simulation.

Moreover, Nepal is able to deal with real network topologies and with the real implementation of protocols, instead of a mathematical model. Thus, it can discover problems related to all the features of protocols, including those features that are impossible to handle with formal verification techniques.

We feel that our approximate methodology could be effectively used in conjunction with traditional methods of verification, giving useful information about the behaviour of a transmission protocol when used with a given network topology.

6 REFERENCES

E. Alba, J. M. Troya, "Genetic Algorithms for Protocol Validation", Proceedings of the 4th conference on Parallel Problem Solving (PPSN IV), Berlin, Germany, September 1996.

K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems'', *Doctoral Dissertion*, University of Michigan, 1975.

J. H. Holland, *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MC (USA), 1975.

J. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, 1990.

ISO - Information Processing Systems - Open Systems Interconnection - *LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807, 1988.

R. J. Linn, Jr., "The features and facilities of ESTELLE", in *Protocol Specification, Testing, and Verification*, V. M. Diaz, ed. New York: Elsevier, 1986, pp. 271-296.

B. A. Mah, "INSANE - An Internet Simulated ATM Networking Environment", available at http://HTTP.CS.Berkeley.EDU/~bmah/Software/Insane/

J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

J. Postel. User Datagram Protocol (UDP), RFC 768, August 1980.

J. Postel. Transmission Control Protocol (TCP), RFC 793, Internet Engineering Task Force, January 1981.

R. R. Razouk and C. V. Phelps, "Performance analysis using timed Petri nets", in *Protocol Specification, Testing, and Verification, IV*, Y. Yemini, R. Strom, S. Yemini, Eds, New York: Elsevier, 1985, pp. 561-576.

W. R. Stevens, *TCP/IP Illustrated: the protocols*, Reading, Mass., Addison-Wesley Pub. Co., 1994.

T. Suzuki, S. M. Shatz, and T. Murata, "A Protocol Modeling and Verification Approach Based on a Specification Language and Petri Nets", *IEEE Transactions on Software Engineering*, Vol. 16, No. 5, May 1990, pp. 523-536.

L. Wall, T. Christiansen, R. L. Schwartz, *Programming Perl, 2nd Edition*, O'Reilly, September 1996.

B. B. Welch, *Practical Programming in TCL and TK*, Prentice Hall Professional Technical Reference, May 1995.

7 BIOGRAPHY

Mario Baldi was born in Cuneo, Italy, on November 9, 1968. He received his M.S. in Electronic Engineering in 1993 from Politecnico di Torino where he is currently Ph.D. student. His research interests include protocols for high speed networks, real-time services over packet switched networks, synchronous packet switched networks, and active networks.

Fulvio Corno was born in Torino, Italy on April 26, 1967. He received the M.S. degree in electronic engineering from the Politecnico di Torino, Italy, in 1991 and the Ph.D. degree in Computer Science from the same institution in 1995. Since 1995 he is assistant professor at the Politecnico di Torino. His research interests include CAD for VLSI design, testing and design for testability of digital systems, high level design, genetic algorithms, and symbolic techniques.

Maurizio Rebaudengo was born in Asti, Italy on November 15, 1966. He received the M.S. degree in electronic engineering from the Politecnico di Torino, Italy, in 1991 and the Ph.D. degree from the same institution in 1995. His research interests include: sequential circuits testing, parallel and distributed algorithm for Electronic CAD, genetic algorithms and dependability analysis of computer-based systems.

Paolo Prinetto was born in Gassino Torinese, Italy, on March 17, 1953. He received his M.S. in Electronic Engineering in 1976 from the Politecnico di Torino, Italy. Since 1990 he is full professor of Computer Science at the same university. His research interests cover systems and tools for CAD for VLSI, with particular emphasis on testing, automated testable synthesis, hardware description languages, formal verification of correctness of digital designs. Since 1994, he is the chairman of ETTTC: the European Group of the IEEE- Computer Society *Test Technology Technical Committee* (TTTC).

Matteo Sonza Reorda was born in Ivrea, Italy, on November 30, 1961. He received the M.S. degree in Electronics and the Ph.D. degree in Computer Science from the Politecnico di Torino, Italy, in 1986 and 1990, respectively. Since 1990, he has been an Assistant Professor with the Department of Computer Science and Automation, Politecnico di Torino. His research interests include CAD for VLSI circuits with particular emphasis on testing and layout, evolutionary computation, and parallel algorithms.

Giovanni Squillero was born in Torino, Italy, on October 23, 1970. He received his M.S. in Computer Science Engineering in 1996 from the Politecnico di Torino, Italy. His main research interests are genetic algorithms and sequential circuits testing.