2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

NetSim-Steer: A Runtime Steering Framework for Network Simulators

Selim Ciraci
Pacific Northwest National Laboratory
Richland, WA, USA
selim.ciraci@pnnl.gov

Bora Akyol
Pacific Northwest National Laboratory
Richland, WA, USA
bora@pnnl.gov

***Abstract*—This paper presents NetSim-Steer, a runtime steering framework for network simulators. With this framework, users can specify execution constraints for the network protocols and network models. In addition to this, users can implement steering rules to be executed when a constraint is violated. These rules allow users to alter the parameters of the protocols/models during the simulation so that they stay within the boundaries of the constraints.**

***Keywords*-runtime steering, network simulator.**

I. Introduction

When testing a protocol or a network setup, it is important to see how it behaves in a certain scenario, such as the operation of the protocol under a heavy load. This test is usually realized as follows: user extends the network simulator (NS) with the new protocol, implements a network model that would follow the desired scenario, executes the network simulator, and analyzes the output of the simulation to verify if the model has actually followed the scenario. If the analysis shows that model has failed to follow the scenario or the new protocol is implemented incorrectly, the user alters the parameters and repeats the steps listed above. However, this is a time-consuming and error-prone task, since the simulation usually takes a long time [1] and generates a large output file. Users can greatly benefit if the verification and alteration of the parameters (when the verification fails) can be realized automatically during the simulation. This can be achieved by providing a *runtime steering* framework to the network simulators.

Runtime steering is the method of *monitoring* the execution of the software systems and altering it according to a set of user-defined *steering rules*. A steering rule consists of a set of *constraints* and *steering actions*. With the constraints, the desired behaviors of the software system are defined. At runtime, if these constraints are violated, the software system is said to be in an undesired state and the corresponding steering actions are executed. The steering actions change the running software system such that the constraints are satisfied again [2], [3].

This paper presents the NetSim-Steer framework that we developed to provide runtime steering for NSs. The monitors of NetSim-Steer collect information about the status of the network model. The user specifies constraints over the collected information with the steering rules that describe a set of alterations in the parameters of the network model. The constraints and the steering rules are expressed in Prolog as it provides the expressiveness to specify complex conditions related to the status of the network model (NetSim-Steer uses Swi-Prolog [4]). During the simulation, NetSim-Steer evaluates whether the network model (including the protocols) follow the specified constraints. If a constraint is violated, NetSim-Steer executes the associated steering rule, which in turn alters the network model. We integrated NetSim- Steer to the popular network simulator ns-3 [5].

II. Related Work

Runtime steering is mainly used for balancing distributed applications [2], providing fault monitoring [3], and adapting the application to different environments [6]. Compared to these approaches, NetSim-Steer is tuned for network simulators: it provides monitors for networking concepts rather than program-specific details. In addition, NetSim-Steer contributes to these approaches by providing the ability to detect conflicting steering rules.

Similar to NetSim-Steer, the VeriSim tool [7] also allows users to verify whether they implemented the (new) protocols correctly. Contrary to NetSim-Steer, this verification is realized after the simulation is completed. Hence, the approach does not provide any means for verifying the network model/protocols during the simulation and steering it.

III. The NetSim-Steer Framework

With the NetSim-Steer framework, users can verify whether their network models follow the desired scenarios or whether they correctly implemented the protocol to be tested during the simulation. In addition, the framework allows the network model to be steered; that is, the users can specify how the network model's parameters need to be changed in order to follow a scenario. When the verification fails, the framework executes these parameter changes.

The NetSim-Steer framework consists of two parts: the NS instrumentation and the evaluation. The NS instrumentation part, implemented in C++, provides the functionality for steering and monitoring the simulator at runtime. The evaluation part includes Prolog rules for implementing steering rules. The functions of the first part are bridged

1526-7539/12 $26.00 © 2012 IEEE
DOI 10.1109/MASCOTS.2012.64

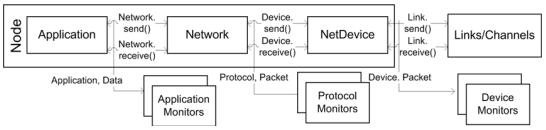

Figure 1. NetSim-Steer's integration to network simulator.

Table I
THE MONITORS AND THE STEERING ACTIONS SUPPORTED BY THE NETSIM-STEER FRAMEWORK

Layer	Monitors	Steering Actions
Simulator	Total # of processed events # of remaining events Events processed/second	Stop Simulator
Application	Total # of packets sent/received Bytes send/received per sec Inspect contents of the data	De/Increase data rate Delay Data Change data
Network	Bytes send/received per sec Destination port/address of packets # of Queued packets Inter packet sent/arrival times	Delay packet Change Destination/Port Change Queue size Reset congestion window
Link	Speed Destination hardware address Bytes send/received per sec # of packets sent/received	De/Increase speed Change destination Set Link up/down Drop packet

with the Prolog interpreter, so that they can be called from Prolog programs. In this way, the framework initialization, the constraints and the steering rules can be expressed in the same language. For steering-enabled simulations, the user provides a steering script in addition to the network model. The steering script is implemented in Prolog, and it consist of the following three parts:

Monitoring Specification: An NS provides a class library that is structured into modules, each representing a network layer. A module consists of classes in which the protocols and the applications related to that layer are implemented. To form a network model, these classes are instantiated with the desired parameters. Every class, belonging to a module corresponding to a network layer, implements a send and a receive method to transmit packets the classes of other modules. In NetSim-Steer, the status of the network model is monitored by intercepting the calls to these send and receive methods as shown in Figure 1. We implemented *monitors* that are attached to the objects belonging to a network layer at runtime, and they extract properties about related to the packet traffic; Table I lists the monitors in NetSim-Steer framework.

The user specifies to which objects the monitors are attached, in what we call the *monitoring specification*. In the steering script, the monitoring specification is implemented in the Prolog rule *monitor(Sim)*. When NetSim-Steer is initialized, it asks the Prolog interpreter to evaluate this rule to setup the monitors. For example, the inter-arrival time monitor observes the time of the calls to the method *receive* of an object belonging to the network devices module. To observe the inter-arrival time of a specific device during simulation, we need to attach the monitor to the object representing this device. This can be achieved with the following Prolog rule:

monitor(Sim):- getNetDeviceByIP(Sim, '10.1.1.2', NetDevice), addInterarrivialMonitor(NetDevice, 'interarrivalcond').

Here, the first predicate *getNetDeviceByIp* returns (unifies) a reference to the instance of the network device that is assigned the IP address $10.1.1.2$ in the third parameter. The second predicate attaches the inter-arrival time monitor to this object. The second parameter of this predicate sets the *constraint* function for the inter-arrival time.

Constraints: Each monitor is assigned a constraint that specifies the desired values for the monitored property. The constraints are also expressed as Prolog rules. For example, the following constraint states that the inter-arrival time of packets to the network device with IP $10.1.1.2$ should be greater than $0.4s$ when simulated time is greater than $1s$:

interarrivalcond(CurTime,InterArrivialTime):- (CurTime<1, true); (CurTime>=1, InterArrivialTime>0.4.

Each constraint rule takes two parameters: the first is the current simulated time and the second is the value corresponding to the monitored property. This second parameter is dependent on the monitor, and NetSim-Steer framework includes Prolog rules for testing the type of this value. When a monitor records a change in the property it is observing, it asks the Prolog interpreter to evaluate associated constraint.

Steering Rules: In a steering rule, an undesired state of the network model or the protocol with the steering actions to recover from this state is specified. Remember that the constraints specify the desired values for the monitored properties. If a constraint (or a set of constraints) is violated, the model or the protocol is not behaving correctly. When the constraint of a monitor fails, the monitor asserts (i.e, inserts into the Prolog database) the Prolog fact *failed (Cons, Time, Ref)*. Here, *Cons* is the name of the constraint that failed, *Time* is the simulated time when the failure has occurred and *Ref* is the reference to the monitored object. An undesired state of the network model or the protocol is expressed as a query for these *failed* facts. Assume that we monitor the bytes/s from network device *NetDevice2*, an instance of a class from the network device module, in addition to the inter-arrival time monitor attached to the network device *NetDevice*. Then, we can state a steering rule that changes the link speed between these two network devices when the constraints on both of these monitors fail at the same time as follows:

steerLink(CurTime):- failed('interarrivialcond', CurTime, NetDevice), failed('bytescond', CurTime, NetDevice2), link(NetDevice, NetDevice2, Link), setSpeed(10000,Link).

Here, *CurTime* is the current simulated time and *bytescond* is the constraint assigned to the monitor for *NetDevice2*. The first two predicates queries whether the two monitors have asserted the *failed* facts within the same simulated time and, if so, the references to the two network devices that violated the constraint is returned. The last two predicates are used for setting the speed of the link between NetDevice and NetDevice2.

NetSim-Steer asks the Prolog interpreter to evaluate the steering rules whenever a constraint is violated. Note that it is possible to express an ordering between the *failed* predicates according to the simulated time in Prolog,

For each network layer module, we have defined a set of steering actions that allows the users to modify the parameters related to the protocols/applications of that layer (These are listed in Table I). We implemented each steering action as a Prolog rule, in which the last predicate signals the desired parameter change to NetSim-Steer. NetSim-Steer buffers the signals and executes them when the simulator is ready to move to the next event.

Executing steering actions such as set a link speed that is more than the initial speed, reducing the queue size when there are packets already queued, or setting a link down when there packets already scheduled to be received can result in an inconsistent network model. We implemented Prolog rules to detect actions that would lead to these inconsistencies. When detected, NetSim-Steer either rejects the steering action or tries circumvent the inconsistency (for example, by canceling the scheduled receive events).

IV. Application of NetSim-Steer

We applied NetSim-Steer in the simulation project of an application layer protocol for controlling smart grid [8]. In this application the following benefits are observed:
i) We wrote a script and automatically verified the grid topology that should be followed by all the network models of the project. Without NetSim-Steer, we would need to manually test all the IP address assignments and application setup.
ii) The application layer protocol tested in this project has a constraint which states that a node can send a packet after it receives 5 packets or 30 seconds after the arrival of the initial packet. We expressed this constraint in a steering script using two monitors that observe the packets sent by the protocol running at a node. In the steering rule, we only used the steering action that stops the simulator. When the simulation with this steering program was executed, NetSim-Steer detected a violation of this constraints and stopped the simulation. With the feedback provided by NetSim-Steer, we located the problem without waiting the simulation to finish.

Benchmarks with a 50 node simulation shows that NetSim-Steer increased the simulation time by 100miliseconds.

V. Conclusion

In network simulation projects, users need to ensure that the network model and the protocol they are testing are implemented correctly. This involves repeated simulations and interpreting long output files. We implemented NetSim-Steer for computer-aided verification of network models and the protocols. In addition to verification, NetSim-Steer allows the simulator to be steered, where the users can change the parameters of the network model and the protocol during the simulation in case the verification fails. To best of our knowledge, NetSim-Steer is the only framework that allows users to express constraints related to networking concepts, provides verification during the simulation and addresses the problem of repeated simulation with the capability of steering.

References

[1] E. Weingartner, H. vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *ICC '09*, june 2009, pp. 1 –5.

[2] W. Gu, G. Eisenhauer, K. Schwan, and J. Vetter, "Falcon: Online monitoring for steering parallel programs," *Concurrency: Practice and Experience*, vol. 10, no. 9, pp. 699–736, 1998.

[3] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Trans. Softw. Eng.*, vol. 30, pp. 859–872, December 2004.

[4] "Swi-prolog http://www.swi-prolog.org/," online, June 2011.

[5] "The ns3 network simulator http://www.nsnam.org/," online, June 2011.

[6] L. Lin and M. D. Ernst, "Improving the adaptability of multi-mode systems via program steering," *SIGSOFT Softw. Eng. Notes*, vol. 29, pp. 206–216, July 2004.

[7] K. Bhargavan and et al., "Verisim: formal analysis of network simulations," *Software Engineering, IEEE Transactions on*, vol. 28, no. 2, pp. 129 –145, feb 2002.

[8] P. Huang and et al., "Analytics and transactive control design for the pacific northwest smart grid demonstration project," in *SmartGridComm'10*, oct. 2010, pp. 449 –454.