Simulation Modelling Practice and Theory 19 (2011) 2063–2075

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

# Fast prototyping of network protocols through ns-3 simulation model reuse ☆

Gustavo Carneiro *, Helder Fontes, Manuel Ricardo

*INESC Porto, Faculdade de Engenharia, Universidade do Porto, Portugal*

ARTICLE INFO

*Article history:*
Received 7 January 2011
Received in revised form 11 May 2011
Accepted 11 June 2011
Available online 11 July 2011

*Keywords:*
Network protocol
Simulation
Implementation
Framework
ns-3

ABSTRACT

In the networking research and development field, one recurring problem faced is the duplication of effort to write first simulation and then implementation code. We posit an alternative development process that takes advantage of the built in network emulation features of Network Simulator 3 (ns-3) and allows developers to share most code between simulation and implementation of a protocol. Tests show that ns-3 can handle a data plane processing large packets, but has difficulties with small packets. When using ns-3 for implementing the control plane of a protocol, we found that ns-3 can even outperform a dedicated implementation.



## 1. Introduction

Research and development today has to keep up with a fast evolving field of research, and communications protocols research is no exception. Considerable time is spent by researchers and developers from the idea of a protocol that solves a specific problem and the deployment of an implementation of that protocol. One of the main contributors for the time spent is the need to develop both a simulation model and then an implementation of the same protocol. What if we could develop a single hybrid simulation/implementation model? In this paper we explore the possibility of doing just that: using the Network Simulator 3 (ns-3) as a framework for developing new protocols that can be first simulated and then deployed in a real communications device with only minimal changes. Such an approach appears to be interesting, at the surface, since it will save development time, among other advantages. However, questions regarding real world performance remain, and one of the objectives of this paper is to discover performance limits of the proposed framework.

This paper includes three main contributions. First, the proposal of a new unified simulation/implementation protocol development process that takes advantage of the existing ns-3 network emulation functionality. We evaluate the packet processing performance, in terms of achievable throughput, packet loss, and round-trip time, of ns-3 working in emulation mode, compared to a pure kernelspace IPv4 forwarding. We propose additional ns-3 classes that significantly improve the emulation of control plane protocols, and simplify the deployment of such protocols.

The remainder of the paper is organized as follows. Section 2 presents an overview of a traditional protocol development process, associated problems, and proposes a new model that uses ns-3 as its basis. Section 3 includes some related work. Section 4 describes the ns-3 emulation framework and evaluates its real world performance. Section 5 again evaluates the ns-3 emulation performance, this time restricted to a scenario where ns-3 only emulates the control plane and not data

☆ This work was funded by the Portuguese government through FCT grants SFRH/BD/23456 /2005 and SFRH/BD/69051/2010.
* Corresponding author. Tel.: +351 222 094 000.
*E-mail address:* gjc@inescporto.pt (G. Carneiro).

1569-190X/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2011.06.002

plane, and proposes an optimization to ns-3 to deal with this case more effectively. Finally, Section 6 draws the major conclusions of the work presented in this paper, and suggests possible future research.

## 2. Protocol development process

### 2.1. Traditional protocol development process

In general, researching and developing a new networking protocol involves several steps. These steps are depicted in Fig. 1:

1. We begin with a notion of a problem the protocol would solve in a given scenario. From the scenario/problem the researcher develops both a simulation model for the protocol and a simulation program that uses the protocol model.
2. Multiple simulations are run, with varying parameters.
3. The results are analyzed. As a result, the protocol model may be debugged, tweaked, or improved. The scenario simulation description may also need to be changed. As a result, we may need many iterations of simulation/analysis/improve.
4. When the simulation results are acceptable, it is time to stabilize the protocol model and write down the specification in a document.
5. From the specification, a protocol implementation is developed, possibly by a different team.
6. A testbed may be used to run the implemented protocol; trial runs may generate logs to be analyzed.
7. The logs are analyzed and if they report widely different results from what was expected from simulations then: (a) implementation is buggy, (b) the trial run exercises a slightly different scenario which has great impact on the protocol performance. In case (a), the implementation simply needs to be debugged, but in case (b) the protocol model may need to be changed, new simulations run, and then the respective implementation changed accordingly, not forgetting the protocol written specification.

The development process has problems that become apparent when focusing on steps 5 and 7. First, there is a lot of duplicated effort when developing both a simulation model and an implementation prototype. Second, when modifications to the protocol are needed, after trial runs, they have to be done in three places at once: simulation model, protocol specification document, and implementation. The risk of inconsistencies between them being accidentally introduced is non-negligible.

One has to wonder what is so different between simulation model and implementation to warrant duplicated code. A network protocol can generally be described as a *Timed Automata* [1], i.e. a state machine in which state transitions are triggered by input messages and constrained by the passage of real time. Both simulation and implementation of the same protocol include the very same automata, only the way messages are received and transmitted, as well as the way time passage is measured, is different between the two. A protocol implementation (e.g. a routing agent) usually has an *event loop*, which is an infinite loop that waits for data to arrive on one or multiple sockets, decodes the data to extract the PDUs, and processes the PDUs according to the protocol. As a result of the processing, new PDUs may need to be transmitted, at which point they are encoded as data and written to one or more sockets. Time based transitions are typically implemented using a system call to suspend the process for the required time, thus saving CPU cycles. For example, in UNIX systems it is frequent to see protocol implementations to use a *select* or *poll* based main loop, allowing them to wait for a certain elapsed time with the process suspended, but at the same time be notified when new data has arrived at a socket. In a simulation environment, on the other hand, time is virtual, represented by a *virtual clock*, which is a simple numeric counter. Passing time in a simulator is represented by an event and does not require the process to be suspended during that time, only to increment the virtual clock by a certain amount. Events are also used to represent the reception of data from a network interface.

In the simulator, the *event scheduler* is essentially an infinite main loop that processes pending events in order. It is very similar to the *select*-based event loop in a protocol implementation. The differences are that (1) in the simulator, elapsed

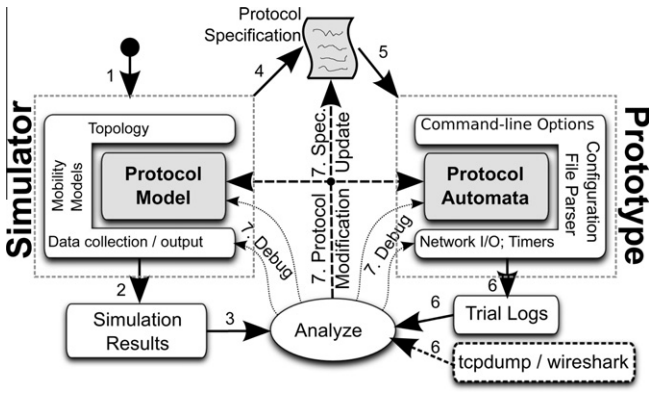


**Fig. 1.** Traditional protocol development process.