Per node clocks to simulate time desynchronization in networks

Matthieu Coudron, Stefano Secci
Sorbonne Universites, UPMC Univ Paris 06,
UMR 7606, LIP6, Paris, France
Email: firstname.lastname@upmc.fr

ABSTRACT

Time is a critical parameter in many protocols: depending on the synchronicity between nodes, protocols may disable themselves or underperform. Nodes in the discrete network simulator ns3 retrieve time from the same simulator clock. As such, they can be considered as perfectly synchronized with regards to the simulator precision. while ns3 provides a maximum precision of a femtosecond with a perfect frequency, i.e. timesteps are comparable, real world clocks tend to drift away from a reference clock due to various reasons (heat, deficient oscillator...), resulting in an offset of a few milliseconds or seconds per day. For synchronization-dependant protocols, this offset can be large enough to take into consideration during the simulation. Hence we present an architecture that implements per-node clock and allows to model node desynchronization.

Keywords

ns3, time synchronization, ntp, dce

1. INTRODUCTION

Different types of synchronization exist such as frequency synchronization, and time synchronization. Frequency synchronization is called syntonization, i.e. the duration of a second is the same on both clocks. Syntonization is required for wireless transmissions such as GSM or LTE [reference needed]. Time synchronization means that clocks share an absolute time reference called the epoch, and is needed to measure the quality of service in networks.

Real world clocks tend to exhibit a drift of a few milliseconds/seconds per day with respect to the TAI (International Atomic Time), which if not corrected, accumulate overtime. In a first part, we present an implementation [1] that can simulate imperfect frequency and/or time synchronization. In the second part, we introduce a possible usage of this feature to simulate time distribution protocols.

2. ARCHITECTURE

PLDI '13 June 16–19, 2013, Seattle, WA, USA

DOI: 10.1145/1235

Ns3 is a discrete time event-based simulator, i.e., first the simulator selects the next available event, then it updates its internal counter (i.e., clock) to the scheduled time of the selected event. Compared with realtime simulators which increase their counter with a constant step, this approach allows to skip periods of time where nothing happens to reduce the simulation duration. It also provides - per construction - frequency and time synchronization. Our objectives are thus:

1. to allow clocks to run with different rates and/or offsets...
2. with the possibility to alter the previous parameters dynamically during the simulation, to simulate clock drift

Currently, events are searched in a single datastructure, in which they are added via the use of *Simulator::Now()*, regardless of the originating node.

From a logical standpoint, it makes sense to schedule events in a member of the node so that the scheduling can adapt to traits of its node, such as the clock frequency or offset. Thus we introduce a *Node::Schedule()* member which should replace most calls to *Simulator::Schedule()*. The node thus tracks its own list of events and is in charge of pushing its events to the global list of events when required. The node maintains at all times its next upcoming event in the global scheduler, i.e., as soon as this event is cancelled or triggered, the node pushes the succeeding event in the global simulator as can be seen on figures 1 and 3.

The last feature consists in the ability to change the parameters of the clock, e.g., its frequency and its offset relative to the simulator settings. Changing a parameter means that the previous projected simulator firing time becomes incorrect: as explained previously, every node maintains a list of events and schedules only the next one in the simulator. Thus the node has to convert the local expected time to trigger the event into the predicted simulator time. The node does so using its current parameters, e.g., if its clock frequency is set to twice the simulator's, 2 local seconds translate to 1 simulator second. Any clock parameter change invalidates the previous prediction. In such case, the node cancels the event registered in the simulator, and reschedules it with an updated prediction of the simulator time.

3. USE CASE: RUNNING A TIME DISTRIBUTION PROTOCOL IN NS3

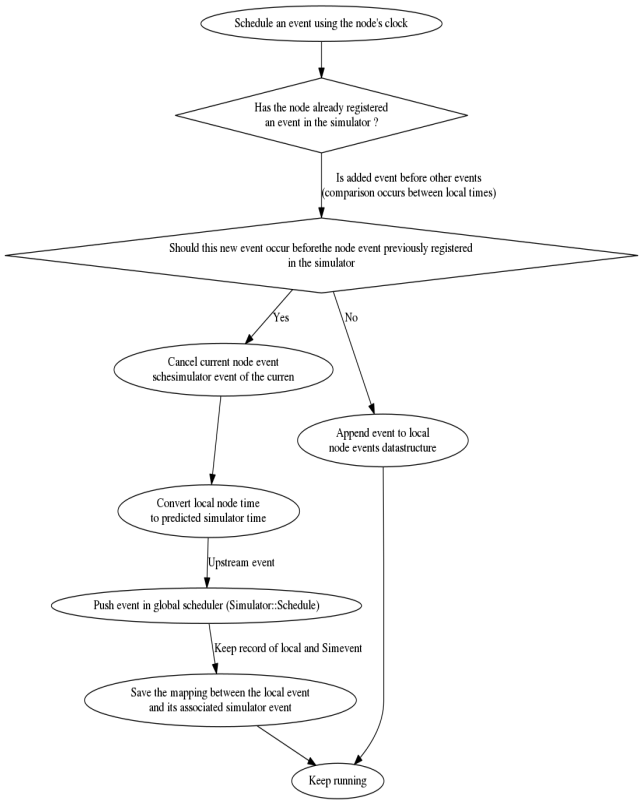

Figure 1: Process of adding an event

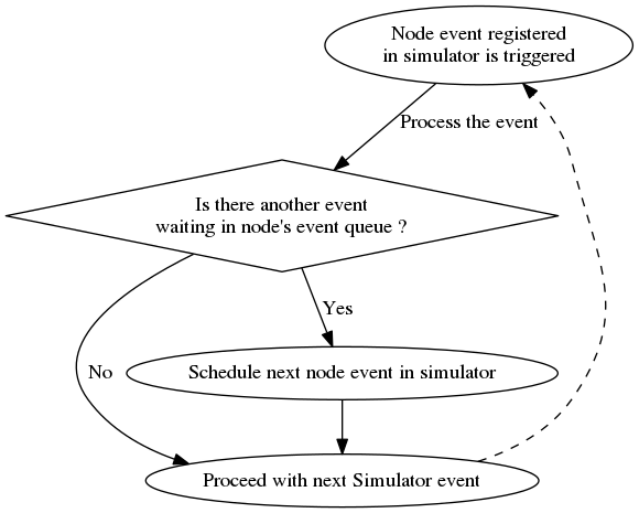

Figure 2: Upon an event firing

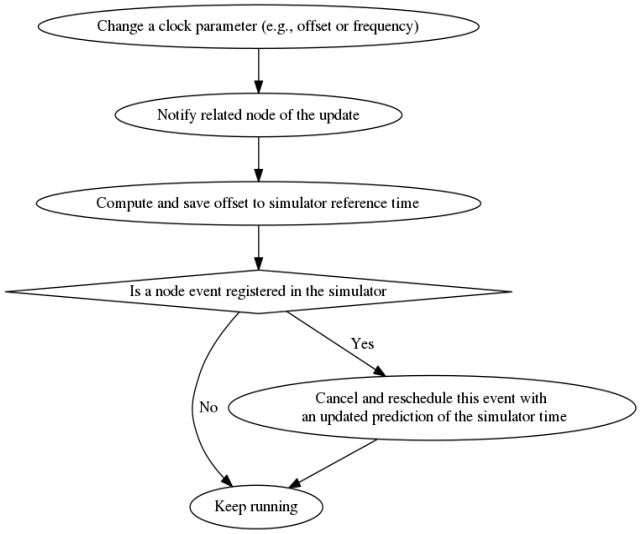

Figure 3: Update of a clock parameter

Several use cases come to mind such as modeling sensor networks that cycle through on/off patterns depending on their clocks. Time distribution protocols also look like an interesting subject of study. Indeed NTP simulations should be able to run tests faster than wallclock, since the typical NTP poll interval is superior to one minute. We present in this section an architecture aimed at running the Network Time Protocol (NTP) reference implementation in ns3 along with linux kernels.

The dissemination of clock synchronization information in packet-switching networks can be achieved through different protocols, with different precision levels. The Network Time Protocol (NTP) [3] is a widely adopted solution where millisecond accuracy is enough; NTP relies on a hierarchy of clocks, the higher the stratum, the lower the precision.

The host retrieves regularly from the network a time reference and steers the local clock (through frequency correction) depending on the offset with this time reference. Thus NTP consists in a userspace daemon to contact remote NTP servers, and a kernel counterpart to correct the clock speed. Considering the complexity of both the linux and the userspace NTP implementations, a sensible course of action seem to rather work on being able to load these implementations rather then writing a specific ns3 model.

Direct-Code Execution (DCE) [5] sounds like the perfect candidate to produce such an environment: DCE is an ns3 extension that allows to load unmodified (though compiled with specific options) applications within the ns3 environment. It is even possible to load a modified linux kernel [2]. We fixed DCE netlink module in order to load the reference NTP implementation [4]. Linux maintains a counter called **jiffies** that is governed by two entities, a *clocksource* realtime clock in charge of keeping the time reference, i.e., epoch and by a *clockevent* device which generate interrupts that update the jiffies counter on a regular basis, typically one jiffie every 10ms. Vanilla DCE bypasses those previous mechanisms to keep the linux jiffies counter perfectly synchronized with the ns3 clock, which prevents any skew to propagate to the kernel. Our proposition is to let the linux

jiffies counter be updated by the conventional means so that NTP adjustments can also affect the jiffie counter via calls to the kernel *xtime_update* function, function in charge of applying the NTP correction.

4. CONCLUSIONS

In this document, we presented an implementation capable to run per-node clocks, and capable of changing the clock parameters dynamically, which broadens the scope of simulations supported by ns3. We also present how this could affect time distribution protocol testing in a realistic simulation running close to unmodified programs.

5. ACKNOWLEDGMENTS

6. REFERENCES

[1] M. Coudron. ns3 implementation with clock support.

[2] T. Hajime, R. Nakamura, and Y. Sekiya. Library Operating System with Mainline Linux Network Stack. In *netdev0.1*, 2015.

[3] D. Mills, U. Delaware, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. *RFC5905*, pages 1–110, 2010.

[4] D. L. Mills. NTP Architecture , Protocol and Algorithms. pages 1–26.

[5] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous. Direct Code Execution : Revisiting Library OS Architecture for Reproducible Network Experiments. *COnference on Emerging networking experiments and technologies (CONext)*, pages 217–228, 2013.