Chapter 15
Related Helper Classes

Helper classes generally not a part of a network, but are used in NS2 as an internal mechanism and/or are network components in a special case. This chapter discusses the details of three main NS2 helper classes. In Sect. 15.1, we first discuss class Timer, which is widely used by other NS2 modules such as TCP to implement conditional time-based actions. In Sect. 15.2, we demonstrate a random number generation process in NS2. In Sect. 15.3, we explain the details of class ErrorModel, which can be used to simulate packet error. Section 15.4 discusses bit masking and bit shifting operations used in various places in NS2. Finally, the chapter summary is given in Sect. 15.5.

15.1 Timers

Timer is a component that can be used to delay actions. Unless cancelled or restarted, a timer takes actions after it has been started for a given period of time (i.e., at the expiration). For example, a sender starts a retransmission timer as soon as it transmits a packet. Unless cancelled by a reception of an acknowledgment packet, the timer assumes packet loss and asks the sender to retransmit the lost packet at the timer expiration.

15.1.1 Implementation Concept of Timer in NS2

As shown in Fig. 15.1, a timer consists of four following states: IDLE, SET WAITING TIME, WAITING, and EXPIRED. A transition from one state to another occurs immediately when the operation in the current state is complete (i.e., by default), or when the timer receives a start message, a restart message, or a cancel message.
When a timer is created, it sets the state to be \textit{IDLE}. Upon receiving a start message, the timer moves to the state \textit{SET WAITING TIME}, where it sets its waiting time to be “\textit{delay}” seconds and moves to the state \textit{WAITING}. The timer stays in the state \textit{WAITING} for “\textit{delay}” seconds and moves to the state \textit{EXPIRED}. At this point, the timer takes predefined expiration actions and moves back to the state \textit{IDLE}. Hereafter, we will say that the timer \textit{expires} as soon as it enters the state \textit{EXPIRED}. Also, we shall refer to the actions taken in state \textit{EXPIRED} as \textit{expiration actions}.

The above timer life cycle occurs by default when the message “start” is received. When a “cancel” messages is received, the timer will stop waiting and move back to the state \textit{IDLE}. If a restart message is received, the timer will restart the waiting process in the state \textit{SET WAITING TIME}.

Implementation of timer in NS2 is a very good example of the \textit{inheritance} concept in OOP. Each timer needs to implement the three following mechanisms: (1) waiting mechanism, (2) interface functions to start, restart, and cancel the waiting process, and (3) expiration actions. The first two mechanisms are common to all timers; however, the last mechanism (i.e., expiration actions) is what differentiates one timer from another. From an OOP point of view, the timer base class must define the waiting mechanism and message receiving interfaces, and leave the implementation of the expiration actions to the derived classes.

In NS2, timers are implemented in both C++ and OTcl. However, both C++ and OTcl timer classes are standalone (i.e., not bound together by TclClass). Relevant functions and variables in both domains are shown in Table 15.1. In both domains, NS2 implements the waiting process by utilizing the Scheduler. Upon entering the state \textit{SET WAITING TIME}, NS2 places a timer expiration event on the simulation timeline. When the Scheduler fires the expiration event, the timer enters the state \textit{EXPIRED} and executes the expiration actions.