FluidSim: A Tool to Simulate Fluid Models of High-Speed Networks

José Incera [a,1], Raymond Marie [b], David Ros [b] and Gerardo Rubino [b]

[a] *ENST Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson-Sévigné Cedex, France* [2]

[b] *IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France* [3]

Abstract

In this paper we present a tool for the simulation of fluid models of high-speed telecommunication networks. The aim of such a simulator is to evaluate measures which can not be obtained through standard tools in reasonable time or through analytical approaches. We follow an event-driven approach in which events are associated with rate changes in fluid flows. We show that, under some loose restrictions on the sources, this suffices to efficiently simulate the evolution in time of fairly complex models. Some examples illustrate the utilization of this approach and the gain that can be observed over standard simulation tools.

Key words: Discrete-event simulation, Fluid simulation, Fluid traffic models, Performance evaluation, Network simulation

1 Introduction

Computer networks continue to evolve, increasing in size and complexity. When we have to analyze some aspect of the behavior of an existing network or to design a new one, the most widely used tool is simulation, both because of its power in representing virtually every possible mechanism and system, and because of its flexibility. The main price to pay is in programming and computation costs: simulation programs are usually difficult to develop; besides, they may need large amounts of resources in time and sometimes also in space. The best alternative to simulation is to use analytical techniques. In general, they offer the advantages of being several orders of magnitude less expensive to apply and, moreover, of frequently

[1] Partially financed by CONACyT, México and by CROUS, France.

[2] E-mail: Jose.Incera@enst-bretagne.fr

[3] E-mail: {marie, dros, rubino}@irisa.fr

leading to a deeper insight into the properties of the analyzed system. The drawback of analytical methods is that they require conditions on the models that are often hard to satisfy. A third possibility is to use numerical techniques, which usually range between simulation and analytical methods, in terms of cost and required assumptions.

In this paper we are interested in the analysis by simulation of high-speed communication networks, in order to quantify important aspects of their behavior. These aspects include performance, dependability and quality of service properties. Communication networks are examples of very complex systems, where simulation is the only tool able to analyze in some detail the associated mechanisms. More specifically, we are interested in the behavior of communication networks where information is sent in discrete units, with complex scheduling mechanisms and interactions between different components of the system, and using realistic models of sources. Our references are ATM networks transporting *cells* and IP networks where the information travels in *packets* (see, for instance, [25]). The classical approach to simulate such a system is to follow the event-driven approach where each message (cell, packet) is represented in the model, together with its evolution through the different nodes [7]. If we consider now a high-speed network, we easily see the problem that may arise when millions or billions of units must be generated and moved through the simulated model. For instance, to validate a design in the ATM area, where the engineer must check that the loss probability of a flow arriving at a specific switch is of the order of 10^{-9}, at least hundreds of billions of cells should be sent to the switch in order to obtain a minimal accuracy on the result.

To deal with this problem, a possible approach is to try more sophisticated simulation techniques that can lead to the same precision with less computational effort (importance sampling, splitting, etc.; see for instance [6], or a more general reference like [4]). The drawback here is similar to that of analytical methods, though less restrictive: the applicability conditions can be too hard to fit. For instance, some techniques of this kind only concern Markov models, whereas others work only for single queues of some particular class.

The approach chosen in this paper is the simulation of continuous-state models (fluid models), where the flow of discrete units traveling through the links, and stored in the buffers, is replaced by fluids going from one container to another. This can lead to a significant reduction in the computational effort. Indeed, when a long burst of cells or packets is sent through a link (which happens quite often), instead of handling each individual unit as with a classical simulator, it suffices here to manage only two events: the beginning of the burst and its end [4] . Our paper describes a tool designed to simulate such a fluid model and to take advantage of this potential computational effort gain.

[4] In fact, this is strictly true only for the sources; nevertheless, it illustrates the key phenomenon.

It must be observed that the use of continuous-state models as approximations of discrete time ones is not new: this has been already done in queuing theory for many years. Think simply of the diffusion processes used as approximations of standard queues or of queuing networks in heavy-traffic conditions [13]. Fluid models representing high-speed communication networks draw from the classical paper by Anick et al., which in turn derives from the work of Kosten and others (see [1] and references therein). Such models are commonly used nowadays, specially in the ATM area, but mainly for analytical purposes. It must be underlined that just a few models can be analyzed this way: mainly single node systems with very simple scheduling mechanisms and strong assumptions on the behavior of the sources. To the best of our knowledge, there are almost no results on multiclass models and/or on models with more than one node.

It appears that just a few published works report on simulators built on the fluid framework. For instance, in [18,19] the authors describe a cell-rate based tool, LINKSIM, useful mainly to assess the fluid simulation method. The goal here is to evaluate the mean cell loss rate in a single ATM buffer. In [12], a simulator is presented, again in the ATM context, used to study the interest of the fluid paradigm in the case of specific sources, switches and scheduling mechanisms. Some results on the comparison between the fluid approach and a discrete-event, "cell-level" simulator are also discussed. In [15], the authors report on a simulator dedicated to evaluate an original resource management algorithm for a multiport, single shared buffer switching node.

Another simulation program is described in [27]. It proposes a time-based simulator, limited to work-conserving feed-forward networks. The metric of interest is the backlogged workload at the servers. The authors show that, for the particular case of single-class fluids, the error introduced by using a discrete-time variable is proportional to the sampling interval. An interesting point here is that their method is easy to parallelize. In [17], the authors present a comparative study of the simulation of fluid models. To this end, they built a set of fluid-based and packet-based models of some components of communication systems within a common simulation framework, the Scalable Simulation Framework. Their goal was to compare in a fair way the accuracy and performance of the fluid paradigm by sharing as much common code as possible between the fluid-based and the packet-based modules. The study focuses on Markov Modulated Process sources with Leaky Bucket shaping and single node topologies and hypercubes of degree 2 to 8.

In this paper, we introduce FluidSim, a discrete-event simulation tool based on fluid models of computer network objects. The aim of this simulator is to evaluate measures which cannot be obtained through standard tools in reasonable time or through analytical approaches. We show that, under some loose restrictions on the sources, this suffices to efficiently simulate the evolution in time of fairly complex models.

The structure of the paper is as follows. Section 2 gives a short description of the dynamics of fluid models, specializing the general relationships to the specific family of sources we are interested in. Section 3 introduces the class of fluid models under consideration. Section 4 discusses some issues related to the simulation of such a system following an event-driven approach. In Section 5, the current state of our tool, FluidSim, is presented and Section 6 illustrates its use by means of three examples. Some conclusions and current research directions close the paper in Section 7.

2 Dynamics of a Fluid Model

Let us consider first a single fluid buffer or "reservoir" of capacity $B \leq \infty$, with a constant output rate $c \in (0,\infty)$, and a work-conserving FIFO service discipline. Let $\Lambda(t) \in [0,\infty)$ be the total rate of fluid being fed into the buffer at time $t \geq 0$. We are interested only in arrival processes in which every sample path $\Lambda(t)$ is a (right-continuous) stepwise function. This condition is not very restrictive since many fluid traffic models satisfy it. Some examples are: on/off rate processes, either Markovian [1] or non-Markovian [20]; Markov-modulated rate processes, for which $\Lambda(t) = \zeta(Z(t))$, where $Z(t)$ is the state of a Markov chain at time t and $\zeta()$ is a given function [20]; renewal rate processes $\Lambda(t) = \sum_{n=0}^{\infty} X_n \mathbb{1}(t \in [S_{n-1}, S_n))$, where the X_i's form a sequence of i.i.d. random variables and the $S_n - S_{n-1}$ form a renewal process independent of the X_i's [10,20]. The volume of fluid arriving in the interval $[0,t]$ is given by: $A(t) = \int_0^t \Lambda(u)\,du$ and is continuous and piecewise linear.

Let $Q(t)$ be the volume ("level") of fluid in the buffer at time $t \geq 0$. The evolution of $Q(t)$ is described by:

$$Q(t) = Q(0) + \int_0^t (\Lambda(s) - c)\, \mathbb{1}(s \in \mathcal{Q})\, ds, \quad t \geq 0, \tag{1}$$

where, for an infinite-capacity buffer, the set $\mathcal{Q}$ is given by [20, Chapter 17]:

$$\mathcal{Q} = \{ s \geq 0 \,|\, \Lambda(s) > c \text{ or } Q(s) > 0 \},$$

and for a finite-capacity buffer,

$$\mathcal{Q} = \{ t \geq 0 \,|\, (\Lambda(t) > c \text{ or } Q(t) > 0) \text{ and } (\Lambda(t) < c \text{ or } Q(t) < B) \}$$

For right-continuous, stepwise input rate functions, this integral reduces to:

$$Q(T_{n+1}) = \min \left\{ B, \left(Q(T_n) + (\Lambda(T_n) - c)(T_{n+1} - T_n) \right)^+ \right\}, \tag{2}$$

where T_n denotes the n-th transition epoch of $\Lambda(t)$; we take $T_0 = 0$. The resulting sample paths $Q(t)$ are piecewise linear, with slope $\dot{Q}(t) = (\Lambda(t) - c)\, \mathbb{1}(t \in \mathcal{Q})$.

Slope changes occur either at the time instants where the buffer becomes full or empty, or at the transition epochs T_n of $\Lambda(t)$. If a finite buffer is full at time s and $\Lambda(s) > c$, *some* of the arriving fluid will be lost. Cumulative fluid losses in the interval $[0,t]$ may be computed by $L(t) = \int_0^t (\Lambda(s) - c)\, \mathbb{1}(s \in O)\, ds$, where O is the set of all overflow periods:

$$O = \big\{ s \geq 0 \,|\, \Lambda(s) > c \text{ and } Q(s) = B \big\}.$$

For a given t, we denote by $t_0(t)$ the beginning of the *next* empty period; likewise, $t_B(t)$ denotes the beginning of the *next* overflow period.

Let $R(t)$ be the output rate of the buffer at $t \geq 0$. $R(t)$ is defined by:

$$R(t) = c\mathbb{1}(Q(t) > 0 \text{ or } \Lambda(t) > c) + \Lambda(t)\mathbb{1}(Q(t) = 0 \text{ and } \Lambda(t) \leq c)\,. \tag{3}$$

For the class of processes Λ considered in this work, we can deduce then that $R(t)$ is also a right-continuous stepwise function. The volume of fluid flowing out of the buffer in $[0,t]$ is: $D(t) = \int_0^t R(u)\, du$.

The model described so far can be applied to the more general case where the buffer is fed by N fluid flows. Let us denote by $\lambda_i(t) \in [0,\infty)$ the rate of the i-th flow at time t. We denote $\boldsymbol{\lambda}(t) = \big(\lambda_1(t), \ldots, \lambda_N(t)\big)$, and we call this the *input flow vector*. The *total* input rate is $\Lambda(t) = \sum \lambda_i(t)$. As before, we shall be interested only in arrival processes such that, for every sample path, $\lambda_i(t)$ is a (right-continuous) stepwise function. It results that $\Lambda(t)$ is also a (right-continuous) stepwise function. In the same way, $r_i(t)$ is the output rate related to the i-th input fluid, at time t, $\mathbf{r}(t) = \big(r_1(t), \ldots, r_N(t)\big)$ is the *output flow vector*, and $R(t) = \sum r_i(t)$ is the *total* output rate.

Let us denote by τ_n the n-th transition epoch of the input flow vector $\boldsymbol{\lambda}(t)$. Because of the FIFO service discipline, a change in $\boldsymbol{\lambda}(t)$ occurring at $t = \tau_n$ will need $Q(\tau_n)/c \geq 0$ time units to propagate to the buffer output (the time needed to flow out the $Q(\tau_n) \geq 0$ volume units already in the buffer). Then, at time

$$\omega_n = \tau_n + Q(\tau_n)/c, \tag{4}$$

the proportion of output components must be the same as the proportion of input components; that is, if $\Lambda(\tau_n) > 0$, then for every flow i, $r_i(\omega_n)/R(\omega_n) = \lambda_i(\tau_n)/\Lambda(\tau_n)$. Note also that a change in $\boldsymbol{\lambda}(t)$ may produce *two* transitions in $\mathbf{r}(t)$ if $Q(\tau_n) > 0$, $\Lambda(\tau_n) < c$ and the buffer becomes empty before the next transition in $\boldsymbol{\lambda}(t)$. The evolution of $\mathbf{r}(t)$ is fully described by:

$$\mathbf{r}(\omega_n) = \begin{cases} \dfrac{R(\omega_n)}{\Lambda(\tau_n)} \cdot \boldsymbol{\lambda}(\tau_n) & \text{if } \Lambda(\tau_n) > 0, \\ \mathbf{0} & \text{if } \Lambda(\tau_n) = 0 \text{ and } Q(\tau_n) = 0 \text{ (in this case, } \omega_n = \tau_n), \end{cases} \tag{5}$$

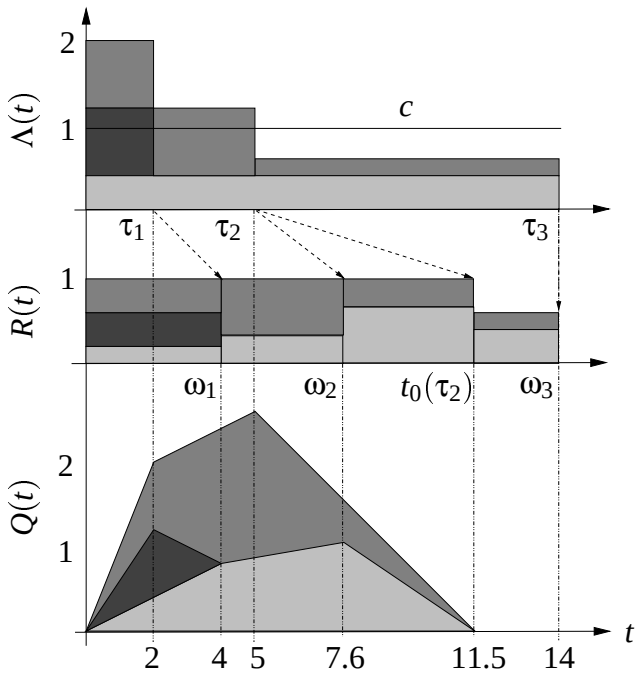

Fig. 1. Dynamics of a FIFO fluid buffer

and

$$\mathbf{r}\big(t_0(\tau_n)\big) = \boldsymbol{\lambda}(\tau_n), \quad \text{if } t_0(\tau_n) < \tau_{n+1} \text{ and } Q(\tau_n) > 0. \tag{6}$$

The dynamics of a fluid buffer is illustrated in Fig. 1. The buffer has an output rate of $c = 1$ units of fluid per unit of time and is initially empty. At $\tau_0 = 0$, it is fed by three flows such that $\boldsymbol{\lambda}(0) = (0.8, 0.8, 0.4)$. $\Lambda(0) > c$ so the queue builds up. The output flow components are in proportion to the input flow vector: $\mathbf{r}(0) = (0.4, 0.4, 0.2)$. At $\tau_1 = 2$, source 2 stops transmitting. By this time, the queue has grown up to $Q = 2$, so $\omega_1 = \tau_1 + Q(\tau_1)/c = 2 + 2 = 4$, and $\mathbf{r}(4) = (0.66, 0, 0.33)$, proportional to $\boldsymbol{\lambda}(2)$. The queue continues to build up at a lower rate. At $\tau_2 = 5$, source 1 changes its rate from 0.4 to 0.2. The output vector will change at $\omega_2 = 5 + 2.6 = 7.6$. Since there is fluid backlogged and the buffer is work-conserving, $\mathbf{r}(\omega_2) = (0.33, 0, 0.66) > \boldsymbol{\lambda}(\tau_2) = (0.2, 0, 0.4)$. The output rates will not change until the buffer becomes empty, at $t_0(\tau_2) = 11.5$, where the output vector equals the input vector as expected from equation 6. Finally, at $\tau_3 = 14$, the rest of the sources stop transmitting. Since by this time no fluid is backlogged, the output rate goes immediately to zero ($\omega_3 = \tau_3$).

We have already introduced the expressions governing the behavior of the basic fluid components that conform a communication network: sources and buffers. The preceding analysis may be applied as well to a network of fluid buffers. Consider for example the case shown in Fig. 2: if all of the $N + M$ sources are stepwise functions, then the input flow vector of the second buffer is also composed of stepwise functions; hence, $Q_1(t)$ and $Q_2(t)$ satisfy (2), while $R_1(t)$ and $R_2(t)$ satisfy (3). Consequently, all the equations presented above can be applied to more general

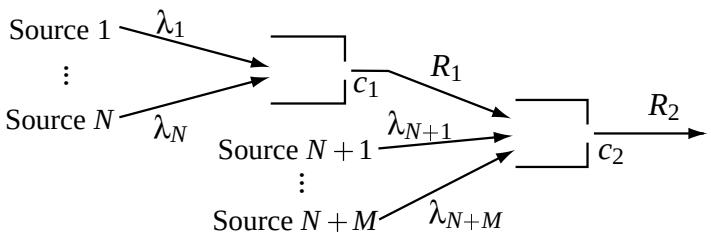

Fig. 2. Two buffers in tandem

topologies than single nodes. The basic mathematical framework of the dynamics of fluid models can be found, for instance, in [1] or [14].

The previous model can be easily extended to deal with a right-continuous, stepwise service rate function $c(t)$. For space and simplicity reasons [5] we do not explicitly consider this case in the paper. Nonetheless, the current version of FluidSim does include buffers with time-varying service rates (as a matter of fact, such buffers are needed for implementing e.g. ABR capabilities in end stations and switches, or Generalized Processor Sharing nodes; see [21] for more details).

3 Fluid Model of a Communication Network

Our tool considers fluid communication networks composed of the following basic elements: fluid sources and sinks, multiplexers (with buffering capacity or bufferless), communication links and switching matrices. Built on these, we currently have higher level objects such as different classes of switches, end-to-end connections, etc. Let us informally describe here these elements from a modeling point of view. In the next two sections, we discuss how our tool implements them, and how more complex objects can be built.

As stated before, we only consider *sources* producing stepwise, right-continuous fluid rate functions. *Sinks* are destination nodes. Sources and/or sinks may perform other functions such as complex control mechanisms, by means of dedicated algorithms. *Multiplexers* are network nodes composed of one or more buffers with capacities ≥ 0. Their functions are to merge the incoming flows according to some policy, possibly to store fluid and, as for sources and sinks, to run in some cases algorithms implementing control protocols. A *communication link* connects two network components. It is a unidirectional element that introduces a constant delay $d \geq 0$ to every flow going through it. A *switching matrix* is simply a mapping between two sets of elements. Its function is to separate the incoming aggregated flow vectors (*demultiplexing*) and to create new flow vectors at its output(s) (*multiplexing*) according to a routing table.

[5] Some additional event classes and event-handling procedures should be added to those described in Section 4.

With the previously described elements, we define *switching nodes*, composed of input and/or output buffers (which are multiplexing elements) and a switching matrix. We also define *unidirectional connections*, formed by a source, a sink, a unique static route and, possibly, an algorithm associated with some control protocol. Only point-to-point permanent connections are taken into account, yet our models can be trivially extended in order to represent the process of connection establishment and cancellation, as well as the multicast case. Applications requiring a bi-directional connection (e.g., ATM's ABR and ABT or Internet's TCP connections) can be represented as two unidirectional connections between the two communicating end points, which are, for instance, a source and a sink with a controller (an algorithm) implementing the corresponding protocol.

While a fluid model has been adopted for representing the network elements, a key feature of this tool is the inclusion of discrete objects, named *fluid molecules*, that can be emitted by the sources. They have no volume, so the buffer level is not changed by the arrival of a molecule. Fluid molecules can be used to compute quality of service (QoS) parameters, like the end-to-end delay (see Section 6.3). They can also represent the behavior of individual entities such as RM (Resource Management) cells while simulating ABR and ABT ATM flows, or ACK messages in the case of a window-based flow control protocol like TCP.

Example of a Fluid Network. Figure 3 shows the state of a section of a fluid network at some instant t. It contains many of the components introduced above. Connection 1, between source S_1 and destination D_1, is represented in light gray and traverses the first buffer and the switching matrix. Source S_2 belongs to a second connection represented in dark gray. Its flow traverses both buffers and the switching matrix and continues downstream. Connection 3, whose flow is represented in black, starts at source S_3 and traverses the switching matrix where it is routed towards the second buffer and continues downstream.

A change in the flow (r_1, r_2) leaving the first buffer, will occur at time $t + Q_1'/c_1$. The new flow will still have two components: the rate of the first connection will be $> r_1$, while the one belonging to connection 2 will be $< r_2$. In a similar way, the current flow $(0, r_3)$ leaving the second buffer will change at time $t + Q_2'/c_2$, producing a rate > 0 for connection 2 and $< r_3$ for connection 3. Molecule m_1, which belongs to connection 1, will wait for Q_1/c_1 time units before it leaves the first buffer.

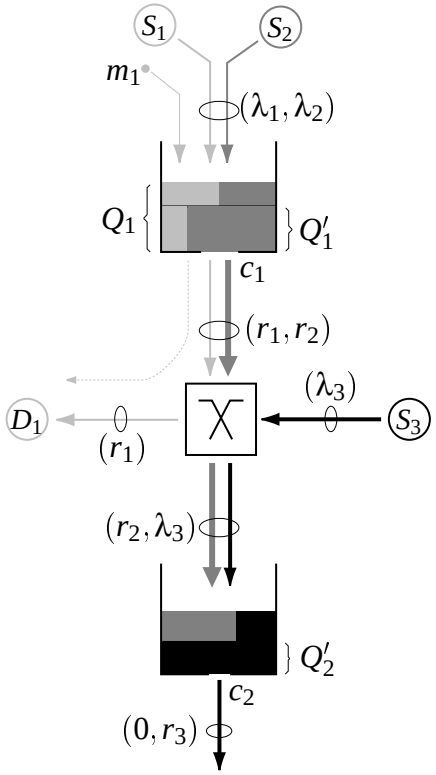

Fig. 3. Example of a fluid network (state at time t)

4 Simulation of a Fluid Network

4.1 Design Principles

The variables defining the behavior of a fluid network (λ_i, Q_j, etc.) fed by stepwise sources *always* follow piecewise linear sample paths. Therefore, in order to completely describe the evolution of such a network, *we only need to know the state of these variables for a denumerable set of time instants*, that is, at the transition epochs for the functions $\lambda_i(t)$, $r_i(t)$ and $\dot{Q}_j(t)$. This strongly hints at the use of *discrete-event simulation* as a simulation technique: every state transition in every variable will be related to an event handled by the scheduler.

In what follows, we describe the event types specific to the objects considered in the model and show how they are handled in the simulation process.

4.2 Event Classes

Discrete events required for the simulation of our fluid network models may be grouped into the following main classes according to the kind of object the event is

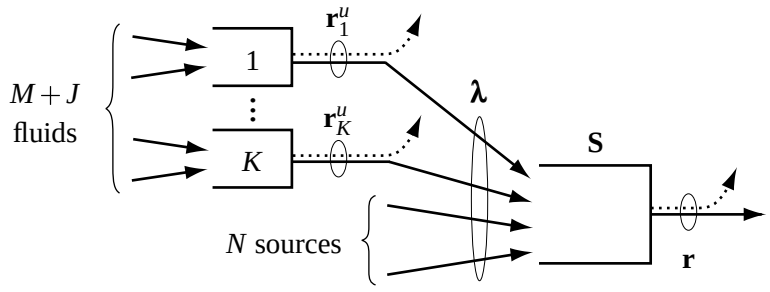

Fig. 4. Example of a buffer in a network

addressed to:

(1) events related to the sources, e.g. "the rate $\lambda_i(t)$ of source i changes";
(2) events related to the buffers: e.g. "the input flow vector $\boldsymbol{\lambda}_j(t)$ of buffer j changes", "the slope of $Q_j(t)$, the backlog of buffer j, changes", "the output flow vector $\mathbf{r}_j(t)$ of buffer j changes";
(3) events related to the switching matrices, e.g. "the input flow vector $\boldsymbol{\lambda}_j(t)$ of matrix j changes";
(4) events related to the communication links, e.g. "the input flow vector $\boldsymbol{\lambda}_j(t)$ of link j changes".

These four event classes will be described in the sequel.

4.3 Event Handling

In order to analyze the handling of events during the simulation of a complex network, we shall concentrate on a single buffer **S** (see Fig. 4), fed by N sources, whose flow rates are denoted by $\lambda_1,\ldots,\lambda_N$, and K upstream buffers, whose output flow vectors are denoted by $\mathbf{r}_1^u,\ldots,\mathbf{r}_K^u$. These buffers are globally fed by $M+J\geq K$ individual fluid flows, coming from other sources and/or buffers; from these, only M flows (numbered $N+1,\ldots,N+M$) are fed into **S**. The input flow vector of **S** is then $\boldsymbol{\lambda}(t)=\big(\lambda_1(t),\ldots,\lambda_N(t),r_{N+1}^u(t),\ldots,r_{N+M}^u(t)\big)$, and its output flow vector is $\mathbf{r}(t)=\big(r_1(t),\ldots,r_N(t),r_{N+1}(t),\ldots,r_{N+M}(t)\big)$. The total input and output rates are respectively denoted by $\Lambda(t)$ and $R(t)$. Without loss of generality, we shall consider in this section that the links connecting sources and buffers introduce no delay.

4.4 Events Related to the Sources

Rate Changes. For the i-th source, a discontinuity in the sample path $\lambda_i(t)$ happening at $t=\tau_n$ implies a transition in $\boldsymbol{\lambda}(t)$, so to guarantee the synchronization of arrivals to **S** we should:

(1) Handle all simultaneous events related to network components located upstream with respect to **S**, i.e., execute all tasks "change the rate $\lambda_j(t)$ at $t = \tau_n$" and "change the vector $\mathbf{r}_j^u(t)$ at $t = \tau_n$" according to (5) and (6).
(2) Schedule the event "$\boldsymbol{\lambda}(t)$ changes at $t = \tau_n$".
(3) Schedule the next rate-change event for this source.

Molecule Generation. An event "source i emits a molecule at $t = t'$" produces the following actions:

(1) Create an object "molecule belonging to flow i" and initialize its particular fields if necessary (e.g. birth date for delay measurement molecules, see Section 6.3).
(2) Schedule the task "a molecule from flow i arrives at $t = t'$" for the (unique) downstream neighbor.
(3) Possibly, schedule the next molecule-emission event for this source.

Notice that the molecule-generation epochs depend on the given application of fluid molecules:

- If molecules are used to simulate a flow control algorithm, emission dates shall be determined by the particular algorithm. For instance, in ATM's ABR a RM-cell molecule shall be sent whenever the source has produced a certain amount of fluid, which is related to ABR source parameters such as the frequency N_{RM} of RM-cell generation (for more details on ABR source behavior see e.g. [8]).
- On the other hand, molecules employed to estimate the end-to-end delay of a single connection (associated to a traffic source) are produced following an algorithm described in Section 6.3.

4.5 Events Related to the Buffers

Input Rate Transitions. An event "$\boldsymbol{\lambda}(t)$ changes at $t = \tau_n$" should trigger the following actions:

(1) Compute $Q(\tau_n)$ and ω_n according to relations (1) and (4) respectively.
(2) Calculate $t_0(\tau_n)$, $t_B(\tau_n)$, as defined in Section 2; then
  (a) schedule the task "**S** begins to overflow at $t = t_B(\tau_n)$", if this has not been done yet and $t_B(\tau_n) < \infty$, and
  (b) schedule the task "**S** becomes empty at $t = t_0(\tau_n)$", if this has not been done yet and $t_0(\tau_n) < \infty$.
(3) If $\omega_n \neq t_0(\tau_n)$, schedule the task "change $\mathbf{r}(t)$ at $t = \omega_n$".

However, computing $t_0(\tau_n)$ and $t_B(\tau_n)$ (and consequently taking the decision of scheduling or not an event "**S** becomes empty" or "**S** overflows") poses some prac-

tical problems, because the evolution of $\Lambda(t)$, and hence of every component of $\boldsymbol{\lambda}(t)$, from $t = \tau_n$ onwards must be known *a priori* to do so. Nevertheless, there is a simpler alternative based on the following definitions.

Definition 1 *An input rate transition at time $t = T_n$ is an* overloading *transition if $\Lambda(T_n) > c$. The* predicted *overflow period will begin at:*

$$\hat{t}_B(T_n) = \begin{cases} \dfrac{B - Q(T_n)}{\Lambda(T_n) - c} + T_n & \text{if } \Lambda(T_n) > c, \\ \infty & \text{if } \Lambda(T_n) \leq c. \end{cases} \tag{7}$$

Definition 2 *An input rate transition at time $t = T_n$ is an* underloading *transition if $\Lambda(T_n) < c$. The* predicted *empty period will begin at:*

$$\hat{t}_0(T_n) = \begin{cases} \dfrac{Q(T_n)}{c - \Lambda(T_n)} + T_n & \text{if } \Lambda(T_n) < c \text{ and } Q(T_n) > 0 \\ \infty & \text{if } \Lambda(T_n) \geq c \text{ or } Q(T_n) = 0. \end{cases} \tag{8}$$

Moreover, note that neither $R(t)$ nor $\mathbf{r}(t)$ change because of an overflow; so, it is not necessary—in principle—to schedule an event "**S** overflows", because the lost volume can be computed *a posteriori*. Therefore, steps 2b-3 above may be replaced by the following three steps:

(2) If $\hat{t}_B(\tau_{n-1}) < \tau_n$ then **S** is overflowing, so take fluid loss measurements (for details, see [23]).
(3) Calculate $\hat{t}_0(\tau_n)$, $\hat{t}_B(\tau_n)$; then
  (a) if there is already an event "**S** becomes empty at $t = t'$" in the event list (with $t' \neq \hat{t}_0(\tau_n)$), cancel it, and
  (b) if $\hat{t}_0(\tau_n) < \infty$, schedule the event "**S** becomes empty at $t = \hat{t}_0(\tau_n)$".
(4) If $\omega_n \neq \hat{t}_0(\tau_n)$, schedule the task "change $\mathbf{r}(t)$ at $t = \omega_n$".

Figure 5 shows a typical sample path for $Q(t)$ and the associated $\Lambda(t)$, as well as predicted overflow and "underflow" times; the curves correspond to the parameter values: $c = 1$, $B = 1$. Note that, in this case, $\hat{t}_B(0) = t_B(0)$ and $\hat{t}_0(4.5) = t_0(4.5)$ but $\hat{t}_B(4) \neq t_B(4)$ and $\hat{t}_0(2) \neq t_0(2)$, i.e., there is an event that gets scheduled and later canceled.

Buffer Underflow. An event "**S** becomes empty at $t = \hat{t}_0(\tau_n)$" should:

(1) make $Q(t) = 0$, and
(2) execute the event "change $\mathbf{r}(t)$ at $t = \hat{t}_0(\tau_n)$", that is, "now".

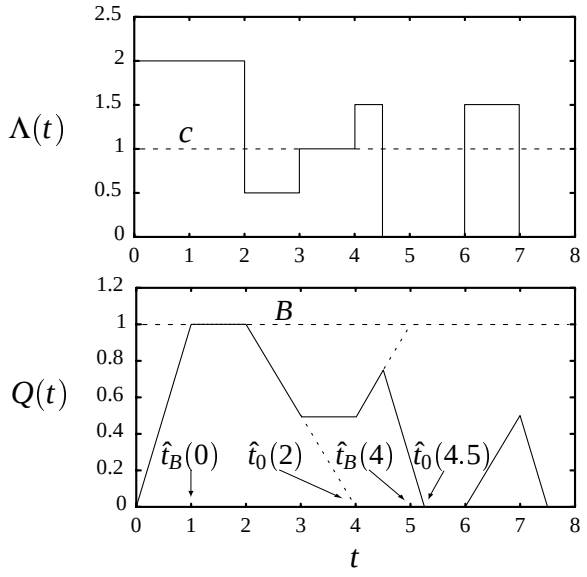

Fig. 5. Estimation of $t_0(\tau_n)$ and $t_B(\tau_n)$

Output Rate Transitions. An event "$\mathbf{r}(t)$ changes at $t = \omega_n$" should trigger the following actions:

(1) Calculate $\mathbf{r}(t)$.
(2) Schedule an event "change $\boldsymbol{\lambda}(t)$ at $t = \omega_n$" for the downstream neighbor.

The previous step should be synchronized with other similar actions, to guarantee the generation of a single "input rate transition" event for every element affected by this transition.

Molecule Arrivals. An event "a molecule arrives at $t = a_n$", with $a_n \in [\tau_m, \tau_{m+1})$, should spawn the following actions:

(1) If $\hat{t}_B(\tau_m) < a_n$ then **S** is overflowing, so the molecule will be destroyed with probability $\big(\Lambda(\tau_m) - c\big)/\Lambda(\tau_m)$.
(2) If the molecule is not destroyed, then:
    (a) Calculate the waiting time $W(a_n)$, as defined in Section 6.3.
    (b) Schedule the task "a molecule arrives at $t = a_n + W(a_n)$" for the downstream network element that should receive this molecule.

Of course, the arrival of special molecules (such as those representing RM cells to buffers that implement ABT/ABR protocols) will activate the procedures specific to them.

4.6 Events Related to the Switching Matrices

Input Rate Transitions. Let us consider a switching matrix with N_{out} output ports and let $\boldsymbol{\lambda}^u(t)$ be the aggregate input fluid vector. An event "$\boldsymbol{\lambda}^u(t)$ changes at $t = \tau_n$" should trigger, for each output port $m = 1, \ldots, N_{out}$, the following actions:

(1) Estimate the new input flow vector $\boldsymbol{\lambda}_m(\tau_n)$ for the port according to the matrix's routing table.
(2) Schedule the task "$\boldsymbol{\lambda}_m(t)$ changes at $t = \tau_n$" for the downstream neighbor connected to port m.

Molecule Arrival. When an event "molecule arrives at $t = a_n$" occurs, the output port the molecule should be forwarded to, is obtained from the matrix's routing table and a similar event "molecule arrives at $t = a_n$" is scheduled for the corresponding downstream neighbor.

4.7 Events Related to the Communication Links

Given that the link model is simply that of a delay element, when any event arrives to a link at $t = t'$, a similar event is scheduled for the downstream neighbor at $t = t' + d$, where $d \geq 0$ is the link delay.

4.8 Efficiency of the Fluid Simulation Approach

The efficiency of the simulation based on fluid models comes from the reduction in the number of events that have to be executed. In a conventional simulator the number of events is proportional to the number of packets produced by the sources, while it is proportional to the number of rate transitions in a fluid-based simulator. Therefore, we expect significant gains if the sources produce large bursts of packets. For instance, Pitts et al. [19] compare an ATM fluid simulator against a cell-based one by computing the cell execution rate measured as the number of cells processed by second. In one of the experiments shown in [19], the mean burst size of the on/off sources was changed from 1,650 bytes to 162,500 bytes. As a consequence, the computing gain grew from 8 to 1,096.

Nevertheless, the number of processed events cannot be *the sole* measure of efficiency for comparing both approaches because the treatment of simulation events in the fluid case is, in general, more complex than in the conventional approach. Compare, for instance, the tasks associated to the event "$\boldsymbol{\lambda}(t)$ changes at $t = \tau_n$" arriving to a fluid buffer, against the handling of a packet arrival to a discrete buffer.

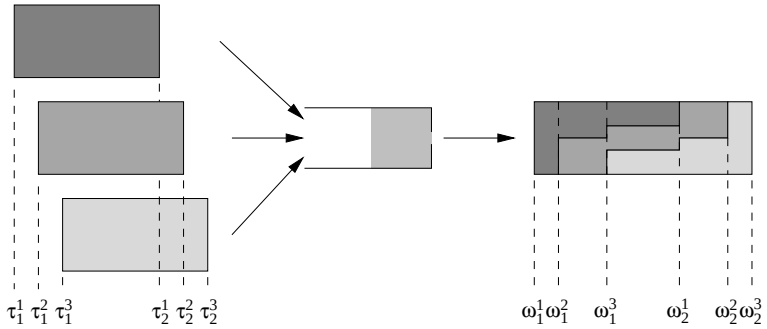

Fig. 6. Flow interaction in a buffer

As an example, Nicol et al. [17] show an extreme case where the simulation of their fluid-based models was two times slower than the simulation of their packet-based models when both executed roughly the same number of events. Notice that both simulations used as much common code as possible.

Another issue that could affect the efficiency of the fluid simulation approach is the so-called *ripple effect* that results from the interaction between flows sharing a buffer. Kesidis et al. [12] described for the first time this effect and show some experimental evidence of the performance degradation that it may produce. Nicol et al. [17] also provide some evidence of the ripple effect in more complex network topologies. Liu et al. [16] use an analytical approach to characterize this phenomenon for the particular case of a tandem of FIFO buffers in a parking-lot topology fed by Markov Modulated on/off fluid sources. Ros [21] discusses this effect in a more general context and emphasizes the difficulty in quantifying its impact in a fluid simulator.

The ripple effect can be defined as follows: a change in the rate of a flow i that arrives at a buffer, may induce a rate change in *all* the flows traversing that buffer. This may produce an increment in the number of events to be executed by the downstream neighbors.

We will illustrate this effect using the example shown in Fig. 6. Three sources that feed the same FIFO fluid buffer produce traffic bursts with peak rate h. The service rate of the buffer is $c = h$. At a certain moment, a burst from each source arrives at the buffer after a small delay, as shown in Fig. 6. When the first burst arrives at $t = \tau_1^1$, the whole bandwidth is assigned to it. No fluid is stored in the buffer. At $t = \tau_1^2$ the second burst arrives and the output rate of the first flow is reduced such that both flows have a fair share of the buffer's service rate. The third burst arrives at $t = \tau_1^3$, and all the output rates must be adjusted again at $t = \omega_1^3$ to account for the arriving flow. A rate change at $t = \tau_2^1$ signaling the end of the first burst induces again a rate change in all the individual flows at $t = \omega_2^1$. Similarly, the termination of the second burst at $t = \tau_2^2$ produces a rate change in the second and third flows at ω_2^2.

It is worth observing that the number of rate changes experienced by the output

flow vector $\mathbf{r}(t)$ as a whole, is the same as the number of rate changes at the input flow vector $\boldsymbol{\lambda}(t)$, six in the example. Hence, as long as these flows travel together downstream, there will be no increment in the overall number of events processed. What has changed, though, is the number of rate changes experienced by each individual flow: it has doubled in this example. So, once the individual flows are demultiplexed, the number of events processed by the downstream components will increase.

It is very difficult to properly estimate how the ripple effect would alter the performance of the fluid simulation approach, since its real impact depends on several factors, such as:

- The interaction between flows and the particular topology. If, for example, $\tau_1^1 < \tau_1^2 < \tau_1^3 < \tau_2^3 < \tau_2^2 < \tau_2^1$ in the system of Fig. 6, then the number of individual transition rates would still double, but its distribution would have changed (e.g., the third flow would not increment its number of transitions whereas the first flow would experience six rate changes) thus the total number of processed events depends on the path taken by each individual flow throughout the network.
- The network load. As long as $Q(t) = 0$, $\mathbf{r}(t) = \boldsymbol{\lambda}(t)$ so the ripple effect will not appear.
- The queueing discipline used at the multiplexers. In models of networks providing different qualities of service by flow isolation using GPS-like schedulers (like e.g. in the Internet's *Diffserv* architecture [26]), such flow isolation may reduce the overall ripple effect.

5 Implementation of the Fluid Simulator

In this section we briefly describe the prototype of our fluid simulation tool, FluidSim. It consists of a modular library of network objects that follow the principles described so far, a discrete-event simulation kernel and support libraries, and an optional set of routines that permit the definition and parameterization of complex fluid networks from a configuration file and/or a graphical interface. These routines also provide the environment for executing various simulation runs of the same network model in order to estimate confidence intervals.

The simulator follows the object-oriented paradigm, has been implemented in `C++` [24] and benefits in an important manner from the recently standardized *Standard Template Library* [2]. The graphical interface has been coded using the `Java` programming language.

FluidSim is event-driven. Network objects communicate by exchanging messages in which simulation events are encoded, either directly (if they are to be executed immediately) or by means of the simulation kernel. An event has several fields; among the most relevant are its type, its scheduled time (i.e., when the event should occur), the recipient of the event and a payload, by means of which information (fluid vectors, molecules) is transferred among the simulated network objects. The simulator kernel stores future events in a time-ordered event list. The scheduler works indefinitely by selecting from the list the next earliest event to execute, invoking the event handler of the concerned object and returning to select the next event to execute. An event usually creates new events that are inserted in the event list according to the simulation time at which they should be executed. This cycle finishes when there are no more events to process or when a special event to stop the simulation is found. In the current implementation, the event list is kept as a doubly-linked list. The modular nature of the design accounts for a fairly transparent migration towards a more efficient data structure (e.g. heap or calendar queue) if desired.

FluidSim's kernel provides some specific facilities to efficiently support fluid models of network elements, such as event cancellation (Section 4.5) and handling of "simultaneous" events happening at the same simulation epoch (Section 4.4). For the former, network objects can store a reference to an event in the event list such that the kernel's cancellation method can directly annulate the event without having to (linearly) search it. Concerning the simultaneous events, the kernel provides methods for fetching the relevant events (i.e., events of the same type, for the same destination and with the same simulation epoch). It also provides a two-priority event scheme (a low priority event that occurs at the same simulation epoch than a high priority one, will be scheduled after this one).

FluidSim provides a family of classes for generating random variates (exponential, Pareto, etc.) from a multiplicative linear congruential random number generator, using the inverse transformation method [7]. Multiple generators may be instantiated simultaneously, providing different streams.

The environment permits the collection of simulation traces in log data files for further processing. The user may decide to use different log files for the different variables being monitored, or he/she may gather them in a common (default) file. Classes are also provided for collecting data and automatically computing statistical information such as arithmetic means and standard deviations, histograms and integral approximations. The latter are used, for instance, for estimating on-line the

mean level of fluid in a buffer, $\overline{Q}$, according to the following recurrence relation:

$$T_{n+1}\,\overline{Q}(T_{n+1}) = \int_0^{T_{n+1}} Q(t)\,dt = T_n\,\overline{Q}(T_n) + \int_{T_n}^{T_{n+1}} Q(t)\,dt, \tag{9}$$

where the value of the last integral depends on the buffer state in $[T_n, T_{n+1}]$:

$$\int_{T_n}^{T_{n+1}} Q(t)\,dt = \begin{cases} \frac{Q(T_n)}{2}\left(t_0(T_n) - T_n\right) & \text{if } t_0(T_n) < T_{n+1}, \\ \frac{Q(T_n)+B}{2}\left(t_B(T_n) - T_n\right) + B\left(T_{n+1} - t_B(T_n)\right) & \text{if } t_B(T_n) < T_{n+1}, \\ \frac{Q(T_n)+Q(T_{n+1})}{2}\left(T_{n+1} - T_n\right) & \text{otherwise.} \end{cases} \tag{10}$$

Another class in FluidSim permits to automatically compute an approximation of the cumulative distribution function of the buffer occupancy, $\Pr(Q \le q), q \in \{0, B/N, 2B/N, \ldots, B\}$, as follows. For each interval $[(i-1)/B, i/B]$, S_i will accumulate the periods of time in which the buffer had fluid up to level i. After an initial warm-up period of length $t_{ss} \ge 0$, every time there is a change in the slope of Q, (i.e., when the buffer becomes empty, full or when there is a rate change in the input flow vector), a new point (level, time) is recorded and the proportion of time in which there was fluid below each interval i, is computed and accumulated. For example, if Q changed from level 5 to level 8 in 3 time units, S_6 will aggregate one time unit, S_7 two and S_8 and above will accumulate 3 time units. The cumulative distribution function is approximated by: $\Pr(Q \le i) \approx S_i/(t_{max} - t_{ss})$.

Confidence intervals for $\overline{Q}$, $\Pr(Q \le i)$ and other measures (e.g. related to the end-to-end delay) are computed simply by the replication method.

5.2 Fluid Objects

The hierarchy of the principal fluid network components implemented at present is shown in Fig. 7. All the network objects are derived from the abstract base class `ActiveSimObj`, from which they inherit the ability to send and process simulation events. Let us describe here some specific aspects of the implementation.

Traffic Sources. Our current implementation provides three kinds of fluid sources that obey the stepwise property.

- On/off sources. They switch between an active state in which they transmit at peak rate, and a silent state where no transmission takes place. Sojourn intervals at each state are i.i.d. random variables with arbitrary distributions.
- Trace-based sources. The instantaneous transmission rate is calculated from files containing traces of real traffic, for instance, sequences of MPEG-1 I/B/P frames.

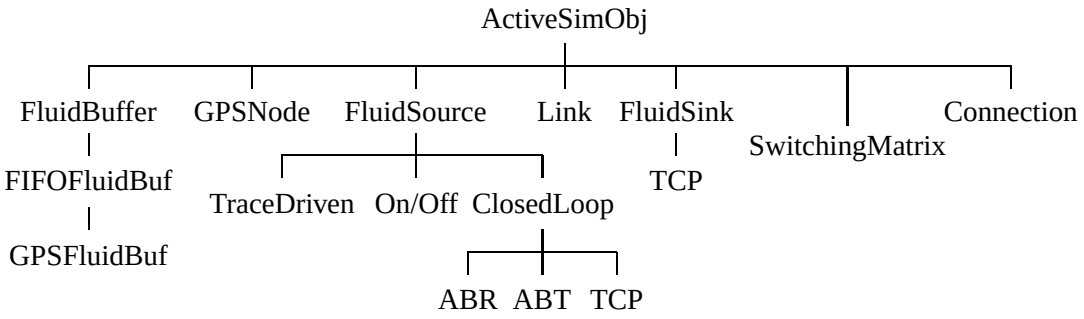

Fig. 7. Main hierarchy of FluidSim classes

The fluid model adopted for such sources assumes that the frequency of images, *fps*, is constant and that the MPEG source generates a smooth flow of uniformly spaced chunks of information.

- Closed-loop sources. At present, we are experimenting with three kinds of closed-loop fluid sources: ABR and ABT sources that transmit RM-type molecules and adapt their rate according to the molecules they receive following different algorithms [21]. A fluid version of TCP's window-based flow control is also under study.

Multiplexers. Basically, two kinds of multiplexing nodes are currently implemented:

- FIFO fluid buffers with capacity $B \leq \infty$. These are the elementary fluid buffers we have discussed in this paper. The limiting case where the buffer capacity is $B = 0$ permits to simulate a *bufferless multiplexing node*.
- Nodes that implement the General Processor Sharing (GPS) service discipline. A GPS node has N buffers $B_1, \ldots, B_N$ (see Fig. 8). Each buffer is served according to a FIFO discipline, but its service rate, $R_i(t)$, may fluctuate in order to satisfy the GPS policy. The global output rate is $R(t) = \sum_{i=1}^{N} R_i(t)$ and the node's service rate is c. A more detailed presentation of these nodes can be found in [23]. A variant *GPS + Best Effort*, as defined by Kesidis [11], is also provided: that is, a $(N+1)$-th buffer is added to the architecture in Fig. 8, and this buffer is served if and only if *all* the other buffers are empty.

Fluid Sinks. Fluid sinks are the end point of a connection where the arriving flow is absorbed and some statistics concerning the individual connection are collected if required. In addition to this basic fluid sink, the library provides objects that complement our closed-loop experimental components: ABT, ABR and TCP sinks.

Links. Communication links implement the elementary delay component presented in Section 3.

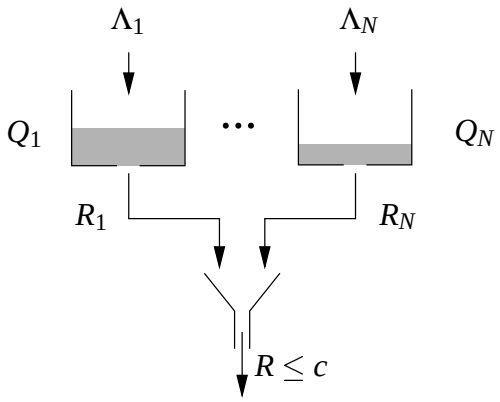

Fig. 8. GPS node

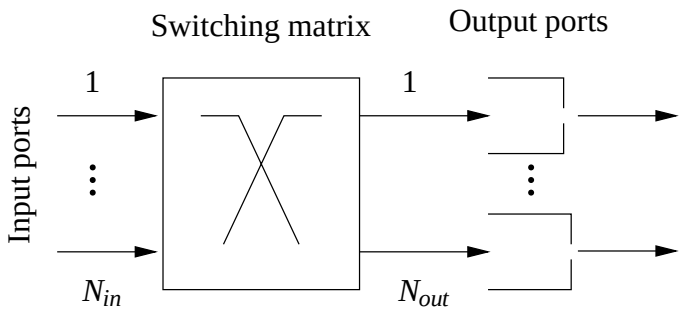

Fig. 9. Switching node

Switching Matrix. Non-blocking, zero-delay switching fabrics are usually found as components of fluid switching nodes, although they are independent objects. They implement the demultiplexing and multiplexing functions described in Section 3.

Switching Nodes. The model adopted for a switching node is an output buffered switch which consists of an arbitrary number, N_{in}, of input ports, a switching matrix and an arbitrary number, N_{out}, of output ports (see Fig. 9). The output ports can be simple FIFO buffers, GPS or GPS + Best Effort nodes, or just bufferless multiplexing nodes. FluidSim also includes nodes that implement complex mechanisms for ATM's ABR and ABT service classes. For instance, a fluid version of the ERICA+ ABR control algorithm [9] has already been implemented [21].

Connections. As mentioned before, currently unidirectional unicast connections are implemented. A connection always starts at an "end station" (fluid source or fluid sink), follows a path composed of different network elements (links, multiplexers, switching matrices, switching nodes) and ends up at an end station.

Fluid Vectors. At any point in the network traversed by $N \geq 1$ connections, the fluid resulting from the aggregation of the individual flows is represented by the composition vector of instantaneous rates $\boldsymbol{\lambda}(t) = \big(\lambda_{j1}(t), \ldots, \lambda_{jN}(t)\big)$. This description of traffic as a vector of independent and distinguishable fluid flows, enables the multiplexing and demultiplexing of those flows; moreover, it permits to gather end-to-end measures and statistics for individual connections. For efficiency reasons, when a transition in $\boldsymbol{\lambda}$ occurs, we consider only those components that actually changed its rate.

Fluid Molecules. Fluid molecules are considered as being part of the flow produced by a connection's source (and it is always possible to identify the connection the molecule belongs to). Recall that if a molecule arrives to a buffer that is overflowing, it is lost with probability $(\Lambda(t) - c)/\Lambda(t)$. Observe that this probability corresponds to the instantaneous fluid loss rate during congestion.

5.3 Configuration Interface

Using FluidSim as a set of C++ libraries is appropriate while developing new network components or if one wants to integrate those libraries to existing C++ programs. However, when the goal is to simulate different network configurations (e.g. *what-if* scenarios) using the objects already provided, a more flexible environment is offered: the network topology, object parameterization and general simulation data may be input via a graphical interface or through a configuration text file. In this mode, the simulator runs as a stand-alone program. A *simulation driver* reads the configuration file, creates the necessary network objects, starts the simulation and collects statistics. If required, it initiates different simulation runs with separate random number sequences for estimating confidence intervals and resets the network object's states between runs.

The graphical interface provides the standard functions associated with this kind of tool: through the aid of icons, menu bars and mouse displacements, the user can create a network topology, define object parameters, copy and delete individual objects or groups of objects (sub-networks), etc. Once the topology is defined, the tool does some validations and creates the configuration file (see Fig. 10).

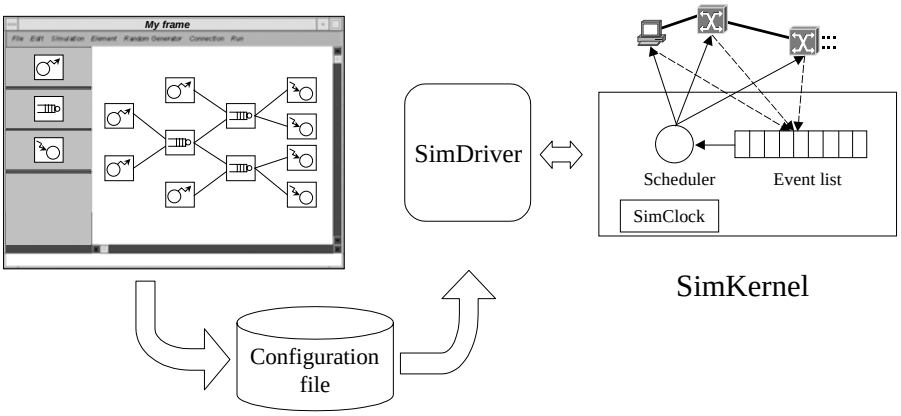

Fig. 10. Stand-alone operation

The configuration file follows a simple syntax. Simulation objects are defined in blocks. A block is composed of a type tag and a body in brackets where the object's parameters are defined. For space reasons we do not detail the whole syntax specified for the configuration files. To illustrate it, let us consider a network composed of two buffers (`BufA`, `BufB`) and two connections. Connection 1 flows from source `Src1` to sink `Snk1` traversing `BufA` and `BufB`. Connection 2 flows from source `Src2` to sink `Snk2` passing through `BufA` only. The corresponding configuration file is presented here in two-column format:

```
# Simple network with 2 connections.

# 1st block. General parameters:
#  - duration and warm up time (in
     simulation time) per run
#  - seeds for default RNG
#  - output file for results
#  - number of runs

SIMULATION {
  DURATION 1000.0,
  WARMUP_TIME 45.0,
  SEEDS 9977581 234234,
  OUT_FILE MyRun.out,
  NUM_RUNS 30
}

# This block specifies 2 RNG.
# First one uses global seeds.
# Random streams will be associated
  with random variates
RANDOM_GENERATOR {
  rng1;
  rng2 SEEDS 1234581 6688774; }
```

```
# Source block.
# First source ON/OFF with different
  distributions for sojourn periods.
# Second source is trace-driven:
  source rates and transition dates
  read from specified file
SOURCE {
  Src1 ON_OFF
    PEAK_RATE   2048000.0,
    SOJOURN_ON
      DETERMINISTIC 0.4,
    SOJOURN_OFF
      PARETO 1.5 0.8 rng2;
  Src2 TRACE_DRIVEN
    FILE ATraceFile.dat; }

# Sink block.
# Stats collection for connection 2
SINK {
  Snk1;
  Snk2 STATS YES; }

# 1st buffer infinite capacity with
  FIFO policy (by default).
# Rate in units of fluid per second.
```

```
#
# 2nd buffer finite of size 512K units.
BUFFER {
  BufA INFINITE
    RATE 51200.25;

  BufB FINITE
    RATE    888812.2315,
    POLICIY FIFO,
    SIZE    512000;
}

# Connection block defines the path
CONNECTION {
  First
    PATH <Src1->BufA->
          BufB->Snk1>;
  Second
    PATH <Src2->BufA->Snk2>;
}
```

6 Examples

In this section we briefly present some results obtained with FluidSim. The first example (Section 6.1) was used for testing the fluid simulation tool. In particular, it allows to check its accuracy by comparison with analytical results. The second example (Section 6.2) was specifically designed to illustrate the gain in efficiency with respect to standard tools; we focus there only on the event processing rate. The last example (Section 6.3) shows how FluidSim can be used for performance evaluation of computer networks.

6.1 Comparison with an Analytical Model

Let us consider a simple model composed of a single finite-capacity buffer fed by ten homogeneous on/off sources, whose on and off periods are exponentially distributed with mean 10 ms, and with a peak rate of 15 Mbps. The buffer has a capacity of 1 Mb and an output rate of 100 Mbps. We are interested in the complementary distribution $\Pr(Q > q)$ of the fluid level Q. This is an interesting topology for validation purposes, because it is simple enough so that *very* long simulation runs can be performed in a reasonable amount of time and, most importantly, the *exact* expression of $\Pr(Q > q)$ is known, see e.g. [14].

We performed 30 independent simulations. Each simulation run corresponded to a simulated interval of 2×10^5 s ($\approx 55,5$ hours) and took about one hour of computing time on a Sun UltraSPARC workstation shared between several users. The number of events processed in each run was $\approx 2 \times 10^8$. Note that the equivalent number of events that should have been processed by a conventional simulator, if the discrete units were for instance ATM cells, would be $\approx 3.5 \times 10^{10}$. Figure 11 shows the fluid level distribution, with 95% confidence intervals (shown as small crosses); note the excellent agreement between theoretical and experimental values.

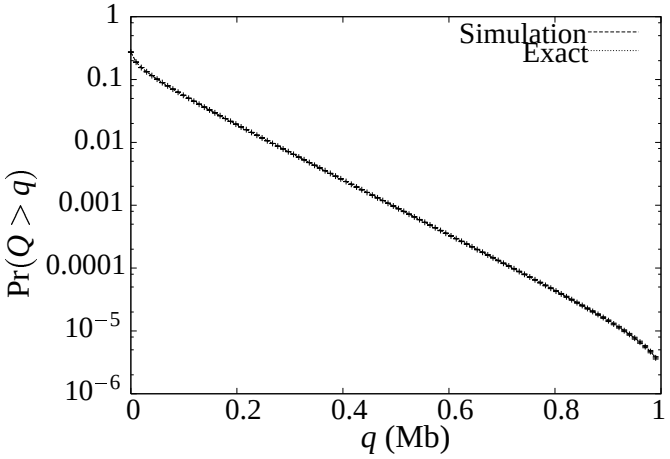

Fig. 11. Example 1: complementary distribution of fluid level Q

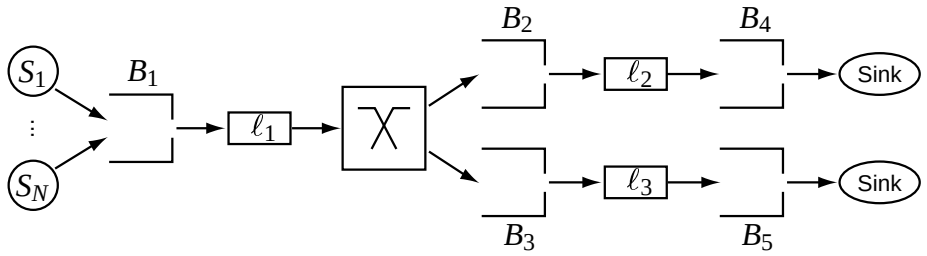

Fig. 12. Example 2: network topology

6.2 Comparison with a Cell-Level Simulation Tool

The following example highlights the efficiency of the fluid simulation paradigm, in comparison to a traditional discrete-event simulation approach. The network under study is shown in Fig. 12; remark that this topology is vulnerable to the "ripple effect" (see Section 4.8).

For comparison purposes, we have selected the well-known NIST simulator [5] which is a cell-level, ATM-oriented tool. We performed 30 one-second simulation runs in order to obtain mean values. The parameters used are as follows:

- 10 homogeneous on/off sources, with peak rate $=$ 18 Mbps and exponentially-distributed on and off periods with mean 10 ms. The mean burst size is then 22.5 KB $\approx$ 424 ATM cells. All buffers are identical, with a capacity of 1 Mb.
- Link ℓ_1 has a transmission rate of 100 Mbps; ℓ_2 and ℓ_3 operate at 50 Mbps.
- Connections of traffic sources $S_1, \ldots, S_5$ follow the route $B_1 \rightarrow \ell_1 \rightarrow$ switch $\rightarrow$ $B_2 \rightarrow \ell_2 \rightarrow B_4$, whereas the remaining connections follow the route $B_1 \rightarrow \ell_1 \rightarrow$ switch $\rightarrow B_3 \rightarrow \ell_3 \rightarrow B_5$.

Simulations of the fluid model showed that $\Pr(Q_1 > 0) \approx 0.66$, i.e., buffer B_1 holds fluid about 66% of the time; hence, about 66% of the transitions in each source will

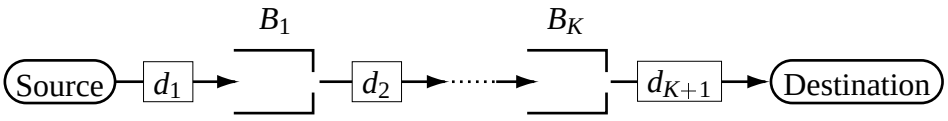

Fig. 13. Reference network for the end-to-end delay.

affect the output rate of *all* flows going out of B_1, and so the influence of the ripple effect should be non-negligible.

Table 1 presents some results regarding the computational cost of cell-level and fluid-level simulations. Both the NIST simulator and FluidSim were compiled using the same development tools and run on the same architecture (a PowerPC-based computer); moreover, the NIST simulator was executed in non-interactive mode, so as to optimize its performance.

Table 1
Example 2: results on simulation efficiency

Simulator	Mean execution time (s)	Mean number of processed events	Event processing rate (events/s)
NIST	18.11	2.502×10^6	1.382×10^5
FluidSim	0.166	1.374×10^4	8.277×10^4

We can see that, even though the treatment of fluid-level events is ≈ 1.67 times more expensive than the processing of cell-level events, the fluid-model paradigm allows for a reduction in the number of events that far outweighs the increased computational cost; remark that this is so in spite of the ripple effect. The speedup factor is $18.11/0.166 = 109.1$.

6.3 End-to-End Delay Estimation

The notion of end-to-end delay for a *single connection* in a fluid model is not as obvious as in a discrete model. Following [3], let us define the waiting time of a molecule arriving at a buffer at time t, and being accepted (i.e. not being lost because of congestion), as:

$$W(t) = \inf\{s > 0 \,|\, cs > Q(t)\} = Q(t)/c \tag{11}$$

Let us examine now what happens when the fluid belonging to a given connection must flow from source to destination through a tandem of K buffers, connected by $K+1$ links having non-zero propagation delays, as in Fig. 13. Let $Q_i(t)$, c_i and $W_i(t)$ be the level, the maximal output rate and the waiting time, respectively, for the i-th buffer. The date $a_i(t)$ at which a molecule produced by the source at time t

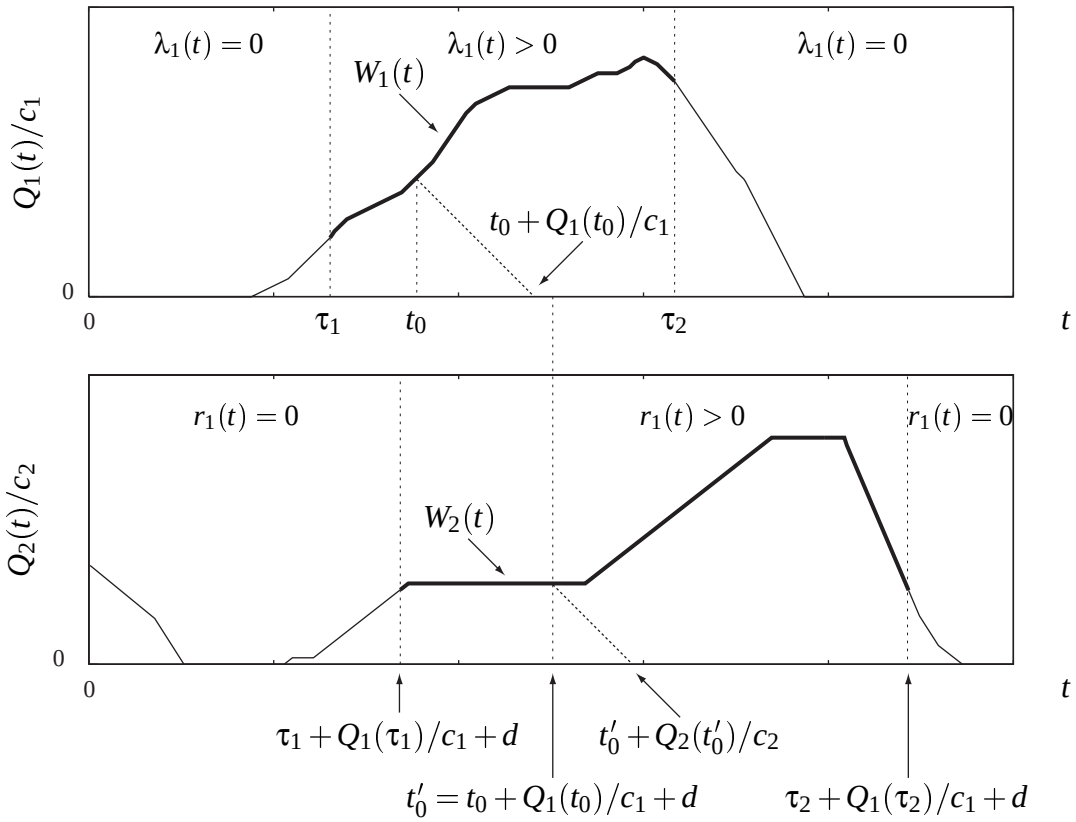

Fig. 14. Waiting time functions for two buffers in tandem.

arrives at buffer B_i ($i=1,\ldots,K$) is given by:

$$\begin{aligned} a_1(t) &= t+d_1 \\ a_{i+1}(t) &= a_i(t)+W_i\big(a_i(t)\big)+d_{i+1}\,, \quad i=1,\ldots,K \end{aligned} \tag{12}$$

where d_i is the delay introduced by link i ($i=1,\ldots,K+1$), and $a_{K+1}(t)$ denotes the arrival date at the destination. Remark that W_i depends on $W_{i-1},\ldots,W_1$.

So, the end-to-end delay for the connection can be defined as:

$$\Delta(t)=a_{K+1}(t)-t=\sum_{i=1}^{K+1}d_i+\sum_{i=1}^{K}W_i\big(a_i(t)\big) \tag{13}$$

Clearly, the function $\Delta(t)$ is defined *only* for those values of t such that $\lambda(t)>0$ (that is, when fluid is produced by the source).

Figure 14 shows an example of the evolution of the waiting time functions for buffers B_1 and B_2, over a time interval such that the source of interest sends fluid only during $[\tau_1,\tau_2)$; remark how a molecule-emission epoch (marked as t_0) results in the departure epoch $t_0'+Q_2(t_0')/c_2$.

Let us point out here that we are interested not only in estimating mean delay values, but also in higher order moments, as well as in the empirical delay distribution

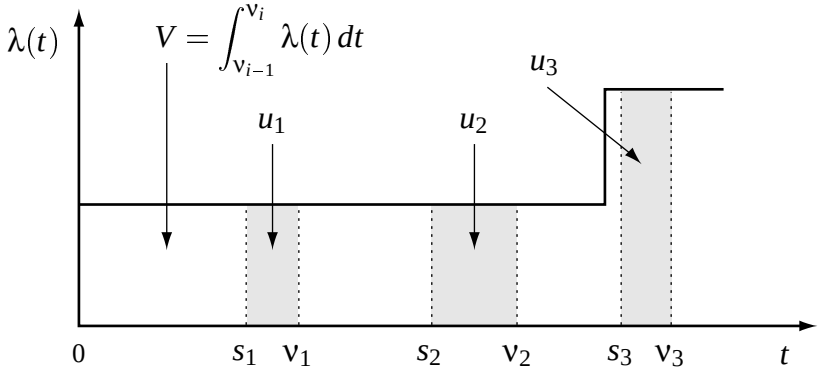

Fig. 15. Method for evaluating $\Delta(t)$.

$\Pr(\Delta > \delta)$. This is so because in communication networks, parameters such as the delay variation or "jitter" are as important as the mean end-to-end delay.

In principle, it would be possible to estimate $\Pr(\Delta > \delta)$ from the traces of $\lambda(t)$ and $Q_1(t), \ldots, Q_K(t)$, by evaluating the waiting time functions

$$W_i(a_i(t)) = Q_i(a_i(t))/c_i$$

in equation (13), only for those values of t such that $\lambda(t) > 0$; then, $\Pr(\Delta > \delta)$ would be computed from the trajectory of $\Delta(t)$, as done for the level distribution function in section 5.1. However, this raises some efficiency concerns [6]. Off-line computation involves saving (potentially) huge amounts of data. On the other hand, on-line operations require either predicting all level-slope transitions at every source transition epoch, or temporally storing such transitions and performing calculations at the end of the source activity interval.

In order to simplify the estimation procedure, the following heuristic method was proposed in [22] (see Fig. 15): Let $\{\nu_i\}$ be the sequence of instants such that $\int_{\nu_{i-1}}^{\nu_i} \lambda(t)\,dt = V$ for $i = 1, 2, \ldots$, with V a real positive constant and $\nu_0 = 0$. Then evaluate $\Delta(t)$ on average once "every V volume units sent by the source", that is to say, at times $t = s_i$ such that $\int_{\nu_{i-1}}^{s_i} \lambda(t)\,dt = V - u_i$ and $s_i \in (\nu_{i-1}, \nu_i]$, for $i = 1, 2, \ldots$; where the u_i's form a sequence of i.i.d. random variables, following a uniform law in $[0, u_{max}]$, with $V > u_{max} > 0$.

In the simulator, sampling of waiting time values is perfomed by timestamped *fluid molecule objects*. These objects are produced by the source at times s_i, then travel through the network, being treated by buffers, switches, etc. as described in section 4; when (and if) they arrive to the destination, end-to-end delay is computed using (13) as: $a_{K+1}(s_i) - s_i$.

The parameter V controls the periodicity of the sampling process. Random fluctuations (controlled by the parameter u_{max}) are introduced in this process in order to

[6] Complexity increases if one or more buffers have non-constant service rates $c(t)$.

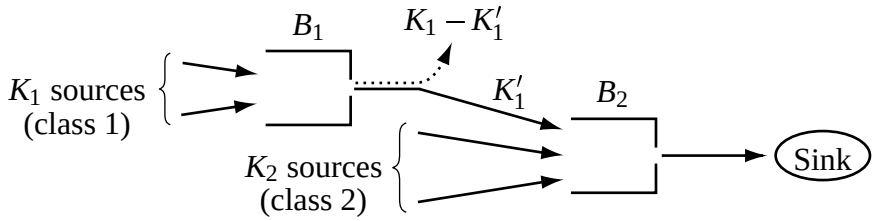

Fig. 16. Example 3: two buffers in tandem.

avoid some undesired bias in the estimation, which may occur if the source rate process is periodical. For a simulation of length T_{sim} and a source with mean rate λ, the method yields about $n_\Delta = \lambda \times T_{\text{sim}}/V$ delay samples, i.e. approx. n_Δ delay-measuring molecules are generated.

6.3.1 Numerical Example.

Next we present an example of delay estimation using the above method. This example, though simple, represents a case of interest for the performance evaluation of existing networks. We are interested in the end-to-end delay per connection of the network shown in Fig. 16.

The parameters chosen for this simulation are:

- $K_1 = 20$ class-1 on/off sources (of which $K_1' = 10$ flow through both buffers), with peak rate = 15 Mbit/s, mean rate = 7 Mbit/s and exponentially distributed on and off periods with mean on period = 0.01 s.
- $K_2 = 10$ class-2 on/off sources, with peak rate = 15 Mbit/s, mean rate = 6 Mbit/s. on and off periods are exponentially distributed and the mean on period is 0.01 s.
- Two identical buffers, with $c = 149.76$ Mbit/s and $B = \infty$.

We performed 30 independent simulations. Each run represented a simulated interval of 10^3 s. Table 2 shows the results concerning the mean delay per class and the mean buffer level. For the first simulation run, the values of V and $u_{\max}$ were 0.6 and 0.1 (in Mbits), respectively, giving $\approx 10^4$ delay samples. To obtain confidence intervals for the statistics, we first computed the autocorrelation $r(k)$ of the sample sequence produced by the first simulation run, then we kept only one sample out of n such that $r(n) \approx 0$. Moreover, we used the first run to tune V such that $r(k) \approx 0, k = 1,2,\ldots$ for the following runs.

Each simulation run involved $O(2.7 \times 10^6)$ events, and took around 15 minutes on a Sun Sparcstation shared between several users. Note that the equivalent number of events that should have been processed by a cell-level simulator is $O(4.7 \times 10^8)$.

Table 2
Example 3: mean delay and mean buffer level.

Class	Mean delay (ms)	95% confidence interval
1	3.488	$(3.448, 3.528)$
2	0.780	$(0.760, 0.799)$
Buffer	**Mean level (kbit)**	**95% confidence interval**
1	393.9	$(390.5, 397.3)$
2	96.41	$(95.73, 97.10)$

7 Conclusions

The simulation tool we have introduced in this paper is particularly well suited to study high-speed telecommunication networks with arbitrary bursty sources. We give some results comparing FluidSim to a well-known simulator, illustrating the gain that can be obtained. From the efficiency point of view, the so-called ripple effect may indeed limit the scalability of the fluid simulation approach, yet this technique offers very attractive performance gains for studying rather complex network topologies.

This tool is devoted to obtaining values of measures that we cannot get by means of analytical approaches, either because we work on the entire network or because we look for sophisticated metrics, or just because the model is not a simple one. Moreover, the flexibility offered by FluidSim and the inclusion of molecule objects in the fluid model make this tool powerful and general enough to evaluate complex protocol mechanisms in an efficient manner.

The tool is written in `C++`, which allows in particular to easily extend it to deal with new architectures. We are currently working on some extensions, namely to deal with TCP's flow control algorithm.

Acknowledgements

We would like to thank the anonymous reviewers for all their suggestions and remarks that helped us improve the quality of the paper.

References

[1] D. Anick, D. Mitra, and M. M. Sondhi. Stochastic Theory of a Data-Handling System with Multiple Sources. *Bell System Technical Journal*, 61(8), pages 1871–1894, October 1982.

[2] U. Breymann. *Designing Components with the C++ STL—A new approach to programming.* Addison Wesley Longman, 2000.

[3] J. Ferrandiz. Analysis of Fluid Models with Markov Modulated Rates. Technical report HPL-92-101, Hewlett-Packard Laboratories, August 1992. `http://www.hpl.hp.com/techreports/92/HPL-92-101.ps`.

[4] G. Fishman. *Monte Carlo.* Springer, 1996.

[5] N. Golmie, F. Mouveaux, L. Hester, Y. Saintillan, A. Koenig, and D. Su. *The NIST ATM/HFC Network Simulator—Operation and Programming Guide, version 4.0*, December 1998.

[6] Z. Haraszti and J. K. Townsend. *Rare Event Simulation of Delay in Packet Switching Networks using DPR-based Splitting.* In *Proceedings of the 2nd International Workshop on Rare Event Simulation*, pages 185–190, Enschede, The Netherlands, March 1999.

[7] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design.* John Wiley & Sons, 1991.

[8] R. Jain, S. Kalyanaraman, S. Fahmy, R. Goyal, and S. Kim. Source Behavior for ATM ABR Traffic Management: An Explanation. *IEEE Communications Magazine*, 34(11):50–57, November 1996. `ftp://netlab.ohio-state.edu/pub/jain/papers/src_rule.ps`.

[9] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and R. Viswanathan. ERICA Switch Algorithm: A Complete Description. Contribution 96-1172, ATM Forum, August 1996. `ftp://ftp.netlab.ohio-state.edu/pub/jain/atmf/atm96-1172.ps`.

[10] P. R. Jelenkovic, A. A. Lazar, and N. Semret. Multiple Time Scales and Subexponentiality in MPEG Video Streams. In *Proceedings of the International IFIP-IEEE Conference on Broadband Communications*, Montreal, Canada, April 1996. `ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/96/JEL96.ps.gz`.

[11] G. Kesidis. *ATM Network Performance*. Kluwer Academic Publishers, 1996.

[12] G. Kesidis, A. Singh, D. Cheung, and W.W. Kwok. Feasibility of Fluid Event-Driven Simulation for ATM Networks. In *Proceedings of IEEE Globecom'96*, Vol. III, pages 2013–2017, London, November 1996. `http://cheetah.vlsi.uwaterloo.ca/~kesidis/fluid.ps`.

[13] L. Kleinrock. *Queueing Systems*, Vol. II. John Wiley & Sons, 1975.

[14] V. G. Kulkarni. Fluid Models for Single Buffer Systems. In J. H. Dshalalow, editor, *Frontiers in Queueing: Models and Applications in Science and Engineering*, chapter 11, pages 321–338. CRC Press, 1997.

[15] K. Kumaran and D. Mitra. Performance and Fluid Simulations of a Novel Shared Buffer Management System. In *Proceedings of IEEE INFOCOM'98*, March 1998. `http://cm.bell-labs.com/cm/ms/who/mitra/papers/infocom.ps`.

[16] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong. Fluid Simulation of Large Scale Networks: Issues and Tradeoffs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, volume IV, pages 2136–2142, Las Vegas, 1999.

[17] D. M. Nicol, M. Goldsby and M. Johnson. Fluid-Based Simulation of Communication Networks Using SSF. In *Proceedings of the 1999 European Simulation Symposium*, October 1999. `http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/papers/ESS99-fluid.pdf`

[18] J. M. Pitts, L. G. Cuthbert, M. Bocci and E. M. Scharf. *Cell Rate Modelling: An Accelerated Simulation Technique for ATM Networks.* In *9th U.K. Performance Engineering Workshop for Computer and Telecommunications*, pages 94–106, July 1993.

[19] J. M. Pitts, L. G. Cuthbert, M. Bocci and E. M. Scharf. *An Accelerated Simulation Technique for Modeling Burst Scale Queueing Behaviour in ATM.* In *Proceedings of the 14th International Teletraffic Congress*, pages 777–786, June 1994.

[20] J. Roberts, U. Mocci, and J. Virtamo, editors. *Broadband Network Teletraffic: Performance Evaluation and Design of Broadband Multiservice Networks—Final Report of Action COST 242*. Number 1155 in Lecture Notes in Computer Science. Springer, 1996.

[21] D. Ros. *Study of High-Speed Networks via Fluid-Model Simulation.* Phd thesis, Institut National des Sciences Appliquées (INSA) de Rennes, France, January 2000. (In French).

[22] D. Ros and R. Marie. Estimation of End-to-End Delay in High-Speed Networks by Means of Fluid Model Simulations. In *Proceedings of the 13th European Simulation Multiconference*, Vol. 1, pages 375–381, Warsaw, June 1999.

[23] D. Ros and R. Marie. Loss Characterization in High-Speed Networks Through Simulation of Fluid Models. To appear in *Telecommunication Systems*, February 2001.

[24] B. Stroustrup. *The C++ Programming Language*. Addison–Wesley, 1998.

[25] A. Tanenbaum. *Computer Networks*. 3rd Ed., Prentice Hall International, 1996.

[26] X. Xiao and L. M. Ni. Internet QoS: A Big Picture. *IEEE Network Magazine*, 13(2):8–19, March–April 1999.

[27] A. Yan and W. Gong. Fluid Simulation for High Speed Networks. University of Massachusetts, Amherst Technical report No. TR-96-CCS-1, 1996. `ftp://soccer.ecs.umass.edu/pub/yan/papers/fldnet.ps.Z`