Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics, Istanbul, Turkey, May 27-29, 2006 (pp403-408)

A fluid model of the RED AQM algorithm and its implementation in a fluid-based network simulator

JOSE INCERA, LUIS CARBALLO
Instituto Tecnológico Autónomo de México
División Académica de Ingeniería
Río Hondo No. 1, Col. Tizapán Sn. Ángel
México D. F., C.P. 01000
MÉXICO

Abstract: The simulation of communication networks using a continuous-state, or *fluid modeling* approach, has shown to be very efficient for a wide range of performance evaluation scenarios. In this paper we present a fluid model of the RED active queue management algorithm for FluidSim, a simulation tool based on the fluid paradigm. We compare the behavior and efficiency of our model against the results provided by NS, a well known packet-based network simulator. The proposed model does capture RED's characteristics with acceptable precision providing good accelerations in typical network evaluation configurations.

Key–Words: Fluid simulation, RED, Active Queue Management

1 Introduction

Discrete-event simulation is the most widespread technique for analyzing the behavior, doing performance evaluations, or designing new communication networks because of its flexibility and its capability for representing virtually every possible mechanism. Unfortunately, this technique may be extremely costly in terms of computing power. Consider for instance a high-speed network being traversed by millions or billions of packets. The classical approach would represent every packet as it makes its way through the (models of the) network elements. One way of dealing with this problem consists in replacing the discrete packet representation by continuous-state models (fluid models) that describe the instantaneous flow rate as it goes from one *container* to another. This can lead to a significant reduction in the computational effort. Indeed, when a burst of packets is emitted (as it often occurs), instead of handling each individual unit, it suffices here to manage only two events: the beginning of the burst and its end. This is the approach followed in FluidSim, a discrete-event simulation tool based on fluid models of communication network objects [5].

A fluid model of the TCP protocol for FluidSim was presented in [6]. Another important mechanism that was lacking in [5] (and, to the best of our knowledge, in any other fluid simulation framework) is the Random Early Detection (RED) Active Queue Management (AQM) algorithm [2]. AQM mechanisms manage queue lengths by dropping (or marking) packets when congestion is building up, that is, before the queue is full. End-systems can then react to such losses by reducing their packet rate, hence avoiding severe congestion. RED intends to avoid congestion by randomly discarding packets based on the average queue size. End-systems can then react to such losses by reducing their packet rate.

While many analytical fluid models of RED have been proposed, the fluid approach has seldom been used for evaluating the performance of RED queues by simulation. This is somewhat surprising given the importance of this mechanism and the interest in fluid simulation [5, 7, 8]. In this paper we present a fluid model of the RED algorithm for FluidSim. Our goal is to evaluate the behavior and performance of RED queues with a precision similar to that offered by the widely used NS packet level simulator but taking benefit of the advantages offered by the fluid approach.

The structure of the paper is as follows. We present in section 2 the fluid model's dynamics of the fundamental network components defined in the FluidSim network simulator. In section 3 we briefly describe the RED algorithm and our fluid representation of this mechanism is introduced in section 4. Several examples conceived to evaluate the fidelity, efficiency and potential of our proposal are the subject of section 5 where our results are compared against those obtained by simulating similar scenarios with NS. Our conclusions and some guidelines for future work close the paper in section 6.

2 Fluid simulation

Let us consider a fluid buffer of capacity $B \leq \infty$, with constant service 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 $t \geq 0$. The volume of fluid arriving in the interval $[0, t]$ is given by: $A(t) \triangleq \int_0^t \Lambda(u)\,du$.

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

$$Q(t) = Q(0) + \int_0^t (\Lambda(s) - c)\,\mathbf{1}_{\{s \in \mathcal{Q}\}} ds, \qquad (1)$$

where, for an infinite-capacity buffer, the set $\mathcal{Q}$ is given by [10]: $\mathcal{Q} = \{s \geq 0 \,|\, \Lambda(s) > c \text{ or } Q(s) > 0\}$, and for a finite-capacity buffer, $\mathcal{Q} = \{s \geq 0 \,|\, (\Lambda(s) > c \text{ or } Q(s) > 0) \text{ and } (\Lambda(s) < c \text{ or } Q(s) < B)\}$. We are interested only in arrival processes in which every sample path $\Lambda(t)$ is a stepwise function. Therefore, equation (1) is reduced to:

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

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

The output rate of the buffer at $t \geq 0$ is:

$$R(t) = \begin{cases} c & \text{if } Q(t) > 0 \text{ or } \Lambda(t) > c, \\ \Lambda(t) & \text{if } Q(t) = 0 \text{ and } \Lambda(t) \leq c. \end{cases} \qquad (3)$$

The model described so far can be applied to the more general case where the buffer is fed by N fluid flows. Let $\lambda_i(t) \in [0, \infty)$ be the rate of the i-th flow at time t. We denote $\boldsymbol{\lambda}(t) \triangleq (\lambda_1(t), \ldots, \lambda_N(t))$, and we call this the *input flow vector*. The *total* input rate is $\Lambda(t) = \sum \lambda_i(t)$. Similarly, $r_i(t)$ is the output rate related to the i-th input fluid at time t, $\mathbf{r}(t) \triangleq (r_1(t), \ldots, r_N(t))$ is the *output flow vector* and $R(t) = \sum r_i(t)$ is the *total* output rate.

Let τ_n be the n-th transition epoch of $\boldsymbol{\lambda}(t)$. Because of the FIFO service discipline, a change in $\boldsymbol{\lambda}(t)$ 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 $\omega_n \triangleq \tau_n + Q(\tau_n)/c$, the proportion of output components must be the same as the proportion of input components.

The discrete-event simulation tool FluidSim, is based on the fluid paradigm and has models of these fluid objects. For it to evaluate fairly complex network topologies, it also incorporates the following components:
Sinks are destination nodes where the arriving flow is simply absorbed and possibly some statistics are collected. *Multiplexers* are network nodes composed of one or more buffers. Their function is 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.
Communication links connect two network components. They are unidirectional elements that introduce a constant delay $d \geq 0$ to every flow traversing them.
Switching matrices are simply a mapping between two sets of elements. Their function is to separate the incoming aggregated flow vectors (*demultiplexing*) and to create new flow vectors at their output(s) (*multiplexing*) according to a routing table.
Fluid molecules can be used to compute some performance metrics; they can also represent the behaviour of individual entities such as TCP's ACK packets.

3 Random Early Detection

In general, a packet arriving to a router will be placed on a buffer until it can be sent away. If the input rate to a particular interface is greater than its capacity to release packets, a queue begins to build up in the buffer. If it gets full, arriving packets will be discarded. This behavior, known as tail-drop, has shown to be highly inefficient. A better way to deal with this congestion event is by applying an Active Queue Management policy within the buffer. The best known and thoroughly studied AQM mechanism is RED [2]. This algorithm avoids congestion by controlling the average queue size $\hat{q}$ and comparing it to two thresholds, min_{th} and max_{th}. See figure 1. Inside this region, packets are discarded[1] with a probability $0 \leq p \leq max_p$ given by a linear function of the average queue size. While $\hat{q}$ exceeds max_{th} all arriving packets are discarded. Keep in mind, however, that the criteria is based on the average queue size, so RED may accept temporal data bursts.

[1] or marked as low priority if a QoS policy is in place. In this paper we will only refer to dropped packets.

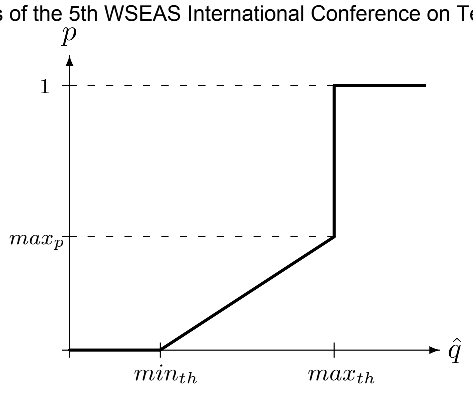

Figure 1. RED **dropping dynamics**

In order to discard packets with a growing probability while the queue is building up and still be able to absorb packet bursts, RED computes an *exponential weighted moving average* queue value based on an *absortion factor* w_q. This factor determines the relative importance that must be given to the average queue value ($w_q \rightarrow 0$) with respect to the *instantaneous* measured queue level ($w_q \rightarrow 1$) according to the following formula:

$$\hat{q} = (1 - w_q) \cdot \hat{q}_{\text{old}} + w_q \cdot Q \tag{4}$$

where Q is the instantaneous queue value and $\hat{q}_{\text{old}}$ is the previous computed average. The recommended value for w_q is 0.002.

4 Implementation

It is fairly easy to implement the RED algorithm in a conventional discrete-event simulator (or in a real switching node), since on the arrival of every packet, we can compute the new value of $\hat{q}$ according to (4) and from it, the discard probability for that particular packet.

In the fluid paradigm, however, things are much more complex because we have lost the packet-level scale. Consider the case in which $\Lambda(u) = k > c$ in $[s, t]$ such that the queue is building up. In the fluid simulator there would only be (in principle) two events at s and t that correspond to the rate changes in Λ. If we only compute $\hat{q}$ at these points, our weighted average would not accurately reproduce RED's behavior. Since the flow rates in FluidSim can be interpreted in packets per second, we obtain a better estimation of $\hat{q}$ by computing it several times in $[s, t]$ according to the following algorithm:

```
m = int((now-lastTime)×OutRate);
while (--m)
    compute (4)
```

where *now* is the current simulation time; *lastTime* is the last event execution time and *m* is the number of packets that would be served in $[s, t]$.

For the computation of the weighted average, the queue idle times (that is, the periods of time when the queue is empty) have to be taken into account as well.

The discarding probability algorithm was based on [2]. We establish first how many *packets* would have traversed in that interval and then we apply the probabilistic algorithm to determine whether to discard or not a particular packet. In the former case, we drop an amount of fluid equivalent to a packet. The details can be found in [1]. The fluid to be dropped must belong to a specific flow which is selected according to the proportion of bandwidth each flow is using.

Finally, we have to determine when to evaluate the previous algorithms. Since we cannot rely on the events produced by the flow's dynamics, we program a future event when we estimate (using a minimum squares aproximation) the queue level will traverse a certain threshold as shown in figure 2.

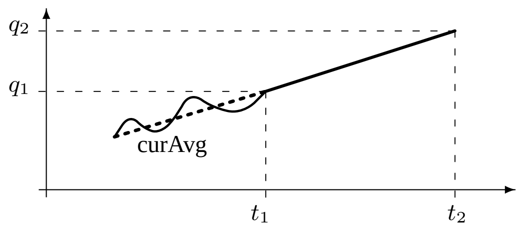

Figure 2. Minimum squares queue estimation

5 Model evaluation

In order to validate our model, we evaluated different network configurations with two simulators: FluidSim equipped with the proposed model, and NS [9]. We have chosen NS because, given its popularity, its models have been intensively validated by the research community [3, 4].

5.1 A single TCP flow

We begin by studying the simple bottleneck configuration (figure 3) in order to identify some of the model's characteristics.

The source sends packets to its destination through a node representing the connection's bottleneck and modelled by a queue of capacity $B = 40$ packets and service rate $c = 100$ packets/s. The round trip time is $RTT = 500\,ms$. The service discipline

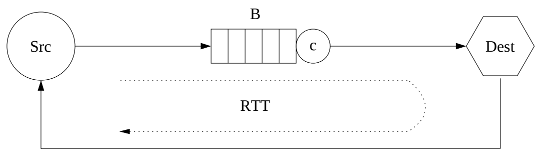

Figure 3. Bottleneck model

is RED with parameters $min_{th} = 15$, $max_{th} = 30$, $P_{max} = 1/50$.

Figure 4 shows the evolution of *cwnd* for both, FluidSim and NS.

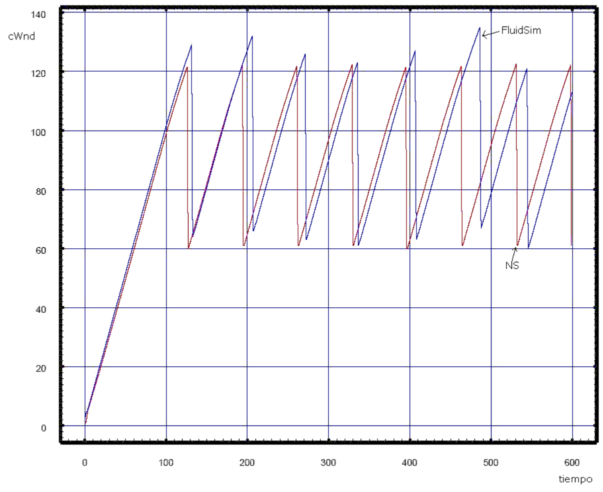

Figure 4. *cwnd* **evolution. One TCP flow**

The long initial *cwnd* growth is related to a particularity of FluidSim already reported in [6]. While our model seems to faithfully reproduce the evolution of *cwnd*, there are a few points that deserve some discussion. In particular, we can see that sometimes (see for instance the interval $[400, 500]$) *cwnd* grows bigger in FluidSim. This is because the discard probability algorithm is evaluated more often in NS, thus slightly less losses are induced in our model.

5.2 Two TCP flows

In our second scenario, two TCP flows share the bottleneck router before arriving to their destinations. The network parameters are those defined in the previous configuration. Src2 begins its flow at time $t = 100\,s$.

We present in figure 5 the evolution of the congestion window for both, NS and FluidSim. The plots in this figure are rather irregular. What we are seeing is RED's ability to break the synchronization of TCP flows. Shortly after the second source gets activated, packets (*fluid*) from both flows are lost (points 1 and 2 show the losses for Src1 in NS and FluidSim respectively). From that point onwards, the losses are no longer synchronized, as can be seen in points $3, 4, 5$ and 6.

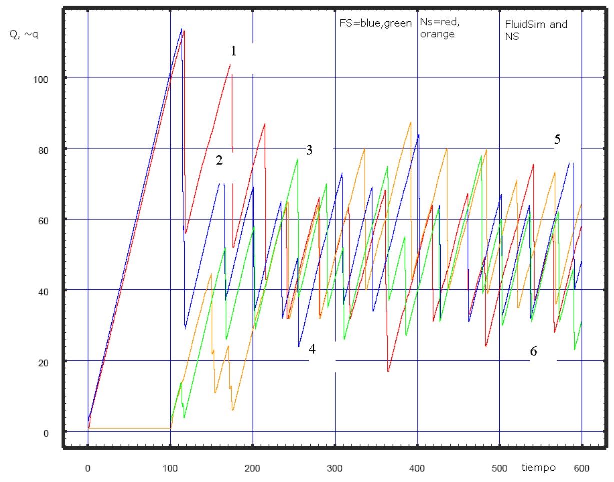

Figure 5. cWnd evolution for the scenario with two flows

As we have mentioned erlier, another property of RED is that it keeps a lower queue occupation. In order to test this behavior, we repeated the previous experiment using an intermitent flow for Src2. It is active in the intervals $[50, 250]$ and $[400, 600]$. The buffer size B was changed to 60 packets. The curves for Q and $\hat{q}$ obtained with and without RED are respectively shown in figures 6 and 7. The mean queue occupation measured with RED was around 18.20 packets, which is consistent with the fact that $min_{th} = 15$ and $max_{th} = 30$. When we turned off RED, the mean occupation grew up to 43.09 packets.

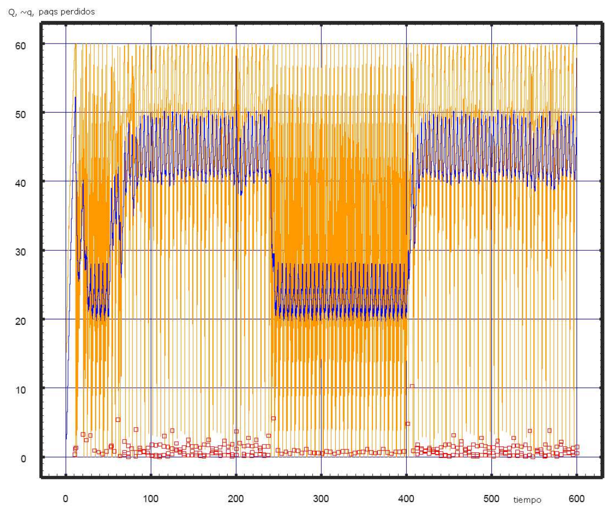

Figure 6. Q and $\hat{q}$ levels with no RED

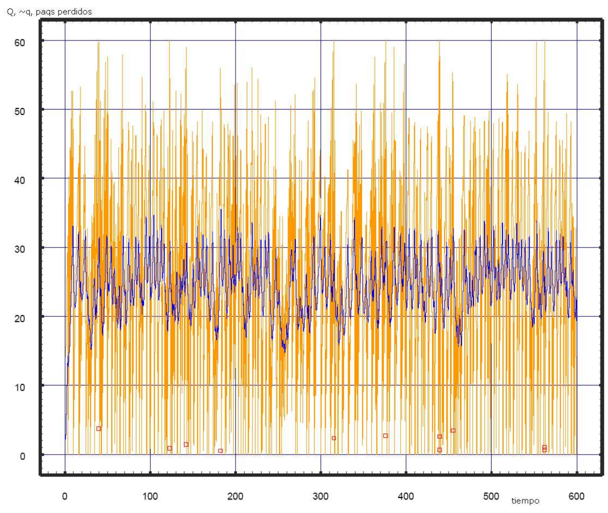

Figure 7. Q and $\hat{q}$ levels with RED activated

5.3 Efficiency

Besides the validity of the results, it is important to compare the efficiency of the fluid model with respect to NS. The efficiency measures we have selected are the *gain* and the *speed-up*, formalised as follows. Let us consider a model $\mathcal{M}$ to be analysed in the time interval $[0, T]$. We define the simulator's *event generation rate* ξ as the number of executed events while studying $\mathcal{M}$ over $[0, T]$, divided by T. We define the *gain* $\mathcal{G}$ of FluidSim (with respect to NS) as the ratio of the event generation rate in both simulators. The advantage of this measure is that it does not depend on the internal characteristics of the simulator (e.g. the type of scheduler). In this sense, it quantifies the efficiency of the fluid paradigm.

The *speed-up* $\mathcal{S}$ is the ratio between the execution time of NS and that of FluidSim when executing $\mathcal{M}$ over $[0, T]$. While this measure is influenced by each simulator's internal details, it is nonetheless relevant because it is the one directly perceived by the users.

The experiments reported below were carried out on an Intel Pentium $660\,MHz$ processor with 256 MB RAM running Linux Fedora v3. We ran the simulations several times; the execution times indicated are simply the averages of the real time observed[2].

Tables 1 and 2 resume the results obtained when simulating the single connection topology shown in figure 3.

We confirm that the fluid model generates far less events than the packet-level approach, so we obtain interesting gains with FluidSim. The speed-up computed is also no negligible even though we have not optimised our implementation. It grows with the simulation time since the advantages of the fluid paradigm are cumulative in this topology.

[2] Our main objective is to estimate the model's feasibility rather that producing very accurate results; therefore, the results presented here should only give a general idea of the accelerations the fluid paradigm could offer.

Simulation	Execution time [s]		No. of events ($\times 10^3$)	
time	NS	FluidSim	NS	FluidSim
600	3.095	0.119	205.32	6.475
1000	4.553	0.173	343.21	10.753
2000	8.992	0.305	726.24	21.447
5000	22.156	0.694	1852.47	53.657
10000	45.959	1.341	3766.07	107.307
15000	66.555	1.990	5623.23	161.102
20000	88.469	2.632	7503.22	214.666

Table 1. Performance values for the single TCP connection topology

Simulation time	ξ NS	ξ FluidSim	$\mathcal{S}$	$\mathcal{G}$
600	342.20	10.79	26.01	31.71
1000	343.21	10.75	26.32	31.92
2000	363.12	10.72	29.48	33.86
5000	370.49	10.73	31.93	34.52
10000	376.61	10.73	34.27	35.10
15000	374.88	10.74	33.44	34.90
20000	375.16	10.73	33.61	34.95

Table 2. Efficiency obtained for the single TCP connection topology

The efficiency data obtained for the two connections topology are presented in table 3.

Simulation time	ξ NS	ξ FluidSim	$\mathcal{S}$	$\mathcal{G}$
600	403.44	27.90	22.53	14.46
1000	412.86	28.36	15.89	14.56
2000	423.14	29.20	16.85	14.49
5000	427.48	29.50	17.82	14.49
10000	428.59	29.83	16.67	14.37
15000	428.85	29.91	15.51	14.34
20000	429.23	29.83	17.05	14.39

Table 3. Efficiency obtained for the topology with two TCP connections

We still observe good performance measures and an order of magnitude acceleration. However, if we compare this data against that obtained for the single connection, we can do two interesting observations. Firstly, the event execution rate has augmented much more in `FluidSim`. This is because the interactions between the connection's flows induce several rate changes that are traduced in supplementary events in the fluid paradigm. In `NS` of course there are more events since there are more packets, but the execution rate growth is less important. Secondly, the speed-up obtained is now decreasing with the simulation time. This is an indication that the flow rate change related events are far more complex to process than those related with the packet treatment.

6 Conclusions

In this paper we have introduced a model of the `RED` algorithm for the object library of `FluidSim`, a fluid based discrete-event simulator. We evaluated the fidelity of our model and the efficiency of the fluid paradigm by comparing similar topologies in both, `FluidSim` and `NS`. When designing the model, we have paid special care on its fidelity. The results presented show that this goal has indeed been achieved.

With regards to the efficiency, we obtained important and promising accelerations in the simple topologies evaluated. However, we have confirmed that these results depend heavily on the complexity of the topology, in particular due to the interactions between the different flows.

While further experiments most be done in order to delimit the model's (and the fluid paradigm's) advantages and limitations, we would prefer to do so after refining the model. To be more explicit in this point, the ideal values for some parameters, like the periodicity with which future `RED` related events should be programmed, have not been deeply evaluated.

References:

[1] Luis Carballo Arévalo. Implementación del algortimo RED en un simulador de red basado en el paradigma de simulación fluida. Master's thesis, ITAM, 2005.

[2] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[3] Sally Floyd. Internet simulations: Issues in defining the model. Presentation at DIMACS Workshop on End-toEnd Modeling and Simulation. `http://www.aciri.org/floyd/talks/sf-dimacs-97.pdf`, October 1997.

[4] John Heidemann, Kevin Mills, and Sri Kumar. Expanding confidence in network simulation. DARPA/NIST Network simulation validation workshop, May 1999.

[5] J. Incera, R. Marie, D. Ros, and G. Rubino. Fluidsim: a tool to simulate fluid models of high-speed networks. *Performance Evaluation*, 44(1–4):25–49, 2001.

[6] J. Incera and G. Rubino. Fluid simulation of TCP flows. In B. R. Haverkort, editor, *Proceedings of the 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB'01*, Aachen, Germany, sep 2001.

[7] 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`.

[8] Benyuan Liu, Yang Guo, Jim Kurose, Don Towsley, and Weibo Gong. Fluid simulation of large scale networks: Issues and tradeoffs. In H. R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'99*, volume IV, pages 2136–2142, Las Vegas, USA, June 1999.

[9] UCB/LBNL/VINT Network Simulator - ns (version 2). `http://www.isi.edu/nsnam/ns/`.

[10] J. Roberts, U. Mocci, and J. Virtamo, editors. *Broadband Network Traffic: Performance Evaluation and Design of Broadband Multiservice Networks-Final Report of Action COST 242*, number 1155 in Lecture Notes in Computer Science. Springer Verlag, 1996.