Results on fluid modelling of packet switched networks* (extended version)

Technical report Laboratoire Heudiasyc UMR CNRS 6599

Fabien Chatté[†], Bertrand Ducourthial[‡], Silviu-Iulian Niculescu [§]

Laboratoire Heudiasyc UMR CNRS 6599
Université de Technologie de Compiègne,
Centre de Recherche de Royallieu,
BP. 20529, 60205 Compiègne CEDEX, France.

Mars 2002

Abstract

This paper focuses on the use of fluid modelling of packet switched networks. A simplified network protocol as well as the corresponding exact (continuous-time) fluid modelling are simulated. Fluid and discrete simulations are compared in different situations and with different parameters, in order to study the influence of the protocol and some network parameters on the matching of the results. Conditions that ensure accurate continuous simulations of packet switched networks are then identified.

Keywords: network modelling, packet switched network, fluid modelling, continuous systems, simulations.

*A short version has been presented at IFAC Worksop on Time Delay System, Santa Fee, December 2001.

[†]e-mail: `Fabien.Chatte@hds.utc.fr`
[‡]e-mail: `Bertrand.Ducourthial@hds.utc.fr`
[§]e-mail: `Silviu.Niculescu@hds.utc.fr`

1 Introduction

The actual development of Internet and communication networks with the need to support interactive applications (which are time constrained), makes the *congestion* an important and critical problem. There exists a lot of methods in the literature for controlling this phenomenon, and we shall not insist on them (see *e.g.* [5, 3, 10, 7, 1, 4]). The main idea consists in controlling the sending rate on the base of a feedback from the destination, in order to avoid congestion that could appear somewhere in the network between the source and the destination.

To obtain simpler models and analysis techniques, we focus on the *fluid approximation* of the networks packets model (see, for instance, [2, 6]). Indeed, the interest of such a continuous network model lies in its simplicity. One of the simplest fluid modelling is given by a chain of integrators with an *uncertain* delay in the input for a source-destination pair of nodes. The number of integrators in the chain corresponds to the number of congestion nodes in the corresponding path (from the source to the corresponding destination). Although the uncertainty in the delay makes the problem more complicated, the advances in robust analysis and control of uncertain system including or not delays [8, 11] allow to be optimistic for improving the congestion control in packets switching networks.

Using a continuous framework for the study and the control of a packet-switched network leads to two main problems. The first one is the *accuracy* of the fluid modelling. The second one is the *equivalence* of discrete and fluid results, depending on network and protocol parameters.

This paper focuses on the second problem, and deals with the *limitations* of the use of the fluid modelling of a discrete protocol with congestion control. To avoid any problem due to some imperfections of the modelling, a simplified protocol is used, for which a continuous fluid model is relatively easy to implement. Simulations using the packet simulator *ns* and *matlab* are discussed for one, two, three, five and ten sources, and one destination. A simple qualitative criterion (*closeness*) is proposed to compare fluid and discrete results. Conditions on the use of the fluid modelling are then given.

The protocol is presented in Section 2, as well as its fluid modelling. Discrete and fluid simulations are discussed in Section 3. Concluding remarks end the paper.

2 Protocol

In this section we present the protocol used in this study.

2.1 Definition and notations

A source node sends data packets to a destination node. The destination node sends back to the source some acknowledgement packets. It is assumed that, at the beginning of the communication, the network is empty (there is no packet in the network) and there is no congestion. Each data packet sent by the source contains a time-stamp in its header (sending date). The destination node copies this time-stamp into the acknowledgment packet before sending it back to the source node. The source can then compute the *round trip time* of the ith packet, denoted as $\texttt{rtt}_i$) by:

$$\texttt{rtt}_\texttt{i} = \text{receive-time} \; - \; \text{time-stamp in ack}$$

The first round-trip-time, computed when the network was still empty, denoted as `rtt_min`, is used as a *reference delay*. We have $\texttt{rtt}_\texttt{i} \geq \texttt{rtt\_min}$ for all further packets i. The source rate increases linearly if

$$\texttt{rtt}_\texttt{i} \leq \texttt{rtt\_min} + \texttt{T}$$

where `T` is a fixed time delay, called *tolerance*. The source rate decreases exponentially if

$$\texttt{rtt}_\texttt{i} > \texttt{rtt\_min} + \texttt{T}$$

Notice that congestion is due to limit bandwidth of links and not to processing power of nodes or routers. It is assumed that there is exactly *one bottleneck link* in the path from the source to the destination and no bottleneck on the way back to the source. The propagation delay `Df` between the source and the bottleneck link, and the delay `Dr` from the bottleneck back to the source are assumed constant (see Figure 1). We call *congestion node* the initial extremity of the bottleneck link (node C). The bottleneck has a fixed rate of μ *kbits/s*, and the size of the queue is infinite (to avoid the packets loss). All data packets sent by the source have the size s, fixed to 1000 bytes.

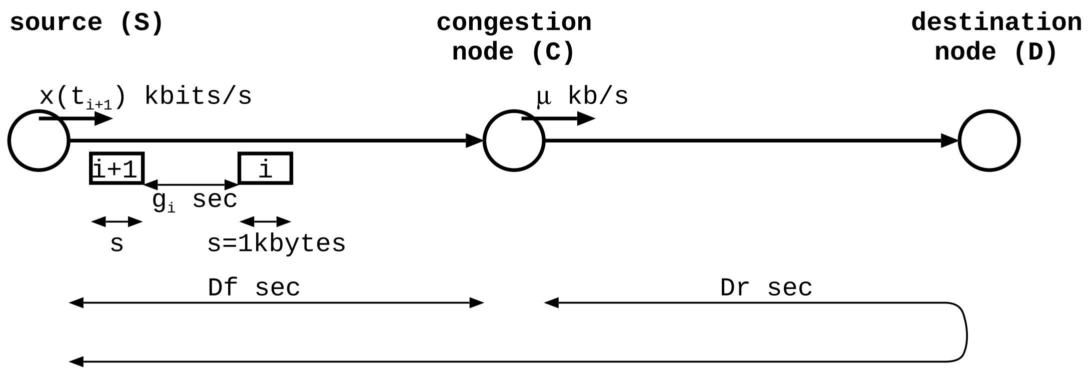

Figure 1: Protocol overview.

The source controls its sending rate $x(t_i)$ by adjusting the inter-packet interval, called g_i. Let t_i be the date at which the packet i is sent. We have:

$$g_i = t_{i+1} - t_i = \frac{s}{x(t_i)}$$

The protocol has a minimal rate of 3000 $bits/s$ in order to avoid a dead locked situation:

$$x(t_i) \geq 3 \; kbits/s, \forall t_i$$

Indeed, if the rate falls close to 0 bit/s, an infinite gap will appear. In this case, the source will not be advertised on the end of the congestion because no packet will be sent and thus no acknowledgements received.

The tolerance `T` can be used to manage the average queue size. For example, if $\texttt{T} = s/\mu$, the presence of one packet in the queue of the bottleneck node C is allowed. Hence, the source rate will enter later in the exponentially decreasing phase.

2.2 Discrete implementation protocol

The simulations of the protocol have been done with the packet simulator *ns* (network simulator) version 2.1b8a [9].

The algorithm used to implement the protocol is presented below. Three parameters are necessary: linear increasing factor α, exponential decreasing factor β, and the tolerance `T`. The larger α is, the faster the rate increases (in an increasing phase). The smaller β is, the faster the rate decreases (in a decreasing phase). In an increasing phase, the instantaneous rate is computed using α, the previous minimal rate `min_rate` (at the beginning of the current increasing phase), and the date `t_update_min` at which it has been computed. Similarly, the instantaneous rate is computed with β, `max_rate`, and `t_update_max` in an decreasing phase. Phases are distinguished due to the variable `phase`. The C++ code used to implement the protocol in *ns* is presented in appendix A.

Algorithm 1: Protocol(α, β, T)

```
phase = increasing
absolute_min_rate = 3000
min_rate = absolute_min_rate
t_update_min = 0
max_rate = absolute_min_rate
t_update_max = 0
rate = min_rate
rtt_min = 0
rtt = rtt_min
S = 1000

sending():
  t = date
  send( S bytes, t ) to the destination node
  if rtt > rtt_min + T then
    ▷ Sending rate must decrease.
    if phase == increasing then
      ▷ Entering in decreasing phase.
      phase = decreasing
      max_rate = rate
      t_update_max = t
    end if
    rate = max( max_rate × exp( (t_update_min - t) / β ) , absolute_min_rate )
  else
    ▷ Sending rate must increase.
    if phase == decreasing then
      ▷ Entering in increasing phase.
      phase = increasing
      min_rate = rate
      t_update_min = t
    end if
    rate = min_rate + α × (t - t_update_min)
  end if
```

```
    call sending() at date t + S / rate
receiving():
    t = date()
    if packet == data then
        ▷ Reception of a data packet by the destination node.
        receive( S bytes, sending_date )
        send( sending_date )  to the source node
    else
        ▷ Reception of an acknowledgment by the source node.
        receive( sending_date )
        if rtt_min == 0 then
            rtt_min = t - sending_date
        else
            rtt = t - sending_date
        end if
    end if
```

2.3 Fluid modelling

The continuous simulations have been done with *matlab*.

The fluid modelling is composed of two time-continuous functions $q(t)$ and $x(t)$: $q(t)$ denotes the bottleneck queue size at time t and $x(t)$ gives the source sending rate at time t. When a data arrives at the bottleneck at time t, its sending rate is equal to $x(t-\texttt{Df})$. Indeed, it was sent by the source at time $t-\texttt{Df}$ (see Figure 1). Data are sent from the bottleneck node with a sending rate equal to μ if the queue is not empty, and a sending rate equal to $x(t-\texttt{Df})$ if the queue is empty (a short time before) and $x(t-\texttt{Df}) \leq \mu$.

Thus we have:

- if $q(t-\epsilon)=0$ and $x(t-\texttt{Df}) \leq \mu$ then $q(t)=0$
- else, $q(t)=\int_{t_{\text{queue not empty}}}^{t}(x(t-\texttt{Df})-\mu)dt$

where $t_{\text{queue not empty}}$ is the moment at which the queue becomes not empty:

$$x(t_{\text{queue not empty}}-\texttt{Df})=\mu$$

A data spends $q(t)/\mu$ time into the queue, where t is the time at which the data reaches the queue. We denote as `queuing_time` the function that gives the time spent by a packet in the queue function of its leaving moment:

$$\texttt{queuing\_time}(t+q(t)/\mu)=q(t)/\mu$$

. If an acknowledgement is received at time t by the source, it means that the corresponding packet left the queue at the moment $t-\texttt{Dr}$, which means that it arrived in the queue at $t-\texttt{Dr}-\texttt{queuing\_time}(\texttt{t}-\texttt{Dr})$.

Then, at the arrival of an acknowledgment at date t, if

$$q(t-\texttt{Dr}-\texttt{queuing\_time}(\texttt{t}-\texttt{Dr}))/\mu > \texttt{T}$$

the sending rate of the source must be reduced exponentially, and if

$$q(t - \mathtt{Dr} - \mathtt{queuing\_time(t - Dr)})/\mu \leq \mathtt{T}$$

it must increase linearly.

Hence, we have:

- if $(q(t - \mathtt{Dr} - \mathtt{queuing\_time(t - Dr)})/\mu) \leq \mathtt{T}$ then $x(t) = \mathtt{min\_rate} + \alpha \times (t - t_{\mathtt{update\_min}})$
- else, $x(t) = max\left(\mathtt{max\_rate} \times exp(\frac{t - t_{\mathtt{update\_max}}}{\beta}), 3000\right)$

with $t_{\mathtt{update\_min}} = t_{\text{waiting}\leq\mathtt{T}} + \mathtt{Dr}$ and $\mathtt{min\_rate} = x(t_{\text{waiting}\leq\mathtt{T}} + \mathtt{Dr})$, $t_{\mathtt{update\_max}} = t_{\text{waiting}>\mathtt{T}} + \mathtt{Dr}$ and $\mathtt{max\_rate} = x(t_{\text{waiting}>\mathtt{T}} + \mathtt{Dr})$, where $t_{\text{waiting}\leq\mathtt{T}}$ and $t_{\text{waiting}>\mathtt{T}}$ are defined by:

$$\begin{array}{ll} \mathtt{queuing\_time}(t_{\text{waiting}\leq\mathtt{T}}) \leq \mathtt{T} & \text{and} \\ \mathtt{queuing\_time}(t_{\text{waiting}>\mathtt{T}}) > \mathtt{T}, & \text{respectively} \end{array}$$

Initially, $\mathtt{min\_rate} = \mathtt{max\_rate} = 3000$, $t_1 = t_{\mathtt{update\_min}} = t_{\mathtt{update\_max}} = 0$, and the queue is empty ($q(0) = 0$).

The code used to implement the fluid model with *matlab* is presented in appendix B.

2.4 Comparison of the fluid and discrete results

The source rate is periodic in the fluid model, and it is quasi-periodic in the discrete case. The comparison of the periods of the two models should then give pertinent information. In order to compare the periods, an average discrete_period has been computed over a simulation of 300 s:

$$\text{discrete\_period} = \frac{\sum \text{period\_duration}}{\text{nb\_periods}}$$

Note that the comparison of the periods is a good indicator of the closeness of the discrete and fluid curves, but it does not take care about phases. Hence, if the periods are close, the only difference that can exist between the models is the phase. In particular, when the periods are close each other, min_rate and max_rate are also close each other (there exists a relation between those parameters). We then define the closeness indicator as follows:

$$\text{closeness} = \left|100 - \left(\frac{\text{discrete\_period}}{\text{fluid\_period}} \times 100\right)\right|$$

3 Simulations

Discrete and fluid simulations have been done on networks with one, two, three, five and ten sources.

3.1 One source network

3.1.1 Influence of coefficients α and β

Hypothesis. This set of simulations has been done on the network of Figure 1. The throughput of the links (S, C) and (C, S) were varying between $1Mbits/s$ and $5Mbits/s$ in order to avoid a congestion at the source node S. The throughput of C was fixed to $\mu = 750kbits/s$. The maximal throughput of the link (D, C) has been fixed to $1Mbits/s$ (asymmetric connection). The

α	20	50	60	100	200	400	800	1400
β	6	4	2	1.5	1.25	1	0.75	0.5
discrete_period (s)	7.98	5.86	6.99	5.84	4.51	3.87	3.64	3.54
traffic load (%)	87.39	87.66	80.91	77.51	75.74	74.17	73.36	71.4
fluid_period (s)	7.9	5.81	6.91	5.78	4.51	3.83	3.6	3.46
closeness (%)	1.01	0.86	1.16	1.04	0.67	1.04	1.11	2.31

α	*1500*	*1600*	187	375	750	1500	*3000*
β	*0.37*	*0.25*	4	2	1	0.5	*0.25*
discrete_period (s)	*9.11*	*8.13*	4.58	4.31	3.9	3.62	*13.34*
traffic load (%)	*64.65*	*51.26*	89.32	83.21	77.02	71.8	*57.5*
fluid_period (s)	*3.24*	*2.99*	4.54	4.25	3.85	3.52	*3.55*
closeness (%)	-	-	0.88	1.41	1.3	2.84	-

Figure 2: Influence of the coefficients α and β, in a network with one sender.

propagation delay of the link (S, C) is `Df` $= 330\ ms$. The delay of the connections (C, D) and (D, C) are both equal to $100\ ms$, and the delay of (C, S) is $130\ ms$. Hence, `Df` = `Dr` $= 330\ ms$. The tolerance `T` was fixed to 0 packets authorized in the queue. Several values of α and β have been tested. The coefficient α was varying between 20 and 3000, and β between 0.25 and 6. In those simulations, the larger α is (*ie.*, more the rate increases quickly), the smaller β is (*ie.* more the rate decreases quickly).

Results. Figure 2 summarizes (and compares with the closeness indicator) the results obtained with a variation of the coefficient α and β. For a large range of value ($20 \leq \alpha \leq 1400$, $0.5 \leq \beta \leq 6$) the closeness is good (under 2.31%). For the other values tested ($\alpha > 1400$, and $\beta < 0.5$), the discrete model was not periodic whereas the fluid model was still periodic, so the fluid and discrete models do not match. Note that when the discrete protocol becomes aperiodic, the traffic load (evaluated using the total number of bits received by the destination D and the duration of the simulation) of the congestion node becomes low, and the protocol is inefficient. The corresponding α and β values should not be used in practice.

Figure 3 display results of discrete (left) and fluid (right) simulations, with $\alpha = 60$ and $\beta = 2$.

As a conclusion, the coefficients α and β should be chosen to obtain a periodic discrete protocol. In the opposite case, the fluid modelling does not match with the discrete protocol.

3.1.2 Influence of the tolerance `T`

Hypothesis. These two sets of simulations have been done on the same network as in the previous case, the propagation delays (`Df` and `Dr`), and the bandwidth of the link were the same. In the first set of simulations, the coefficient α was fixed to 800 and β to 0.75. And respectively, α was fixed to 375 and β to 2 in the second case. Several values of the tolerance `T` have been tested. The tolerance was varying between 0 and 35 packets authorized in the queue for the first set and between 0 and 30 for the second one.

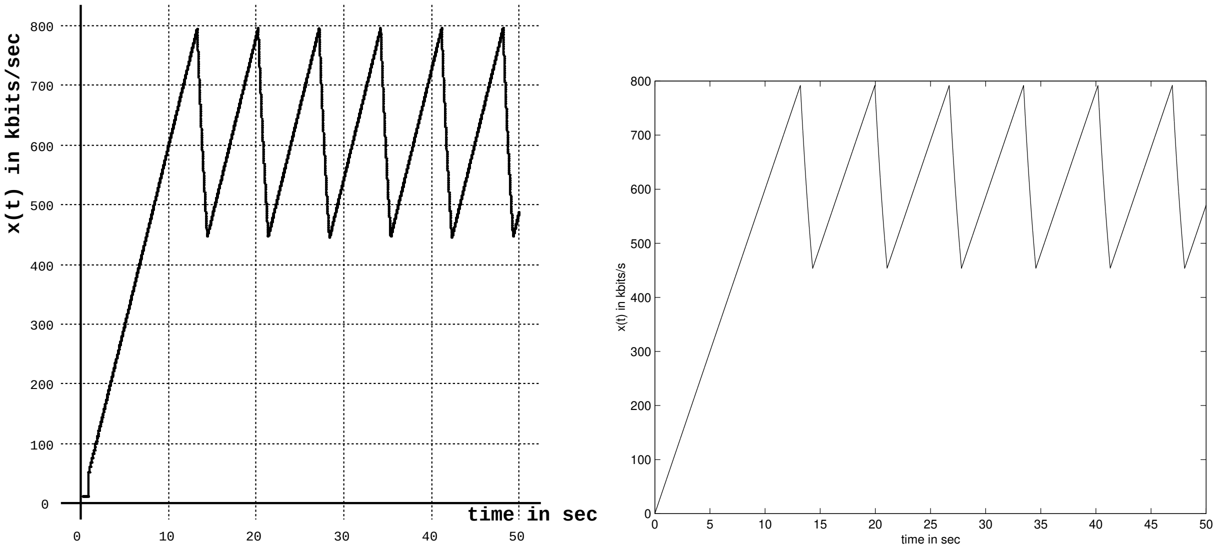

Figure 3: Discrete and fluid simulations, $\texttt{T} = 0$ packets, $\mu = 750\ kbits/s$.

3.1.3 Results

Figure 4 summarizes (and compares with the closeness indicator) the results obtained with a variation of the tolerance. In both sets of simulations, the results are good ($0.42\% \leq$ closeness $\leq 3.31\%$) for a tolerance under thirty five packets. For values of T equal to 35, the closeness becomes very bad, over 27%. Hence, the value of the tolerance T has an influence (which is difficult to analyze) on the pertinence of the results obtained by the fluid modelling of the packet switching protocol. If the tolerance is too large, the results become bad.

3.2 Two sources network

3.2.1 General hypothesis

These sets of simulations have been done on the network of Figure 5. The rate of the links between the source nodes and the congestion node are symmetric and varied between 1 $Mbits/s$

T	0	1	2	4	6	8	10	15	20	30	35
discrete_period (s)	3.64	4.08	4.31	4.63	4.9	5.15	5.37	5.89	6.37	7.49	9.83
traffic load (%)	73.36	74.93	75.91	77.11	77.84	78.47	79.41	80.45	81.23	77.61	66.46
fluid_period (s)	3.6	4.04	4.26	4.57	4.86	5.1	5.33	5.86	6.33	7.25	7.7
closeness (%)	1.11	0.99	1.17	1.31	0.82	0.98	0.755	0.51	0.63	3.31	27.66

T	0	1	2	3	4	8	10	15	20	30
discrete_period (s)	4.31	5.02	5.36	5.64	5.88	6.64	6.96	7.68	8.32	9.46
traffic load (%)	83.21	83.61	83.85	83.9	84.1	84.46	84.69	85.22	85.55	86.14
fluid_period (s)	4.25	4.96	5.31	5.6	5.83	6.61	6.9	7.64	8.26	9.42
closeness (%)	1.41	1.21	0.94	0.71	0.86	0.45	0.87	0.52	0.73	0.42

Figure 4: Influence of the tolerance, in a network with one sender.

and 5 $Mbits/s$ (in order to avoid a congestion at the source node). The link between the congestion node and the receiver is asymmetric: on the forward way, μ was fixed to 750 $kbits/s$, and to 1 $Mbits/s$ on the way back.

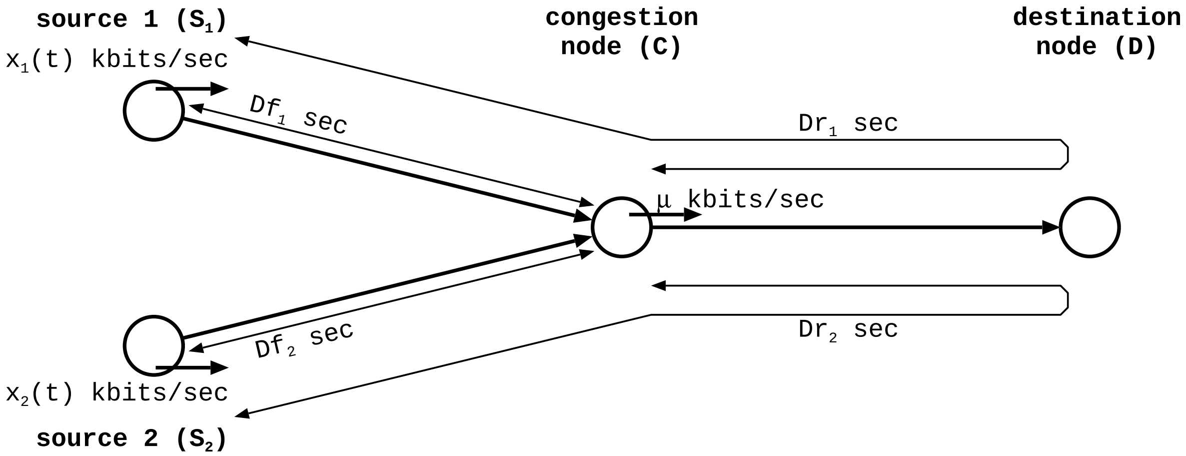

Figure 5: Two sources network overview.

3.2.2 Influence of the tolerance

Hypothesis. In this set of simulations, the coefficients α_1 and α_2 are both fixed to 60, while β_1 and β_2 are both fixed to 2. The propagation delays $\mathtt{Df}_1$ and $\mathtt{Df}_2$ are both fixed to 330 ms, and $\mathtt{Dr}_1$ and $\mathtt{Dr}_2$ are also fixed to 330 ms. In order to check if the tolerance `T` has an influence on the closeness of the two models, the tolerance was varying between 1 and 30 packets.

Results. Figure 6 summarizes (and compares with the closeness indicator) the results obtained with a variation of the number of packets authorized in the queue. In a network with two sources, two packets can arrive at the same time to the congestion node. In this case, if there is a tolerance of zero packet, one of the two sources could conclude that there is a beginning of congestion since one of its packets has waited in the queue. This source will reduce its rate, whereas in fact, there is no congestion in the network. This problem is avoided if the tolerance has a minimum value of one packet. The results obtained show that the closeness is always good for a tolerance less or equal to 30 packets.

T	1	2	3	4	5	8	10	15	20	25	30
discrete_period (s)	5.86	6.39	6.78	7.1	7.37	8.05	8.48	9.32	10.06	10.73	11.34
traffic load (%)	80.93	80.68	80.75	80.7	80.76	80.63	80.42	80.53	80.27	80.27	80.07
fluid_period (s)	5.94	5.94	6.75	7.03	7.3	7.99	8.38	9.24	9.97	10.65	11.27
closeness (%)	1.35	1.35	0.44	1	0.96	0.75	1.19	0.87	0.9	0.75	0.62

Figure 6: Influence of the Tolerance, in a network with two senders.

$\alpha_1 = \alpha_2$	20	50	60	80	90	100	200	400	800	*1100*
$\beta_1 = \beta_2$	6	4	2	1.75	1.5	1.5	1.25	1	0.75	*0.63*
discrete_period (s)	6.9	5.25	5.24	4.79	4.65	4.51	3.88	3.66	3.73	*13.31*
traffic load (%)	91.26	90.86	85.25	84.31	82.75	82.92	82.37	82.46	81.42	*76.11*
fluid_period (s)	6.94	5.36	5.2	4.75	4.61	4.46	3.82	3.6	3.62	*3.6*
closeness (%)	0.58	2.05	0.77	0.84	0.87	1.12	1.57	1.67	3.04	-

$\alpha_1 = \alpha_2$	*1400*	*1600*	70	70	70	70	70	70	70
$\beta_1 = \beta_2$	*0.5*	*0.25*	1	1.25	1.5	2	2.5	3	3.5
discrete_period (s)	*13.93*	*25.29*	5.47	5.24	5.12	4.97	4.93	4.96	4.94
traffic load (%)	*76.77*	*86.3*	77.12	80.11	82.3	85.54	87.67	89.28	90.45
fluid_period (s)	*3.51*	*2.83*	5.39	5.18	5.05	4.93	4.9	4.94	5
closeness (%)	-	-	1.48	1.16	1.39	0.81	0.61	0.4	1.2

Figure 7: Influence of the coefficients α and β, in a network with two identical sources.

3.2.3 Influence of the coefficients α and β

Hypothesis for two identical sources. The simplest case consists in using two identical sources: $\alpha_1 = \alpha_2$, $\beta_1 = \beta_2$. To manage this set of simulations, a tolerance of three packets was used. The propagation delay is symmetric on the link between the sources and the congestion node (100 ms, $\texttt{Df}_1 = \texttt{Df}_2 = 100$). The propagation delay on the link between the congestion node and the receiver is symmetric too and fixed to 100 ms ($\texttt{Dr}_1 = \texttt{Dr}_2 = 300$). In order to verify if the coefficients have an influence on the closeness, the coefficients α_1 and α_2 were varying between 20 and 1600, and the coefficients β_1 and β_2 between 0.25 and 6.

Results for two identical sources. The tables in Figure 7 summarize (and compare with the closeness indicator) the results obtained with a variation of the coefficients α and β. With two identical sources, the range of values such that the two models are close, is smaller than the range obtained for one source network ($20 \leq \alpha < 800$ and $0.75 < \beta \leq 6$). Conditions to obtain a periodic behavior of the discrete protocol with two sources are stronger than for one source; the range of acceptable values is then smaller.

Hypothesis for two different sources. In order to observe the influence of two different sources, a set of simulations has been performed, with $\alpha_1 \neq \alpha_2$ and $\beta_1 \neq \beta_2$ and a tolerance of two packets. The propagation delays were the same than those used for two identical sources.

Results for two different sources. Figure 9 displays fluid model (left), and discrete model (right), a result among others. Figure 8 summarizes (and compares with the closeness indicator) the results obtained with a variation of the coefficients α and β. The closeness between the two models does not seem to be influenced by the difference between the sources. If α_1, α_2, β_1 and β_2 are chosen in the range defined in 3.2.3, the closeness is always under 3%.

3.2.4 Influence of the propagation delays

In this set of simulations, the coefficients $\alpha_1 = \alpha_2$ are both fixed to 60, and $\beta_1 = \beta_2$ are both fixed to 2. The tolerance was fixed to 2 packets authorized in the queue. The results obtained

$\alpha1$	50	50	50	50	50	50
$\beta1$	4	4	4	4	4	4
$\alpha2$	75	100	150	200	400	750
$\beta2$	3	2.5	2.25	2	1.5	1
discrete_period (s)	4.93	4.85	4.61	4.43	4.09	3.76
traffic load (%)	90.01	89.34	88.90	88.32	86.88	85.02
fluid_period (s)	5.08	4.84	4.59	4.42	4.07	3.73
closeness (%)	2.95	0.21	0.25	0.43	0.49	0.8

Figure 8: Influence of the coefficients α and β, in a network with two senders.

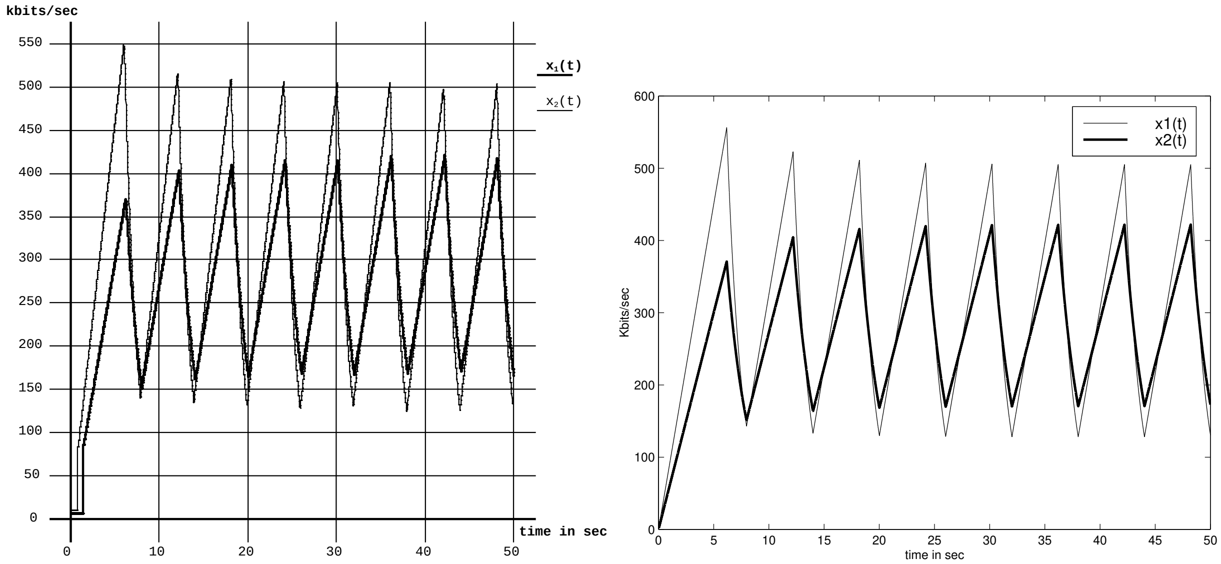

Figure 9: Discrete and fluid simulation, $\alpha_1 = 90$, $\alpha_2 = 60$, $\beta_1 = 1.32$, $\beta_2 = 2$, $\mathtt{Df}_1 = \mathtt{Df}_2 = \mathtt{Dr}_1 = \mathtt{Dr}_2 = 330\ ms$.

indicate that the propagation delays do not have a real influence on the closeness of the models.

3.3 Three sources network

In this set of simulations we focus on the influence of the tolerance on the closeness.

3.3.1 Hypothesis

This set of simulations has been done in a network similar to the previous one (see Figure 5): the network was composed of three source nodes, one congestion node and one receiver node. The bandwidth of the links between the source node and the congestion node are symmetric and fixed to 1 $Mbits/s$. The link between the congestion and the receiver nodes is asymmetric: on the forward way μ was fixed to 1.1 $Mbits/s$, and on the way back the bandwidth was fixed to 2.1 $Mbits/s$. The propagation delay on the link between the sources and the congestion node is asymmetric: on the forward way it was fixed to 330 ms, and to 130 ms on the way back. The propagation delay on the link between the congestion node and the receiver one is symmetric

Df1 (ms)	330	330	330	330	330	100	100	100	100	100
Dr1 (ms)	330	330	330	330	330	300	300	300	300	300
Df2 (ms)	330	660	990	1320	1650	100	200	400	800	1600
Dr2 (ms)	330	660	990	1320	1650	300	400	600	1000	1800
discrete period (s)	6.39	7.97	9.25	10.32	11.15	4.83	5.43	6.4	7.97	8.86
traffic load (%)	80.68	76.87	74.88	74.33	73.70	85.42	83.67	81.02	78.37	85.41
fluid_period (s)	6.4	8.01	9.3	10.37	11.23	4.84	5.43	6.45	8.040	8.92
closeness (%)	0.16	0.5	0.54	0.48	0.36	0.21	0	0.78	0.87	1.96

Figure 10: Influence of the propagation delay, in a network with two senders.

T	2	3	4	5	8	10	15	20	25	30
discrete_period (s)	6.01	6.31	6.59	6.82	7.38	7.69	8.39	8.97	9.52	10
traffic_load (%)	80.93	80.95	80.87	80.74	80.63	80.65	80.49	80.47	80.34	80.54
fluid_period (s)	6	6.31	6.55	6.77	7.31	7.63	8.29	8.89	9.43	9.91
closeness (%)	0.17	0	0.61	0.74	0.96	0.79	1.21	0.9	0.95	0.91

Figure 11: Influence of the Tolerance, in a network with three senders.

and fixed to 100 ms. The propagation delays $\mathtt{Df}_1$, $\mathtt{Df}_2$, $\mathtt{Df}_3$, $\mathtt{Dr}_1$, $\mathtt{Dr}_2$, $\mathtt{Dr}_3$ are fixed to 330 ms. All the sources were identical: $\alpha_1 = \alpha_2 = \alpha_3 = 60$ and $\beta_1 = \beta_2 = \beta_3 = 2$.

3.3.2 Results

Figure 11 summarizes (and compares with the closeness indicator) the results with a variation of the number of packets authorized in the queue for a three source network. Once again, the closeness is very good (less or equal to 1.21 %) for a tolerance less or equal to thirty packets. We can observe that the greater the number of sources is, the better the closeness of the two models is. In this set of simulation, like the two sources network case, for all values of the tolerance, the two models are close.

3.4 Five sources network

In this set of simulations we focus on the influence of the tolerance on the closeness.

3.4.1 Hypothesis

This set of simulations has been done in a network similar to the precedent (see Figure 5): the network was composed of five source nodes, one congestion node and one receiver node. The bandwidth of the links between the source nodes and the congestion node are symmetric and fixed to 1 $Mbits/s$. The link between the congestion node and the receiver one is asymmetric: on the forward way μ was fixed to 2 $Mbits/s$, and on the way back the bandwidth was fixed to 3 $Mbits/s$. The propagation delay on the link between the sources and the congestion node is asymmetric: on the forward way it was fixed to 330 ms, and to 130 ms on the way back. The propagation delay on the link between the congestion node and the receiver one is symmetric and fixed to 100 ms. All the propagation delays $\mathtt{Df}_x$ and $\mathtt{Dr}_x$ ($x \in \{1, ..., 5\}$) are fixed to 330 ms. All the sources were identical: $\alpha_x = 60$ and $\beta_x = 2$.

T	4	5	6	7	8	10	15	20	25	30
discrete_period (s)	6.24	6.42	6.59	6.73	6.88	7.11	7.63	8.07	8.45	8.81
traffic_load (%)	80.88	80.86	80.91	80.83	80.73	80.73	80.73	80.6	80.44	80.52
fluid_period (s)	6.28	6.44	6.58	6.72	6.85	7.1	7.59	8.03	8.4	8.76
closeness (%)	0.64	0.31	0.15	0.15	0.44	0.14	0.53	0.5	0.6	0.57

Figure 12: Influence of the Tolerance, in a network with five senders.

3.4.2 Results

Figure 12 summarizes (and compares with the closeness indicator) the results with a variation of the number of packets authorized in the queue for a five sources network. The results obtained confirm that larger the number of sources is, better is the closeness.

3.5 Ten sources network

In order to see if there is a relation between the number of sources, the tolerance and the closeness, we have done several simulations with ten sources. Using the same type of topology: each source has the coefficient α fixed to 60, and β fixed to 2; the propagation delays of each source are the same ($\texttt{Df} = 330ms$, $\texttt{Dr} = 330ms$). For all the values of tolerance tested ($9 \leq \texttt{T} \leq 40$), the closeness is always under 1%.

4 Conclusion

In this work, the limitation in the use of a fluid modelling of a discrete congestion control protocol has been investigated. A simple protocol and a simple network has been used in order to obtain an exact fluid modelling. The results obtained by comparison of both discrete and fluid periods indicate that a fluid modelling can be a pertinent approximation of a discrete packet switched protocol. For all tested topologies, the fluid modelling gives good results, and we extend here results of [2].

However, there are some conditions, depending on the protocol or the network parameters, for which the continuous modelling is not pertinent. Indeed, to obtain a good closeness, the discrete protocol should have a periodic behavior (in our case, α and β must be chosen in a specific range). The variation of the source rate must be smooth, and more the number of sources is important, more the rate must varying smoothly. Note that the propagation delays do not seemed to have any influence on the closeness. Results should be completed with more simulations, more configurations, and the comparison of the discrete and fluid curves could take care of the phase. Hence, the main conclusions of our work are (i) as indicate in [2], fluid modeling can give very good approximation of some packet switching protocols, (ii) this is still true with more than one source, but (iii) only for specific ranges of parameters. Still more works are needed to understand the relationship between fluid modelling and discrete protocol.

A ns code

This first appendix gives the C++ code used to simulate the protocol with ns.

A.1 Dec_net.cc

```
// UTC Mai 2001

#include "dec_net.h"
#include <stdlib.h>

int hdr_dec_net::offset_;
static class Dec_netHeaderClass : public PacketHeaderClass {
public:
        Dec_netHeaderClass() : PacketHeaderClass("PacketHeader/Dec_net",
                                               sizeof(hdr_dec_net)) {
                bind_offset(&hdr_dec_net::offset_);
        }
} class_dec_nethdr;

static class Dec_netClass : public TclClass {
public:
        Dec_netClass() : TclClass("Agent/Dec_net") {}
        TclObject* create(int argc, const char*const* argv) {
          if (argc == 8)
            return (new Dec_netAgent(argv[4],argv[5],argv[6], argv[7]));
          else
            printf ("Error");

}
} class_dec_net;




Dec_netAgent::Dec_netAgent(const char* nom, const char* A, const char* B,
const char* tol) : Agent(PT_DEC_NET)
{

  if( strlen(nom) <= TAILLE_NOM)
    (void)strcpy(monNom, nom);
  else
    {
      (void)strncpy(monNom, nom, TAILLE_NOM-1);
      (void)strcat(monNom, ".");
    }
```

```
  // Variables initialisation.
  // For more details about the variables, see dec_net.h.
  c=0;
  tempo = 0;
  phase = 1;
  premier_pkt_data = 1;
  rtt_min = 0;
  rtt_inst = 0;
  premier_ack = 0;
  min_rate = 0;
  rate = 0;
  max_rate = 0;
  t_max_rate = 0;
  t_min_rate = 0;
  alpha= atof(A)*1000 ; // Conversion  of kbits/s in bits/s.
  beta = atof(B) ;
  last_recv = 0;
  inter_recv = 0;
  tolerance = atof(tol) + 0.0001;
  nb_pkts_recv = 0;
  periode_tot = 0;
  nb_periode = -1;
  t_temp = 0;
  deb_max = 0;

  // Shared variables, in order to use those variables in the tcl code.
  bind("packetSize_", &size_);
  bind("rtt_min_",&rtt_min);
  bind("rtt_inst_", &rtt_inst);
  bind ("inter_paquet_",&tempo);
  bind ("rate_",&rate);
  bind("c_",&c);
  bind("inter_recv_",&inter_recv);
  bind("nb_pkts_recv_",&nb_pkts_recv);
  bind("nb_periode_",&nb_periode);
  bind("periode_tot_",&periode_tot );
  fflush(stdout);
}

int Dec_netAgent::command(int argc, const char*const* argv)
// This function is called by the tcl code. The tcl code give in argument
// one of the follwing words: ``resultats'', ``nb_paquets'', ``send''.
{

  if(argc == 2)
    {
      if (strcmp(argv[1], "resultats") == 0)
      // This command is usually called at the end of the tcl code.
```

```
    // It returns the number of periods, and the average duration of
    // the periods.
    {
     periode_tot = periode_tot / nb_periode ;
     printf("\n The agent %s had a period of  %f s, and there has been %d
     priods \n",monNom, periode_tot, nb_periode);
     printf("The agent %s had max rate of %f bit/s\n",monNom, deb_max);
     return TCL_OK;
    }

    if (strcmp(argv[1], "nb_paquets") == 0)
    // This command is usually called at the end of the tcl code.
    // It return, the number of  transmited packets between the begin of the
    // connexion and the instant at which it has been called.
    {
     printf("\nThe agent %s received %d packets\n",monNom, nb_pkts_recv);
     return TCL_OK;
    }


  // The following command is used in the tcl code for start to send paquets.
  if (strcmp(argv[1], "send") == 0) {
    // Creation of a new packet.
    Packet* pkt = allocpkt();
    // Access to the Dec_net packet header.
    hdr_dec_net* hdr = hdr_dec_net::access(pkt);
    // Ret is put to 0 for specifie that it is a data packet and
    // the receiver must send an ack.
    hdr->ret = 0;
    // The sending date is copied in the header in order to compute
    // the rtt when the ack will arrive.
    hdr->send_time = Scheduler::instance().clock();
    // Sending of the packet.
    send(pkt, 0);
    // inter_packets_delay function is called to compute the new gap
    // before the next send.
    tempo =  inter_packets_delay();
    Tcl& tcl = Tcl::instance();
    // Recursive call of the command send at the date now + new gap.
    tcl.evalf (" global ns
    set now [$ns now]
    $ns at [expr $now+ %f] \"$%s send\"
    ",tempo,monNom);
    return (TCL_OK);

  }

}
```

```
  return (Agent::command(argc, argv));
}


void Dec_netAgent::recv(Packet* pkt, Handler*)
{
  // Access to the IP header.
  hdr_ip* hdrip = hdr_ip::access(pkt);
  // Access to the Dec_net header.
  hdr_dec_net* hdr = hdr_dec_net::access(pkt);
  // Verification if ret = 0 (to send an ack or not).
  if (hdr->ret == 0)
    {
    // Sending of an ack.
    t_recv = Scheduler::instance().clock();
    // Keep the packet sending date to copy it in the ack.
    double stime = hdr->send_time;
    // Destruction of the data packet.
    Packet::free(pkt);
    // Creation of a new packet.
    Packet* pktret = allocpkt();
    hdr_dec_net* hdrret = hdr_dec_net::access(pktret);
    // ret is put to  1 to specifie taht it is an ack.
    hdrret->ret = 1;
    // Copy the sending date (of the data packet) in the ack header.
    hdrret->send_time = stime;
    // Verify if it is the first reception of a data packet.
    nb_pkts_recv ++;
    // Add one packet to the total number of received packets.

    if  (premier_pkt_data == 1)
      {
        // If it is the first receipt of data packet, the receipt date
        // is copy in the variable last_recv.
        last_recv = t_recv;
        // The flag is put to 0 to specifie that the first data packet
        // has been received.
        premier_pkt_data = 0;
      }
    else
      {
        // The time between the last reception and this reception is computed.
        inter_recv = t_recv - last_recv;
        // The variable last_recv is update.
        last_recv = t_recv;
      }

    // Sending of the ack.
```

```
   send(pktret, 0);
   }

else
   {
   // Reception of an ack.
   // Save the reception date of the ack.
   t_recv = Scheduler::instance().clock();

   // Check if it is the first ack.
   if (premier_ack == 0)
     {
       // If it is the first ack, the reference rtt is compute.
       premier_ack = 1;

       rtt_min = t_recv - hdr->send_time;



       printf("\nRtt_min = %f for the connexion which for sender %s\n",
       (rtt_min), monNom);
       // The rate is in an increasing phase.
       phase = 1;
     }
   else
     {
       // If it is not the first ack.
       // The rtt is compute.
       rtt_inst = t_recv - hdr->send_time;
       // If the rtt is gearter than rrt_min + tolerance (the reference),
       // that signifie that the queue is too large.

       if ((rtt_inst > (rtt_min + tolerance))  && (phase==1) )
         {
          // If it is the first period, all the variables are initialized.
          if (nb_periode < 0)
             {
               t_temp = t_recv;
               nb_periode = 0;
               periode_tot = 1;
             }

           // If it is not the first period, the duration of the
           // period is compute.
           else
             {
               periode_temp = t_recv - t_temp;
               if (periode_temp >= 1.5)
```

```
            // This checks if it is a real period or just a
            // micro oscillation.
              {
                periode_tot = periode_tot + periode_temp;
                t_temp = t_recv;
                nb_periode++ ;
                }
          }

        update_rate(); // Update the rate before memorized it.

        // Keep the maximum rate (when the queue is considered too large).
        max_rate = rate;

        if (max_rate > deb_max){
          deb_max = max_rate;}

        // Keep the date of the detection.
        t_max_rate =  t_recv;

        // Change of phase, the rate goes into a decreasing phase.
        phase = 0;
         }

    // If the rtt is equal to rtt_min + tolterance, the queue has a
    // normal or small size.
    if  ((rtt_inst <= (rtt_min + tolerance))&&(phase==0))
      {
        update_rate();
         // Keep the minimum rate (when the queue has a normal size).
        min_rate = rate;
        // Keep the date of the detection.
        t_min_rate = t_recv;
        // Change of phase, the rate goes into an increasing phase.
        phase = 1;

      }
  }

  // Destruction of the packet.
  Packet::free(pkt);
 }
}
```

A.2 Dec_net.h

```
// UTC Mai 2001


#ifndef ns_dec_net_h
#define ns_dec_net_h
#define TAILLE_NOM 10
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "random.h"
#include <math.h>

struct hdr_dec_net {
  char ret;
  double send_time;
  double c;

        // Header access methods.
        static int offset_; // Required by PacketHeaderManager.
        inline static int& offset() { return offset_; }
        inline static hdr_dec_net* access(const Packet* p) {
                return (hdr_dec_net*) p->access(offset_);
        }
};


class Dec_netAgent : public Agent
{
private:
  char monNom[TAILLE_NOM];
  // Name of the agent.
  double tempo;
  // Time to wait between two sends.
  int premier_pkt_data;
  // flag used to know if it is the first data packet or not.
  int premier_ack;
  // flag used to know if it is the first ack or not.
  double rtt_min;
  // Value of the first rtt.
  double rtt_inst;
  // Value of the rtt.
  int phase;
  //  Flag used to know in which phase the rate is
  // (phase = 0: decreasing pahse, phase = 1: increasing pahse).
  double min_rate;
  // Rate of the source during the last change of decreasing phase to
```

```
  // increasing phase.
  double max_rate;
  // Rate of the source during the last change of increasing phase to
  // decreasing phase.
  double rate;
  // Rate in bits/sec.
  double t_max_rate;
  // Date of the last change of increasing phase to decreasing phase.
  double  t_min_rate;
  // Date of the last change of decreasing phase to increasing phase.
  double t_recv;
  // Date of ack receipt.
  double alpha, beta;
  // Coefficients used in the rate equation.
  double c;
  // Variance of the packet size (default value 0).
  double last_recv;
  // Receipt date of the last data packet.
  double inter_recv;
  // Delay between receptions of data packets.
  double tolerance;
  // Tolerance in te queue.
  int nb_pkts_recv;
  // Number of received packets.
  double periode_tot, t_temp, periode_temp;
  // Temporary variables used for compute the period.
  int nb_periode;
  // Number of priods.
  double deb_max;
  // Max rate during the simulation.


public:
  Dec_netAgent(const char*, const char*, const char*, const char*) ;
  virtual int command(int argc, const char*const* argv);
  virtual void recv(Packet*, Handler*);

 char *nomme(const char *nom) // Return the name.
  {
    if( nom != NULL )
      strcpy(monNom, nom);

    return monNom;
  }

 void update_rate()  // Update the rate by using the equations.
   {
     float t;
```

```
      t = Scheduler::instance().clock();

      if (phase==1)
        {
          rate = min_rate + alpha * (t-t_min_rate);
        }
      else
        {
          rate = max_rate * exp (-((t-t_max_rate)/beta));
          if (rate < 3000)
             rate = 3000;
        }

   }


double  inter_packets_delay()
// This function computes the gap between two emissions of packets.
   {
     float delais_interpaquet ;
     int size_bit;
     float t;
     t = Scheduler::instance().clock();
     update_rate();
     size_=  (double)Random::normal(1000, c);
     size_bit = (int)size_ * 8;
     delais_interpaquet = (size_bit / rate);
     return(delais_interpaquet);
   }
};



#endif
```

B Matlab code

Those appendixes present the matlab code used to simulate the fluid model of a network composed of two senders, one receiver, and one congestion node.

B.1 fluid_model_2sources.m

```
function [Time, taille_file, x1, x2, duree_attente_sortie, duree_attente] =
fluid_model_2sources(alpha1, beta1, dF1, dR1, alpha2,beta2, dF2, dR2, mu,
duree, pas,tolerance)
% alpha1 : coefficient used in the increasing phase of the source 1.
% beta1 :  coefficient used in the decreasing phase of the source 1.
```

```
% dF1 : Propagation delay between the source node and the congestion node (in ms).
% dR1 : Propagation delay between the congestion node and the soucre node (in ms).
% It is the same for the second source (alpha2, beta2, dF2, dR2).
% Nb :  dF1, dF2, dR1, dR2 = propagation delay + packet size / rate of the links.
% mu : rate of the queue (in kbits/s).
% duree : duration of the simulation (in s).
% pas : granularity of the simulation ms.
% tolerance : delay autorized to wait in the queue (in ms).


% Variables initialisation
disp(['Beginning of the initialization'])
flag = 0;
duree = duree * 1000 / pas;
Time = [0 : duree];

d1 = dR1+dF1;
d1_it = d1/pas - mod(d1/pas,1); % Theoretical rtt in function of the granularity.
dR1_it = dR1/pas - mod(dR1/pas,1); % Dr in function of the granularity.
dF1_it = dF1/pas - mod (dF1/pas,1); % Dfin function of the granularity.
periode1=0;  % Flag used to compute the period.
nb_periode1=-1; % Number of periods.
t_temp1=-1; % temporary variable used to compute the period.

d2 = dR2+dF2;
d2_it = d2/pas - mod(d2/pas,1);
dR2_it = dR2/pas - mod(dR2/pas,1);
dF2_it = dF2/pas - mod (dF2/pas,1);
periode2 =0;
nb_periode2=-1;
t_temp2=-1;

flag_x1 = 1; % flag used to know in which phase is the rate.
t1_lin_temp = 0; % Date at which the rate enter in the last increasing phase.
t1_expo_temp = 0; % Date at which the rate enter in the last decreasing phase.
x1_max_temp = 0;  % max value reached by the rate in the last increasing pahse.
x1_min_temp =0; % min value reached by the rate in the last decreasing pahse.
x1_min = mu; % initialisation of the min an max rate.
x1_max =0;

flag_x2 = 1;
t2_lin_temp = 0;
t2_expo_temp = 0;
x2_max_temp = 0;
x2_min_temp =0;
x2_min = mu;
x2_max =0;
```

```
taille_max_file = 0; % Initialisation of the queue max size.

% Initialisation of all the tables used in the simulation.
i_max= size(Time);
for i=1:i_max(2)
   taille_file(i)=0;
   x1(i) = 0;
   x2(i) = 0;
   duree_attente_sortie(i)=-1;

end;

disp(['End of the initialization'])

disp(['Beginning of the simulation'])

for i=1:i_max(2)

   t = Time(i); % t in function of the granularity.
   t_sec = (t * pas)/1000 ; % t in seconde.

   if i>1
      % call of the function file_attente_2soucres.
      % This function returns the delay spent in the queue by a packet sent
      % at the date t.
      [duree_attente(i), taille_file_reel(i), taille_file(i)] =
      file_attente_2sources(x1, x2, mu, (t-dF1_it),(t-dF2_it),
      taille_file_reel(i-1), pas);

      % ds denotes the date at witch the packet sent at time t will
      % leave the queue.
      ds = (duree_attente(i)/pas) - mod ((duree_attente(i)/pas),1);
      ds = ds+i;
      if ds > i_max(2)
         ds = i_max(2);
      end;

      % Save the time spent in the queue by the packet which leave
      % the queue at ds.
      duree_attente_sortie(ds) = duree_attente(i);

      if (duree_attente_sortie(i)==-1);
         duree_attente_sortie(i)=duree_attente_sortie(i-1);
      end;

      % Check if the queue has reached a maximum size.
      if taille_file(i)>taille_max_file
         taille_max_file = taille_file(i);
```

```
      end;
   else % If it is the first step of the simulation.
      taille_file(i) = 0;
      taille_file_reel(i) = 0;
      duree_attente(i)=0;
      duree_attente_sortie(i)=0;
      ds = 0;

   end;

   if (t-d1_it) >= 0
      T1 = (i-dR1_it); % Denotes the date at witch the packet received at
                       % time t has left the queue (T1).
   else
      T1 = 1;
   end;

   if (duree_attente_sortie(T1) <= tolerance)
   % If the delay spent in the queue is less or equal to the tolerance,
   % the rate is in an increasing phase.
      if flag_x1 == 0 % If just before, the rate was in a decreasing phase.
         flag_x1 = 1; % the rate change of phase,
         % the temporary min rate is computed,
         x1_min_temp = x1_max_temp*exp(-(t_sec-t1_expo_temp)/beta1);
         % and the date of the change is saved.
         t1_lin_temp = t_sec;

         % If the temporary min rate is less than the min rate, it is saved.
         if x1_min_temp < x1_min
            x1_min = x1_min_temp;
         end;
      end;
      % Compute the new rate.
      x1(i)= alpha1 * (t_sec - t1_lin_temp) + x1_min_temp;
   end;


if (duree_attente_sortie(T1)>tolerance)
   % If the delay spent in the queue is less or equal to the tolerance,
   % the rate is in an increasing phase.
   if flag_x1 == 1 % If just before, the rate was in a increasing phase
         flag_x1 = 0; % the rate change of phase,
         % the temporary max rate is computed,
         x1_max_temp = alpha1 * (t_sec - t1_lin_temp) + x1_min_temp;
         % and the date of the change is saved.
         t1_expo_temp = t_sec;

         if t_temp1 == -1 % If it is the first decreasing phase
```

```
            t_temp1 = t1_expo_temp; % the date of change is saved in order to
                                    % compute the period later.
         end;

         % Compute the new sum of all the periods.
         periode1= periode1 + t1_expo_temp - t_temp1;
         % Add one more period to the total number of periods.
         nb_periode1 = nb_periode1+1;
         % Saved the last date of change in t_temp1.
         t_temp1 = t1_expo_temp;

         % If the temporary min rate is less than the min rate, it is saved.
         if x1_max_temp > x1_max;
            x1_max = x1_max_temp;
         end;
    end;
    % Compute the new rate.
    x1(i)= x1_max_temp*exp(-(t_sec-t1_expo_temp)/beta1);

end;

if x1(i)<3 % Ensure the minimal rate of 3000 bits/s.
   x1(i)=3;
end;


   if (t-d2_it) >= 0
      T2 = (i-dR2_it);
   else
      T2 = 1;
   end;

   if (duree_attente_sortie(T2) <= tolerance)
      if flag_x2 == 0
         flag_x2 = 1;
         x2_min_temp = x2_max_temp*exp(-(t_sec-t2_expo_temp)/beta2);
         t2_lin_temp = t_sec;

         if x2_min_temp < x2_min
            x2_min = x2_min_temp;
         end;
       end;
       x2(i)= alpha2 * (t_sec - t2_lin_temp) + x2_min_temp;
   end;


if (duree_attente_sortie(T2)>tolerance)
   if flag_x2 == 1
```

```
         flag_x2 = 0;
         x2_max_temp = alpha2 * (t_sec - t2_lin_temp) + x2_min_temp;
         t2_expo_temp = t_sec;
         if t_temp2 == -1
            t_temp2 = t2_expo_temp;
         end;
         periode2 = periode2 + t2_expo_temp - t_temp2;
         nb_periode2 = nb_periode2+1;
         t_temp2 = t2_expo_temp;

         if x2_max_temp > x2_max;
            x2_max = x2_max_temp;
         end;
   end;

   x2(i)= x2_max_temp*exp(-(t_sec-t2_expo_temp)/beta2);

end;

if x2(i)<3
   x2(i)=3;
end;

end;

% Compute the average period.
periode1 = periode1/nb_periode1;
periode2 = periode2/nb_periode2;

disp(['End of the simulation'])
% Display the results.
disp(['Max rate of the source1: 'num2str(x1_max,6) 'kbits/s']);
disp(['Max rate of the source2: 'num2str(x2_max,6) 'kbits/s']);
disp(['Period of the source1: 'num2str(periode1,6) 's']);
disp(['Period of the source2: 'num2str(periode2,6) 's']);
disp(['Number of periods of the source1: 'num2str(nb_periode1,6)] );
disp(['Number of periods of the source2: 'num2str(nb_periode2,6)] );
disp(['Max size of the queue: 'num2str(taille_max_file,6) 'paquets']);

Time=Time * pas /1000;

plot(Time,x1,'k');
hold on
plot(Time,x2,'k','linewidth',2);

legend('x1(t)', 'x2(t)');
xlabel('time in sec');
ylabel('Kbits/sec');
```

B.2 file_attente_2sources.m

```
function [attente, file_reel, val_q] =
file_attente_2sources(x1, x2, mu, tdf1, tdf2, file_pre, pas)
% x1, x2 : rate of the sources in kbits/s.
% mu : rate of the queue.
% tdf1, tdf2 : time at whhich the function has been called  - df.
% file_pre : last size of the queue in kbits.
% pas : granularity of the simulation.

if (tdf1<=0 & tdf2<=0 ) % If no packet has still reached the queue.
   val_q = 0;

else
        val_q = file_pre; % Save in val_q the last queue size.

        if (tdf1 > 0) % If some packets of the source 1 have reached the queue.
        % Add to val_q the number of bits from the source 1 whhich has
        % reached the queue
        val_q =  val_q + x1(tdf1)*(pas/1000);
   end;

   if (tdf2 > 0) % If some packets of the source 2 have reached the queue.
        % Add to val_q the number of bits from the source 2 which has
        % reached the queue.
        val_q =  val_q + x2(tdf2)*(pas/1000);
        end;

        % Remove to val_q the number of kbits which have left the queue.
     val_q = val_q - mu*(pas/1000);
  end;

% A queue can not have a negative number of kbits.
if val_q < 0
        val_q = 0;
end;

% Save the size of the queue in kbits.
file_reel = val_q;

val_q_pkt = val_q/8; % Change val_q in kbytes.

if mod(val_q_pkt,1)>0  % Change the size of the queue in number of packets.
    val_q = (val_q_pkt - mod(val_q_pkt,1) +1);
 end;
```

```
attente =  ((val_q * 8)/mu) * 1000; % Time spent in the queue in ms.
```

References

[1] M. Alman, V. Paxson, and W. Stevens. TCP congestion control. Request For Comment 2581, RFC editor, http://www.rfc-editor.org/, March 1999.

[2] J.-C. Bolot and A. U. Shakar. Analysis of a fluid approximation to flow control dynamics. In *Proc of IEEE INFOCOM'92*, pages 2398–2407, 1992.

[3] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[4] S. Floyd, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of SIGCOMM 2000*. ACM, May 2000.

[5] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM'88*, pages 314–329, 1988.

[6] S. Mascolo. Congestion control in high-speed communication networks using the Smith principle. *Automatica*, 35(1999):1921–1935, 1999.

[7] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[8] S.-I. Niculescu. *Delay effects on stability: A robust control approach*. LNCIS. Springer-Verlag, Heidelberg, May 2001. 383 pages.

[9] The ns project, http://www.isi.edu/nsnam/ns/ns-documentation.html. *The ns manual*, August 2000.

[10] J. Walrand and P. Varaiya. *High-performance communication networks*. Morgan Kaufmann, San Francisco, CA, USA, 1996.

[11] K. Zhou, J. Doyle, and K. Glover. *Robust and optimal control*. Prentice Hall, New Jersey, USA, 1995.

Contents