

Modélisation d'une connexion Compound TCP isolée

Alberto Blanc — Denis Collange — Konstantin Avrachenkov

N° 6778

Decembre 2008

Thème COM



*Rapport
de recherche*

Modélisation d'une connexion Compound TCP isolée

Alberto Blanc ^{*}, Denis Collange ^{*}, Konstantin Avrachenkov [†]

Thème COM — Systèmes communicants
Équipes-Projets Maestro

Rapport de recherche n° 6778 — Decembre 2008 — 27 pages

Résumé : Compound TCP (CTCP) a été conçu par Tan et al. pour améliorer l'efficacité des transferts TCP sur les réseaux haut-débit sans pénaliser les autres flux concurrents. Nous proposons et analysons dans cette étude un modèle détaillé d'une connexion CTCP isolée, et nous le comparons avec des simulations ns-2 utilisant l'implémentation Linux de CTCP. Ce modèle permet d'identifier différents modes de fonctionnement de CTCP, selon les paramètres du système. Nous montrons que durant la phase où la fenêtre de congestion est considérée comme constante par les auteurs du protocole, la fenêtre observe en réalité des oscillations importantes. Ces fluctuations peuvent dégrader les performances de la connexion CTCP, mais aussi celles des autres flux. Le modèle que nous avons développé permet également de calculer le débit moyen d'une connexion CTCP, ainsi que l'occupation induite du buffer du goulet d'étranglement. Ces métriques dépendent du mode de fonctionnement de CTCP. Pour des hauts débits, l'efficacité d'une connexion TCP isolée tend vers 75%.

Mots-clés : TCP, Compound TCP, modèle fluide

^{*} Orange Labs, 905 rue Albert Einstein, 06921 Sophia Antipolis, France, email: `First-Name.LastName@sophia.inria.fr`

[†] INRIA, 2004 Route des Lucioles, Sophia-Antipolis, France, email: `First-Name.LastName@sophia.inria.fr`

Modelling an Isolated Compound TCP Connection

Abstract: Compound TCP (CTCP) was designed by Tan et al. to improve the efficiency of TCP on high speed networks without unfairly penalizing other connections. In the present work we construct and analyze a detailed model of an isolated CTCP connection. We validate it with ns-2 simulations using a Linux implementation of CTCP. The detailed model allows us to identify and classify significantly different CTCP operating regimes depending on the system parameters. We show that in the “constant window” phase the congestion window can in fact have significant oscillations with non-negligible effect on the performances and which can, also, induce additional jitter in the cross traffic. Using this model we calculate the average throughput and average backlog size at the bottleneck link. These performance metrics depend on the CTCP operating regime. Under certain circumstances, an isolated CTCP connection on a high speed link utilizes around 75% of the link capacity.

Key-words: TCP, Compound TCP, fluid model

1 Introduction

With the increasing popularity of faster access links like Fiber To The Home, the current Standard TCP is not always ideal. As indicated by Floyd [9] the current Standard is not able to reach these rates in realistic environments, i.e. with typical packet loss rates. Many new transport protocols have been proposed and are currently being studied to replace it. Some of them are already implemented in the latest versions of some operating systems, like Compound TCP on Windows, and Cubic (and others) on Linux. Others are implemented in network equipment. At least for the next few years, the protocols already implemented will play an increasing role in the resource sharing between flows in the Internet. Yet the behavior, the performance, and the impact on the network of these protocols are not well-known. A method to evaluate the new protocols has just been specified [10], and test scenarios are still under discussion. For most of these new protocols there are only experimental or simulation studies, with conflicting results. Analytical models, describing their behavior and impact on the network, exist only for a few protocols and for simple cases. In this paper we develop an analytical model of Compound TCP, to analyze in detail its behavior for an isolated connection.

Compound TCP (CTCP) has been presented by Microsoft Research in [15] and [16] in 2006. It is currently submitted as a draft to the IETF Network Working group with minor differences [13]. CTCP is enabled by default in computers running Windows Server 2008 and disabled by default in computers running Windows Vista [8]. It is also possible to add support for CTCP to Windows XP. An implementation of CTCP, based on [16, 13], is also available for Linux [3].

As the proposal of CTCP is still recent, there are only a few published evaluations of it. The only analytical model of CTCP in [16] assumes a constant window size in the third phase of Figure 1. To the best of our knowledge there are no complete theoretical models of CTCP that can be used to analyze in details the behavior of this new protocol. While we have shown in [7] that the sending window oscillates during this phase, and that these oscillations may have a significant impact on the performance of CTCP. Other evaluations are based on experiments. However, except the one of Li [12], all the other experimental evaluations [5, 11, 4] use Linux implementations of CTCP whose behavior differs from the Windows implementation, according to [4].

The main objective of the authors of CTCP [16] is to specify a transport protocol which is efficient, using all the available bandwidth, fair and conservative, limiting its impact on the network. They propose to combine the fairness of a delay-based approach with the aggressiveness of a loss-based approach. The sending window (w) is defined as the sum of two components : the classical New Reno congestion window (w_c) and a delay-based window (w_d).

When the source detects an under-utilized network, it quickly increases the delay-based component until the sending window exceeds the estimated bandwidth delay product. Conversely, the delay-based component is decreased if the source detects increasing network delays. The delay-based component is then adjusted to maximize the efficiency and to minimize the backlog in network queues. At the i -th round trip the backlog in network queues is estimated as :

$$\Delta_i = w_i (1 - \tilde{\tau}/\tau_i). \quad (1)$$

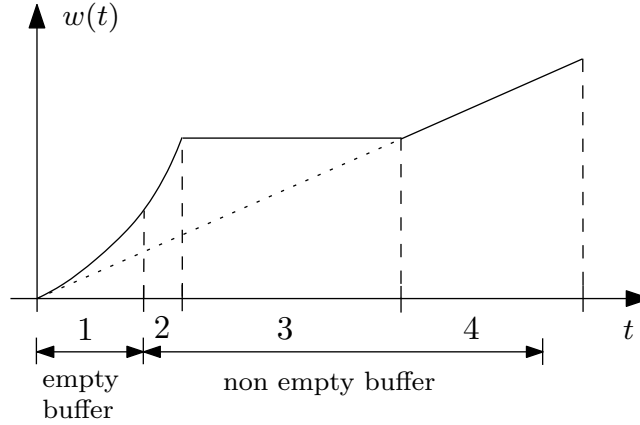


FIG. 1 – The four different phases

where w_i is the sending window, $\tilde{\tau}$ the smallest round trip time ever observed and τ_i the latest sample of the round trip time. The sending window is then quickly increased if Δ_i is lower than a threshold γ , and decreased otherwise :

$$w_{i+1} = \begin{cases} w_i + \alpha w_i^k & , \text{if } \Delta_i < \gamma \\ w_i - \zeta \Delta_i + 1 & , \text{if } \Delta_i \geq \gamma. \end{cases} \quad (2)$$

Where γ was initially a fixed threshold, with proposed value equal to 30 [16]. It is now dynamically adjusted between 5 and 30, according to the window size on loss events [14], using the TUBE algorithm. For the sake of simplicity, and due to the fact that it has been only recently introduced we do not explicitly model TUBE in the remainder of the paper. The authors of CTCP proposed to set $\alpha = 1/8$, $k = 3/4$ and $\zeta = 1$. Unless otherwise specified, we are going to use the same values for all the numerical examples and simulations.

2 Different Phases of CTCP

In order to model an isolated CTCP connection we consider a simple fluid system comprised of a sender and receiver connected by a FIFO queue, with rate μ and buffer size b . The sender uses a single CTCP connection to send data to the receiver. For the sake of simplicity we will assume that there is no exogenous traffic in the FIFO queue ; that the sender has an unlimited amount of data to send ; and that the advertised window is never a limiting factor.

We are going to consider different cases corresponding to different phases in the evolution of the sending window. Each particular realization will take a different path, depending on the specific values of all the parameters involved. But given the initial state it is always possible to determine the final state at the end of the first phase. Based on this, the next phase is chosen and the process is repeated.

We assume that the time origin coincides with the starting time of the phase being discussed. Therefore all the times are meant as an *offset from the start of each phase*. The initial conditions as well always refer to those of the phase in question.

In all phases w is the total congestion window, while w_c and w_d are the congestion and the delay component respectively. Let $\tau(t)$ be the round trip time at time t and $\tilde{\tau}$ be the propagation delay (including processing delays) so that :

$$\tau(t) \triangleq \tilde{\tau} + \frac{x(t)}{\mu}$$

where $x(t)$ is the backlog at time t . In the case of an isolated connection the smallest round trip sample coincides with the propagation delay, and we use $\tilde{\tau}$ to indicate both quantities.

Let $R_{\text{in}}(t)$ be the input process, that is the total amount of traffic sent by the source up to (and including) time t . Similarly, let $R_{\text{q}}(t)$ be the output process of the FIFO queue, that is the total amount of traffic that has left the queue since the beginning of the current phase.

As done in previous works (see, for example, [1]) we are going to model the flow of acknowledgments from the receiver back to the sender with a process equal to the output process of the queue delayed by a fixed propagation delay ($\tilde{\tau}$). Clearly this is not what happens in a real network but it is a correct abstraction as the acknowledgments do inform the sender of the total amount of traffic arrived at the receiver. Using this model we have that :

$$R_{\text{in}}(t) = R_{\text{out}}(t) + w(t)$$

where $R_{\text{out}}(t) = R_{\text{q}}(t - \tilde{\tau})$, that is the output of the queue delayed by $\tilde{\tau}$. Without the loss of generality we will assume that $R_{\text{out}}(0) = 0$ at the beginning of each phase so that $R_{\text{in}}(0) = w(0)$ and the total amount of traffic sent between 0 and t is $R_{\text{in}}(t) - R_{\text{in}}(0) = R_{\text{in}}(t) - w(0)$. We are also going to assume that whenever the window is instantaneously increased by a certain amount R_{in} increases instantaneously by the same amount. In other words whenever the window has a discontinuity (a jump) so does the input process. This is the same as assuming that the capacity of the link connecting the sender to the FIFO queue is much greater than μ .

Figure 1 shows the four different phases of CTCP, we will use a number to identify each of them and its corresponding parameters. For example t_i is the end time of the i -th phase and w_i^* is the value of the window at the beginning of the i -th phase. For each phase we explicitly compute the expression of $w(t)$, which can be used to compute the final value of w and the total amount of traffic sent during each phase. It is also possible to compute w_c and then find w_d using the relation $w = w_c + w_d$. The final values of w and w_c can be used as the initial values of the following phase. In section 4, we will show how several different combinations of these phases are possible.

2.1 Empty Buffer and Superlinear Increase

Let w_1^* be the window at the beginning of the phase. Given that the buffer is empty $\tau(t) = \tilde{\tau}$. During this phase the sender increases w by αw^k every $\tilde{\tau}$ units of time. We have not been able to find a closed form expression for the recursion $w_{n+1} = w_n + \alpha w_n^k$ so that it is not possible to find the exact expression of $w(t)$. Instead we use the same approach as in [16] by approximating the derivative of

w with its increment during one round trip time¹ :

$$\frac{dw}{dt} \simeq \frac{\alpha w^k}{\tilde{\tau}},$$

combining this with the initial condition $w(0) = w_1^*$ we have :

$$w(t) = \left(\frac{(1-k)\alpha}{\tilde{\tau}} t + w_1^{*1-k} \right)^{\frac{1}{1-k}}. \quad (3)$$

In a fluid system the buffer would be empty as long as the instantaneous arrival rate is greater than μ . As $R_{\text{in}}(t)$ represents the total amount of traffic received until time t , its first derivative (provided it exists) is the instantaneous arrival rate. Using the fact that $R_{\text{in}}(t) = R_{\text{out}}(t) + w(t)$ and that, in this phase, $R_{\text{out}}(t) = R_{\text{in}}(t - \tilde{\tau})$ it is possible to compute $R'_{\text{in}}(t)$. But it is not possible to find a closed form solution for the equation $R'_{\text{in}}(t) = \mu$.

Given that we cannot explicitly compute when $R'_{\text{in}}(t) = \mu$ we will use a different approach. Clearly, if $w > \mu\tilde{\tau}$ the backlog has to be greater than zero : if there are more than $\mu\tilde{\tau}$ bits in flight some of them have to be in the buffer waiting to be served. This is also true in a packet based system and we will use this value to determine when this phase ends. To be more precise in a packet based system the backlog is non-zero every time a packet arrives at the buffer before the departure of the previous one and this can happen even for small values of the window but only for limited periods of time. Provided $\mu < \mu\tilde{\tau}$, even if the sender can send some packets in a burst such burst is not big enough to cause a persistent backlog. In the sense that before the beginning of the next round trip time the backlog will be zero.

In both the fluid and the packet based system the backlog will be non-zero at some point before $w = \mu\tilde{\tau}$ but we can use this value as an approximation to compute the end time of this phase. The simulations results in Section 4 validate this assumption. From (3) it follows that $w < \mu\tilde{\tau}$ until time

$$t_1 = \frac{\mu^{1-k}\tilde{\tau}^{2-k} - \tilde{\tau}w_1^{*1-k}}{(1-k)\alpha}.$$

In order to compute $R_{\text{in}}(t)$ we would need to know $R_{\text{out}}(t)$ starting from time 0 but this depends on what happened before this phase. At the same time, given that the buffer is empty at time 0, after $t = \tilde{\tau}$ the system will loose memory of what happened before the beginning of this phase. For the sake of simplicity we are going to assume that $R_{\text{out}}(t) = \frac{w_1^*}{\tilde{\tau}}t$ for $0 \leq t \leq \tilde{\tau}$. That is that, during the first round trip time, acknowledgments arrive at a constant rate. In this case $R_{\text{out}}(t) = R_{\text{in}}(t - \tilde{\tau})$ after $\tilde{\tau}$, and :

$$R_{\text{in}}(t) = \begin{cases} \frac{w_1^*}{\tilde{\tau}}t + w(t) & , \text{ if } t < \tilde{\tau} \\ R_{\text{in}}(t - \tilde{\tau}) + w(t) & , \text{ if } t \geq \tilde{\tau}. \end{cases}$$

The total amount of traffic sent during this phase is :

$$R_{\text{in}}(t_1) = \frac{w_1^*}{\tilde{\tau}}z + \sum_{n=0}^N w(t_1 - n\tilde{\tau}), \quad (4)$$

¹The same approximation can be obtained by considering how much each acknowledgment contributes to the increase of the window and then assuming that the acknowledgments arrive at rate $\frac{w}{\tilde{\tau}}$.

where we have “decomposed” t_1 as the sum of N propagation delays and a remainder z , that is $t_1 = N\tilde{\tau} + z$. Using (4) we can compute the average sending rate as $\bar{\lambda}_1 \triangleq \frac{R_{\text{in}}(t_1)}{t_1}$.

Alternatively we can exploit the fact that $\lambda = \frac{w(t)}{\tau}$ and compute the total amount of traffic as :

$$\begin{aligned} R_{\text{in}}(t) &= \int_0^{t_1} \frac{w(t)}{\tilde{\tau}} dt \\ &= \frac{(\mu\tilde{\tau})^{2-k} - w_1^{*2-k}}{(2-k)\alpha}. \end{aligned}$$

Using this approach the average sending rate is² :

$$\frac{R_{\text{in}}(t_1)}{t_1} = \frac{(1-k)(w_1^{*k}(\mu\tilde{\tau})^2 - w_1^{*2}(\mu\tilde{\tau})^k)}{(2-k)\tilde{\tau}(w_1^{*k}\mu\tilde{\tau} - w_1^*(\mu\tilde{\tau})^k)}.$$

In order to correctly characterize the following phase we also need to compute the final value of the two components of the window. During this phase w_c grows by 1 each round trip time so that

$$w_c(t) = w_{c,1} + \frac{t}{\tilde{\tau}} \quad (5)$$

where $w_{c,1}$ is the initial value of the congestion component of the total window at the beginning of this phase. Therefore at the end of this phase :

$$w_c(t_1) = w_{c,1} + \frac{(\mu\tilde{\tau})^{1-k} - w_1^{*1-k}}{(1-k)\alpha}.$$

As $w(t_1) = \mu\tilde{\tau}$ and $w = w_c + w_d$ it follows that

$$w_d(t_1) = \mu\tilde{\tau} - w_{c,1} - \frac{(\mu\tilde{\tau})^{1-k} - w_1^{*1-k}}{(1-k)\alpha}.$$

Even though in the fluid model a buffer overflow is not possible in this phase, in a packet based system this can happen if the increment of window during one round trip time is bigger than the buffer size. As we are assuming that the capacity of the link connecting the source to the bottleneck is much greater than the bottleneck capacity, whenever the sender updates its sending windows using the $w = w + \alpha w^k$ relationship, the sender will inject a burst of size αw^k into the network. Clearly if this burst is bigger than the size of the buffer there will be an overflow. If b is the buffer size this will happen when $\alpha w^k \geq b$ that is when $w \geq (b/\alpha)^{\frac{1}{k}}$. Given that this phase will end when $w = \mu\tilde{\tau}$ we also need $(b/\alpha)^{\frac{1}{k}} < \mu\tilde{\tau}$. In other words there will be a buffer overflow in this phase only if $b < \alpha(\mu\tilde{\tau})^k$. If this is true the phase will end not when $w = \mu\tilde{\tau}$ but when $w = (b/\alpha)^{\frac{1}{k}}$. In this case we have :

$$\begin{aligned} t_1 &= \frac{\tilde{\tau}}{(1-k)\alpha} \left(\left(\frac{b}{\alpha} \right)^{\frac{1-k}{k}} - w_1^{*1-k} \right) \\ w_c(t_1) &= w_{c,1} + \frac{1}{(1-k)\alpha} \left(\left(\frac{b}{\alpha} \right)^{\frac{1-k}{k}} - w_1^{*1-k} \right) \end{aligned} \quad (6)$$

²Clearly in the final version only one of the two approaches will be presented. I have to check which one give the best estimate.

where we have used (5) to compute the final value of w_c .

2.1.1 Safe Values for k

While, at this time, we do not have any different recommendations, we would like to point out the consequences of using value of k greater than one. If $k > 1$ then, for any positive value of α there exist a value of the window, say \hat{w} , such that if $w > \hat{w}$ then the increment of the window is greater than during slow start, that is $\alpha w_n^k > w_n$. This is because if $k > 1$ and $w_n > \alpha^{\frac{1}{1-k}}$ then $\alpha w_n^k > w_n$. By choosing small values of α the value of \hat{w} can be increased. But such an approach would leave the possibility of having an exponential increase of the window in certain network scenarios. The case $k = 1$ would lead to an increase similar to slow start, which would probably be excessive in several cases. Because of this we are going to assume that $0 < k < 1$.

2.2 Non Empty Buffer and Superlinear Increase

Again let w_2^* be the window and x_2 the backlog at the beginning of the phase. In this case $w_2^* \geq \mu\tilde{\tau}$ so that either $x(0) > 0$ or $x(0 + \epsilon) > 0$ for any $\epsilon > 0$, that is the backlog is always non-zero immediately after time 0. Note that in throughout this phase the round trip time is not a constant anymore. We also know that the acknowledgments are arriving with rate μ as the buffer is non-empty.

Given that the window is increased when w acknowledgments are received we can say that, on average, each acknowledgment increases the window by $\alpha w^k/w$. This is true also if delayed acknowledgments are used : as explained in the draft standard ([13], section 6) the round trip time is estimated matching some of the arriving acknowledgments with the corresponding segments and computing the difference between the arrival time of the acknowledgment and when the segment was sent. Delay acknowledgments do diminish the number of available samples but they should not radically change the value of the estimates. Especially considering that only a fraction of the packets is used as samples anyway, in order to diminish the computing overhead.

Between t and $t + \delta$ we receive $\mu\delta$ acknowledgments so that

$$w(t + \delta) = w(t) + \mu\delta\alpha w^{k-1},$$

and

$$\frac{w(t + \delta) - w(t)}{\delta} = \mu\alpha w^{k-1}. \quad (7)$$

By taking the limit as δ approaches zero of (7) we have that

$$\frac{dw}{dt} = \mu\alpha w^{k-1}$$

which implies

$$w(t) = ((2 - k)\alpha\mu t + w_2^{*2-k})^{\frac{1}{2-k}}, \quad (8)$$

where we have used the initial condition $w(0) = w_2^*$.

The superlinear increase will continue as long as the backlog is less than γ . More precisely until the $\Delta_i < \gamma$, where Δ_i is the backlog estimate computed

by the sender each round trip time. If we assume that this estimate is correct, this phase will end when $x(t) = \gamma$. Given that $w(t) = x(t) + \mu\tilde{\tau}$ (that is all the bits in flights are either in the buffer or in the delay element) we can use (8) to compute t_2 such that $x(t_2) = \gamma$:

$$t_2 = \frac{\theta^{2-k} - w_2^{*2-k}}{(2-k)\alpha\mu}$$

where $\theta = \gamma + \mu\tilde{\tau}$ is the final value of w during this phase. Note that this expression holds for any value of $\theta > w_2^*$ (we will use this fact shortly to consider the case when this phase ends because of a buffer overflow and not because $w = \gamma + \mu\tilde{\tau}$).

During this whole phase the congestion window component grows by 1 each round trip time. Using the same argument as above we have that each acknowledgment increases the congestion window by $\frac{1}{w}$ so that

$$\frac{dw_c}{dt} = \frac{\mu}{mw}, \quad (9)$$

where m is the number of packets acknowledged by each acknowledgment.

We know that, until t_2 , $w(t)$ is given by (8) so that (9) becomes (see [6] for the solution) :

$$\frac{dw_c}{dt} = \frac{\mu}{m} \left((2-k)\alpha\mu t + w_2^{*2-k} \right)^{\frac{1}{k-2}}.$$

which implies

$$w_c(t) = \frac{1}{(k-1)m\alpha w_2^{*k}} \left((w_2^{*2-k} - (k-2)t\alpha\mu)^{\frac{1}{k-2}} \times \right. \\ \left. \times ((k-2)t w_2^{*k} \alpha\mu - w_2^2) + w_2 \right) + w_{c,2} \quad (10)$$

where we have used the initial condition $w_c(0) = w_{c,2}$. At time t_2 we know that $x(t) = \gamma$ and that

$$w_c(t_2) = w_{cb} + \frac{w_2^{*1-k} - (\gamma + \mu\tilde{\tau})^{1-k}}{(k-1)m\alpha}.$$

During this phase $R_{\text{out}}(t) = \mu t$, so that $R_{\text{in}}(t) = R_{\text{out}}(t) + w(t) = \mu t + w(t)$, given that $R_{\text{in}}(0) = w_2^*$. As $w(t_2) = \theta$ we have that

$$R_{\text{in}}(t_2) - R_{\text{in}}(0) = \theta + \frac{\theta^{2-k} - w_2^{*2-k}}{(2-k)\alpha} - w_2^*$$

so that

$$\bar{\lambda}_2 = \frac{\mu\theta^k (w_2^* - w_2^{*k}\alpha(\theta - w_2^*)(k-2)) - w_2^{*k}\theta^2\mu}{w_2^2\theta^k - w_2^k\theta^2} \\ = \mu + \frac{(2-k)\alpha(w_2^* - \theta)(w_2^*\theta)^k\mu}{w_2^{*2}\theta^k - w_2^{*k}\theta^2} \\ \bar{\lambda}_2 = \mu + \frac{(2-k)\alpha(w_2^* - \theta)(w_2^*\theta)^k\mu}{w_2^{*2}\theta^k - w_2^{*k}\theta^2}$$

If $\gamma > b$ the backlog cannot reach γ and this phase will, instead, end with a buffer overflow when $w(t) = b + \mu\tilde{\tau}$. In this case we can use all the above expressions with $\theta = b + \mu\tilde{\tau}$.

2.3 Constant Window

As a first approximation we are going to assume that during this phase the window is constant and equal to θ , as in [16]. That is we are assuming that w_d is decreasing at the same rate as w_c is increasing so that the total window is constant. During this phase (9) becomes

$$\frac{dw_c}{dt} = \frac{\mu}{m\theta}$$

so that

$$w_c(t) = \frac{\mu}{m\theta}t + w_{c,3}. \quad (11)$$

Where $w_c(0) = w_{c,3}$ is the initial value of w_c for this phase. This expression holds until time t_3 such that $w_c(t_3) = \theta$. From (11) we have

$$t_3 = \frac{m\theta(\theta - w_{c,3})}{\mu}.$$

In this phase $R_{\text{out}}(t) = \mu t$ as well so that $R_{\text{in}}(t) = \mu t - \theta + w(t) = \mu t$. The total amount of traffic sent is $R_{\text{in}}(t_3) = m\theta(\theta - w_{c,3})$ and the average sending rate is $\bar{\lambda}_3 = \mu$.

2.4 Linear Increase

In this phase the delay component is zero (i.e. $w_d = 0$) so that $w = w_c$ and

$$\frac{dw_c}{dt} = \frac{dw}{dt} = \frac{\mu}{mw}.$$

In this case

$$w(t) = \sqrt{\frac{2\mu}{m}t + w_4^{*2}} \quad (12)$$

where we have used the initial condition $w(0) = w_4$.

This phase will end when the buffer overflows, that is when $x(t) = b$. Using (12) we can compute t_4 such that $x(t_4) = b$, that is $w(t_4) = b + \mu\tilde{\tau}$:

$$t_4 = \frac{m((b + \mu\tilde{\tau})^2 - w_4^{*2})}{2\mu}.$$

Using once more the fact that $R_{\text{out}}(t) = \mu t$ we can compute $R_{\text{in}}(t)$ as $R_{\text{in}}(t) = \mu t - w_4^* + w(t)$ so that

$$R_{\text{in}}(t_4) - R_{\text{in}}(0) = b - w_4 + \mu\tilde{\tau} + \frac{m}{2}((b + \mu\tilde{\tau})^2 - w_4^*)$$

and

$$\bar{\lambda}_4 = \frac{2\mu(b - w_4^* + \mu\tilde{\tau} + \frac{m}{2}((b + \mu\tilde{\tau})^2 - w_4^*))}{m((b + \mu\tilde{\tau})^2 - w_4^*)}.$$

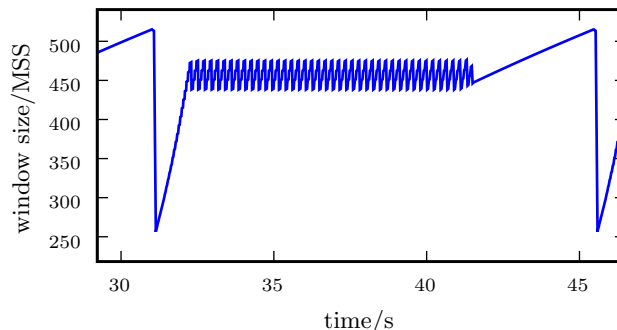


FIG. 2 – Oscillations of w (ns-2 simulation : $\mu = 100$ Mb/s, $\tilde{\tau} = 50$ ms)

3 Oscillations During Phase 3

As noted in [7], during phase 3, the algorithm described in [16] and [14] causes the window to oscillate around a constant value. Figure 2 shows the evolution of the sending window between two packet losses. Clearly, during phase 3, the window is constantly incremented and decremented. In this particular case the oscillations are realized by increasing the window three times and then decreasing it once but this is not the only possible pattern : depending on the system parameters it is possible to have different patterns. In each case a series (two or more) of increasing phases is followed by one (or more) decreasing phase(s). We use two integers $m : n$ to indicate the type of oscillations, with m and n representing the number of increasing and decreasing phases, respectively.

While considering the window as a constant can be a useful approximation, it is not always possible to ignore the oscillations during this phase. In at least two cases it is important to consider them. First, if the window were kept constant (such that $w = \mu\tilde{\tau} + \gamma$) phase 3 would take place as long as $b > \gamma$, but, because of the oscillations, the window will reach a value greater than $\mu\tilde{\tau} + \gamma$ causing a buffer overflow and a premature end of phase 3. If θ_{\max} and θ_{\min} are the maximum and minimum values reached by w during each oscillation, phase 3 will take place only if $\mu\tilde{\tau} + b < \theta_{\max}$. If this condition does not hold phase 2 ends with a buffer overflow with a potentially adverse effect on the throughput. In most cases, if phase 2 ends with a buffer overflow, the buffer will be empty after each window cut, reducing the throughput. Second, even if phase 3 does take place, the oscillations of the window will cause the backlog to oscillate as well, which can have a negative impact on the other traffic going through the same bottleneck link. Even if in this work we do not consider exogenous traffic we, nonetheless, think that it is important to highlight this consequence of the oscillations.

While it is possible to use a fluid model to estimate the size of the oscillations we believe it is easier to use the discrete event model presented in [7] to precisely characterize these oscillations. In the remainder of this section, after a brief presentation of the model used in [7], we will extend that work by proving the existence of the fixed point solution (in one case) and by showing how it is possible to determine the $m : n$ pattern of the oscillations based on the bandwidth delay product.

Note that all the results presented in this section depend on properties of the specific Linux implementation we have used [3]. So some care should be taken in applying them to other implementations. At the same time we believe that these issues will be present in any implementation of the algorithm presented in [16, 14]. Furthermore the rest of the model depends only on θ_{\max} and θ (the maximum and average value of the window during phase 3) so that it suffices to find these two values for each implementation.

3.1 Linux Implementation

Once every round trip time the Linux implementation, that we used for the simulations, instead of using (1), computes Δ_i as :

$$\Delta_i = w_{i-1} (1 - \tilde{\tau}/\tau_i) \quad (13)$$

where w_{i-1} is the size of sending window the last time the window was updated (that is one round trip time before). As the sender uses acknowledgments to estimate the round trip time any such estimate refers to the packet being acknowledged. Given that this packet was sent one round trip time ago it is more appropriate to use w_{i-1} rather than w_i .

The round trip time estimate τ_i is the smallest value of all samples collected during the last round trip time. This choice is explained, by a comment in the source code, as a way to minimize the impact of delayed acknowledgments.

In the case of a single connection with no cross traffic τ_i depends only on the window dynamics so that it is possible to express it as a function of past values of the window. In particular we have that :

$$\tau_i = \begin{cases} \min \left[\frac{w_{i-2}+1}{\mu}, \tilde{\tau} \right] & , \text{ if } w_{i-1} > w_{i-2} \\ \min \left[w_{i-1} \frac{\tilde{\tau}}{\tau_{i-1}}, \tilde{\tau} \right] & , \text{ if } w_{i-1} < w_{i-2} \end{cases} \quad (14)$$

That is if the window was not reduced at the last update the first expression is used, while if the window was reduced at the last update then the second one is used (see [7] for more details).

3.2 Fixed Points

Given an initial value for the window it is possible, using equations (13), (14) and the window update function $w_{i+1} = w_i + \alpha w_i^k$, to explicitly compute the evolution of the window. For example, in the case of the 3:1 cycle if w_1 is the initial value of the window, the final value will be $w_4(w_1) - \Delta_4(w_1) + 1$, where w_4 is the value after w_1 was updated three times, (the value at the beginning of the fourth step in the cycle). And Δ_4 is the value of Δ_i at the beginning of the same cycle. The plus one takes into account the fact that this cycle covers four round trip times and in three of them the window is incremented while in the fourth one w_d is decremented while w_c is still incremented by one. Using the above equations it is possible to compute the values of $w_4(w_1)$ and $\Delta_4(w_1)$ in closed form but the expressions are lengthy and do not offer any insight and are not presented here.

In Figure 2 the oscillations quickly reach a steady state, as they do in all the simulations we have observed. As discussed in [7], if a steady state solution

does exist it must satisfy the condition that the final value of one cycle is the same as the initial value of the following one. If $f(w_1)$ is the final value of a cycle starting with $w = w_1$ we can find the steady state solution by solving the fixed point equation $f(w_1) = w_1$ for w_1 . Note that that the expression of $f(w_1)$ depends on the type of oscillations. In general, for the $m : 1$ cases $f_{m:1}(w_1) = w_{m+1}(w_1) - \Delta_{m+1}(w_1) + 1$ and $f_{m:2} = w_{m+1}(w_1) - \Delta_{m+1}(w_1) - \Delta_{m+2}(w_1) + 2$ for the $m : 2$ cases.

Given that $0 < k < 1$ it is not possible to find a closed form expression for the solution of $f_{m:n}(w_1) = w_1$ (it is possible to write $f_{m:n}$ in closed form but the expression is somewhat long and it is not reported here). At the same time it is possible to use efficient numerical algorithms to find the solution as $f_{m:n}$ is a continuous function. The following theorem shows that, in the $3 : 1$ case, such solution does indeed always exist. We believe that a similar argument can be used for the other cases as well. For the proof see the companion technical report [6].

Theorem 1 $f_{3:1}(w_1) = w_1$ has always one solution in $[\frac{\mu\tilde{\tau}}{2}, \infty)$, provided $0 < k < 1$, $\frac{\mu\tilde{\tau}}{2} > \alpha^{-\frac{1}{k}}$, and $\frac{\mu\tilde{\tau}}{2} > \alpha^{\frac{1}{1-k}}$.

Proof Using 13,14 we have that $f_{3:1}(w_1) = w_4(w_1) - \Delta_4(w_1) = w_4(w_1) - w_3(w_1) \left(1 - \frac{\mu\tilde{\tau}}{w_2+1}\right)$ and we can rewrite $f_{3:1}(w_1) = w_1$ as $g_1(w_1) = g_2(w_1)$ where $g_1(w_1) = w_4(w_1) - w_1$ and $g_2(w_1) = w_3(w_1) \left(1 - \frac{\mu\tilde{\tau}}{w_2(w_1)+1}\right)$. We are going to show that $g_1(\frac{\mu\tilde{\tau}}{2}) \geq g_2(\frac{\mu\tilde{\tau}}{2})$ and that

$$\lim_{w_1 \rightarrow +\infty} [g_2(w_1) - g_1(w_1)] \geq 0.$$

As $f_{m:n}(w_1)$ is the composition of continuous function is itself continuous (for $w_1 > 0$), and so are g_1 and g_2 . To see why $f_{m:n}$ is continuous consider that $w_1 > 0$ implies $w_i > 0$ and (as $\tau_i \geq \tilde{\tau} > 0$) $\Delta_i > 0$. Using the intermediate value theorem, we can, therefore, conclude that $g_1(w_1) = g_2(w_1)$ for some $w_1 \in [\frac{\mu\tilde{\tau}}{2}, \infty)$.

We will first show that $g_1(\frac{\mu\tilde{\tau}}{2}) \geq g_2(\frac{\mu\tilde{\tau}}{2})$. We have that, as $w_4 > w_2$:

$$\begin{aligned} g_1(w_1) &= w_4(w_1) - w_1 \\ &> w_2(w_1) - w_1 \\ &= \alpha w_1^k. \end{aligned}$$

Where the last equality follows from the definition of w_i . If $\frac{\mu\tilde{\tau}}{2} > \alpha^{-\frac{1}{k}}$ then $g_1(\frac{\mu\tilde{\tau}}{2}) > 1$. For g_2 we have :

$$\begin{aligned} g_2\left(\frac{\mu\tilde{\tau}}{2}\right) &= w_2\left(\frac{\mu\tilde{\tau}}{2}\right) + 1 - \mu\tilde{\tau} \\ &= -\frac{\mu\tilde{\tau}}{2} + 1 + \alpha\left(\frac{\mu\tilde{\tau}}{2}\right)^k \\ &\leq -\frac{\mu\tilde{\tau}}{2} + 1 + \frac{\mu\tilde{\tau}}{2} \\ &= 1, \end{aligned}$$

where the first two equalities follows from the definition of g_2 , w_i and Δ_i ; the inequality holds because $\alpha(\frac{\mu\tilde{\tau}}{2})^k \leq \frac{\mu\tilde{\tau}}{2}$ (as $\frac{\mu\tilde{\tau}}{2} \geq \alpha^{\frac{1}{1-k}}$). Therefore $g_2(\frac{\mu\tilde{\tau}}{2}) \leq g_1(\frac{\mu\tilde{\tau}}{2})$.

Next we will show that for $w_1 \rightarrow \infty$ the opposite inequality holds. Using the definitions of g_2 and g_1 we have :

$$g_2(w_1) - g_1(w_1) = w_1 - \alpha(w_3(w_1))^k - \frac{w_3(w_1)}{w_2(w_1) + 1} \mu\tilde{\tau}.$$

Starting with the last term we have :

$$\begin{aligned} \lim_{w_1 \rightarrow \infty} \frac{w_3(w_1)}{w_2(w_1) + 1} \mu\tilde{\tau} &= \\ &= \lim_{w_1 \rightarrow \infty} \frac{w_2(w_1) + \alpha(w_2(w_1))^k}{w_2(w_1) + 1} \mu\tilde{\tau} \\ &= \lim_{w_1 \rightarrow \infty} \frac{w_2(w_1)}{w_2(w_1)} \mu\tilde{\tau} + \lim_{w_1 \rightarrow \infty} \frac{\alpha(w_2(w_1))^k}{w_2(w_1) + 1} \mu\tilde{\tau} \\ &= \mu\tilde{\tau} \end{aligned}$$

where the first equality follows from the definition of $w_3(w_1)$ and the last one from the fact that $k < 1$ and $\lim_{w_1 \rightarrow \infty} w_i(w_1) = \infty$. For the first two terms consider that if $f(x)/h(x) \rightarrow 0$ and if $f(x) \rightarrow \infty$ then $f(x)-h(x) \rightarrow \infty$ (where all the limits are as $x \rightarrow \infty$). Therefore it suffices to show that $\alpha(w_3(w_1))^k/w_1 \rightarrow 0$. Using the definitions of w_3 and w_2 and then l'Hôpital's rule we have

$$\begin{aligned} \lim_{w_1 \rightarrow \infty} \frac{\alpha(w_3(w_1))^k}{w_1} &= \\ &= \lim_{w_1 \rightarrow \infty} \frac{\alpha \left[w_1 + \alpha w_1^k + \alpha (w_1 + \alpha w_1^k)^k \right]^k}{w_1} \\ &= \lim_{w_1 \rightarrow \infty} \alpha k \left[w_1 + \alpha w_1^k + \alpha (w_1 + \alpha w_1^k)^k \right]^{k-1} \times \\ &\quad \left[1 + \alpha k w_1^{k-1} + \alpha k (w_1 + \alpha w_1^k)^{k-1} (1 + \alpha k w_1^{k-1}) \right] \\ &= 0 \end{aligned}$$

where the last step follows from the fact that $k < 1$.

3.3 Different Oscillation Cycles

The fixed point equations presented in the previous section can be used to calculate the maximum and minimum values of the oscillations but do not indicate which cycle type will take place. For any value of the bandwidth delay product it is always possible to have a certain type of oscillations, provided the window is reduced by the appropriate amount at the appropriate time. But CTCP calls for the window to be reduced only when $\Delta_i \geq \gamma$ so that, for any specific bandwidth delay product, certain patterns are not feasible because they need the window to be reduced by a factor smaller than γ .

Using simulations, we have observed several different types : 3 :1, 5 :2, 4 :2 and 3 :2 but also 4 :1, 5 :1 and 6 :1. In most cases, the type of oscillations

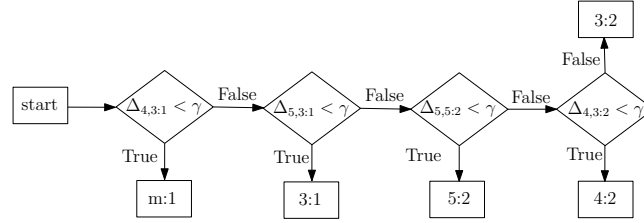
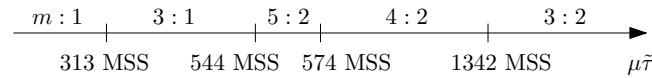


FIG. 3 – How to find the type of oscillations

case	cond. 1	cond. 2	cond. 3	$\mu\tilde{\tau}$ interval ($\gamma = 30$)
2 :1	$\Delta_2 < \gamma$	$\Delta_3 \geq \gamma$	$\Delta_4 < \gamma$	[545, 558]
3 :1	$\Delta_3 < \gamma$	$\Delta_4 \geq \gamma$	$\Delta_5 < \gamma$	[312, 545]
5 :2	$\Delta_5 < \gamma$	$\Delta_6 \geq \gamma$	$\Delta_7 \geq \gamma$	[546, 573]
4 :2	$\Delta_4 < \gamma$	$\Delta_5 \geq \gamma$	$\Delta_6 \geq \gamma$	[559, 1411]
3 :2	$\Delta_3 < \gamma$	$\Delta_4 \geq \gamma$	$\Delta_5 \geq \gamma$	[1342, ∞)

TAB. 1 – Limits for $\Delta_{i,m:n}$ FIG. 4 – Type of oscillations for $\gamma = 30$

depends on the bandwidth delay product and does not change during the course of the simulation. This can be easily explained by observing that, every round trip time, the sender will compute Δ_i and compare it with γ . If $\Delta_i \geq \gamma$ then w_d (and therefore the sending window) decreases, otherwise it increases. For example the 3 :1 oscillations can only take place if three conditions are met. The first one is $\Delta_{4,3:1} \geq \gamma$, so that the window will be cut at the fourth step. The second is that $\Delta_{5,3:1} < \gamma$, otherwise the window would be reduced another time and the cycle type would be 3 :2. The third one is that $\Delta_{3,3:1} < \gamma$, so that the window is not cut at the third step, and only two increments, in which case the cycle would be 2 :1.

For each case it is possible to use a similar argument to find the boundary values for the appropriate $\Delta_{i,m:n}$. Table 1 shows the three conditions for several cases, where the Δ_i 's in each row are those of the corresponding case : for example Δ_3 on the second row is a shorthand for $\Delta_{3,3:1}$ and Δ_5 on the third row represents $\Delta_{5,5:2}$. The values in the last column correspond to the case when $\gamma = 30$, with $\mu\tilde{\tau}$ expressed in terms of MSS of 1500 B. As indicated by the last column of Table 1, these are necessary conditions for a certain oscillation type but they are not sufficient in the sense that it is possible for two oscillation types to be feasible for the same value of the bandwidth delay product. For example if $546 \leq \mu\tilde{\tau} \leq 558$ both the 2 :1 and the 5 :2 oscillations are possible and if $559 \leq \mu\tilde{\tau} \leq 573$ the 4 :2 and 5 :2 cases are possible.

Based on an extensive set of simulations it seems that certain conditions have “priority” in the sense that as long as they are satisfied the corresponding case will take place. For example the 5 :2 case happens whenever $546 \leq \mu\tilde{\tau} \leq 573$

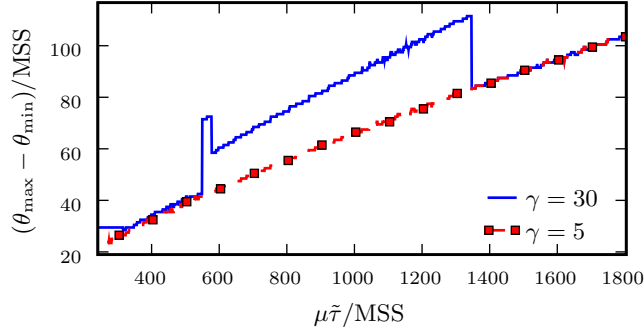


FIG. 5 – Amplitude of the oscillations

even though the 2 :1 and 4 :2 cases could take place as well. In only one case ($\mu\tilde{\tau} = 557$) we were able to observe the 2 :1 oscillations. In some cases, for values very close to some of the brake points reported in table 1, the type of oscillations observed in the simulations is not the same as the one given by the model but, instead, it is the neighboring one (e.g. 3 :2 instead of 4 :2 or vice versa).

Given that we did not run a simulation for every possible value of the bandwidth delay product we might have missed some other exceptions but we are reasonably confident that it is possible to use these “priorities” to find the oscillation type. Figure 4 shows the type of oscillations obtained using this method for $\gamma = 30$ while Figure 3 presents it in the form of a flow chart. In both figures we use the oscillation type $m : 1$ to represent the 4 :1, 5 :1 and 6 :1 oscillations. We have not written more detailed tests for these cases because they happen only when the window is small with respect to γ so that four or more increments are needed before the window can be reduced (recall that the size of the reduction is always greater than γ). The window will oscillate around such small values only when the bandwidth delay product is small and, in this case, the duration of phase 3 (“constant window”) is much smaller than the duration of phase 4.

As discussed in [7] the Linux kernel does not use floating point instructions, so that the implementation we used approximates all the operations using integer operations. All the numerical values used in this section and the following ones are computed using the same approximations as the Linux implementation as there can be non-negligible differences between using floating point and integer operations. Especially when computing for which value of the bandwidth delay product certain Δ_i ’s are equal to γ . (See [7] for more details.)

3.4 Oscillation Amplitude

Using the algorithm presented in Figure 3 it is possible to compute the size of the oscillations as a function of the bandwidth delay product. Figure 5 shows the amplitude of the oscillations for $\gamma = 30$ and $\gamma = 5$. These two values of γ are the maximum and minimum values suggested in [14]. For $\gamma = 30$ the type of oscillations changes with the bandwidth delay product (see Figure 4) explaining the discontinuities in the curve. While for $\gamma = 5$ the only type of oscillations is always 3 :2.

When, for $\gamma = 30$, the oscillations are $m : 1$ ($m \in \{4, 5, 6\}$) the difference between the two curves is not too big but it is the case that the oscillations are smaller when $\gamma = 5$ (in this case the oscillations have the same order of magnitude as γ). For the 3 : 1 case the difference becomes smaller but it is still true that the oscillations are slightly smaller when $\gamma = 5$. The difference between the two curves becomes significant for the 5 : 2 and 4 : 2 cases. While for the 3 : 2 case the oscillations are the very same whether $\gamma = 30$ or $\gamma = 5$. This is because the larger window sizes, due to the larger bandwidth delay product, cause each increment of the sending window to be fairly large. For example, if $w = 1500$ then $\alpha w^k = 30.13$ and the difference between Δ_i and Δ_{i+1} is comparable. As a consequence, Δ_i is either 0 or of the same order of magnitude of αw^k and any value of γ smaller than this will not make any difference in the behavior of the sender. This indicates that even using smaller values for γ (and/or the TUBE algorithm [14]) would not limit the size of the oscillations for larger values of the bandwidth delay product. On this figure we also see that the oscillations might have an impact on other flows sharing the same bottleneck link.

4 Combining Multiple Phases

In this section we are going to analyze all the possible combinations, six in total, of the phases described in section 2, identifying under what conditions each combination will take place. In most case, we will also compare the evolution of the window given by the model with ns-2 simulations. For these we used ns-2.33 and the CTCP implementation [3].

We assume that CTCP uses fast recovery and fast retransmit. As a consequence, assuming that a single packet is lost, after receiving the third duplicate acknowledgment and after reducing the window by a factor of $(1 - \beta)$, the sender will not transmit until it has received acknowledgments for $w_f - (1 - \beta)w_f$ worth of data, where w_f is the value of the window *just before* the cut. As in [16] we will use $\beta = 1/2$.

According to [2] after receiving the third duplicate acknowledgment the sender will retransmit the lost packet and set its window to $(1 - \beta)w_f + 3$. Then it will keep incrementing the window by one for each acknowledgment received. As we are assuming that a single packet was lost the sender will keep receiving the same sequence number in each acknowledgment for one round trip time so that the number of unacknowledged packets does not change preventing the sliding window mechanism used by TCP from moving the window. Therefore the sender can send a new packet only when the window reaches again w_f . As the window is increased by 1 for each acknowledgment received this will take a time equal to $(\beta w_f - 3)/\mu$, assuming acknowledgments arrive at rate μ .

At this point the sender will send data at the same rate at which acknowledgments are arriving. We assume that this rate is μ , which is true as long as the buffer is non-empty, so that the total length of this round trip time is w_f/μ and the total amount of traffic sent (during this round trip time) is $(1 - \beta)w_f + 3$. As it is sending at rate μ for a period of time equal to $(w_f - \beta w_f + 3)/\mu$. After a time equal to w_f/μ after the loss was detected (i.e. at the end of the round trip time) the acknowledgment for the lost packet, which was re-transmitted at the beginning of the round trip time, will finally arrive and the sender will set its window equal to $(1 - \beta)w_f$.

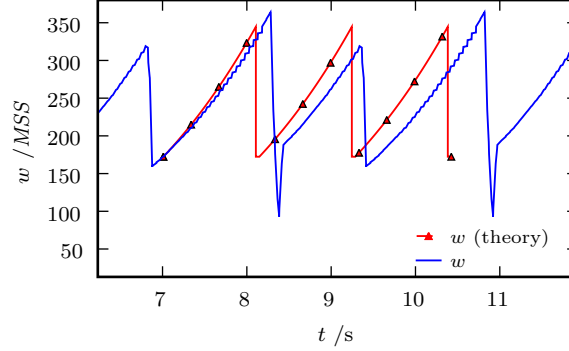


FIG. 6 – The window in case 1

Combining all this we have that the length of this round trip is w_f/μ , the length of sender's pauses, immediately after detecting the loss, is $(\beta w_f - 3)/\mu$ and the total amount of data sent during this round trip is $(1 - \beta)w_f + 3$. For each phase we are going to consider this last round trip, starting from when the loss is detected, as an integral part of the current phase.

4.1 Case 1 : Phase 1 Only

As discussed in section 2.1 if $b < \alpha(\mu\tilde{\tau})^k$ the buffer will overflow when $w = (b/\alpha)^{\frac{1}{k}}$. and t_1 is given by (6). According to equation (4) of [16] the total window will then be reduced to $(1 - \beta)(b/\alpha)^{\frac{1}{k}}$. According to the fluid model, in this phase, the round trip time is constant, and equal to the propagation delay $\tilde{\tau}$. In a packet based system, instead, the round trip time is constantly changing. We are going to assume that the behavior during the round trip time following the reduction of the window is the same as in the other phases, that is that fast recovery/retransmit will be used. The assumption that acknowledgments arrive at rate μ can be justified by observing that, in a packet based system, the source would send bursts of packets that are served at rate μ at the bottleneck link so that acknowledgments do arrive at rate μ , at least for all the packets that were sent during the same burst. At the same time, given that these bursts are not big enough to cause a non-zero backlog throughout the round trip, this assumption does not hold during part of each round trip.

The total duration of this phase and the total amount of traffic sent during this phase are :

$$T_1 = t_1 + \frac{1}{\mu} \left(\frac{b}{\alpha} \right)^{\frac{1}{k}}$$

$$R_1 = R_{in}(t_1) + (1 - \beta) \left(\frac{b}{\alpha} \right)^{\frac{1}{k}}$$

where t_1 and $R_{in}(t_1)$ are given by (6) and (4), respectively, and where we have used the fact that $w_f = (b/\alpha)^{\frac{1}{k}}$.

Figure 6 compares the model with the corresponding simulation ($\mu = 100$ Mb/s, $\tilde{\tau} = 50$ ms, $b = 10$ pkts, $MSS = 1500$ B). The mismatch between the two curves

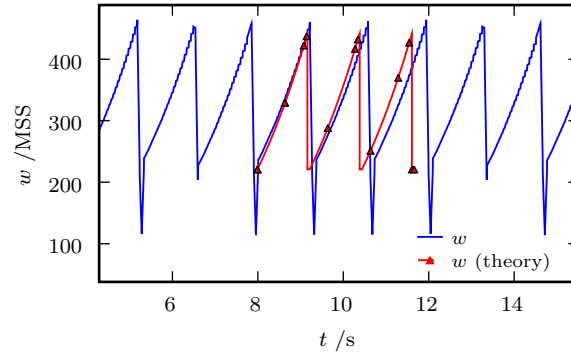


FIG. 7 – The window in case 2

is not too surprising, given that, in this phase, the fluid model is not the best approximation for a packet based system. As we have previously pointed out, the fluid model predicts a constant round trip time and an empty buffer and we have to consider packet bursts in order to get an estimate of the window size when a packet is dropped. Furthermore, we have assumed that a single packet is dropped at each congestion event so that the sender can use fast retransmit/recovery. Given that, in this case, packets are dropped when the window is rapidly increasing, the single packet drop assumption does not necessarily hold. Indeed, in the simulation shown in Figure 6, it is the case that either two packets are dropped during the same round trip time or two packets are dropped in two consecutive round trip times, causing the window to be cut twice (from 364 to 92). Even if the Linux implementation seems to be capable of quickly recovering, the window dynamics after a congestion event are not accurately described by the fast retransmit/recovery mechanism that we have described above. We believe that for this reason the model underestimates the length of each congestion epoch while average value of w is roughly correct.

4.2 Case 2 : Phases 1 and 2

If $\alpha(\mu\tilde{\tau})^k < b < \theta_{\max} - \mu\tilde{\tau}$ phase 2 does take place and ends with a buffer overflow when $w = w_f = b + \mu\tilde{\tau}$. If $b < \frac{\beta\mu\tilde{\tau}}{1-\beta}$ the buffer will be empty at the end of the pause time following the triple duplicate acknowledgment so that the connection will be in phase 1 at the beginning of the following cycle.

Figure 7 compares simulation and theoretical model for this case ($\mu = 100$ Mb/s, $\tilde{\tau} = 50$ ms, $b = 25$ MSS, MSS=1500 B, and $\gamma = 30$ MSS). As in the previous case the single packet drop assumption does not hold. At each congestion event in the simulation, between 6 and 12 packets are dropped during the same round trip time or during two consecutive round trip times. This explains why the window is reduced twice most of the times in Figure 7 (more precisely, for five congestion events, four times the window is reduced twice and once is reduced once).

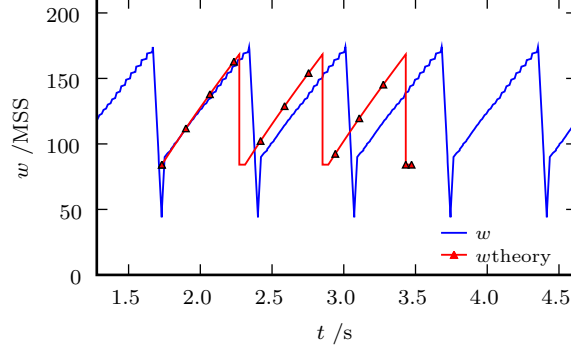


FIG. 8 – The window in case 3

4.3 Case 3 : Phase 2 Only

As in the previous case if $\alpha(\mu\tilde{\tau})^k < b < \theta_{\max} - \mu\tilde{\tau}$ phase 2 does take place and ends with a buffer overflow when $w = w_f = b + \mu\tilde{\tau}$. But, if $b > \frac{\beta\mu\tilde{\tau}}{1-\beta}$ the buffer will never be empty.

Note that for this case to take place γ would need to be incremented to an unrealistic value. Given that we need $\frac{\beta\mu\tilde{\tau}}{1-\beta} < b < \theta_{\max} - \mu\tilde{\tau}$ to be in this case, and increasing γ is the only way to increase θ_{\max} while still having $\theta_{\max} - \mu\tilde{\tau} > \frac{\beta\mu\tilde{\tau}}{1-\beta}$. In practice we believe that this case will be encountered rarely and it is presented only for completeness. In the simulations, as in the previous two cases, multiple packets are dropped at each congestion event : in this case six packets are dropped each time and the window is always reduced twice.

Figure 8 shows the window according to the model and the simulation with $\mu = 50$ Mb/s, $\tilde{\tau} = 20$ ms, $b = 85$ MSS, MSS=1500 B, and $\gamma = 90$ MSS.

4.4 Case 4 : Phases 1, 2, 3 and 4

If $\theta_{\max} - \mu\tilde{\tau} < b < \frac{\beta}{1-\beta}\mu\tilde{\tau}$ the connection goes through all the four phases and phase 4 ends with a buffer overflow. Therefore $w_f = b + \mu\tilde{\tau}$ and $(1-\beta)w_f < \mu\tilde{\tau}$ implying that the buffer will empty after the overflow.

Figure 9 compares the evolution of w from the simulation with the fluid model. In this case $\mu = 100$ Mb/s, $\tilde{\tau} = 50$ ms, $b = 80$ MSS, MSS=1500 B, and $\gamma = 30$ MSS. The match is quite accurate, even though it is possible to notice that after three cycles there is a slight difference between the two curves. This is due to the fact that the time between two packet drops predicted by the fluid model it is not exactly the same as the one in the simulation.

In the simulation w oscillates during the “constant window” phase, as discussed in section 3. In Figure 9 the value used for the fluid model during this phase is calculated as the average between the maximum and minimum values given by the solution of the corresponding fixed point equation, presented in section 3.2.

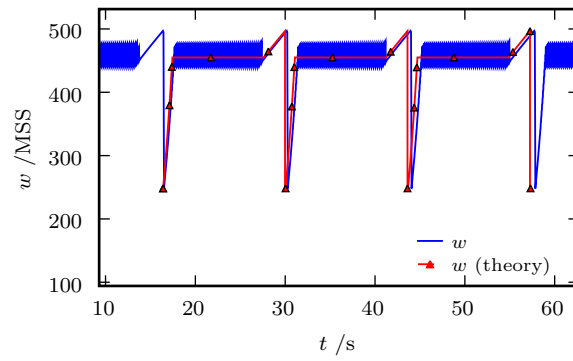


FIG. 9 – The window for case 4

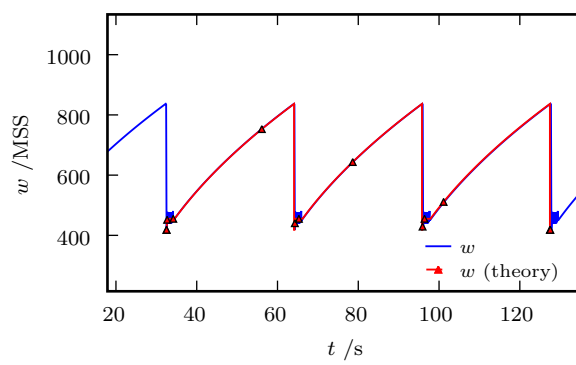


FIG. 10 – The window for case 5

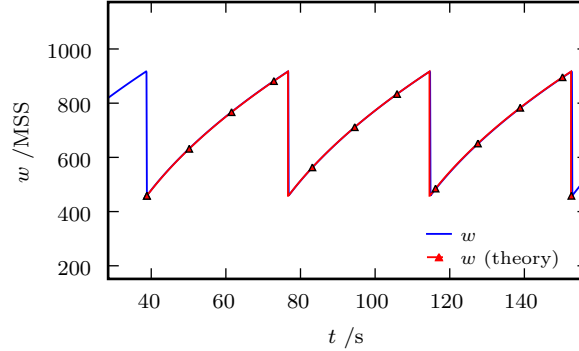


FIG. 11 – The window for case 6

4.5 Case 5 : Phases 2, 3 and 4

If $\frac{\beta}{1-\beta}\mu\tilde{\tau} < b < \frac{\beta\mu\tilde{\tau}+\gamma}{1-\beta}$ the buffer is always non empty and phase 1 doesn't take place. Figure 10 compares the fluid model with the simulation when $\mu = 100$ Mbit/s, $\tilde{\tau} = 50$ ms, $b = 420$ MSS, $MSS=1500$ B and $\gamma = 30$ MSS.

In order to be in this case the backlog needs to be always non-zero so that the window needs to reach bigger values than in case 4 and the constant window phase will always be small compared to the total time between packet drops.

4.6 Case 6 : Phase 4 Only

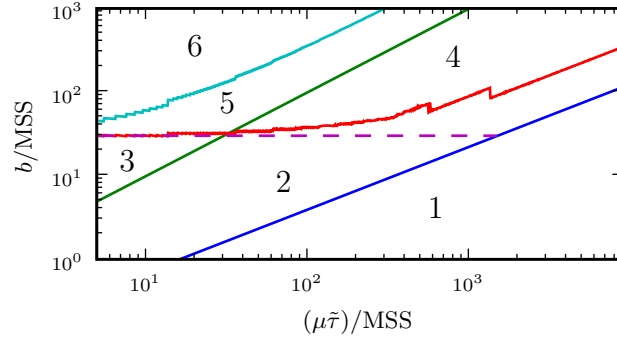
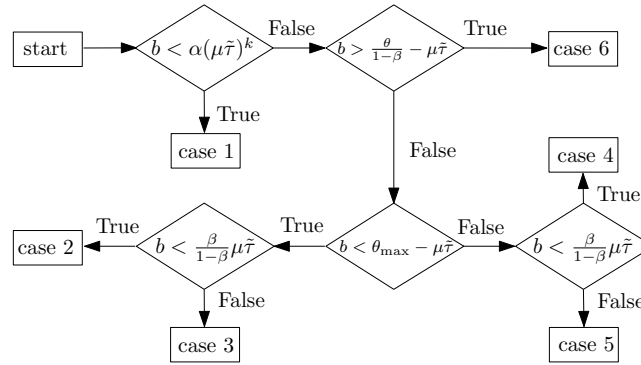
If $b > \frac{\beta\mu\tilde{\tau}+\gamma}{1-\beta}$ the backlog is always greater than γ so that phase 4 is the only one. Figure 11 compares the fluid model with the simulation when $\mu = 100$ Mb/s, $\tilde{\tau} = 50$ ms, $b = 500$ MSS, $MSS=1500$ B and $\gamma = 30$ MSS. In this case the behavior of CTCP is the same as Reno and there is a better, even though not perfect, match between the fluid model and the simulations.

4.7 Case Selection

In the case of an isolated connection only one of them will take place, depending on the system parameters (b and $\mu\tilde{\tau}$).

In the previous sections we have described all the possible six cases for the evolution of the sending window. Figure 12 shows the different cases on the b - $\mu\tilde{\tau}$ plane. Note that both axis use a logarithmic scale. The discontinuities and the “fuzziness” in the lines between cases 2 and 4 and between cases 5 and 6 are due to the integer approximations used to compute θ_{\max} and to different oscillation types (3 :1, 5 :2, etc.).

The dotted line in the middle of the figure represents the boundary between cases 2 and 4 if the oscillations during phase 3 are ignored, that is if we take $\theta_{\max} = \mu\tilde{\tau} + \gamma$. This is another example of the consequence of these oscillations. The behavior in cases 2 and 4 is significantly different : in case 2 packets are dropped before the “constant window” phase causing large oscillations in the window and lower throughput while in case 4 the connection goes through all the phases with a throughput close to the link capacity. Ignoring the oscillations

FIG. 12 – Different cases on the b - $\mu\tilde{\tau}$ planeFIG. 13 – Flow chart for selecting the case based on $\mu\tilde{\tau}$ and b

would lead to the wrong conclusion that for large values of the bandwidth delay product case 2 is no longer possible and that the only possible issue would be if the buffer is much smaller than the bandwidth delay product so that case 1 would take place. Due to the oscillations, even much larger values of the buffer might not be enough to guarantee a high throughput.

Figure 12 was plotted using $\gamma = 30$ and the standard values for all the other parameters. For arbitrary values of the parameters it is possible to use the flow chart in Figure 13 to find the corresponding case. In order to use this flow chart one needs to know the values of the all the parameters and of θ_{\max} , which can be computed as discussed in section 3 for the Linux implementation we have used. As we have previously mentioned for different implementations θ_{\max} will be different but it is the only implementation-specific parameter used.

5 Steady State Performances

5.1 Throughput

Using the model presented in the previous sections it is possible to compute the average throughput of an isolated CTCP connection. More precisely solving the fixed point equations presented in section 3.2 and then using the flow chart in Figure 3 it is possible to compute θ_{\max} . Using this value and the flow chart

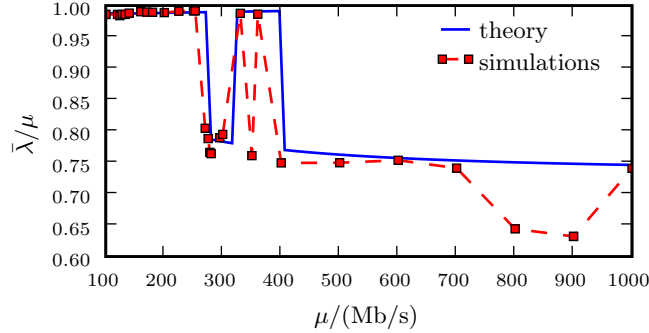


FIG. 14 – Normalized throughput

in Figure 13, the corresponding case can be found so that the evolution of the window is known. We have implemented this procedure in order to compute the average throughput ($\bar{\lambda}$) for different values of the bottleneck capacity. Figure 14 compares the normalized throughput ($\bar{\lambda}/\mu$) computed using the theoretical model and the results of some simulations with $b = 100$ MSS, $\tilde{\tau} = 50$ ms, MSS=1500 B and $\gamma = 30$ MSS (each square corresponds to a simulation). To compute the throughput in the simulations we ignore the slow start phase. In a realistic setting, where some of the connections send small amounts of data, ignoring the slow start might not be appropriate. But here we are interested in comparing the simulations with the theoretical model we presented, which does not model slow start.

The first change in throughput at $\mu = 273$ Mb/s is caused by a transition from case 4 to case 2, due to the increase in the size of the oscillations during phase 3. At $\mu = 320$ Mb/s the oscillation type changes from 4 :2 to 3 :2 causing smaller oscillations and a transition back to case 4 from case 2 (this corresponds to the reduction at $\mu\tilde{\tau} = 1340$ MSS in Figure 5). In this region θ_{\max} is close to $b + \mu\tilde{\tau}$ and it is possible for a packet to be dropped before the oscillations reach steady state (recall that θ_{\max} is the maximum value of the window during the oscillations in steady state). This is confirmed by the simulations : for $\mu = 330$ Mb/s and $\mu = 360$ Mb/s phases 3 and 4 do take place so that we are in case 4 while for $\mu = 350$ Mb/s packets are dropped during phase 2 and we are in case 2. At $\mu = 400$ Mb/s the oscillations during phase 3 are sufficiently large to cause a buffer overflow causing the transition from case 4 to case 2.

Until $\mu = 273$ Mb/s, that is during case 4, there is a very good match between the model and the simulations. For case 2, instead, the match is not as good. As already discussed in section 4.2, in this case multiple packets are dropped at each congestion event, violating one of the assumptions of the model. We have also noticed that, in several cases, the window will oscillate a few times before packets are dropped, extending the length of each congestion epoch. At the same time we do not have an explanation for the fact that there is a bigger difference between the model and the simulation for $\mu = 800$ Mb/s and $\mu = 900$ Mb/s.

As discussed in section 3.4, even for $\gamma = 5$ the amplitude of the oscillations is an increasing function of μ . For example, if $\gamma = 5$ the transition from case 4 to case 2 takes place when $\mu \simeq 410$ Mb/s, indicating that the TUBE algorithm can shift this problem to higher bit rates but it does not solve it completely.

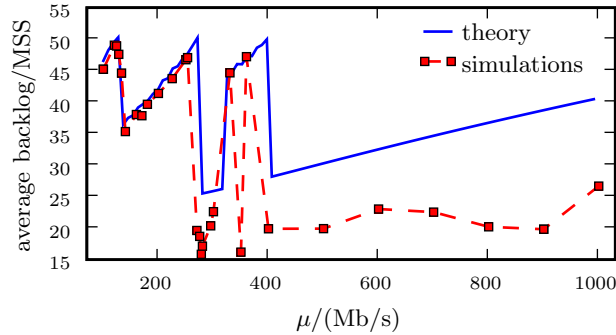


FIG. 15 – Average backlog size

5.2 Average Backlog

Using the equations describing the evolution of the sending window, presented in section 4, and with some simple but tedious algebra, it is possible to compute the average backlog at the bottleneck link. Using the same procedure described for the throughput it is possible to determine which case the connection will follow and then use the appropriate formula for the average backlog.

Figure 15 shows the average backlog for different values of μ , according to the model and to the simulations (with the same parameters used for the throughput analysis). As in the previous section each square corresponds to a simulation, ignoring the slow start phase to compute the average backlog. The rapid decrease at $\mu = 131$ Mb/s corresponds to the oscillations changing from 3 :1 to 5 :2. The 5 :2 oscillations are present between 131 Mb/s and 138 Mb/s (which correspond to the peak between $\mu\bar{\tau} = 546$ MSS and $\mu\bar{\tau} = 574$ MSS in Figure 5). When the oscillations change from 3 :1 to 5 :2 the double reduction causes the backlog to decrease further at each oscillation and the average value is lower as well. Between $\mu = 138$ Mb/s and $\mu = 273$ Mb/s the 4 :2 oscillations have increasing amplitude, causing the average backlog to increase as well. It is interesting to note how these different types of oscillations (3 :1, 5 :2 and 4 :2) have a negligible effect on the throughput while they have a significant impact on the backlog. This is because during the oscillations, which are present only during phase 3, the backlog is nonzero most of the time, leading to high throughput, but the window oscillations cause similar oscillations in the backlog size affecting its average value.

For values of μ above 273 Mb/s Figures 15 and 14 have the same discontinuities, caused by the same underlying changes in the evolution of the sending window, as described in the previous section.

It is interesting to note how the oscillations during phase 3 do affect the performance of CTCP, both in terms of throughput and of average backlog size. Modifying the protocol in order to significantly reduce their amplitude would be a worthwhile endeavor. One such way could be to use smaller increments and decrements after full utilization is detected, that is after Δ_i is positive but such modifications are outside the scope of this work.

6 Conclusions

We have presented a complete model for an isolated CTCP connection and we have validate it with ns-2 simulations using a Linux implementation of CTCP. To the best of our knowledge this is the first complete model of CTCP which has led us to identify its significantly different behaviors depending on the system parameters. While the basic idea of combining a delay and a loss based approach is fairly simple but the resulting protocol is far from it and its behavior is much more complicated to analyze than that of TCP Reno, even in the simple case of an isolated connection.

While the model does not always match the simulations with great precision it does accurately predict the overall evolution of the window and it has allowed us to identify the different regimes of CTCP. As the performances, in terms of throughput and average backlog size at the bottleneck link, do depend on these regimes the model can be used to analyze them, helping to find for which values of the system parameters the performances will change.

We have also highlighted how the oscillations during the “constant window” phase do have a non negligible impact on the performance and how it is not possible to reduce their size by simply reducing the value of the parameter γ . We believe that modifying the protocol in order to significantly reducing the size of the oscillations could have a significant impact. Especially given that, even though we have not addressed the issue, they can adversely effect the other traffic sharing the same links.

Références

- [1] R. Agrawal, R. Cruz, C. Okino, and R. Rajan. Performance bounds for flow control protocols. *Networking, IEEE/ACM Trans. on*, 7(3) :310–323, June 1999.
- [2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.
- [3] L. Andrew. Compound TCP Linux module. available at <http://netlab.caltech.edu/lachlan/ctcp/>, Apr. 2008.
- [4] L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, and I. Rhee. Experimental evaluation of delay/loss-based TCP congestion control algorithm. In *Proc. 6th Int. Workshop on Protocols for FAST Long-Distance Networks*, Mar. 2008.
- [5] A. Baiocchi, A. Castellani, and F. Vacirca. YeAH-TCP : Yet Another Highspeed TCP. In *Proc. 5th Int. Workshop on Protocols for FAST Long-Distance Networks*, Mar. 2007.
- [6] A. Blanc, D. Collange, and K. Avrachenkov. Modelling an isolated Compound TCP connection. Tech. Report 6778, INRIA, Dec. 2008.
- [7] A. Blanc, D. Collange, and K. Avrachenkov. Oscillations of the sending window in Compound TCP. In *Proc. 2nd NetCoop Workshop*, 2008.
- [8] J. Davies. Performance enhancements in the next generation TCP/IP stack. The Cable Guy <http://www.microsoft.com/technet/community/columns/cableguy/cg1105.aspx>, 2007.

- [9] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), Dec. 2003.
- [10] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), Mar. 2008.
- [11] K. Kumazoe, M. Tsuru, and Y. Oie. Performance of high-speed transport protocols coexisting on a long distance 10-Gbps testbed network. In *Proc. 1st Int. Conf. on Networks for Grid Applications*, Oct. 2007.
- [12] Y. Li. Evaluation of TCP congestion control algorithms on the Windows Vista platform. Technical Report SLAC-TN-06-005, Stanford Linear Accelerator Center, June 2005.
- [13] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. Compound TCP : A new TCP congestion control for high-speed and long distance networks. Internet draft, Internet Engineering Task Force, Oct. 2007. (Work in progress).
- [14] K. Tan, J. Song, M. Sridharan, and C. Ho. CTCP-TUBE : Improving TCP-friendliness over low-buffered network links. In *Proc. 6th Int. Workshop on Protocols for FAST Long-Distance Networks*, Mar. 2008.
- [15] K. Tan, J. Song, Q. Zhang, and M. Sridharan. Compound TCP : A scalable and TCP-friendly congestion control for high-speed networks. In *Proc. 4th Int. Workshop on Protocols for FAST Long-Distance Networks*, Mar. 2006.
- [16] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound tcp approach for high-speed and long distance networks. In *INFOCOM 2006. Proc. 25th IEEE Int. Conf. on Computer Communications.*, 2006.



Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
<http://www.inria.fr>
ISSN 0249-6399