Modelling an Isolated Compound TCP Connection

Alberto Blanc and Denis Collange
Orange Labs
905 rue Albert Einstein
Sophia Antipolis, France
{Email: alberto.blanc,denis.collange}@orange-ftgroup.com

Konstantin Avrachenkov
I.N.R.I.A.
2004 route des lucioles
Sophia Antipolis, France
Email: k.avrachenkov@sophia.inria.fr

***Abstract*—Compound TCP (CTCP) was designed by Tan at al. to improve the efficiency of TCP on high speed networks without unfairly penalizing other connections. In this work we analyze an isolated CTCP connection, identifying and classifying 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. These oscillations can also induce additional jitter in the cross traffic. 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.**

I. 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 [7] 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 [8], 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 [13] and [14] in 2006. It is currently submitted as a draft to the IETF Network Working group with minor differences [11]. CTCP is enabled by default in computers running Windows Server 2008 and disabled by default in computers running Windows Vista [6]. It is also possible to add support for CTCP to Windows XP. An implementation of CTCP, based on [11], [14], is also available for Linux [1].

As the proposal of CTCP is still recent, there are only a few published evaluations of it. The only analytical model of CTCP in [14] 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 [5] 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 [10], all the other experimental evaluations [2], [3], [9] use Linux implementations of CTCP whose behavior differs from the Windows implementation, according to [2].

The main objective of the authors of CTCP [14] 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:

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

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:

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

Where γ was initially a fixed threshold, with proposed value equal to 30 [14]. It is now dynamically adjusted between 5 and 30, according to the window size on loss events [12], using the TUBE algorithm. For the sake of simplicity, and

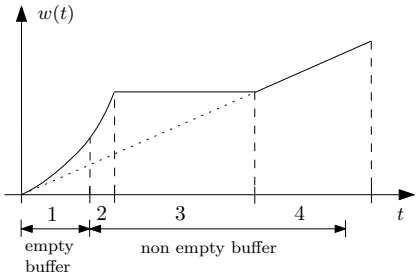

Figure 1. The four different phases

due to the fact that is 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.

As depicted in Figure 1 the evolution of the CTCP has four different phases: during phases 1 and 2 the window is quickly increased using the first row of (2) (in this case $\Delta_i < \gamma$). The only difference between phases 1 and 2 is that in phase 2 the buffer at the bottleneck link is empty while in phase 2 non-empty. During phase 3 the window is kept constant because the network is already fully utilized and, finally , in phase 4 the window is increased by one packet each round trip (as in TCP Reno).

In this work 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. Under these assumptions the system is deterministic and it is possible to model the evolution of the window during each phase (see [4] for a complete description of the model). While this is a very simple model it already allows us to identify the different behaviors of CTCP and it can also be used to validate different implementations.

The remainder of the paper is organized as follows: in section II we analyze the oscillations during phase 3, in section III we explore how the four phases of CTCP can be combined and in section IV we analyze the steady state throughput and backlog size.

II. Oscillations During Phase 3

As noted in [5], during phase 3, the algorithm described in [14] and [12] causes the window to oscillate around a constant value. 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. 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.

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 [5] to precisely characterize these oscillations. In the remainder of this section, after a brief presentation of the model used in [5], we will extend that work 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 [1]. 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 [12], [14]. Furthermore the rest of the model depends only on $\theta_{\max}$ 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.

A. Linux Implementation

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

$$\Delta_i = w_{i-1}\left(1 - \tilde{\tau}/\tau_i\right) \tag{3}$$

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:

$$\tau_i = \begin{cases} \min\left[\frac{w_{i-2}+1}{\mu}, \tilde{\tau}\right] & , \text{if } w_{i-1} > w_{i-2} \\ \min\left[w_{i-1}\frac{\tilde{\tau}}{\tau_{i-1}}, \tilde{\tau}\right] & , \text{if } w_{i-1} < w_{i-2} \end{cases} \tag{4}$$

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 [5] for more details).

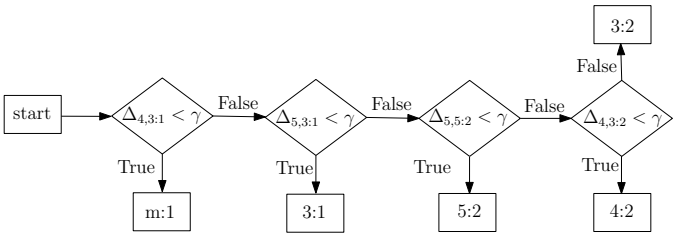

Figure 2. How to find the type of oscillations

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	$\Delta_3 < \gamma$	$\Delta_4 \geq \gamma$	$\Delta_5 \geq \gamma$	$[1342, \infty)$

Table I
LIMITS FOR $\Delta_{i,m:n}$

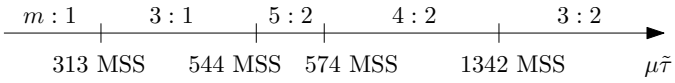

Figure 3. Type of oscillations for $\gamma = 30$

B. Fixed Points

Given an initial value for the window it is possible, using equations (3), (4) 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.

As discussed in [5], 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 [4].

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 $\frac{\mu\tilde{\tau}}{2} > \alpha^{\frac{1}{1-k}}$.

C. 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 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 I 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 I, 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 $546 \leq \mu\tilde{\tau} \leq 573$ 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 I, the type of oscillations observed in the simulations is not the same as the one given by the model

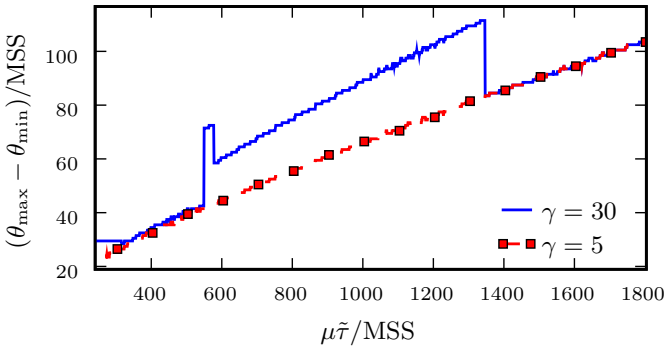

Figure 4. Amplitude of the oscillations

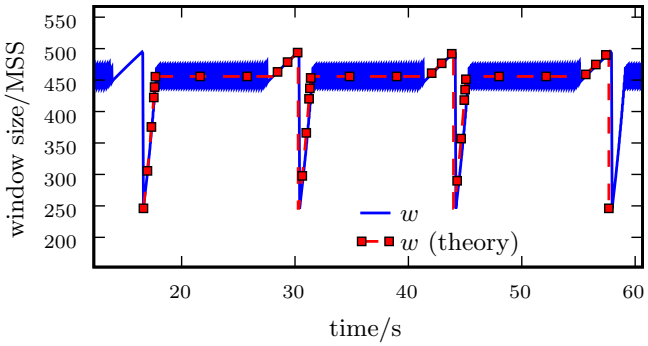

Figure 5. The window for case 4

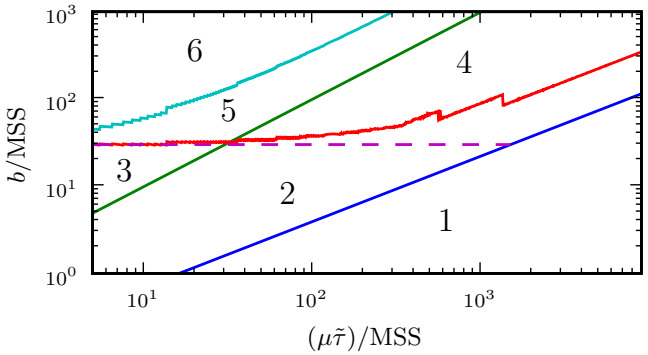

Figure 6. Different cases on the b-$\mu\tilde{\tau}$ plane

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 3 shows the type of oscillations obtained using this method for $\gamma = 30$ while Figure 2 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 [5] 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 [5] for more details.)

Using the algorithm presented in Figure 2 it is possible to compute the size of the oscillations as a function of the bandwidth delay product. Figure 4 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 [12]. For $\gamma = 30$ the type of oscillations changes with the bandwidth delay product (see Figure 3) explaining the discontinuities in the curve. While for $\gamma = 5$ the only type of oscillations is always 3:2.

This indicates that even using smaller values for γ (and/or the TUBE algorithm [12]) 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.

III. Combining Multiple Phases

As previously mentioned, the evolution of the CTCP window is characterized by four different phases. Depending on the bandwidth-delay product and on the buffer size it is possible to identify six different ways of combining these phases:

1) phase 1 only
2) phases 1 and 2
3) phases 2 only
4) phases 1,2,3 and 4
5) phases 2,3, and 4
6) phase 4 only.

In each one of these cases the evolution of the window will be different. As an example, Figure 5 shows the evolution of the window for case 4. The solid line corresponds to a ns-2 simulation using ns-2.33 and the CTCP implementation [1]. In this case the agreement between the model and the simulation is excellent, as for cases 5 and 6. In the other cases (1,2 and 3) packets are dropped when the window is increasing quickly, causing multiple packets to be dropped. In this case sender will sometimes reduce its window more than once. Our model does not take this into account so that the evolution of the window is predicted with a smaller accuracy but other quantities (like the average throughput) have a smaller error (see [4] for a complete discussion).

Figure 6 shows the different cases on the b-$\mu\tilde{\tau}$ plane. Note that both axis use a logarithmic scale. The discontinuities and

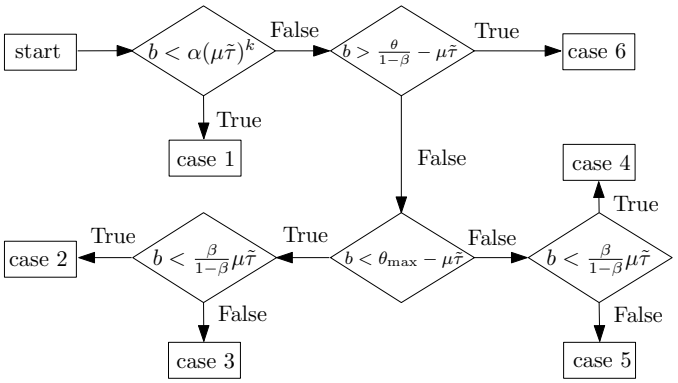

Figure 7. Flow chart for selecting the case based on $\mu\tilde{\tau}$ and b

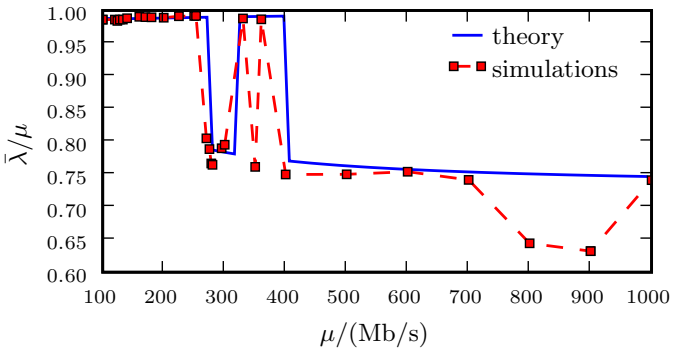

Figure 8. Normalized throughput

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 $\theta_{\max}$ 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 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 6 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 7 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 $\theta_{\max}$, which can be computed as discussed in section II for the Linux implementation we have used. As we have previously mentioned for different implementations $\theta_{\max}$ will be different but it is the only implementation-specific parameter used.

IV. Steady State Performances

A. 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 II-B and then using the flow chart in Figure 2 it is possible to compute $\theta_{\max}$. Using this value and the flow chart in Figure 7, 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 8 compares the normalized throughput ($\bar{\lambda}/\mu$) computed using the theoretical model and the results of some simulations with $b = 100\,\mathrm{MSS}$, $\tilde{\tau} = 50\,\mathrm{ms}$, MSS=1500 B and $\gamma = 30\,\mathrm{MSS}$ (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 $\mu = 273\,\mathrm{Mb/s}$ 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 $\mu = 320\,\mathrm{Mb/s}$ 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 $\mu\tilde{\tau} = 1340\,\mathrm{MSS}$ in Figure 4). In this region $\theta_{\max}$ 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 $\theta_{\max}$ is the maximum value of the window during the oscillations in steady state). This is confirmed by the simulations: for $\mu = 330\,\mathrm{Mb/s}$ and $\mu = 360\,\mathrm{Mb/s}$ phases 3 and 4 do take place so that we are in case 4 while for $\mu = 350\,\mathrm{Mb/s}$ packets are dropped during phase 2 and we are in case 2. At $\mu = 400\,\mathrm{Mb/s}$ the oscillations during phase 3 are sufficiently large to cause a buffer overflow causing the transition from case 4 to case 2.

Until $\mu = 273\,\mathrm{Mb/s}$, 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 mentioned in section III, 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 $\mu = 800\,\mathrm{Mb/s}$ and $\mu = 900\,\mathrm{Mb/s}$.

As discussed in section II-C, 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 $\mu \simeq 410\,\mathrm{Mb/s}$, indicating that the TUBE algorithm can shift this problem to higher bit rates but it does not solve it

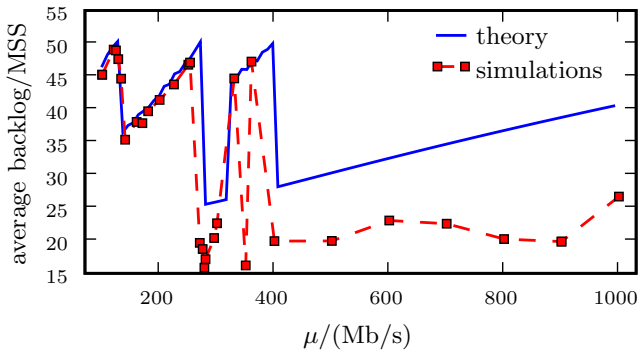

Figure 9. Average backlog size

completely.

B. Average Backlog

Using the equations describing the evolution of the sending window, presented in [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 9 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 $\mu = 131\,\mathrm{Mb/s}$ corresponds to the oscillations changing from 3:1 to 5:2. The 5:2 oscillations are present between $131\,\mathrm{Mb/s}$ and $138\,\mathrm{Mb/s}$ (which correspond to the peak between $\mu\tilde{\tau} = 546\,\mathrm{MSS}$ and $\mu\tilde{\tau} = 574\,\mathrm{MSS}$ in Figure 4). 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 $\mu = 138\,\mathrm{Mb/s}$ and $\mu = 273\,\mathrm{Mb/s}$ 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.

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.

V. Conclusions

In this paper we have presented a model for an isolated CTCP connection. 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, 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. Using this model we have analyzed the average throughput and backlog size, showing how these quantities do depend on the different regimes of CTCP.

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.

References

[1] L. Andrew. Compound TCP Linux module. available at http://netlab.caltech.edu/lachlan/ctcp/, Apr. 2008.

[2] 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.

[3] 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.

[4] A. Blanc, D. Collange, and K. Avrachenkov. Modelling an isolated Compound TCP connection. Tech. Report 6778, INRIA, Dec. 2008.

[5] A. Blanc, D. Collange, and K. Avrachenkov. Oscillations of the sending window in Compound TCP. In *Proc. 2nd NetCoop Workshop*, 2008.

[6] J. Davies. Performance enhancements in the next generation TCP/IP stack. The Cable Guy http://www.microsoft.com/technet/community/columns/cableguy/cg1105.mspx, 2007.

[7] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), Dec. 2003.

[8] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), Mar. 2008.

[9] 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.

[10] 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.

[11] 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).

[12] 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.

[13] 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.

[14] 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.