# Study and enhancement of DCCP over DiffServ Assured Forwarding class

Emmanuel Lochin, Guillaume Jourjon, Laurent Dairaine

# Study and enhancement of DCCP over DiffServ Assured Forwarding class

Emmanuel Lochin
National ICT Australia Ltd,
Locked Bag 9013,
Alexandria, NSW 1435
Australia

Guillaume Jourjon
National ICT Australia Ltd,
University of
New South Wales, Sydney
Australia

Laurent Dairaine
ENSICA - LAAS/CNRS,
1, place Emile Blouin
31056 Toulouse Cedex 5
France

{emmanuel.lochin, guillaume.jourjon}@nicta.com.au, laurent.dairaine@ensica.fr

## Abstract

*The Datagram Congestion Control Protocol (DCCP) has been proposed as a transport protocol which supports real-time traffic. In this paper, we focus on the use of DCCP/CCID3 (Congestion Control ID 3) over a DiffServ/AF class. This class of service is used to build services that provide only a minimum throughput guarantee without any delay or jitter restrictions. This minimum throughput guarantee is called the target rate. In this context, the throughput obtained by DCCP/CCID3 mainly depends on RTT and loss probability. As a result, the application does not always get the negotiated target rate. To cope with this problem, we propose to evaluate a simple adaptation of the CCID3 congestion control mechanism, allowing the application to reach its target rate whatever the RTT value of the application's flow is. As this adaptation can be seen as an extension to the DCCP with CCID3 congestion control, we call it gDCCP for guaranteed DCCP. Results from simulations are presented to illustrate the improvements of the proposed modification in various situations. Finally, we investigate the deployment of this proposal in terms of security.*

## 1 Introduction

This paper studies the behaviour of DCCP protocol [12] in a DiffServ/AF quality of service class. In particular, we focus on the DCCP/CCID3 congestion control protocol based on the TFRC mechanism. The aim is to explore the feasibility of using DCCP protocol over a DiffServ network and look at the service received from the DCCP user point of view.

Still nowadays, the classical protocol used by real-time applications is UDP. These time-constrained applications (such as audio and video streaming, voice over IP (VoIP), video on demand (VoD)) use UDP by default because TCP is not adapted to their time constraints. The main problem with UDP remains the lack of congestion control compelling the applications to implement congestion control mechanisms. Nevertheless, many applications don't integrate any of these mandatory controls, leading to a bad popularity of the UDP protocol.

In Internet, UDP traffic is often filtered for security reason. In the context of DiffServ network, it raises difficulties when mixed with TCP. Indeed, when responsive TCP flows and non-responsive UDP flows share the same class of service, there is an unfair bandwidth distribution and TCP flows throughput are affected [25]. Theoretically, the fair allocation of excess bandwidth can be achieved by giving different treatments to out-profile traffic of both kinds of flows. The general approach is to define two specific queues in the core network in order to separate the non-responsive (UDP) from the responsive (TCP) traffics [18].

In this context, the use of DCCP is particularly interesting since it offers a way to avoid this traffic segregation, allowing to mix both kinds of traffics. Moreover, since the DCCP/CCID3 protocol is based on the TCP equation defined in [5], we can expect to use the same DiffServ conditioners designed for TCP traffic as those defined in [2, 4, 8, 13]. Nevertheless, as TCP does, measurements presented in this paper show that DCCP/CCID3 congestion control doesn't reach the target rate previously negotiated by the application. To cope with this problem, we evaluate in this paper a lightweight add-on to DCCP/CCID3 congestion control in order to improve its behaviour in the context of DiffServ/AF. As a results, extensive measurements show its ability to provide the user a bandwidth in agreement with the network QoS guarantee.

This paper is organized as follows. Section 2 presents related work about DCCP. Section 3 points out the per-

formance problem associated with the use of DCCP into the DiffServ/AF class and introduces the *g*DCCP proposal. Section 4 presents the simulation testbed and comments results about the studied scenarios. We discuss about security problem involved with this proposal in section 5. Finally, section 6 concludes the paper.

## 2 Related Work

### 2.1 DCCP and congestion control

Nowadays, many real-time applications appear on the Internet and give birth to new types of unreliable real-time traffics. TCP is not appropriate to the quality of service requirements of these applications. Then, due to its simplicity and the lack of efficient alternative, UDP is currently largely used for the transmission of real-time traffic. UDP does not implement any traffic control and its use ideally requires the application to implement such mechanisms. Multimedia developers prefer the use of UDP without congestion control due to the complexity of implementing such mechanisms. This lack of congestion control mechanism leads to an unfairness problem in case of mixing UDP with TCP traffic.

The Real-time Transport Protocol (RTP) and its companion protocol RTCP [24] have been proposed as a support for time constrained multimedia applications. RTP is implemented as a library directly integrated into the application. Currently, RTP is employed by a large number of distributed multimedia applications over Internet (such as VideoLan[1]). This protocol integrates information such as timestamps and sequence numbers. These fields are used to implement the appropriate mechanisms to detect and recover losses, reorder data, discard obsolete data and synchronize data flows. Nevertheless, RTP and RTCP do not directly implement congestion control mechanism and user still needs to implement or to use ad-hoc mechanism.

The Rate Adaptation Protocol (RAP) [23] is an end-to-end TCP-friendly protocol using the Additive Increase and Multiplicative Decrease (AIMD) algorithm [28] to enable fair sharing of the bandwidth with TCP traffic. Implemented at user-level, these proposals raise some security problems as users can easily modify them and eventually, involve great disorder in the network bandwidth distribution. In order to provide a congestion control function that cannot be modified by users, DCCP has been proposed. DCCP is a transport protocol for real-time communications that supports a congestion control mechanism at kernel level.

DCCP has two main objectives: first, its implementation is to remain as simple as possible [2]. So unlike TCP, it does not support reliable data delivery. The second objective is to provide a TCP-friendly congestion control mechanism because TCP traffic is still pervasive in the current Internet. DCCP includes multiple congestion control algorithms which can be selected in regards to the user needs. An algorithm is identified through its Congestion Control ID (CCID). Two CCIDs are now being standardized by the Internet Engineering Task Force (IETF). CCID2 [6] is a window based congestion control algorithm like TCP, and CCID3 [7] is a TCP-Friendly Rate Control (TFRC) algorithm. CCID2 is appropriate and particularly useful for senders who would like to take advantage of the available bandwidth in an environment with rapidly changing conditions as bursty real-time traffic such as traffic from compressed encoded video and network games. TFRC[3] is a congestion control mechanism for unicast flows operating in a best-effort Internet environment [9]. CCID3 is suitable for traffic with smooth changes in sending rates, such as telephony or video streaming. This CCID3 is based on the TCP throughput equation (1) and is designed to be reasonably fair when competing with TCP flow.

$$X = \frac{s}{(RTT \cdot \sqrt{\frac{p \cdot 2}{3}} + RTO \cdot \sqrt{\frac{p \cdot 27}{8}} \cdot p \cdot (1 + 32 \cdot p^2))} \quad (1)$$

Where $X$ the sending rate depends on the packet lost rate $p$, the mean packet size $s$ and the Round Trip Time $RTT$. $RTO$ refers to the TCP retransmission timeout value. One of the main TFRC properties is to generate a flow with a much lower variation of throughput over time than TCP. This is the reason why it is particularly suitable for multimedia applications such as video streaming or telephony over the Internet.

In contrast of best effort networks, QoS networks such as DiffServ networks offers services guarantees. In the particular case of the DiffServ/AF class, a minimal bandwidth is provided (in-profile traffic part) with the possibility to reach higher bandwidth (out-profile traffic part) depending on the level of congestion of the network. Multimedia applications are natural candidates for the use of this service class. Unfortunately, as it is the case for classical TCP flows, the use of DCCP/CCID3 over such a network service produces unexpected results in terms of user expectations.

### 2.2 DCCP over DiffServ/AF

To our best knowledge, there are no study of DCCP behaviour over a DiffServ network. Nevertheless, in [11], the

[1] `http://www.videolan.org/`

[2] `http://www3.ietf.org/proceedings/05aug/dccp.html`

[3] identified as Congestion Control ID 3 in DCCP protocol

authors investigate AF-TFRC performances and give a service provisioning mechanism allowing an ISP[4] to build a feasible DiffServ system. As CCID3 should react in similar manner to the TCP AIMD congestion control principle, a good starting point is to look at the results obtained for TCP in the DiffServ/AF class.

The problem of TCP throughput guarantee using DiffServ/AF class is already well-known and not new. There have been a number of studies that focused on assured service for TCP flows. In [25], five factors have been studied (RTT, number of flows, target rate, packet size, non responsive flows) and their impact has been evaluated to provide a predictable service.

As the TCP protocol uses the Additive Increase Multiple Decrease (AIMD) congestion control algorithm which aims to share fairly the available bandwidth, the only mean to obtain a service differentiation with TCP is to use DiffServ traffic conditioners such as the token bucket color marker (TBCM) [10] or time sliding window color marker (TSWCM) [3].

The behaviour of those traffic conditioners has a great impact on the service obtained by TCP flows in terms of bandwidth provided to the application. Several others conditioners have been proposed to improve the efficiency of the token bucket-like conditioners [2][19]. In the context of multimedia flows, even if CCID3 congestion control mechanism efficiently replaces the AIMD congestion control algorithm for multimedia applications, its behaviour remains similar to TCP over the DiffServ/AF service class.

In the present study, the CCID3 congestion control mechanism becomes aware of the target rate negotiated by the application with the DiffServ network. Thanks to this knowledge, the application's flow is sent in conformance with the negotiated QoS while staying TCP-friendly in the out-profile traffic part.

## 3 Problem statement

In the assured service class, the throughput of a flow breaks up into two parts: a fixed part which corresponds to a minimum assured throughput and an elastic part which corresponds to an opportunist flow of packets. Packet belonging to the first part are marked in-profile and packets belonging to the second part are marked out-profile. In the case of network congestion, the in-profile packets are considered inadequate for loss. At the contrary, out-profile packets are conveyed on the principle of "best-effort" (BE) service and are dropped first if a congestion occurs.

In case of excess bandwidth in the network, the application could send more than its target rate and the network should mark the excess traffic out-profile. Then, if the network becomes congested, out-profile packets losses occur and the optimal rate estimated by TFRC could fall down under the target rate requested by the application. TCP would react in the same way by halving its congestion window.

As for TCP in the AF class [25], the TFRC mechanism is not aware that the loss corresponds to out-profile packet and that it should not decrease its actual sending rate less below the target rate negotiated. For TCP, the solution was to introduce a conditioner able to better mark the TCP flows by taking into account the sporadic nature of the TCP traffic [2, 15]. But the proposed conditioners are not all really efficient in certain network conditions such as long RTT and are sometimes complex to use.

In order to solve this problem, we propose to make the mechanism QoS-aware following the proposal presented in [16]. This previous study had shown the benefit of using a TFRC QoS-aware congestion control in a DiffServ Assured Forwarding (AF) class. We extend this study by firstly enhancing the way to compute the sending rate; secondly by tackling concrete protocol problems such as security and congestion management; finally, by evaluating the proposal inside a DCCP implementation.

In contrast to TCP, as TFRC explicitly computes the actual sending rate with (1), it is possible to directly act on this rate in order to avoid the under-usage of the network service. Therefore, the present proposal consists in making the sending rate estimator aware of the target rate. This scheme avoids the indirect processing of traffic conditioners while enhancing efficiently the performances in terms of application throughput and TCP-friendliness.

We suppose the application is aware of the target rate, next to a QoS negotiation. This target rate is then known by the transport layer at socket creation time. The target rate parameter can be set e.g., by the `setsockopt()` function. We will discuss in section 5 of security issue concerning this setting functionality.

After a classical slowstart phase, the $g$DCCP modification consists in computing the maximum between the TFRC rate estimation $X_{calc}$ and the target rate $g$. The rest of the CCID3 mechanism follows entirely the DCCP/CCID3 specification and the slowstart phase remains active and unchanged.

In the context of a DiffServ/AF class, a network can be either over or exactly or under-provisioned. A network over or exactly-provisioned means that the amount of in-profile traffic is below or equal to the resource allocated to the AF class. On the contrary, an under-provisioned network means that this amount is higher. This case could occur if the Bandwidth Broker [20] of a DiffServ network sends or receives false information. In a DiffServ context, if the $g$DCCP source emits below its target rate and if the $g$DCCP flow gets losses, it means that the in-profile traffic is no guaranteed anymore in the network. In order to tackle this problem, two approaches are possible:

[4]Internet Service Provider

- The first one is to pursue to emit at the guarantee $g$. This behaviour is legitimate since the service provider must provide to the client the service which is paid;
- The second one is to react to this congestion. This can be done by adding a second ratio ($\lambda$) to $g$DCCP. This ratio can be applied as following: if the rate returned by the TFRC equation $X_{calc}$ computed by the sender is $\lambda$ times below the target rate $g$, the sender must follow the standard TFRC algorithm. Indeed, when $X_{calc} < g/\lambda$, it means that a bunch of losses has occurred in the in-profile part and that the congestion could be due to a wrong setting.

In the TFRC algorithm, when the loss probability $p$ is not nil, the update of the sending rate is basically computed as follow[5]:

$$X = min(X_{calc}, 2 * X_{recv}) \quad (2)$$

With $X_{calc}$ the computed rate and $X_{recv}$ the estimated received rate. We can see that the sending rate is limited to at most twice $X_{recv}$. If we add the guarantee $g$ and take into account the specific case of misconfiguration, the $g$DCCP mechanism is now computed with (3):

$$X = \begin{cases} min(X_{calc}, 2 * X_{recv}) & \text{if } X_{calc} < g/\lambda \\ min(max(X_{calc}, g), 2 * X_{recv}) & \text{otherwise} \end{cases} \quad (3)$$

At least, in case of wrong setting, the sender should not emit above $2 * X_{recv}$.

We believe that the selfish user problem and the misconfiguration problem should not be solved inside $g$DCCP itself and should remain under the responsibility of the service provider. As a consequence, we do not evaluate the under-provisioned network case in this paper as it doesn't give further information on $g$DCCP performances. Nevertheless, we will discuss about this issue in section 5.

We show in the next section that thanks to this adaptation, the application's flow is sent in conformance to the negotiated QoS while staying TCP-friendly in its out-profile part.

## 4 Evaluation and analysis

$g$DCCP is evaluated over a DiffServ network using simulation. It has been implemented in ns-2.29 simulator and the Nortel DiffServ model [22]. We achieved simulation on the testbed illustrated in the figure 1 with the two following scenarios: when the network is exactly-provisioned (i.e. there is no excess bandwidth for the out-profile traffic) and when the network is over-provisioned (i.e. there is excess bandwidth). When only two flows are emitted over the testbed, they cross the paths $(A, D)$ and $(B, E)$. In case of cross-traffic scenario, a flow is emitted between $(C, F)$.

[5]For the sake of simplicity, we do not represent the maximum with $s/t_{mbi}$ which means that at least, if the rate computed is very low, one packet is emitted every 64 seconds

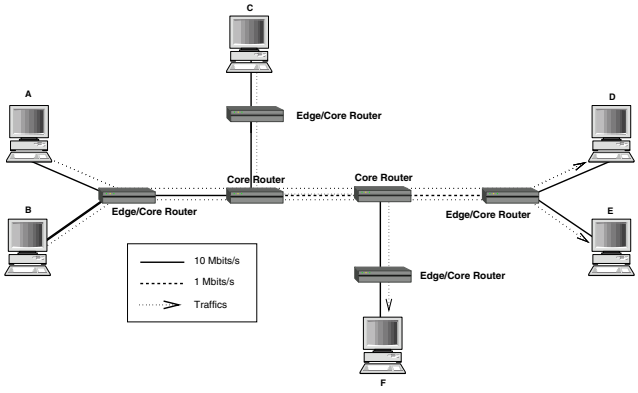



**Figure 1. The simulation topology for DiffServ experiments**

All experiments are achieved using the following conditions:

- packet size is fixed to 1500 bytes;
- TCP version used is NewReno;
- the traffic QoS conditioner is a two-color token bucket marker with a bucket size of $10^4$ bytes;
- the queues size are 50 packets and RIO parameters are: $(min_{out}, max_{out}, p_{out}, min_{in}, max_{in}, p_{in}) = (10, 20, 0.1, 20, 40, 0.02)$;
- the bottleneck is fixed to $1000Kbits/s$.

For each experiment, we evaluate the instantaneous throughput at the server side. The results are presented in the next section.

### 4.1 DCCP/CCID3 over DiffServ AF results

This first part presents the problem encountered with the use of DCCP/CCID3 in the DiffServ/AF class. In order to better understand which problem is tackled in this study, we remind in figure 2 the behaviour of two TCP flows crossing a DiffServ network. Both flows have a target rate of $300Kbits/s$ and a respective RTT of $300ms$ and $50ms$. These two flows are not in the worst condition to reach their desired target rate since the network is not overloaded

(there is only two flows) and we have $40\%$ of excess bandwidth. We can see in figure 2 that both TCP flows reach at least their target rate. On the other hand, this figure also shows that the flow with the lowest RTT obtains the highest amount of bandwidth. This problem is due to the well-known out-profile unfair sharing between both TCP flows as explained in [25].

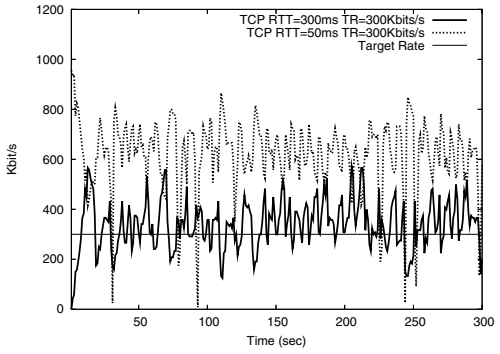



**Figure 2. Two TCP flows in an over-provisioned network with same target rate and different RTT**

Both DCCP flows with the CCID2 congestion control present a similar behaviour. This is a logical result as the CCID2 congestion control reproduces a window based congestion control algorithm TCP-like. As expected, we see in figure 3 that the result obtained is similar to the figure 2.

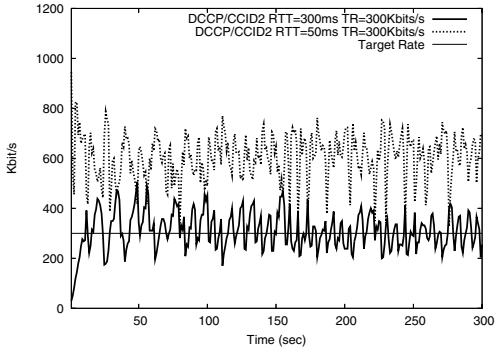



**Figure 3. Two DCCP/CCID2 flows in an over-provisioned network with same target rate and different RTT**

Finally, we made an experiment with two DCCP flows with the CCID3 congestion control in the same conditions. Figure 4 still shows that in a long term perspective, both flows obtain on average the amount of bandwidth obtained previously by TCP and CCID2. The instantaneous throughput is smoother with the CCID3 congestion control mechanism than with CCID2 or with standard TCP congestion control mechanism. This smoothing behaviour is a well-known property of the TFRC congestion control mechanism. Furthermore, the flow with the highest RTT takes a long time to reach its target rate. Several studies have demonstrated that TFRC reacts to transient congestion slower than TCP [1] [26]. This characteristic is also deeply detailed in Jorg Widmer thesis [27]. Even if both flows are closer to their target rate compared to TCP or CCID2 mechanisms, the pace of convergence is unacceptable in case of DiffServ context as the flow with the highest RTT can take more than 100 seconds to reach its own target rate.

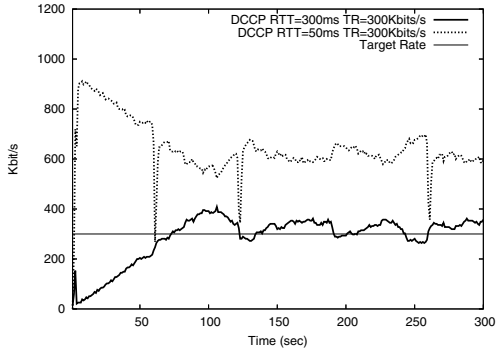



**Figure 4. Two DCCP/CCID3 flows in an over-provisioned network with same target rate and different RTT**

To briefly summarize all these results, figure 5 gives the cumulative average throughput of the three previous experiments related in figures 2, 3, 4. The cumulative average throughput allows to give the perception that the user has in terms of final throughput. This confirms that the convergence of the DCCP/CCID3 protocol in terms of average throughput is not optimal.

*g*DCCP copes with this problem, as it is shown in the figure 6. Both flows reach their target rate in the first second of the experiments like for TCP or DCCP/CCID2 but with a more stable instantaneous throughput. Thanks to the knowledge of the target rate, the protocol is able to send always above its negotiated target rate.

## 4.2 Exactly-provisioned network

In this part, we drive experiments in an exactly-provisioned network. Both flows have different target rates and RTT. One flow has the worst conditions to reach its own target rate. Indeed, the flow with the highest target rate has the highest RTT. As this is the case for TCP flows [25], we see that a DCCP flow with a high RTT and target rate will

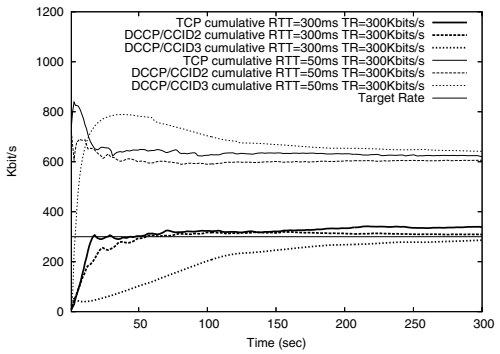



**Figure 5. Cumulative average throughput**

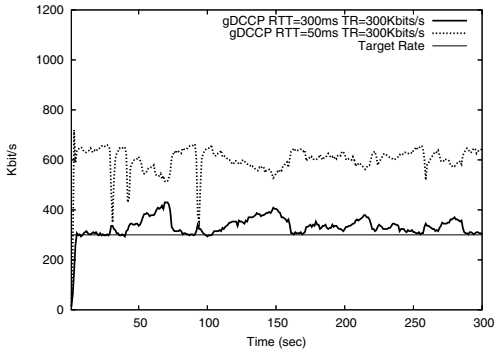
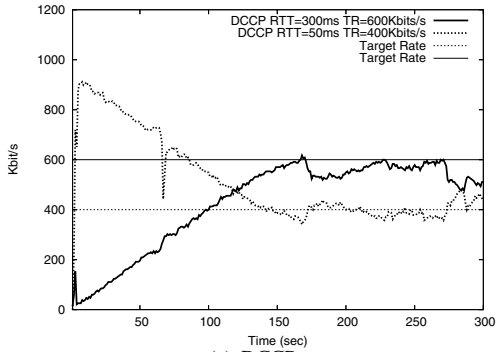



**Figure 6. Two $g$DCCP flows in an over-provisioned network with same target rate and different RTT**

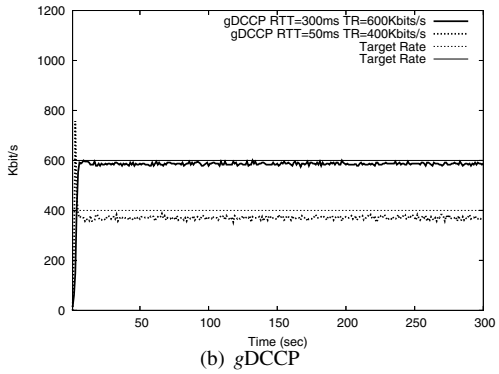
always have the most difficulty to reach its target rate. But, in case of an exactly-provisioned network, Park and Choi [21] show that there is no more unfairness problem.

As the CCID3 congestion control mechanism is based on the TFRC mechanism which models the TCP congestion control mechanism, we can expect that the behaviour of the DCCP/CCID3 flows is similar to the TCP flows on average.

In [27], it is shown that the TFRC mechanism obtains a similar behaviour compared to TCP. Nevertheless, TFRC is smoother than TCP, and TFRC takes longer to reach the link bandwidth than TCP. Concerning DCCP/CCID3, this pace convergence problem is still present as shown in figure 7 (a) since this convergence problem is only linked to the RTT value and not to the network state [27].



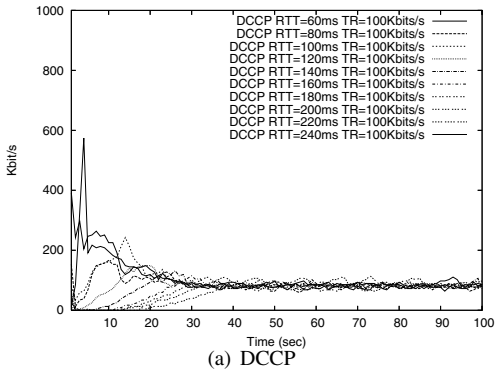

(a) DCCP



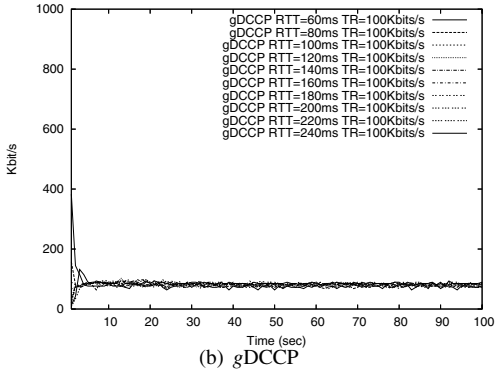

(b) $g$DCCP

**Figure 7. Two flows in an exactly-provisioned network with different target rate and RTT**

Thanks to the $g$DCCP mechanism, figure 7 (b) shows that both flows reach their target rate soon and that the convergence problem disappears. Finally, we conclude this case by sending ten DCCP/CCID3 and ten $g$DCCP flows with the same target rate and various RTT. Figures 8 present the obtained results and show the efficiency in terms of target rate achievement and pace of convergence with the $g$DCCP mechanism.

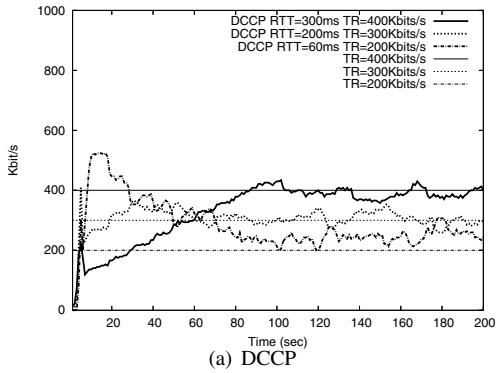



(a) DCCP

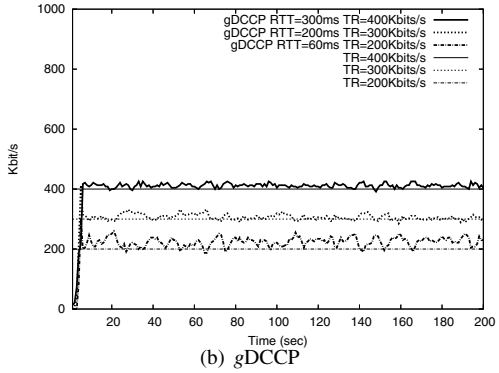



(b) $g$DCCP

**Figure 8. Ten flows in an exactly-provisioned network with same target rate and different RTT**




(a) DCCP




(b) $g$DCCP

**Figure 9. Three flows in an over-provisioned network with different target rate and RTT**

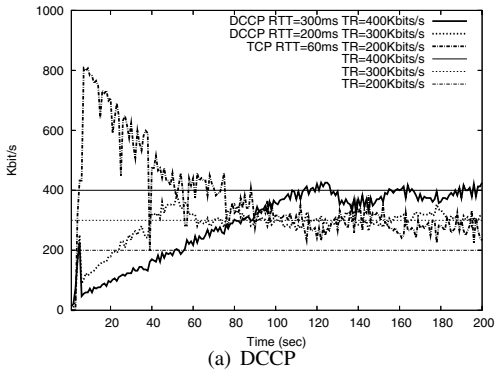
### 4.3 Over-provisioned network with crossing traffic

This section deals with the case of an over-provisioned network with crossing traffic. Two flows are emitted on the testbed 1 between $(A, D)$ and $(B, E)$ and a third one is emitted between $(C, F)$ as crossing traffic. We made experiments with an increasing number of flows and crossing traffic and obtained similar results. For the sake of readability, we give in this section the results obtained with only three flows. For all next simulations, $(A, D)$ has an RTT of $300ms$ and a target rate of $400Kbits/s$; $(B, E)$ has an RTT of $200ms$ and a target rate of $300Kbits/s$ and $(C, F)$ has an RTT of $60ms$ and a target rate of $200Kbits/s$. As expected, the flow $(A, D)$ with the highest RTT and target rate has difficulty to reach its target rate as shown in figure 9 (a). This is also the case for the $(B, E)$ flow. If we use the $g$DCCP mechanism, we see in figure 9 (b) that all flows reach their target rate at the same time.

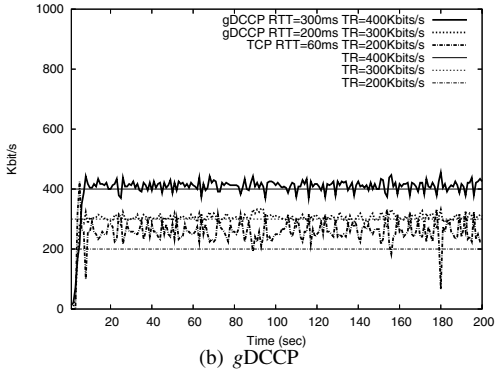
In the next experiments, we replace one of these flows with a TCP flow. In figures 10, the TCP flow is emitted between $(C, F)$ and has the lowest RTT and target rate. Due to the aggressive nature of TCP, figure 10 (a) shows more oscillations than in the case of using only DCCP/CCID3 flows. We also see that TCP outperforms its target rate. In the case of using $g$DCCP with TCP, as shown in figure 10 (b), TCP is not disturbed by these flows and always outperforms its target rate. Concerning the others $g$DCCP flows, we can see that they reach both their target rate. Moreover, the $g$DCCP flows stay close to their target rate. Without $g$DCCP flows, in figure 10 (a), we can see that sometimes, the $(A, D)$ flow falls under its target rate as this is the case between $t = [200, 250]$.

In figures 11, the TCP flow crosses the $(A, D)$ path and is now in the worst conditions to achieve its target rate. Indeed, it gets the highest RTT and the highest target rate. Figures 11 show the results obtained in this case. Figure 11 shows that the TCP flow has no difficulty to reach its target rate with DCCP/CCID3 or with $g$DCCP . It means that the $g$DCCP mechanism does not influence the behaviour of TCP in the DiffServ/AF class when these flows are mixed together.

Finally, in the last experiments, twenty flows cross the testbed with a RTT ranging from $50ms$ to $1000ms$. Ten between $(A, D)$ and ten between $(C, F)$. Each flow has a

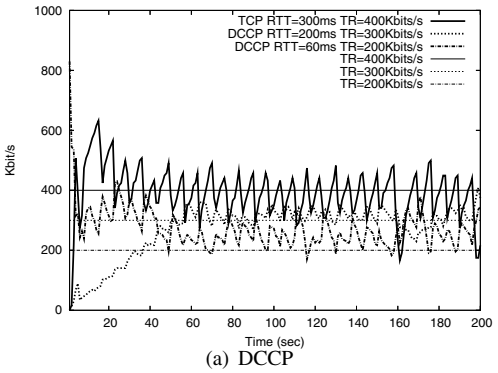



(a) DCCP

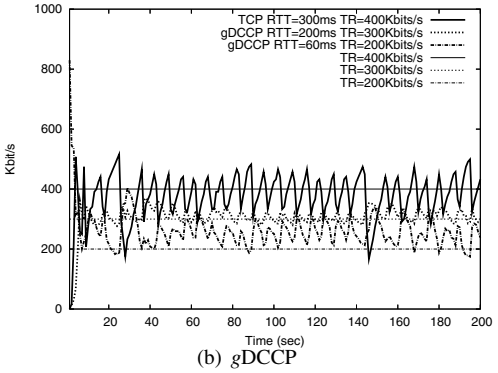



(b) $g$DCCP

**Figure 10. One TCP with the lowest RTT flow versus two DCCP flows in an over-provisioned network**




(a) DCCP




(b) $g$DCCP

**Figure 11. One TCP with the highest RTT flow versus two DCCP flows in an over-provisioned network**

target rate of $50Kbits/s$. The bottleneck in this experiment is $1500Kbits/s$. So there is $500Kbits/s$ of excess bandwidth. Figure 12 compares the results obtained with twenty DCCP and twenty $g$DCCP and give the average throughput obtained for each flow at the end of the experiment. The x-axis in figure 12 gives the RTT value of the measured flow. The y-axis value is corresponding to the average throughput. The test time is set to $100sec$. As already seen before, the DCCP flows with the highest RTT have difficulties to reach the target rate and the flows with the lowest RTT occupy the most part of bandwidth. With $g$DCCP , we see that all flows, whatever their RTT, are closed to the target rate.

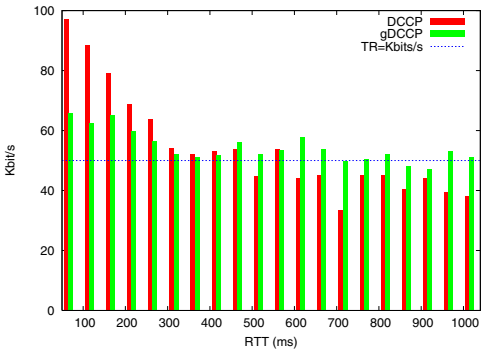



**Figure 12. Twenty flows in an over-provisioned network with same target rate and different RTT**

## 5 Discussion about this proposal

In this section, we propose to discuss on security and misconfiguration problems of the guarantee $g$. Two cases can occur: the first one is a volunteer misconfiguration from the user. In this case, the network provider controls the misbehaving user's traffic as any other kind of traffic and the network drops the excess part. Nevertheless, this case is similar to mix UDP and TCP traffics in the same service class and even if there is a marking strategy at the edge of the network, others flows could suffer of this misbehaving traffic. The second one is a misconfiguration from the QoS provider. In this context, both conditioning method (i.e. the

target rate of the token bucket marker) and the guarantee $g$ are erroneous. In this specific case, it could be better that the protocol reacts to the resulting congestion. The next two parts deals with these problems.

### 5.1 Security concern

As we give the possibility to instantiate through a `setsockopt()` function the target rate negotiated between the QoS network and the application, we can imagine that a misbehaving person could abuse of this functionality by giving an higher value to the guarantee $g$. In this case, the misbehaving person sends an UDP-like traffic and increases its out-profile traffic. In the context of a DiffServ/AF class, the edge router will still mark in-profile the packets in respect with the negotiated profile and out-profile the excess part. As a result, in case of network congestion, the dropping precedence set in the core router will drop this excess traffic. The misbehaving person will not take advantage of the situation as the number of losses of its own flow increases as well. Then, the in-profile traffic remains protected in the network and the others out-profile traffics, sharing the same link, perceives a kind of flooding attack. As the out-profile traffic is a best-effort traffic, this case of use does not disturb the management of the DiffServ network. Furthermore, even if the in-profile part remains protected, the behaviour of the out-profile part is not share in the same manner. As underlined in [18], we obtain a better control by separating non responsive to responsive traffic even over a specific QoS service. Nevertheless, we believe that this security concern is out of the transport level scope. As for any kind of transport protocol, we claim that it is definitely not the responsibility of the protocol to detect a selfish user behaviour.

### 5.2 Misconfiguration from the QoS Service Provider

This case is more problematic and occur when the network configures and gives a wrong configuration both on client side and network side. In the previous section, the misbehaving traffic was the out-profile traffic. Now, the in-profile becomes not legitimate and gets losses as the QoS provider doesn't manage this traffic in respect with the bandwidth available. We are in an under-provisioned network and the in-profile is not protected anymore. In section 3, we have proposed to add a second ratio to our computation proposal in order to deal with this case. This second ratio allows the flow to react to an abnormal number of losses in the in-profile part. The flow's reaction to this unexpected losses can be seen as in contradiction with the DiffServ principles. Nevertheless, it could be in the end system's best interests to send in a congestion controlled manner rather than getting losses. This problem is currently under discussion [17] and is difficult to solve as it combines users' perception and network interests.

## 6 Conclusion

This paper studies the behaviour of DCCP over DiffServ with Assured Forwarding class. TCP and DCCP with both congestion control mechanisms appear to present limitation in this context. To cope with these limitations, we propose to evaluate an evolution of DCCP, named $g$DCCP integrating a QoS aware congestion control based on TFRC. Thanks to this knowledge, we show that $g$DCCP reaches easily its target rate whatever the RTT or the target rate of a flow. It requires only the target rate negotiated by the application. We have demonstrated through many experiments its use over a standard AF/DiffServ class. Finally, we have discussed about problems related to security and real deployment over a DiffServ network of this proposal. We have recently integrated this modification into a real DCCP implementation originally developed at Lulea University of Technology Sweden. Large number of measurements are expected over a real network. See [14] for details.

## Acknowledgments

This research work has been conducted in the framework of the EuQoS European project (http://www.euqos.org). Emmanuel Lochin and Guillaume Jourjon has been supported by funding from National ICT Australia (NICTA). The authors thank Sebastien Ardon and the members of the IETF's TSVWG working group for their comments on this mechanism.

## References

[1] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenkerr. Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms. In *Proc. of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[2] M. El-Gendy and K. Shin. Assured Forwarding Fairness Using Equation-Based Packet Marking and Packet Separation. *Computer Networks*, 41(4):435–450, 2002.

[3] W. Fang, N. Seddigh, and AL. A Time Sliding Window Three Colour Marker. Request For Comments 2859, IETF, June 2000.

[4] A. Feroz, A. Rao, and S. Kalyanaraman. A TCP-Friendly Traffic Marker for IP Differentiated Services. In *Proc. of IEEE/IFIP International Workshop on Quality of Service - IWQoS*, June 2000.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, pages 43–56, Stockholm, Sweden, Aug. 2000.

[6] S. Floyd and E. Kohler. Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control. Request For Comments 4341, IETF, Mar. 2006.

[7] S. Floyd, E. Kohler, and J. Padhye. Profile for DCCP Congestion Control ID 3: TRFC Congestion Control. Request For Comments 4342, IETF, Mar. 2006.

[8] A. Habib, B. Bhargava, and S. Fahmy. A Round Trip Time and Time-out Aware Traffic Conditioner for Differentiated Services Networks. In *Proc. of the IEEE International Conference on Communications - ICC*, New-York, USA, Apr. 2002.

[9] M. Handley, S. Floyd, J. Pahdye, and J. Widmer. TCP-Friendly Rate Control (TFRC): Protocol Specification. Request For Comments 3448, IETF, Jan. 2003.

[10] J. Heinanen and R. Guerin. A Single Rate Three Color Marker. Request For Comments 2697, IETF, Sept. 1999.

[11] Y.-G. Kim and C.-C. J. Kuo. TCP-Friendly Assured Forwarding (AF) Video Service in DiffServ Networks. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Bangkok, Thailand, May 2003.

[12] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). Request For Comments 4340, IETF, Mar. 2006.

[13] K. Kumar, A. Ananda, and L. Jacob. A Memory based Approach for a TCP-Friendly Traffic Conditioner in DiffServ Networks. In *Proc. of the IEEE International Conference on Network Protocols - ICNP*, Riverside, California, USA, Nov. 2001.

[14] E. Lochin. A DCCP and gDCCP kernel patch for FreeBSD 6.1, Aug. 2006. http://mobqos.ee.unsw.edu.au/∼lochin/#coding.

[15] E. Lochin, P. Anelli, and S. Fdida. Penalty shaper to enforce assured service for TCP flows. In *IFIP Networking*, Waterloo, Canada, May 2005.

[16] E. Lochin, L. Dairaine, and G. Jourjon. gTFRC: a QoS-aware congestion control algorithm. In *Proc. of the International Conference on Networking - ICN*, Mauritius, Apr. 2006.

[17] E. Lochin, L. Dairaine, and G. Jourjon. Guaranteed TCP Friendly Rate Control (gTFRC) for DiffServ/AF Network. Internet Draft draft-lochin-ietf-tsvwg-gtfrc-02, IETF, Aug. 2006.

[18] B. Nandy, J. Ethridge, A. Lakas, and A. Chapman. Aggregate Flow Control: Improving Assurances for Differentiated Services Network. In *Proc. of IEEE INFOCOM*, pages 1340–1349, 2001.

[19] B. Nandy, P.Pieda, and J. Ethridge. Intelligent Traffic Conditioners for Assured Forwarding based Differentiated Services Networks. In *IFIP High Performance Networking*, Paris, France, June 2000.

[20] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. Request For Comments 2638, IETF, July 1999.

[21] E.-C. Park and C.-H. Choi. Proportional Bandwidth Allocation in DiffServ Networks. In *Proc. of IEEE INFOCOM*, Hong Kong, Mar. 2004.

[22] P. Pieda, J. Ethridge, M. Baines, and F. Shallwani. A network simulator differentiated services implementation. Technical report, Open IP, Nortel Networks, 2000.

[23] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-Based Congestion Control Mechanism for Real-time Streams in the Internet. In *Proc. of IEEE INFOCOM*, pages 1337–1345, 1999.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Technical Report 3550, IETF, July 2003.

[25] N. Seddigh, B. Nandy, and P. Pieda. Bandwidth Assurance Issues for TCP Flows in a Differentiated Services Network. In *Proc. of IEEE GLOBECOM*, page 6, Rio De Janeiro, Brazil, Dec. 1999.

[26] M. Vojnovic and J. Boudec. The long-run behavior of equation-based rate control. In *Proc. of ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[27] J. Widmer. *Equation-Based Congestion Control*. Diploma thesis, University of Mannheim, Germany, Feb. 2000.

[28] Y. Yang and S. Lam. General AIMD Congestion Control. Technical Report TR-200009, Department of Computer Science, University of Texas at Austin, May 2000.