Combining ACK Rate Control and AQM to Enhance TCP Performance over 3G Links

Juan J. Alcaraz, Fernando Cerdán
Department of Information Technologies and Communications
Polytechnic University of Cartagena
Plaza del Hospital, 1, 30202 Cartagena, SPAIN
34 968326544
{juan.alcaraz, fernando.cerdan}@upct.es

ABSTRACT
When TCP is carried over 3G links, overbuffering and buffer overflow at the RLC layer degrades its performance. In order to prevent these undesired interactions, we propose an algorithm that combines two strategies. If the congestion state of the downlink buffer is moderate, the protocol uses ACK rate control to adjust the sending rate of the source. If the buffer occupancy reaches certain threshold, a single packet is discarded to trigger TCP congestion control measures. By means of extensive simulation experiments, we disclose the influence of each parameter on TCP performance, finding several configurations that improve end-to-end goodput while reducing the delay. The proposal is compared to other approaches, showing a better and more stable behaviour. The operation of the algorithm is deterministic; it does not require changes in 3G specifications nor in end user's protocol stack, and preserves TCP end-to-end semantics.

Categories and Subject Descriptors
C.2.1 [**Network Architecture and Design**]: Wireless communication.

C.2.2 [**Network Protocols**]: Applications – TCP/IP.

General Terms
Algorithms, Performance.

Keywords
TCP over 3G links, Radio Link Control (RLC).

1. INTRODUCTION
While third generation cellular networks are becoming an important part of the Internet, the interest on TCP performance over 3G links is gaining momentum. In the radio access network, data transport services are supported by the Radio Link Control (RLC) [1], which can recover from propagation errors by means of a selective-repeat Automatic Repeat reQuest (ARQ) algorithm. The performance of TCP over a reliable radio bearer (RB) is better than over a raw wireless link, because it avoids the triggering of unnecessary congestion control measures [2] when packets are lost due to propagation errors. However, continuous variations in propagation contitions cause changes on avaliable bandwidth as well as high and variable latency in 3G links. These characteristics may cause buffer overflow of RLC downlink buffers [3], which have undesired effects on TCP performance.

Problems in RLC buffers are, to some extent, similar to those in a buffer of an Internet congested router. Based on this, some previous works proposed the application of Active Queue Management (AQM) techniques at the RLC layer [4, 5]. Several characteristics make RLC different from a router link. RLC works in a per-user basis, and therefore multiplexes much less TCP flows and its bandwidth is much lower than that of a router link. Hence, AQM techniques should be adapted or specifically designed for its use at RLC buffers. A novel AQM scheme, named Packet Discard Prevention Counter (PDPC) was presented in [5]. This protocol has many desirable features for its implementation in RLC: it is simple and deterministic.

A slightly different approach is to control the rate of upgoing ACKs at RLC. This mechanism was presented in [6] as an algorithm aimed to improve the performance of TCP over satellite links. The underlying idea is to delay acknowledgements travelling through a node where its forward connection is congested. As shown in [3], this algorithm can be adapted to RLC buffers, enhancing TCP performance.

In this paper we show that end-to-end performance can be improved even more with a new algorithm consisting of a combination of PDPC and ACK rate control. In our proposal, the ACK rate computation relies on a function of the buffer occupancy (*BO*), instead of using a constant ACK delay value like the original algorithm does. This generalization gives us a wider point of view to find optimum configurations. In our algorithm, a packet discard forces a temporal change in ACK rate control policy. The consequence is that the function that relates *BO* with ACK inter-departure time shows a sort of hysteris cycle. We provide a feasible implementation of the algorithm, and by means of extensive simulation tests, we discuss the influence of each parameter and propose configurations that clearly improve end-to-end performance compared to the conventional RLC operation.

PM²HW²N'06, October 2, 2006, Torremolinos, Malaga, Spain.

Table 1. Simulation Parameters

3G link parameters	Setting
PDU payload size	320 bits
TTI (Transm. Time Interval)	10 ms
Transmission window	1024 PDUs
maxDAT	10
In-order-delivery	true
Status Prohibit Timer	60 ms
Missing PDU detection	true
Poll Timer	60 ms
Wireless Round Trip Delay	50 ms
Normalized doppler frequency	0,01
Poll window	50 %
Last PDU in buffer Poll	yes
Last retransmitted PDU Poll	yes
Frame Error Ratio (FER)	10%
TCP parameters	**Setting**
Maximum TCP/IP packet size	1500 bytes
Maximum allowed window	64 kbytes
Initial window	1
Wired Network Round Trip Delay	200 ms

We compare our proposal with PDPC and ACK rate control algorithms separately, providing performance figures and traces. This let us provide a quantitative and qualitative comparison, showing that our proposal benefits form the synergy between both schemes.

The rest of the paper is organized as follows. Section 2 describes the characteristics of 3G radio bearers that degrade TCP performance. Section 3 explains our proposed algorithm in detail. Section 4 provides a brief description of the simulation environment. Section 5 discusses the influence of each parameter in end-to-end performance based on extensive simulation results. Section 6 compares our proposal with PDPC and ACK delay control. The paper concludes in section 7.

2. MOTIVATION

Previous experimental [7] and simulation [8] results provide a clear view of the characteristics of 3G wireless links. The behaviour of the link buffer occupancy has shown a great impact on TCP performance.

At a reliable RLC layer, Service Data Units (SDUs) are stored in the downlink buffer until they are fully acknowledged by the receiver side. The consequence is that, as described in [9] frame losses in the downlink channel result in higher RLC buffer occupancy at the network side. Considering that the current RLC specification propose a drop-tail scheme, the buffer may overflow causing consecutive packet losses. This situation is especially harmful in the first stages of a TCP connection (slow start) and has a higher impact in TCP Reno, which can only recover from consecutive packet losses with a Retransmission TimeOut (RTO). An RTO reduces TCP transmission window to one, causing the highest reduction of the source rate.

The buffer should be large enough to avoid frequent overflow. However, excessive queuing causes some additional problems [7] like Round Trip Time (RTT) inflation, unfairness between competing flows and viscous web surfing.

Figure 1 illustrates the end-to-end goodput and delay of TCP over an RB for different RLC buffer sizes and a number of flows ranging from 1 to 4. The goodput represents the successfully received packets at the receiver and the delay is the transfer time of a packet in the downlink direction, at the TCP layer. The buffer size is given in RLC Service Data Units (SDU) of 1500 bytes. Table 1 shows the parameter configuration for the RLC and TCP protocols. The RLC parameters were set according to the optimizing considerations described in [3] and [8]. Further details on the simulator are provided in Section 4. As expected, Figure 1 reveals that a larger buffer benefits the goodput performance but the overbuffering increases the latency.

Figure 2 shows the trace of a TCP connection over a 384 kbit/s radio bearer multiplexing one TCP flow. The curve at the top shows the RLC buffer occupancy (*BO*). The buffer size is limited to 50 SDUs. The TCP congestion window (*cwnd*) is depicted below, and the curve at the bottom shows the sequence number of the packets when they are sent, received and dropped. At the first stages of the connection, multiple packets are dropped due to buffer overflow, causing an RTO. The buffer is drained because the connection reduces its rate, with the consequent underutilization of the resources. After that, the rate is recovered slowly and the overbuffering appears again, causing high delay and additional packet losses.

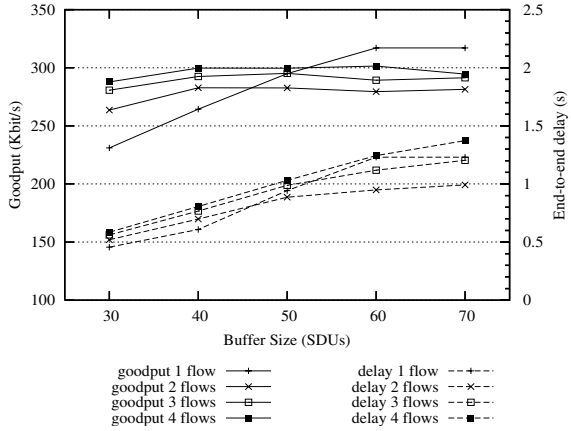

Figure 1: TCP performance over a 384 kbit/s RB.

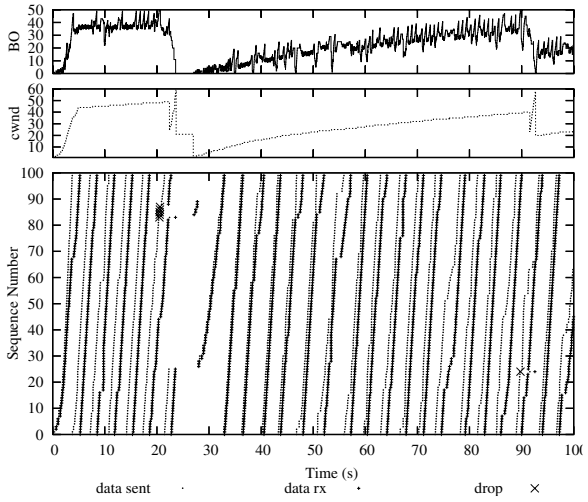

Figure 2: TCP trace over a 384 kbit/s RB.

3. DESCRIPTION OF THE ALGORITHM

The proposed algorithm comprises two protection levels, based on the channel quality perceived with the observation of the *BO* level. When *BO* is below certain threshold (*minth*), the algorithm assumes that the channel quality is good and there is no congestion. Therefore no congestion control measure is applied in this state. When *BO* is between certain bounds (*minth* and *maxth*), the algorithm applies the first mechanism to prevent congestion: the ACK rate control. When *maxth* is reached, the second mechanism, consisting on a single packet discard, takes over. In contrast to AQM methods proposed for Internet routers, *BO* is the instantaneous queue size. This is better for RLC buffers compared to the queue averaging done in RED-like proposals for Internet routers, because it simplifies protocol implementation and provides a faster reaction when the channel condition changes, as argued in [3] and [5].

3.1 First Mechanism: ACK Rate Control

The motivation of the ACK rate control algorithm relies on TCP self-clocking operation. TCP is basically a sliding window algorithm. The transmission window limits the total amount of data that the source can transmit without receiving any acknowledgement from the receiver side. The window moves forward and its length increases with the arrival of ACKs, thus the source can inject new segments in the network. In consequence, the throughput of a TCP source is ultimately determined by the arrival rate of ACKs.

The proposed algorithm adjusts the rate at which TCP ACKs cross the RLC layer, in order to adapt the sending rate of the TCP source to the link situation. The objective is to avoid overbuffering and buffer overflow in the RLC layer while achieving full utilization of the link bandwidth. *BO* is the variable used to perceive the channel quality, and the rate control measures are applied only when certain threshold is reached (*minth*). During this phase of the algorithm, i.e. when no SDU has been discarded recently, the function used to compute inter-departure time, t_{id}, of ACKs traversing the RLC layer is $t_{id} = f_1(BO)$, which obviously should be a continuous and only-increasing function because the departing rate of ACKs has to be reduced when *BO* increases.

The motivation of the variable delay strategy, in contrast to the fixed delay applied in [6], is to gradually adapt the source's perception of the available bandwidth and the round trip delay. Higher buffer occupancies require higher reductions in the sending rate of the source, while for a short error burst an excessive delay on the ACKs could be unnecessary. This should also help to avoid excessive delay spikes causing spurious TCP timeouts.

It may be argued that a drawback of the ACK delaying strategy is the Retransmission Time Out (RTO) inflation caused by the increment on the perceived RTT. The RTO inflation slows down the reaction of the source upon the eventual loss of consecutive packets. However, the higher delay of the reverse path is compensated by a lower buffer occupancy, which reduces the downlink packet latency. In addition, a main goal of this algorithm is the full avoidance of buffer overflow, thus TCP timeouts are expected to happen rarely.

3.2 Second Mechanism: Single Packet Discard

When *BO* reaches *maxth*, the algorithm discards an SDU in the downlink RLC buffer. Like in other AQM schemes, a packet drop is in fact a signal directed to the source aimed to reduce the sending rate of TCP. When a TCP Reno source detects a single packet loss, with the reception of three duplicate ACKs, it starts the Fast Retransmit Fast Recovery mechanism which forces a retransmission and halves TCP transmission window.

Instead of using a probabilistic discard, like most RED-like algorithms [10] do, our scheme is deterministic, i.e. as soon as the queue length equals *maxth*, a packet is dropped. This strategy was first used in the Packet Discard Prevention Counter (PDPC) algorithm proposed in [5]. Once a packet is dropped, any additional discard is prevented until certain amount of data (*n* bytes) is drained from the buffer. This quantity, *n*, is an estimation of the TCP transmission window, assuming that the buffer multiplexes just one connection. The objective of this preventing measure is to avoid the loss of more than one packet in the same transmission window, which causes an RTO in a TCP Reno source. The implementation requires a single variable (*Counter*), which is set to *n* when a packet is dropped. When the link layer successfully delivers a SDU, *Counter* is decremented by the size of this SDU. When *Counter* equals 0, the discard prevention period ends. We have to consider the impact of this algorithm on ACK rate control.

After a packet discard the t_{id} computing function switches to a different one, $f_2(BO)$. When $Counter = 0$, $f_1(BO)$ is used again. Several ideas are behind this temporal change. The ACK delivering rate should be accelerated because the source should detect the packet loss as soon as possible. Moreover, after the rate reduction, the link usage should not decay too much. Therefore, $f_2(BO)$ is aimed to adjust the t_{id} while *Counter* decreases. However, as will be shown in the simulation results, the effect of $f_2(BO)$ is not too significant. This let us implement the protocol in a simpler way, using $f_2(BO) = 0$, i.e. disabling the ACK rate control mechanism while $Counter > 0$.

Initializing *Counter* after a packet drop. In accordance to previous works [9], our simulation experiments show that, for a single TCP flow the RLC buffer occupancy corresponds approximately to the window size, given that the radio bearer is the bottleneck link. When several flows are multiplexed, the window sizes are smaller than *BO*. An oversized *Counter* value is not desirable, because it reduces the algorithm responsiveness. Therefore, we propose to set *Counter* equal to the *BO* value when the packet is dropped (generally *maxth*). This value was proved accurate in the simulations, as expected.

Choosing the right packet to discard. The packet chosen for discard will be as close as possible to the front of the queue, in order to reduce the reaction time. Additionally, the algorithm should not discard a packet if its transmission over the RLC link has already started. Otherwise, upon a packet discard, the RLC would start the signalling procedure required to synchronize RLC sender and receiver sides [1]. Consequently, our protocol discards the first packet whose transmission has not started, thus reducing complexity and avoiding changes in the 3GPP specification itself.

3.3 Combining Both Mechanisms

Figure 3 shows the ACK rate control and the discard schemes combined in a single algorithm. The implementation relies in the use of a single timer (*id_timer*) which delays ACKs assuring that they are spaced by t_{id}. The pseudocode identifies four possible events: the arrival and departure of an SDU in the downlink

buffer, the arrival of an ACK packet at the uplink buffer and the

```
for each SDU successfully delivered (downlink)
   if (Counter > 0)
        Counter ← max{ Counter - SDU size, 0}
for each SDU arrival (downlink)
      if ((BO >= maxth) and (Counter == 0))
         drop an SDU
         Counter ← BO
for each ack_arrival (uplink)
   insert ack in uplink buffer
   if (id_timer off)
      delay_and_send
for each id_timer_expiration
   send first ack in buffer
   LADT ← current_time
   if (uplink buffer nonempty)
      delay_and_send
function delay_and_send
   delay ← 0
   if (BO>=minth)
      if (Counter == 0) t_id = f1(BO)
      else t_id = f2(BO)
      actual_t_id = current_time - LADT
      if (actual_t_id < t_id)
         delay ← t_id − actual_t_id
   if (delay == 0)
      send first ack in buffer
      LADT ←current_time
   else
      id_timer ← delay
      activate id_timer
Events:
SDU successfully delivered (downlink)
SDU arrival (downlink)
ack_arrival (uplink)
id_timer_expiration
Saved Variables:
LADT: Last Ack Departure Time
Counter: Discard Prevention Counter
Other:
current_time: system clock
delay: artificial delay applied to ACKs
```

Figure 3. Proposed ACK rate control algorithm.

expiration of the *id_timer*. Figure 4 illustrates the operation of the algorithm. For the sake of clarity, the particular case of $f_2(BO) = 0$ is considered, therefore, every ACK arriving after the packet drop is delivered without any additional delay, until Counter = 0.

3.4 Calculating the Inter-Departure Time

The original proposal of reverse ACK rate control relies on a fixed delay [6]. In a previous work [3] we introduced the possibility of applying variable delay, setting a linear relation between the delay and the downlink queue length. In this paper we consider a more general function (1) to estimate t_{id} with BO, where *maxd* is the maximum value for t_{id}, and *maxth* is the discarding threshold.

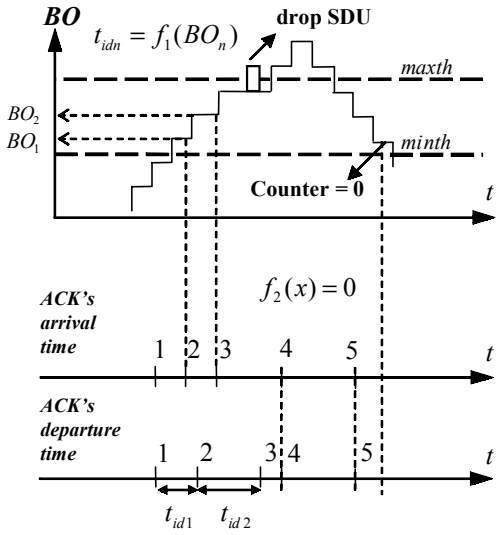

Figure 4. Operation of the algorithm.

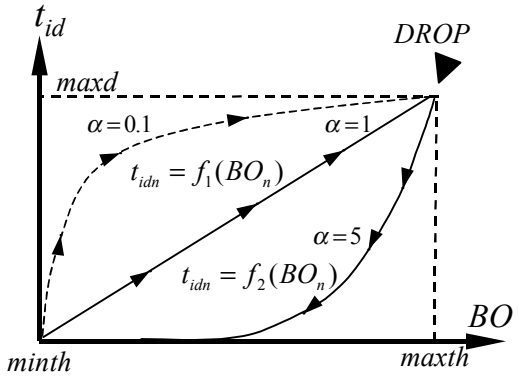

Figure 5. t_{id} computing function. Hysteresis cycle.

$$t_{id} = maxd\left(\frac{BO - minth}{maxth - minth}\right)^{\alpha} \qquad (1)$$

This function let us evaluate the effect of the variation rate of t_{id} respect BO. This variation rate, which is related to the "aggressiveness" of the algorithm, is determined by α (≥0). Lower values of α are tied to more aggressive delaying policies. Figure 5 shows the shape of (1) for different values of α.

In (1) it is easy to see that, if $\alpha \rightarrow 0$, *f(BO)* tends to the step function, i.e. t_{id} is constant when BO ≥ *minth*. On the other hand, if $\alpha = 1$, *f(BO)* is a linear function. Therefore, the step and the linear functions are particular cases of (1). Moreover, the absence of delay (*f(BO)* = 0) is also a particular case of this function, when $\alpha \rightarrow \infty$ and *BO* < *maxth*.

Figure 5 shows a couple of functions $f_1(BO)$ and $f_2(BO)$ for a particular configuration of the algorithm. As explained, when *BO* reaches *maxth*, a packet is discarded and the t_{id} computation function switches from $f_1(BO)$ to $f_2(BO)$ in order to accelerate the ACK serving rate when the buffer occupancy starts to decay. The arrows in the graphic stand for the expected *BO* evolution after and before the packet drop event. This behavior is somewhat

similar to a hysteresis process, except for the fact that the return from $f_2(BO)$ to $f_1(BO)$ is triggered when *Counter* = 0.

4. SIMULATION ENVIRONMENT

The simulation environment for this research has been developed in OMNeT++ [11] and comprises a complete implementation of TCP and RLC protocols. Similar simulators were described in [8] and [9]. The simulation topology, shown in Figure 6 consists of one or several TCP sources connected to their respective receivers in the user's equipment (UE). The end-to-end connection consists of two sections, the wired network and the radio bearer. The wired network comprises the Internet and the 3G core network. The radio bearer has a round trip time (RTTw) of 50 ms [12] and a bidirectional nominal rate of 384 kbit/, representing the bottleneck link, which is the situation expected in most cases [13]. The wired network is modeled with a 1 Mb/s link with a round trip delay (RTTf) of 200 ms.

The wireless channel generates error bursts according to the model described in [14] where the Doppler frequency, f_d, of the UE determines the average burst length. Lower f_d causes longer bursts of errors. It is usual to employ the normalized Doppler frequency, equal to the product of f_d and the radio frame duration (10 ms).

In order to obtain more realistic results, the error probability is the same in the uplink and in the downlink direction. The frame loss ratio is 10%, a typical UMTS design value [12].

The simulation results exposed in this paper are obtained averaging 20 runs per sample. Each run is a 60 second download session. The confidence intervals are obtained with a confidence degree of 90% according to a t-student distribution.

The TCP flavour employed is TCP Reno, one of the most extended in the Internet [13]. RLC and TCP parameter setting is shown in Table 1.

5. SIMULATION RESULTS

In order to disclose the effect of each parameter in end-to-end performance, multiple parameter combinations were tested. The value of *maxth* ranged from 20 to 45 SDUs in steps of 5. To evaluate the efect of *maxd* we changed the ratio *maxd*/*maxth* from 5 to 30 in steps of 2.5. This way, given a certain ratio value, the shape of the curve $f_i(BO)$ is similar for different *maxth* values. The effect of delaying ACKs after a packet discard was evaluated configuring $f_2(BO)$ with $\alpha = 1$ and $\alpha = \infty$ ($f_2(BO) = 0$) in each scenario, i.e. two extreme options. The values used for α in $f_1(BO)$ are 1, 0.5 and 0.2. For $\alpha = 1$, *minth* = 0 and for $\alpha < 1$, *minth* = 10 SDUs. As explained later in this section, these values assure that ACKs are not delayed when there is no congestion. The tests were repeated with the RB multiplexing 4 TCP connections.

In the figures shown in this section, the goodput and delay performance for the conventional RLC drop-tail (DT) operation is also shown as a reference value. Our objective is to find the range of parameter values that improve one or both performance figures.

Figure 7 shows the performance for one TCP flow and several parameter configurations. We define the aggressiveness of the algorithm as the tendency to reduce the source's rate for each *BO* level. It is clear that, if the protocol discards a packet at low *BO* values (low *maxth*) or if t_{id} grows fast (high *maxd*/*maxth*), the aggressiveness is high. As expected, more aggressive configurations yields lower delay values, because the buffer occupancy is kept at lower levels. Because packet loss bursts are avoided, several configurations also improve the goodput. If the

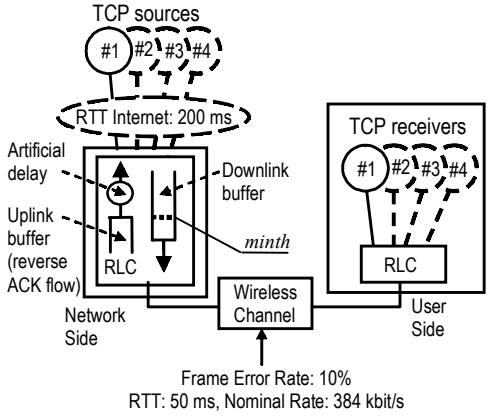

Figure 6. Simulator topology.

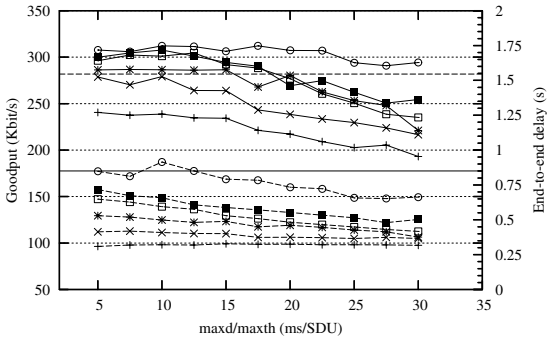

Figure 7 (a). $\alpha = 1$ in $f_1(BO)$; $f_2(BO) = 0$.

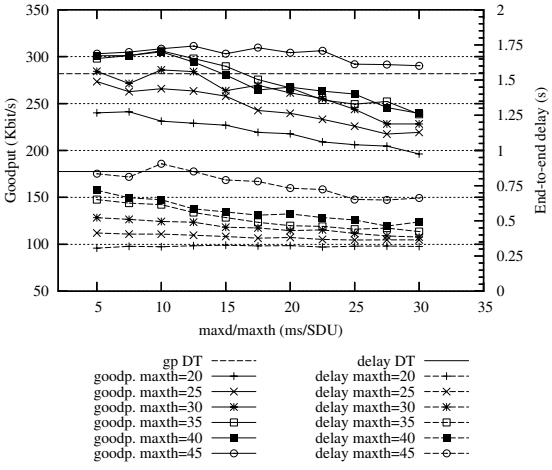

Figure 7 (b). $\alpha = 1$ in $f_1(BO)$ and in $f_2(BO)$.

configuration is too aggressive, the average *BO* is too low, and thus the available bandwidth is underused and the goodput decays. As can be seen in Figure 7, several configurations are able to improve the goodput while reducing the delay. Figure 7 also

shows that the effect of $f_2(BO)$ in the overall performance has low impact. In the rest of the figures $f_2(BO) = 0$.

Our simulations show that best performing configurations maintain the average buffer occupancy around 16 SDUs. This

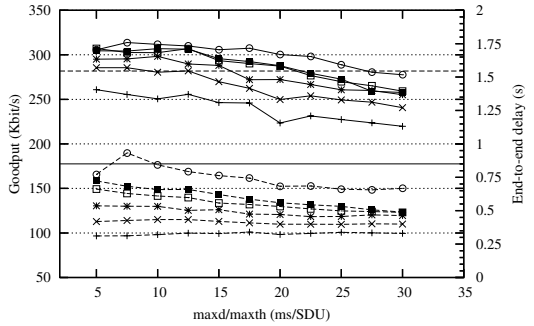

Figure 8. $\alpha = 0.5$ in $f_1(BO)$.

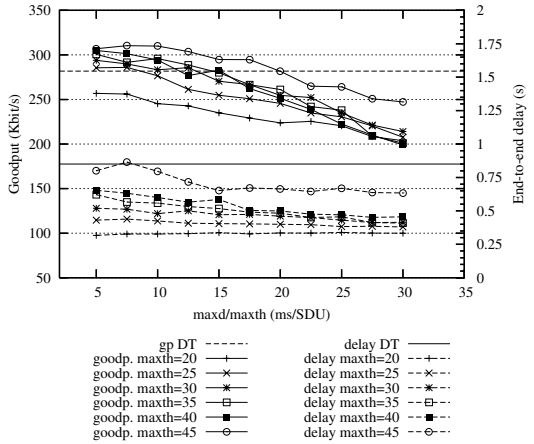

Figure 9. $\alpha = 0.2$ in $f_1(BO)$.

level is obviously above the bandwidth delay product, and therefore allows a full usage of the available resources. At the same time, it is low enough to accommodate sudden increments of the queue length caused by periods of low channel quality, avoiding buffer overflow. Of course, it is lower compared to the conventional RLC, which is around 24 SDUs, and is the main reason of packet latency reduction.

Figures 8 and 9 show the performance measurements with a single TCP flow for $\alpha = 0.5$ and $\alpha = 0.2$. Given certain *BO* level, lower values of α produce higher t_{id} values. In consequence, the goodput reduction occurs for lower *maxd*/*maxth* values. On the other hand, the protocol is less influenced by the value of *maxth* (curves are closer), given that for higher α, it is less probable that *BO* reaches the discarding threshold.

From the simulation tests, we can derive a few rules that give adequate performance of the proposed algorithm.

1. Set the discarding threshold high enough. In the proposed environment, best performance figures are obtained for *maxth* set to 35 or 40 SDUs. In our proposal, the discarding measure is tied to noticeable congestion, and should be used when the ACK rate is not capable to handle it.
2. When *BO* is low, no delay should be applied to outgoing ACKs. A rule of thumb should be not to delay ACKs if BO is below the BDP of the connection. In most TCP implementations, one ACK packet is sent when two TCP

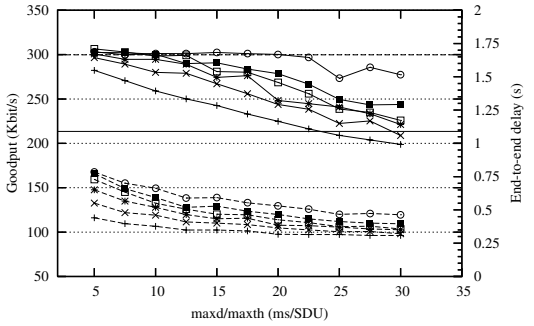

Figure 10. $\alpha = 1$ in $f_1(BO)$; 4 TCP flows.

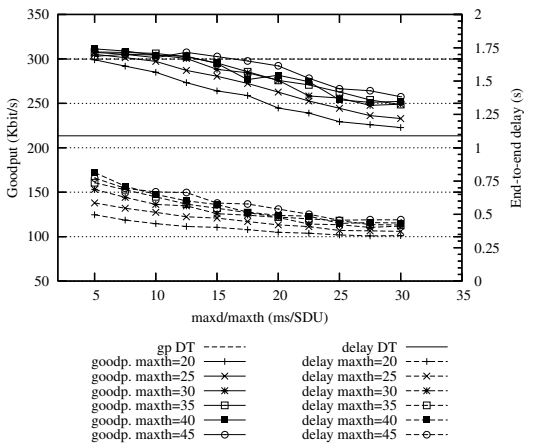

Figure 11. $\alpha = 0.5$ in $f_1(BO)$; 4 TCP flows.

segments are received in sequence. In our environment it implies that, in normal operation, two consecutive ACKs are spaced by, approximately, 140 ms. Therefore, if t_{id} is around this value, the effect of the algorithm is imperceptible. Our simulations show that optimum configurations follow a similar pattern: t_{id} is lower than 140 ms for BO values below 10 SDUs (a value around the BDP).

3. When the occupancy is near the discarding threshold, *maxth*, the rate should be reduced by a factor of 3, i.e. t_{id} should be around 400 ms.

5.1 Effect of the Number of Sources

As argued in [5, 13], TCP performance over wireless links should be evaluated considering a number of sources ranging from 1 to 4 TCP flows, which is the expected number of simultaneous TCP connections established by one user. This fact has great influence on the parameter setting decisions for an AQM scheme at the RLC buffer, given that, if the link layer is not aware of the number of multiplexed flows, the AQM configuration should be accurate for one and for several flows.

Figures 10 and 11 shows the performance figures for $\alpha = 1$ and $\alpha = 0.5$ in a 4 flow scenario. The algorithm does not improve the goodput performance because a 50 SDUs buffer is enough for the

standard RLC to achieve optimum goodput performance when multiplexing 4 connections. In this scenario, the main problem is overbuffering (see Figure 1). With the proposed algorithm there is a set of possible parameter values that can reduce the latency in up to a 50% while maintaining the maximum goodput. The range of configuration values that improve performance match roughly with those identified for the one-flow case.

6. COMPARISON STUDY

In this section we compare PDPC, ACK rate control and the proposed algorithm. By means of multiple simulation tests we found configuration options achieving best performance for each scheme. Therefore we can compare best performance figures of each algorithm. Moreover, in order to obtain a qualitative vision, we provide traces of TCP over RLC for each mechanism.

1. PDPC. One of the main advantages of PDPC is that it only relies in one parameter, the discarding threshold *maxth*. In Table 2, the value of *maxth* for this scheme is 30, which does not give the best goodput for the single flow case, but gives a better balance between goodput and delay. Increasing the goodput in 10 kbit/s for the 1 flow case, implies an increment of 200 ms in average delay (for 1 and 4 flows).

Table 2. Performance figures of each scheme

TCP flows	Algorithm	Goodput (kbit/s)	Delay (s)
1	Conventional RLC	281.9 ± 8.9	0.85±0.05
1	ACK Rate Control	311.2±2.5	0.76±0.01
1	PDPC	296.2±5.6	0.54±0.02
1	Combined Scheme 1	304.6±4.2	0.57±0.02
1	Combined Scheme 2	308.9±4.6	0.63±0.02
4	Conventional RLC	299.9 ± 3.9	1.09±0.02
4	ACK Rate Control	306.3±6.9	0.56±0.03
4	PDPC	305.1±3.7	0.78±0.01
4	Combined Scheme 1	301.1±4.6	0.51±0.03
4	Combined Scheme 2	306.2±3.1	0.63±0.02

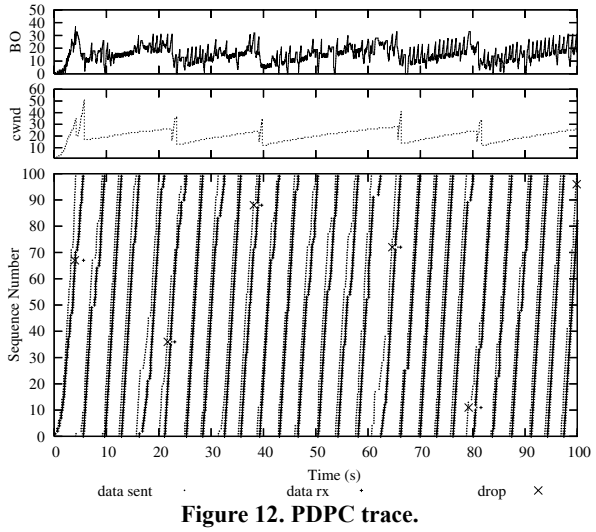

Figure 12. PDPC trace.

2. ACK rate control. Using this algorithm without packet discard is just a particular case of the proposed algorithm, with *maxth* ≥ buffer size. Therefore, there is also a set of possible configuration options giving good performance. For the sake of simplicity, Table 2 shows the figures for a lineal t_{id} computing function ($\alpha = 1$), with *minth* = 0. Configuration rules are a bit different in this scheme, and best performance is obtained with a *maxd*/*maxth* ratio around 20.

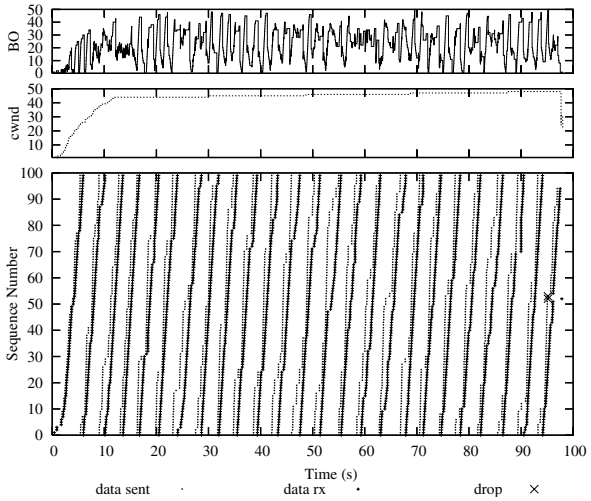

Figure 13. ACK Rate control trace.

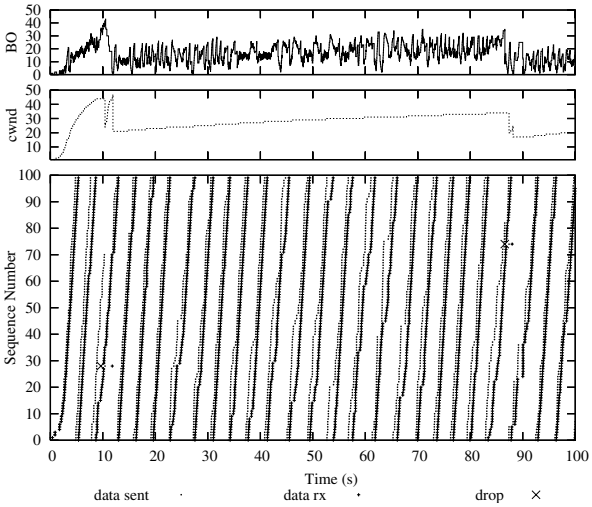

Figure 14. Trace of the proposed scheme

3. Combined Scheme. For our proposal, we have selected two possible configurations:

- Combined Scheme 1: with $\alpha = 1$, *maxd*/*maxth* = 12.5 and $f_2(BO) = 0$.
- Combined Scheme 2: with $\alpha = 0.5$, *maxd*/*maxth* = 10 and $\alpha = 1$ in $f_2(BO)$.

In Table 2 we see that, although ACK rate control presents higher goodput figures, Combined Scheme 1 obtains very low latency in both scenarios, with goodput values near the maximum (only 5 kbit/s below Delayed ACK values). Combined Scheme 2 achieves better goodput figures with slightly higher delay. The following

discussion provides some insight about the behaviour of each scheme.

Figure 12 shows a trace of a single TCP connection over an RLC buffer implementing PDPC, with *maxth* = 30. Delay control is achieved by means of multiple packet drops. In consequence, the goodput does not reach its maximum, because each packet drop is tied to a rate reduction. On the other hand, buffer overflow is successfully avoided, and therefore the overall performance is better than that of the conventional RLC. PDPC goodput improves when using a higher *maxth* value, which obviusly increases *BO* and delay. Figure 13 shows a similar trace for ACK rate control with *maxd*/*maxth* = 20. Although buffer overflow is also avoided by this scheme, the buffer occupancy process shows strong oscillations. As a consequence, the latency experiences continuous variations and its average value is greater than that of other schemes.

Finally, a trace of our proposal (Combined Scheme 1) is shown in Figure 14. In this case, BO is kept more stable, with lower oscillations. Thanks to the ACK rate control algorithm, applied in moderate congestion state, the algorithm does not need to discard too many packets. In addition, if the channel condition degrades too much (higher *BO*), the packet discarding with the deactivation of the ACK rate control provides more stability than t_{id} increment.

7. CONCLUSIONS

We have presented an algorithm capable of enhancing TCP performance over 3G links, solving undesired effects of conventional RLC operation on TCP. For the single flow case, our approach improves goodput and delay performance simultaneously. For several flows, the delay is reduced in up to 50%. Our mechanism adapts two previous existing ideas, ACK rate control, a scheme suggested to enhance TCP over satellite links, and PDPC, a deterministic AQM algorithm for RLC. ACK rate control is used in situations of moderate congestion in RLC buffer, and consists of regulating the inter departure time, t_{id}, of upgoing ACKs to gradually adapt source's rate to the link's available bandwidth. Instead of applying a constant delay to outgoing ACKs, like the original proposal does, our approach uses a generalized t_{id} computing function depending on BO.

When the BO level indicates noticeable congestion, PDPC operation starts. A packet drop is tied to a switch in the t_{id} computing function, which behaviour is therefore similar to a hysteresis cycle.

By means of extensive simulation tests we provide insight about the influence of each parameter, which let us find multiple possible configurations improving overall performance. Configurations showing similar performance figures (goodput and delay) present a similar rate reduction pattern for low and high BO levels.

Our proposal was compared to ACK rate control and PDPC separately. Extensive simulation tests were conducted in order to find the best configuration for each scheme. Our proposal obtained the most balanced results, showing goodput figures near the maximum while the delay values were low for 1 and also for 4 TCP flows. In the traces of TCP connections the combined approach showed the most stable BO process, with a relatively small number of packet discards.

Our mechanism inherits the advantages of the algorithms combined in it. An important one is its deterministic operation, which reduces the computational cost of the algorithm, compared to random or RED-like algorithms, and makes it more feasible for its implementation at the RLC level where the buffering is done in a per-user basis. Some additional advantages of the proposal are that it does not require changes in 3G specifications nor in TCP itself and preserves TCP end-to-end semantics.

8. ACKNOWLEDGMENTS

This work was supported by the Spanish Inter Ministerial Sciece and Technology Commission under project TEC2005-08068-C04-01/TCM.

9. REFERENCES

[1] 3GPP TS 25.322, "Radio Link Control (RLC) protocol specification", v. 6.4.0., Jun. 2005.

[2] H. Inamura et al., "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks" IETF RFC 3481, Feb. 2003.

[3] J. J. Alcaraz, F. Cerdan and J. García-Haro, "Optimizing TCP and RLC Interaction in the UMTS Radio Access Network", *IEEE Network,* vol 20, no. 2, Mar. 2006, pp. 56 – 64.

[4] J. J. Alcaraz and F. Cerdan, "Using Buffer Management in 3G Radio Bearers to Enhance End-to-End TCP Performance", in *Proc. IEEE 20th AINA*, vol. 2, Apr. 2006, pp. 437-441.

[5] M. Agfors, R. Ludwig, M. Meyer and J. Peisa, "Queue Management for TCP Traffic over 3G Links", in *Proc. IEEE WCNC 2003*, pp. 1663 -68.

[6] Jing Wu et al., "ACK Delay Control for Improving TCP Throughput over Satellite Links", Proc. IEEE ICON'99, pp. 303- 312.

[7] R. Chakravorty, et. al., "Using TCP Flow-Aggregation to Enhance Data Experience of Cellular Wireless Users", *IEEE J. Select. Areas Commun.*, vol. 23, no. 6, Jun. 2005, pp. 1190-1204.

[8] M. Rossi, L. Scaranari and M. Zorzi, "On the UMTS RLC Parameters Setting and their Impact on Higher Layers Performance", in *Proc. IEEE 57th VTC*, vol. 3, Oct. 2003, pp. 1827- 32.

[9] R. Bestak, P. Godlewski and P. Martins, "RLC Buffer Occupancy when Using a TCP Connection over UMTS", in *Proc. IEEE PIMRC*, vol. 3. Sep. 2002, pp. 1161-65.

[10] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, pp. 397-413, Aug. 1993.

[11] A. Varga, "The OMNeT++ Discrete Event Simulation System", in *Proc. European Simulation Multiconference*. June 2001.

[12] H. Holma, A. Toskala, *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications, Third Edition*, Wiley, July 2004.

[13] A. Gurtov, S. Floyd, "Modeling Wireless Links for Transport Protocols", *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, Apr. 2004, pp. 85-96.

[14] A. Chockalingam and M. Zorzi, "Wireless TCP Performance with Link Layer FEC/ARQ", in *Proc. IEEE ICC*, June.1999, pp. 1212-16.