Department of Computer Sciences
University of California, Los Angeles

# Technical Report No. 990021

# Improving wireless handoff with GWA-TCP *

M. Gerla[1], R. Lo Cigno[2], W. Weng[1]
[1] Computer Science Department – Boelter Hall – UCLA
[2] Dipartimento di Elettronica – Politecnico di Torino

**Abstract**

The performance of applications running on TCP/IP in cellular networks is heavily influenced by the TCP reaction to handovers, i.e., to short periods when the communication channel is unavailable and packets are lost.

This paper proposes a way of de-coupling the congestion control of TCP from its reaction to the loss of burst of packets. The scheme is a modification of Generalized Window Advertizing (GWA) TCP, recently proposed by the authors.

The performance of the new scheme is compared against traditional TCP implementation and router behaviors. RED buffer management, as well as ECN capable TCP and routers are considered in the comparison. All the results show that the new scheme attains higher performance and allows a better utilization of the network resources.

*This work was done while R Lo Cigno was on leave at the Computer Science Department of the University of California – Los Angeles under grant No. 203.15.8 issued by the Italian National Research Council (CNR).

# 1 Introduction

Recent innovation in cellular networks and wireless networks, makes the provision of data services one of the key priorities for network operators. The parallel evolution of mobile IP makes it difficult to ignore the behavior of the TCP/IP protocol suite in a cellular environment. Recent papers have studied the behavior of TCP connections on wireless, lossy links, as well as its reaction to link unavailability due to handovers [1, 2, 3, 4], i.e., to the procedures used in cellular networks to handle the communication from one access point to the next while the mobile terminal roams through the network.

Both in the case of random losses and handovers, TCP performance shortcoming stems from the impossibility of discriminating between packet losses driven by network congestion and those due to link errors or temporary channel unavailability during handovers. In this paper we concentrate on handovers: Mobile speed increase, associated with cell size reduction make the behavior during handovers of the utmost importance. It is important to notice that, in TCP/IP networks, the latency introduced by handover procedures may be much larger than the sheer time needed to transfer the physical access point from one Base Station (BS) to the other. Indeed, before normal operation can be resumed, the mobile IP forwarding address must be updated and, quite often, TCP operation is completely stalled because a whole window of information gets lost. Under these circumstances TCP will recover only after one or more timeout.

In [1, 2], several proposals for improving TCP performance in wireless and cellular networks are analyzed. They focus on data-link level procedures that may help TCP to recover faster from random losses and link failures. In this paper we propose a mechanism that is instead embedded at the TCP level. We mainly focus on the response to handovers, which, from the TCP point of view, can be modeled as periods of channel unavailability in both directions.

Recently, a new TCP implementation named Generalized Window Advertizing GWA-TCP [5, 6] has been proposed. When all the routers along the connections are GWA-capable, GWA-TCP not only protects the network from congestion. It also ensures that no packets are dropped because of congestion. In this paper we extend the ideas that lead to GWA-TCP and show how a careful GWA-TCP implementation, adapted to the wireless and cellular scenario, can overcome most of the drawbacks of TCP in coping with handovers.

# 2 TCP performance enhancements

Congestion control in the current Internet is primarily a function of TCP. The current IP routers, in their simplest implementation, maintain a single queue with first-come first-served (FCFS) queuing discipline. Incoming packets are dropped when the buffer is full. In other words, routers generally do not control congestion in that they do not control the traffic source directly. They just discard excessive traffic. TCP, on the other hand, has no direct view of the congestion situation in the network. It must infers the congestion state of the network implicitly from data and signalling – arrival of acknowledgements (acks), timeouts, and receipt of the duplicate acks.

As we mentioned earlier, the simplest version of router is Drop-Tail. The Drop-Tail router exhibits rather poor performance in congested situations since TCP gets congestion information only after the network has become congested. Recent studies have proposed to add new features in the routers, such as Random Early Detection (RED) [7] and Explicit Congestion Notification (ECN) [8]. Through these features TCP can get an early congestion notification BEFORE the buffer is full.

In the RED scheme, routers detect incipient congestion by computing the average queue length and comparing it with predetermined thresholds. Congestion can be detected even before the buffer is full, thus controlling the average queue size. RED applies randomization in selecting the connections to notify, with the objective of avoiding global synchronization and applying a dropping probability proportional to the inherent bandwidth used by the various connections. A RED router maintains two buffer occupancy thresholds: a low threshold and a high threshold. No packets are dropped when the average queue length is below the low threshold. When the average queue length exceeds the high threshold, all incoming packets are dropped with probability one. When the average queue length is between the two thresholds, the incoming packet is dropped with a prespecified probability.

The ECN scheme proposed by [8] suggested to mark the packets instead of dropping them when the average queue length is between the two thresholds. This will avoid packet drop when average queue length is below the high threshold. With these additional functions in the routers, it has been showed that TCP can get valid indication of incipient congestion. It can thus provide better performance, especially for TCP Reno in which fast retransmit and fast recovery can avoid to reduce the window size to 1. In this paper, we investigate the performance of TCP Reno interoperating with Drop-Tail, RED, and ECN routers in the wireless networks and compare them with the new scheme proposed next.

# 3 GWA-TCP in Cellular Networks

TCP [10, 11, 12] performs two different, though inter-related, tasks: i) end-to-end flow control to avoid overflowing the receiver buffer; ii) connection congestion control, adapting the amount of information injected in the network to the network congestion status. This latter function is based on *congestion recovery* (rather than prevention). TCP increases the traffic offered to the network until the network is forced to drop some packets; then it backs off to give the network time to relieve from congestion, and starts again to increase the offered load to force another congestion episode.

The key idea of GWA TCP is to "generalize" the interpretation of receiver's advertised window and use it to convey *both* receiver's available buffer space and network congestion status, such as the available buffer for each connection along the path. The latter can be obtained by using IPv6 with an *available buffer space* (ABS) field in its extended. Each router along the path updates this field if its available buffer for this flow is smaller than the value in the ABS (see [6] for detailed description). The per-flow implementation of the GWA TCP control is proven to converge as a first order control system, without fluctuations, guaranteeing MAX-MIN fairness across all flows. A simpler implementation, based on measurements of the common FIFO queue was also implemented, yielding suboptimal, yet very good results [6].

GWA TCP uses network feedback to control its congestion window and avoid packet loss; thus, it ensures high performance in wired networks. Moreover, when it is applied to the wireless, cellular network environment, it does NOT erroneously interpret the loss caused by a temporarily unavailable link situation (eg, deep fade, handoff) or a noisy channel situation as congestion (and thus reduce the congestion window). This is because GWA TCP reacts to queue length rather than packet loss.

For backward compatibility, it is advisable to combine GWA TCP with and existing version of TCP. To this end, GWA can be "superimposed" to any version of TCP. In this paper, we present a GWA TCP which is implemented in TCP-Reno.

Here, we list the key features of GWA-TCP:

1. GWA-TCP RECEIVER compares the ABS field delivered from the network with its available buffer space and puts the minimum of the two in the RECEIVER's advertised window field.

2. GWA TCP SOURCE reacts to a new ACK the same way as TCP-Reno except that its congestion window is set to the RECEIVER's advertised window if the former is larger than the latter.

3. When three duplicate ACKs are received, GWA TCP source retransmits three packets without changing its window size and threshold. We call this mechanism *immediate recovery*. This design choice is based on the assumption that handovers usually cause multiple packet losses. After the fast retransmit, the source is allowed to send one more packet for each duplicate cumulative ACK received as long as it does not exceed the receiver's advertised window.

4. When a retransmit timer expires, GWA-TCP sends one data packet for probing link recovery and resets the timeout with the previous retransmit time value . This modification disables the exponential backoff of the retransmission timers and provides more efficient recovery from link unavailablility. If the retransmitted data packet is ACKed, the source resumes transmission of subsequent packets without reducing window size and threshold. I.e, the timeout is interpreted as caused by link noise/temporary anavailability rather than by link congestion.

We implemented GWA TCP in the *ns-2* simulator[1], developed at UC Berkeley within the VINT project[2]. We used the available *ns-2* modules wherever possible. The GWA-TCP model for the wireless environment was obtained by modifying the TCP-Reno implementation already available in *ns-2* . We introduce the ABS field in the IPv6 header and make routers to update the ABS field if needed. Two types of GWA-capable routers are implemented in *ns-2* . One is per-flow Round Robin (RR) queuing discipline, and the other has a single queue with first-come first served (FCFS) queuing discipline. For routers with per-flow queue, the available buffer for a flow $i$ is computed by

$$B_i = \frac{B}{n} - q_i$$

where $B$ is the total buffer size, $n$ the number of active flows, and $q_i$ the queue length for the $i$th flow. For routers with a single queue, the available buffer for a flow $i$ is obtained by

$$B_i = \frac{B - q}{n}$$

[1] *ns-2* is the last version of the simulator; all necessary information and the software can be retrieved at the URL `http://www-mash.cs.berkeley.edu/ns/ns.html`

[2] See the URL `http://netweb.usc.edu/vint` for additional information on VINT

where $q$ is the total queue length. We call the schemes using GWA-TCP with per-flow queue and with single queue routers GWA-RR and GWA-FCFS, respectively. The simulation results obtained from the two schemes along with the other three schemes described in the previous section are presented in the next section.

# 4 Experimental Results

Four different TCP implementations and router behavior are considered for comparison purposes; namely:

- **DropTail**: standard TCP-Reno with routers that discard packets only when buffers are full;
- **RED**: standard TCP-reno with RED [7];
- **ECN**: ECN enabled TCP and RED routers [8];
- **GWA**: the proposal in this paper, either with Round Robin (RR) or with First Come First Served (FCFS) queueing discipline.

In our experiment, we set the low and high thresholds to 1/4 and 3/4 of the router's buffer size, respectively, for RED and ECN routers. First, we consider the experiment shown in Fig. 1. We use this very simple setup to illustrate phenomena that occur in current cellular networks and which can affect the performance of TCP between fixed and mobile hosts. A single TCP source S is connected to a router/base station, which in turn forwards data to and receives ACKs from the mobile destination R. The receiver's window size is assumed to be 32 Kbytes and the buffer size at the base station is 32 Kbytes as well. This configuration ensures that the router buffer never overflows; however, when a RED router is used packets can still be dropped (as a congestion prevention measure). We assume that the source is engaged in a large ftp file transfer. We focus on handovers, which occur when the mobile R moves from one cell to the next (see Fig. 1). For simplicity, we model the handovers as periods during which the communication channel is unavailable. More precisely, if Mobile IP is used, R registers with the new BS (Foreign Agent), which in turn reports to a Home Agent (not shown in Fig. 1). The HA then establishes a tunnel to the FA. Only at this point, packets from source S can be redirected to the new cell. More efficient handover schemes than the one just described can be used. For instance, schemes which can anticipate the transition from one cell to the next and thus redirect traffic accordingly.

In general, however, no matter how efficient the handover scheme is, it is realistic to assume that the channel is unavailable for some interval during handover. When the channel is unavailable (or down), both downlink and uplink communications are blocked and packets are lost.

In our experiments, we assume that a TCP connection is "handed over"from one cell to the next every 2 seconds. Albeit not realistic in present day cellular networks, this situation may occur in future UMTS systems with very small cell size (microcells). Consider for example a car speeding at 100 km/h along a high way covered by microcells with 50 m diameter. In any event, the short interval between handovers allows us to extract statistically meaningful results even with relatively short simulation runs. Moreover, the relative ranking

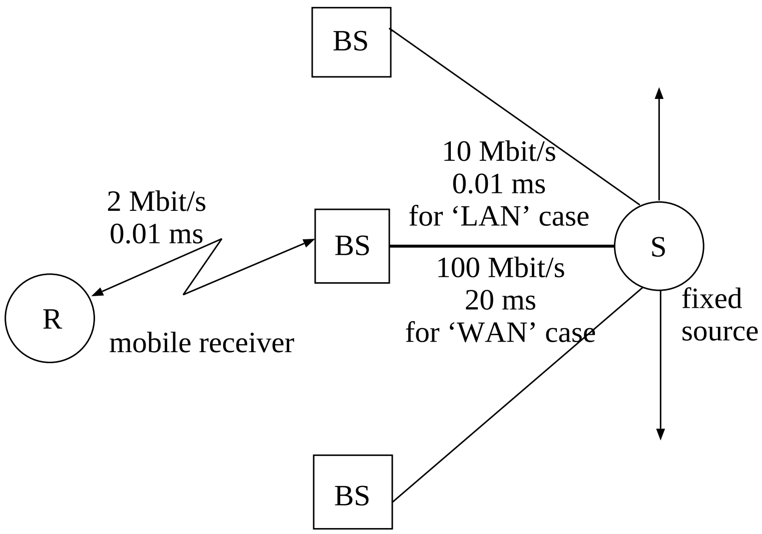


Figure 1: Experimental setup for the analysis of the impact of handovers on a single TCP connection

of the various schemes is not affected by the frequency of handovers. The link down periods are random variables uniformly distributed over $[0, 2\mu]$, where $\mu$ is the average value.

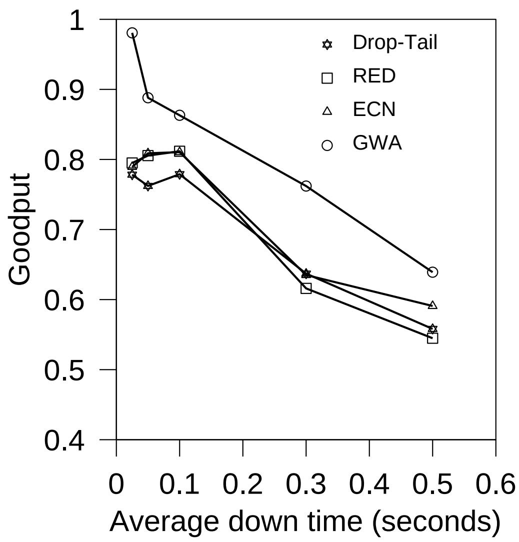


Figure 2: Goodput vs. average down time $\mu$ in a LAN environment

Fig. 2 reports the goodput versus average down time for the four cosidered implementations. Results are averaged over 100 seconds of simulation. GWA always yelds a higher goodput than the other implementations. The difference in performance increases as the average down time becomes smaller. The reason of this behaviour can be easily understood by examing the evolution in time of the transmitted packet number and of the congestion window.

Figs. 3 and 4 report the packet number (left colum) and the congestion window (right column) as a function of time for the four considered implementations for the leftmost ($\mu = 0.025\,\mathrm{s}$) and the next to the rightmost ($\mu = 0.3\,\mathrm{s}$) points of Fig. 2. The channel goes

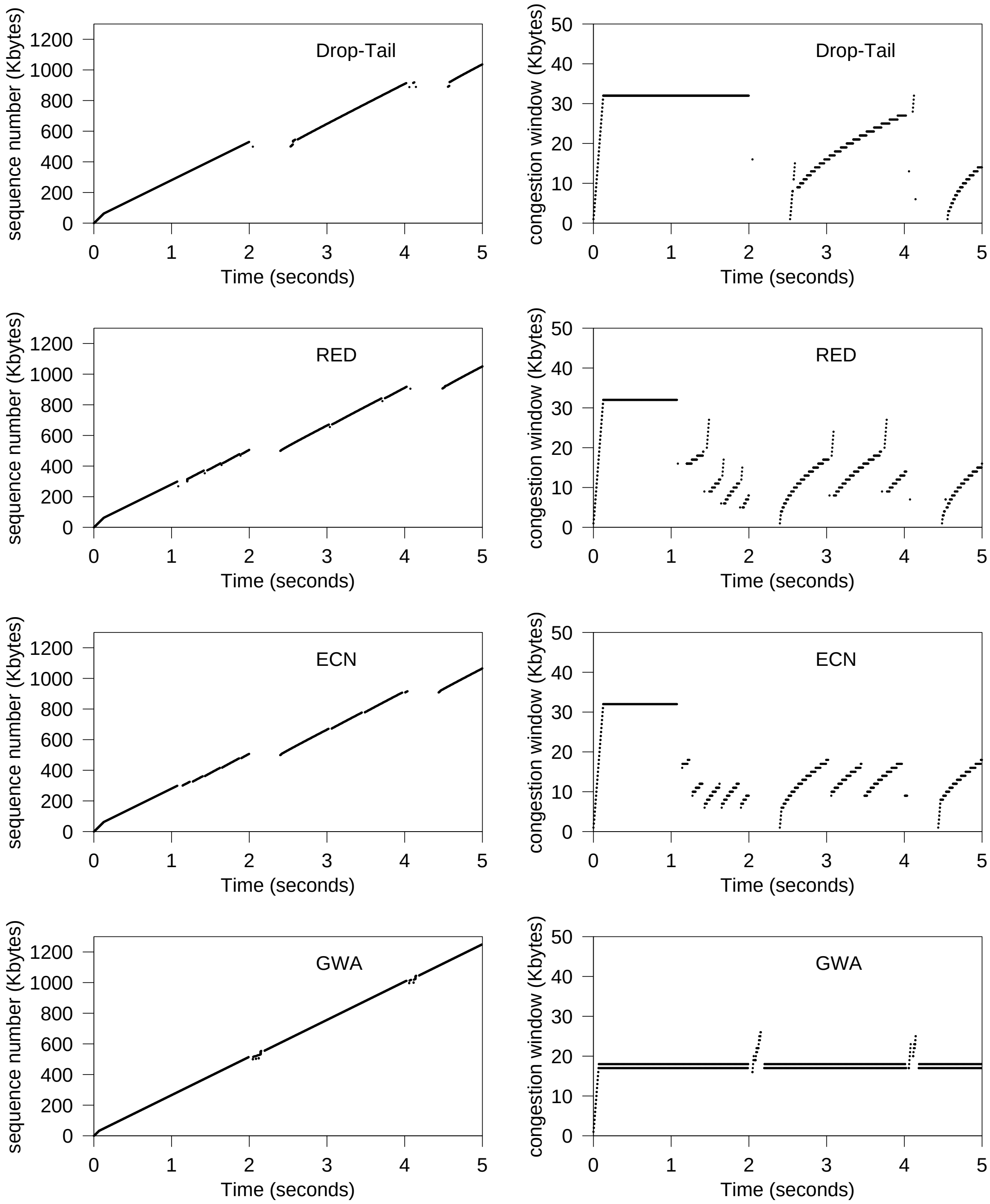


Figure 3: Packet number and congestion window vs time: $\mu = 0.025\,\mathrm{s}$

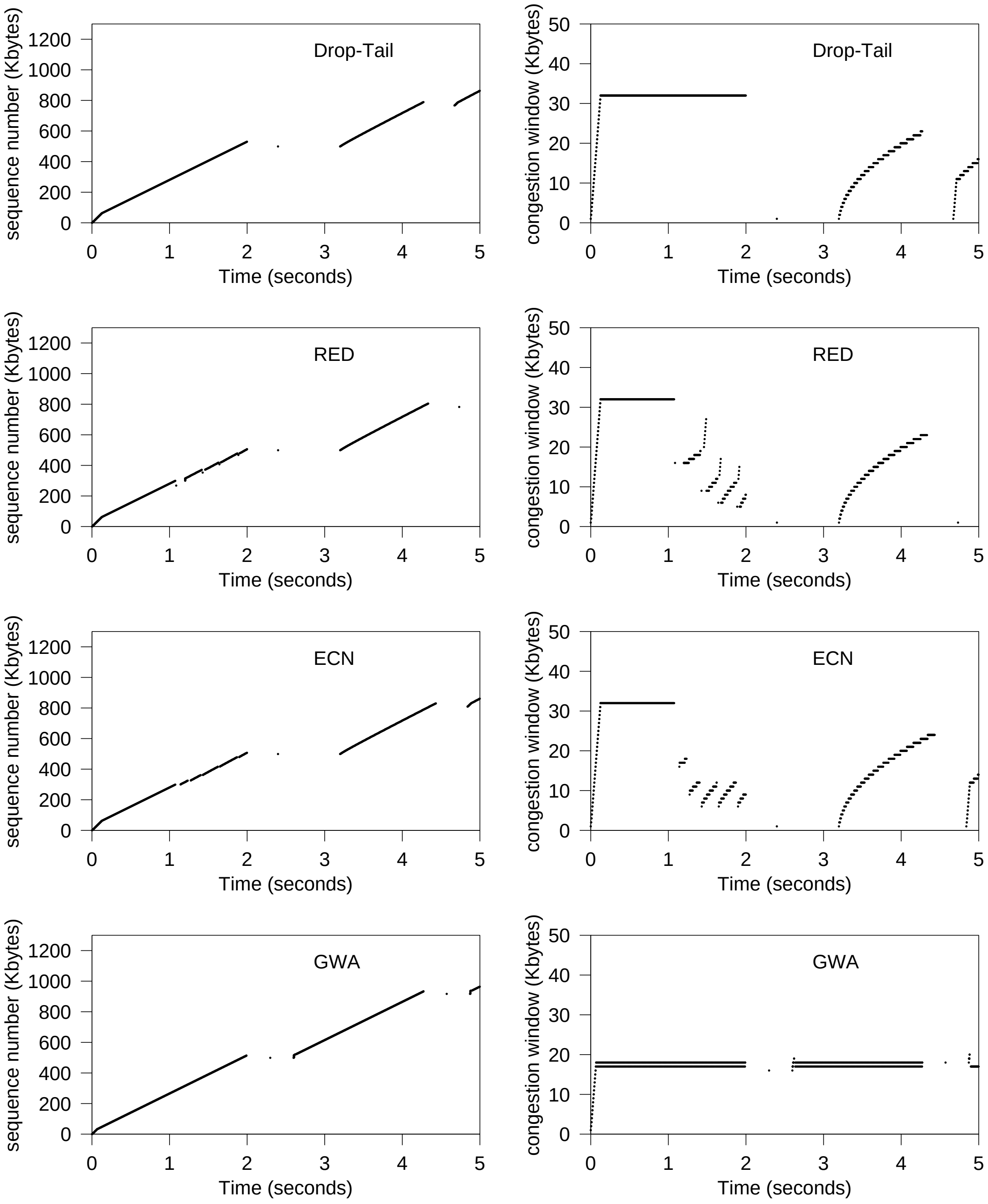


Figure 4: Packet number and congestion window vs time: $\mu = 0.3\,\mathrm{s}$

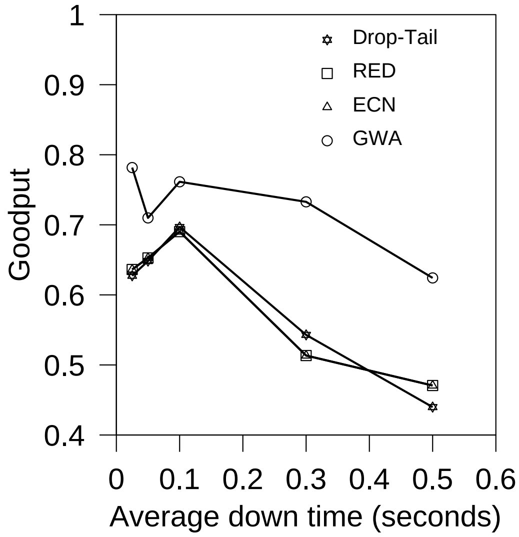


Figure 5: Goodput vs. average down time $\mu$ in a WAN environment

down at t= 2s and some time between t= 4s and t= 5s in both experiments. Of course, the down time is much larger in Fig. 4.

When the link down time is large ( in our case about 0.45s and 0.33s for first and second event, respectively, see Fig. 4 ), none of the implementations can avoid a timeout.This is because one whole window of data gets lost (the channel is unavailable for a period of time larger than the round trip time). In this case the gain obtained with GWA is due to two reasons: the slow start avoidance and the fixed timer for timeout (recall that in GWA the exponential backoff is disabled). Notice that we used a "fast tick" equal to 0.1 s, resulting in a minimum timeout of 0.3 s. Larger ticks, as are currently used in TCP implementations, would results in a larger goodput degradation for any implementation.

When the down time is shorter than the round trip time, several packets and/or ACKs are dropped, but generally not the entire window. Duplicate ACKs are received and may trigger a fast retransmit. The traditional fast recovery implementation, however, does not seem to be able to prevent a timeout (see the plots for DropTail, RED and ECN), since several packets are missing in a window. The *immediate recovery* technique used in GWA, on the other hand, allows for a practically seamless handover.

Fig. 5 reports the goodput when the round trip time is large, i.e., WAN environment ($>$ 40 ms). The receiver and the BS buffer are now 64 kbytes, to allow full link utilization. Again, GWA exhibits higher goodput as compared with traditional implementations. The gain in this case increases with the average down time. All implementations show a dip in performance when the channel is down for less than a round trip time (about .04s in this case). In this situation, fast retransmit is still possible. However, the multiple losses generally result in one or more timeouts, causing additional performance degradation. Due to the larger window size, also GWA is not immune from timeout.

The above experiments were done in a drastically simplified environment. They were tuned to show that the timeout and retransmission policy in TCP can heavily affect performance in mobile, wireless networks, even if the handover time is very small.

Let's now shift to a different scenario, and analyze the performance of handovers in the network shown in Fig. 6. There are 10 (almost) identical connections competing for resources

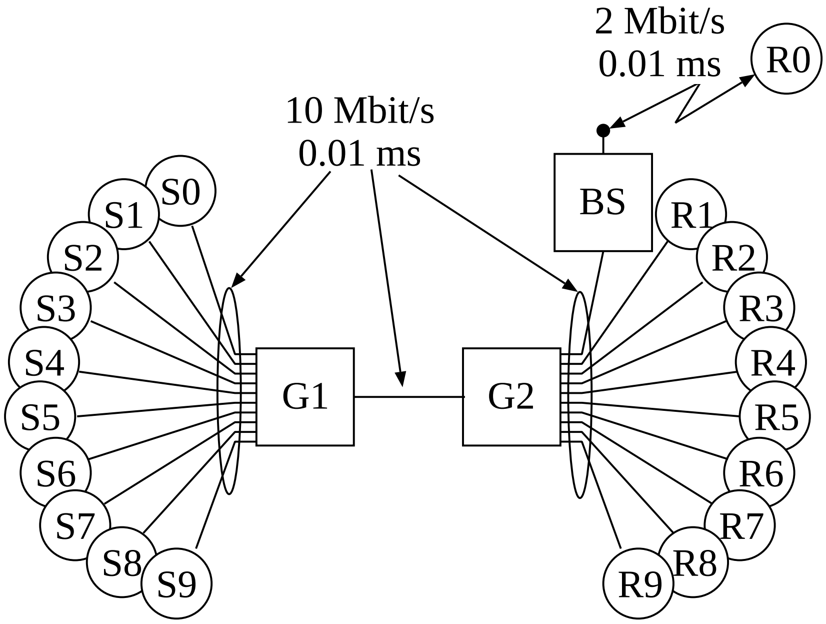


Figure 6: Experimental setup for the analysis of the impact of handovers on competing connections

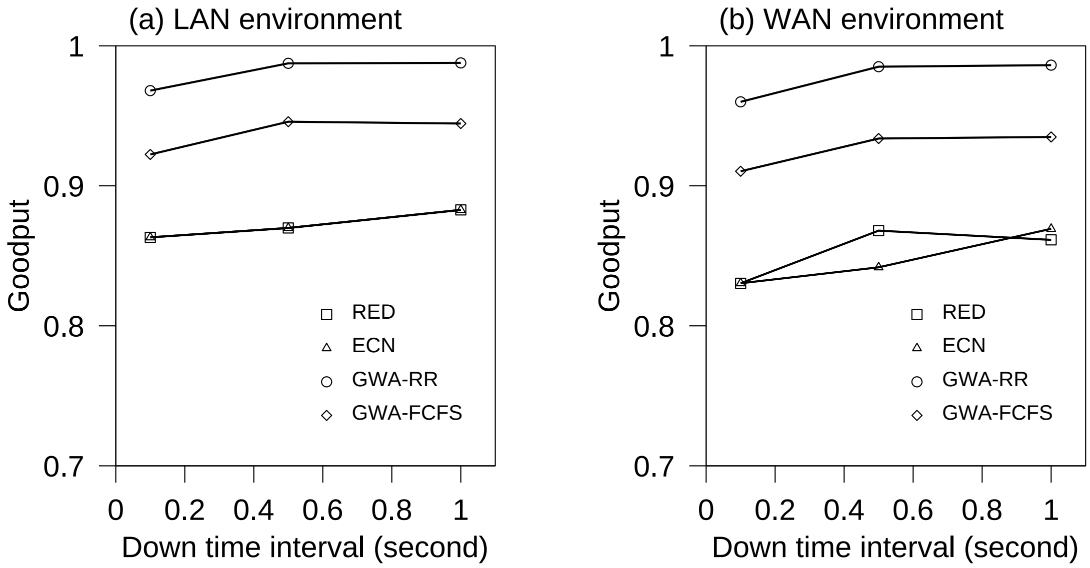


Figure 7: Goodput vs. $\mu$ in a WAN environment

on a bottleneck (the link between the two routers **G1** and **G2**). Out of 10 connections 9 are wired, the 10th is wireless and roaming. The network geography shown in Fig. 6 is tipical of a LAN. The routers G1 and G2 have buffer size of 64 Kbyte. The buffer size for the base station is 32 Kbyte. As a difference from the previous case, we use a fixed interval for handover time but random interval between handovers. More precisely, we set the average channel downtime to 1% and change both the duration $\mu$ and the frequency of the handovers. For instance a value $\mu = 0.5$ s means that the handover rate is 1 every 50 s.

Fig. 7(a) reports the normalized goodput obtained by the mobile host as a function of $\mu$ for the topology shown in Fig. 6. The goodput is normalized with respect to the fairshare at the bottleneck. Thus, the maximum achievable goodput is 0.99, since the channel is not available for 1% of the time. As anticipated, the goodput increases with $\mu$, since a larger $\mu$ means less handovers and hence less transients after the handover channel disruption. Fig. 7 reports only the results relative to RED, ECN, GWA-RR and GWA-FCFS. As for DropTail, the experiments yield a goodput smaller than 0.2, too small to report in Fig. 7(a). The reason for this poor behavior lies in the unfairness of TCP in DropTail routers that drop packets only when the buffer is full. While the wireless channel is down, the wired connections grab the resources and fill the buffer. When the wireless connection tries to recover after a handover, it finds the buffer full and tends to be "pushed out" of the network. Fig. 7(b) reports goodput for WAN environment, whose topology is very similar to Fig. 6 except that the link propagation delay is 5 ms between sources and G1, 10 ms between G1 and G2, 5 ms between G2 and receivers and the buffer size is 128 Kbyte at the congested link. The comparison between Fig. 7(a) and Fig. 7(b) indicate that the goodput decreases with the increase of the round trip delay.

Again, GWA yields the highest goodput. If RR routers are used and handovers are not too frequent, the goodput is larger than 0.98, practically full link utilization during up time. GWA-FCFS suffer a phenomenon similar to the one described for DropTail, ie, a recovering connection may find no empty buffer; however, the fact that congestion is avoided and GWA does not go through slow start because of packet loss, allows for a fast re-balancing of the resources. RED and ECN schemes attain similar goodput, about 10 to 12% percent smaller than GWA-RR.

# 5 Conclusions

This paper has analyzed the behavior of different TCP implementations after the connectivity loss due to handovers in cellular networks. Traditional TCP implementations, including ECN, react to packet losses assuming they are due to congestion phenomena, hence reducing the load offered to the network. In a cellular environment, where handovers lead to the frequent loss of entire packet bursts, this behavior leads to poor performance and resource wastage.

Starting from the consideration above, we propose a TCP implementation, based on the Generalized Window Advertizing (GWA) scheme, that is suitable for wireless, cellular networks, decoupling the congestion control algorithm from the reaction to the loss of one or more packets. The proposed TCP implementation is backward compatible.

Simulation results obtained with *ns-2* show that the performance gain is remarkable, both in LAN and WAN scenarios. The proposed scheme yields a higher throughput than traditional TCP implementations in all the setup we considered, though it is not possible to report all the results here. Besides the increased throughput, the new scheme offers advantages in reducing the time the connection remains idle after the network connectivity has been restored.

# References

[1] R. Cáceres, L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments", *IEEE JSAC*, Vol. 13, No. 5, pp. 850-857, June 1995

[2] H. Balakrishnan, N. Padmanabhan, S. Seshan, R. H. Katz, "A Comparison of Mechanism for Improving TCP Performance over Wireless Links", *IEEE/ACM Transaction on Networking*, Dec. 1997.

[3] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, L. Wang "TCP over Wireless Multihop Protocols: Simulation and Experiments", To appear in Proceedings of *IEEE ICC'99*, Vancouver, Canada, Jun. 1999.

[4] M. Gerla, K Tang, R. Bagrodia, "TCP Performance in Wireless Multihop Networks", To appear in Proceedings of *IEEE WMCSA'99*, New Orleans, LA, Feb. 1999.

[5] S. Mascolo, "Smith's Predictor for Congestion Control in TCP Internet Protocol", To appear in Proceedings of *American Control Conference 1999*.

[6] M. Gerla, R. Lo Cigno, S. Mascolo, W. Weng, "Generalized Window Advertising for TCP Congestion Control", CSD-TR 990012, UCLA, CA, USA, Feb. 1999.

[7] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transaction on Networking*, Vol. 1, No. 4, pp. 397–413, Aug. 1993.

[8] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, V. 24 N. 5, pp. 10-23, Oct. 1994.

[9] K. K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, IETF, Jan. 1999.

[10] R. Stevens, "TCP/IP Illustrated", Vols. 1 & 2 Addison Wesley, 1994.

[11] J. Postel, "Transmission Control Protocol – DARPA Internet Program Protocol Specification", RFC 793, DARPA, Sept. 1981.

[12] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, IETF, Jan. 1997.