TCP ACK Division Revisited

Andrés Arcia-Moret, Nicolas Montavont, Jean-Marie Bonnin, David Ros

***Abstract*— In TCP, receivers usually delay the emission of acknowledgements (ACK) packets for efficiency purposes (e.g., alleviate the processors charge or piggyback information in telnet connections). However, just as a TCP receiver may send less than one ACK per incoming data packet, it might also send more than one ACK per data packet without breaking the fundamental ACK semantics. In this article we investigate the impact of systematically increasing the ACK frequency and we discuss the uses and misuses of the technique. Interestingly, even when the ACK division seems applicable to unfairly gain bandwidth, results are not straightforward. There are several considerations that limit the impact of ACK division, such as the interactions with link layer protocols, the inners of TCP, the background traffic and the TCP congestion control algorithms.**

***Keywords*—ACK division, congestion control, TCP, cwnd.**

I. Introducción

Nowadays, the Transmission Control Protocol (TCP) is an omnipresent data transport protocol. Indeed, TCP controls about the 90% of the 200000 terabytes that flows through the Internet every second. As such, TCP must face a large heterogeneity of data networks, including challenging networks such as sensor networks, vehicular network or satellite networks in which TCP ACK frequency does not fit well [1]. However, as the legacy TCP specification states [2] the frequency for ACK packets is most of the time constant and independent from the network conditions. This means that the so-called ACK clocking — the one that maintains a fair rate — was thought and still continues to be a trans- port layer conception.

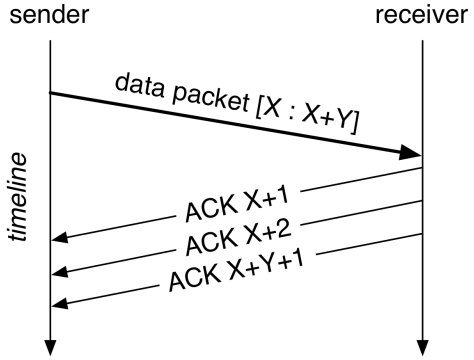

FIG. 1. ACK Division (divack) technique.

Artículo recibido el 18 de Diciembre de 2009.

A.A-M. está con la Universidad de Los Andes, Sector La Hechicera, Facultad de Ingeniería, Escuela de Ingeniería Eléctrica, Mérida, Estado Mérida, Venezuela, Tlf. +58-274-2402990, Fax: +58-274-2402890, E-mail: amoret@ula.ve, N.M., J-M.B, D.R.. está con TELECOM Bretagne, 2 rue de la châtaigneraie, Cesson Sévigné. 35576 CEDEX. France. Tlf: +33-299127045, E-mail: nicolas@montavont.net, jm.bonnin@telecom-bretagne.eu, david.ros@telecom-bretagne.eu.

In TCP, regular receivers usually delay the emission of acknowledgements (ACK) packets for efficiency purposes. However, just as TCP receiver may send less than one ACK per incoming data packet, it might also send more than one ACK per segment without breaking the fundamental ACK semantics. As shown in Fig. 1, a data packet transporting bytes X to X + Y (say an MSS segment) can be acknowledged in smaller parts by several ACKs, namely ACK X + 1, ACK X + 2 and the full-ACK X + Y + 1.

Savage et al. [3] studied first the ACK division phenomenon (divacks for short), that consists in sending a massive number of ACKs per segment. Eventually, misbehaving receivers may attain an unfair share of the available bandwidth through the sending of divacks. On the other hand, using divacks has also been proposed to improve TCPs reaction when it faces abrupt bandwidth changes in the last hop of a wired-cum-wireless network [4]. In both cases, the idea is to take advantage of the increased TCP Congestion Window (cwnd) growth rate, and thus the increased throughput.

Different from previous work, in this paper we present an extended evaluation of divacks and so, we discuss some limits in more realistic scenarios. As already considered in early contexts, divacks may block the regular TCP ACK clocking [5]. On the other hand, an aggressive TCP connection has to be lucky enough to pass its packets through the bottleneck without suffering a loss in performance. However, as we will see in Section IV-B, in such case ACK division (used by a misbehaving client) can be refrained by using a byte counting technique that allows a uniform increase of the cwnd.

We have also remarked that, as a design premise, legacy TCP does count the received segments to open the congestion window [6]. Recently, RFC 5681 [7] suggests the prevalent use of byte counting to open the cwnd in slow start, and optionally to open in congestion avoidance. However, there have been reports in communities of developers suggesting that byte counting strategies harm the performance of short TCP connections [8], [9]. Therefore, our interest on studying both benefits and shortcomings of using ACK division as a technique when the sender counts packets.

This paper is organized as follows. In Section II we survey the divacks applications when considered as a mechanism to improve performance. In Section III we present different shortcomings that divacks may face in different deployment scenarios. In Section IV we evaluate ACK division facing congestion, moreover we evaluate a byte counting technique to regulate the ACK clocking. Finally, in Section V we conclude the paper.

II. Uses of TCP ACK division in Wireless Networks

Several researchers have proposed taking advantage of divacks to improve TCP performance in wireless links, without

having to modify the TCP sender. In such proposals, divacks are produced in a controlled manner to help the sender in speedily recovering the value of cwnd after it has been unduly reduced.

Jin et al. [10] propose a method for coping with decreases in cwnd due to wireless random losses, in a wired-cum-wireless network configuration. They assume a TCP sender on a fixed host in the wired part of the network, and a TCP receiver on a host in the wireless part. The base station (BS) keeps a timer that tries to parallel the TCP retransmission timeout (RTO) of the TCP sender; when there is a loss and the timer expires, the BS infers that the sender at the fixed host has entered the slow start phase. The BS then sends a fixed number of divacks to the fixed host when the retransmitted data packet arrives. By means of simulations, the authors show how cwnd rapidly recovers its size prior to the loss, thereby improving TCP performance.

Matsushita et al. [11] use divacks to quickly adapt the cwnd size after an upward vertical handover in a heterogeneous wireless networks composed of a WAN service such as a 3G cellular network and IEEE 802.11g-based wireless Local Area Network (LAN). They assume that the mobile node is able to detect the available bandwidth in the new access network with a higher bandwidth-delay product (BDP). Once the node enters into the new network, it sends divacks in order to rapidly increase cwnd in congestion avoidance mode.

Hasegawa et al. [12] developed a receiver-oriented, end-to-end solution to recover from undue cwnd decreases caused by wireless random losses, in a wired-cum-wireless network with an asymmetric full-duplex Universal Mobile Telecommunications System (UMTS) access link. Their proposal considers three main aspects: a mechanism for differentiating between wireless losses and congestion losses, another for controlling the duration of the divack-generation interval and a final one for controlling the divack sending rate. Ideally, divacks would only be sent when wireless losses take place. To control the duration of the ACK-division period, they keep at the mobile node an estimate of the senders cwnd size. So, when a random loss is detected, they send as many divacks as needed to recover the cwnd achieved by the sender just before the loss.

Besides the research proposals described before, we are aware of one implementation of TCP ACK division in a commercial product. Cisco routers implement a so-called Rate Based Satellite Control Protocol (RBSCP) [13], which was designed as a solution for improving the performance of transport protocols in wireless links with high bit-error rates, like satellite links

III. DIVACKS SHORTCOMINGS

In this section we discuss particular shortcomings that can clearly limit the performance induced by divacks both in link and transport layer.

A. Interactions with Shared Media

Nowadays, most Internet Mobile terminals and Cell Phones are equipped with 802.11 interfaces. A typical mobile user uses TCP-based applications to access emails, browse the web or download files from the Internet. Thus the selfish interest for a user to have a better performing version of TCP.

The sharing of the transmission media represents a particular characteristic of IEEE 802.11 networks that regulates the effect of divacks. Thus, the access to the media is shared for the data packets and ACKs. Our intention is to illustrate the impact of the MAC layer on the divacks performance.

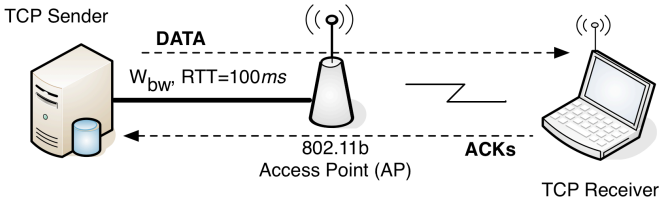

FIG. 2. Wired-cum-wireless Topology.

We illustrate how the variation of the bottleneck on a data access scenario, can determine the success or failure of the divack technique.

Using a medium access control technique such as distributed coordination function (DCF) in 802.11, divacks and data packets have equal media access opportunities due to DCF contention nature [14]. We show that the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism with the random exponential back-off interferes with the expected performance of massive sending of divacks. This observation is valid as long as the bottleneck of the network remains in the wireless part. Thus, when the bottleneck is in the wired part of a wired-cum-wireless network, data packets are delayed and divacks gain access more frequently during the period of time in which the APs buffer is empty.

We implemented and tested the divacks mechanism in the topology shown in Fig. 2. The experiment consisted on progressively decreasing the bandwidth at the wired part of the network while observing the impact on the cwnd dynamics. We used the default settings for the ns-2 802.11b access. That is, a data rate of 11 Mbps and a basic rate (for control frames) of 2 Mbps.

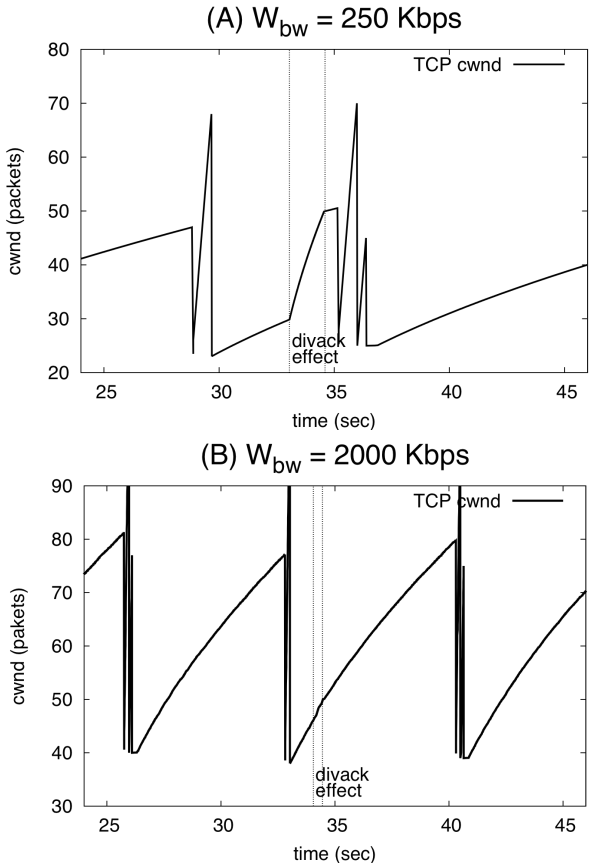

FIG. 3. Effect of divacks in different wired bottlenecks.

In the experiments results shown in Fig. 3 we sent 10 divacks per data packet for 80 in-order data packets (i.e., 800 divacks in total) during the congestion avoidance phase at the time indicated between the vertical bars. Although we are conscious that there is an important impact on the slow start phase, we know that for long transfers, the congestion avoidance phase is more important and thus our interest.

We tested for various rates of divacks per data packet, but we show only the relevant ones to illustrate our findings. Contrary to the common wisdom we found that a misbehaving receiver noticeably benefits from divacks only if the bottleneck is located at the wired part of the network, which does not represent the commonly found deployment scenario. Fig. 3.A shows how more transmission opportunities are given to the TCP receiver as long as the bandwidth of the wired part is smaller than the 802.11b access bandwidth, i.e., for W_{bw} = 250Kbps. On the other hand, in Fig. 3.B, when the wired bandwidth is increased to W_{bw} = 2000 Kbps, the effect on the cwnd is unnoticeable.

B. *Interactions with Nagle Algorithm*

The Nagle algorithm [15] was proposed to avoid the massive sending of small-sized segments called tinygrams which caused problems in Wide Area Networks (WANs) at the time of bandwidth constraints.

The solution consists in just delaying the emission of a full-sized segment if less that a Maximum Segment Size is acknowledged by the receiver (say, by the use of divacks) or in the case of an interactive TCP connexion. The Nagle algorithm allows to send less TCP segments when receiving several divacks and to accelerate the data sending when receiving a proper number of ACKs [16].

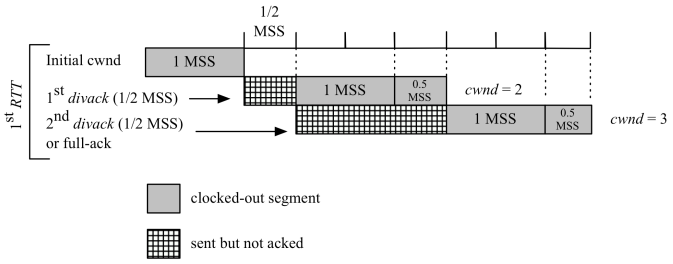

FIG. 4. Interactions between slow start and divack when Nagle algorithm is not active.

The Nagle algorithm, as it is used today, may eventually lead to a last-packet deadlock. That is, in the case in which the packetization of certain stream leads to a last packet size lower than 1 MSS (which may be the common case), in certain scenarios [17] the Nagle timer (that allows the sending of the last packet) accounts for the biggest component of the Rounds Trip Time (RTT). Thus, disabling the Nagle algorithm is convenient for certain scenarios. In such a case, if Nagle algorithm is disabled, the sender behaves as shown in Fig. 4 when facing divacks. In this particular example, the receiver acknowledges segments in two parts (i.e., two divacks per segment). As the sender is in slow start phase, according to the algorithm every received divack increases the cwnd by one MSS and the divack (half an ACK) slides the cwnd by half a segment which is also sent.

The behavior described before generates in the first RTT a total of 4 segments, two of which are full-sized segments and the other two half-sized. After few RTTs one can easily observe how the sender will start bursting small-sized segments and eventually, most of them will transport just one byte. Remark that, in this particular phase, slow start is the most convenient phase for accelerating the transfer, since in congestion avoidance most of the packets will be the small ones.

Although the clocking of segments is accelerated with divacks, the traffic in number of data packets per time unit is clearly increased and may eventually lead to more congestion (in terms of packets). Moreover, the recover of small-sized packets (with no constant size) will lead to unnecessary retransmissions, that is the retransmitted full-sized segments may contain bytes that have already arrived.

IV. ACK Division Facing Congestion

In this section we would like to answer to the question: To what extent divacks represent a problem when facing congestion?

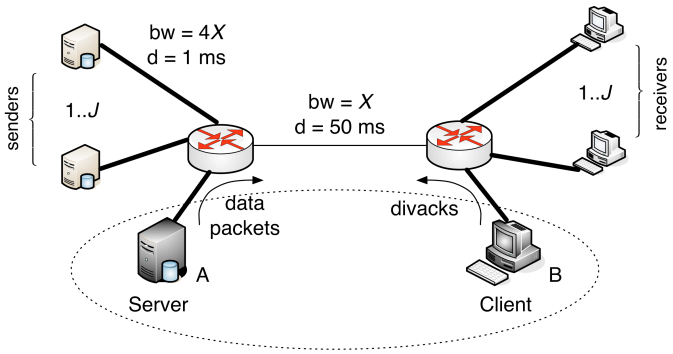

FIG. 5. Simulation topology for divack evaluation.

To tackle this question we use the simulation topology that is shown in Fig. 5. For the whole set of experiments, we consider a dumbbell topology. The bottleneck link has a 50-ms delay and a band- width of X Mb/s; simulations were run for X = 10 and X = 1.5, but since we found analogous tendencies for both values we will only show results for the former one. For the bottleneck buffers, we use droptail queues in packet mode, with buffer size equal to the bandwidth-delay product (computed assuming a packet size of 1500 bytes). Thus, we use an 83-packet queue for a 10 Mbps bandwidth and a 13-packet queue for a 1.5 Mbps link. Access links are of 1-ms delay and the bandwidth for each of them corresponds to four times the bottleneck bandwidth X.

Remark that all flows have the same RTT. The rationale behind this (usually unrealistic) setting is that we want to avoid any bias due to RTT unfairness; performance gains, if any, would be solely due to the use of divacks.

In every scenario, there is a single receiver doing ACK division (the "client" in Fig. 5), and a number of background TCP flows which use standard delayed ACKs. There are only J back- ground flows in the forward direction (i.e., data packets are traveling in the direction A → B only), so there is no reverse data traffic. In a given scenario (i.e., varying the congestion) the receiver doing divacks downloads a file after a long warm-up period (lasting 400s) for the background traffic.

We tested various ACK division policies from the "client"

that intended to obtain better transfer times. Our method, varying the aggressiveness of the transmission, consisted in sending divacks for every data packet (method I or, M-I for short) or sending divacks for every other data packet (method II or, M- II). Moreover, when sending divacks for M-I or M-II, we could send them selectively to the sender during the slow start or congestion avoidance. Combining all restrictions we obtain four possible policies: divss1 and divca1 for M-I and divss2 and divca2 for M-II.

For the experimental results, we used as a reference the two standard versions of TCP: 1 ACK per data packet (1-apdp) and Delayed ACKs. Note that 1-apdp performed better than Delayed ACKs in 100% of the cases. That is, the time to transfer a file (for all the tested sizes) was always lower for 1-apdp than for Delayed ACKs. Thus, we adopt them as the reference for total transfer time for the evaluated divack policies.

A. *Bandwidth Stealing*

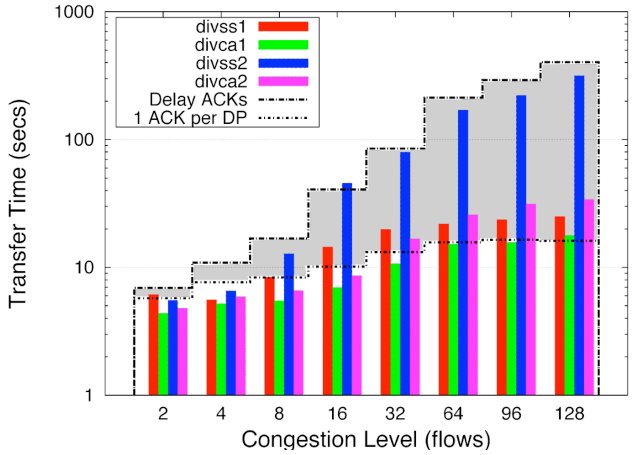

FIG. 6. Transfer time for large files (1.5 MB) without ABC.

In Fig. 6, we show how a client may improve its performance by using divack when sending 16 divacks per data packet considering all policies. Under conditions of low congestion ($J \leq 32$) the client is able to improve the transfer time by using divacks in any of the two methods. The divacks percolate easily through congestion and the induced aggressiveness on the sender results in no significant harm for the divack flow. How- ever, note that the from J = 64 on, transfer times fairly approximates or enter into the shadowed region which means that transfer time does not improve nor damage the performance of the flow.

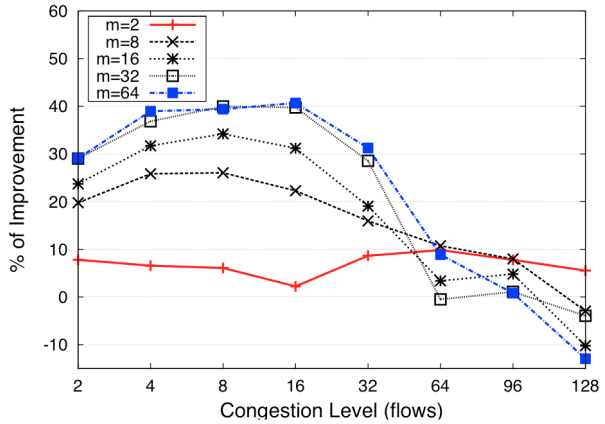

FIG. 7. Improvements in thorughput for different frequencies.

In Fig. 7 we extend our analysis for various rates of divacks per data packet. In this specific case, we took divss1 for transfers of 1.5 MB and we varied the number of divacks sent, i.e., we sent m divacks for every incoming data packet in a range from 2 to 64. The figure shows a fundamental result for divacks: it is possible to obtain better transfer times when increasing the number of divacks. However, note that this improvement is effective as long as the congestion is characterized by $J \leq 32$ flows. Moreover, there is a limit in the effective number of divacks to get the desired effect. Observe that m = 32 and m = 64 give similar performance.

B. *Regulating the ACK clocking*

As previously shown in Section IV-A, the use of divacks can lead to an unfair use of the bandwidth by a misbehaving receiver. That is, by sending divacks a receiver can force a sender to increase the throughput even through a congested bottleneck. As previously discussed, one way of refraining the sender to increase the throughput by the use of divacks is to count bytes instead of packets. Byte counting techniques were specified by Allman in [18], [6]. We tested the Appropriate Byte Counting (ABC) technique at the sender side to regulate the ACK clocking.

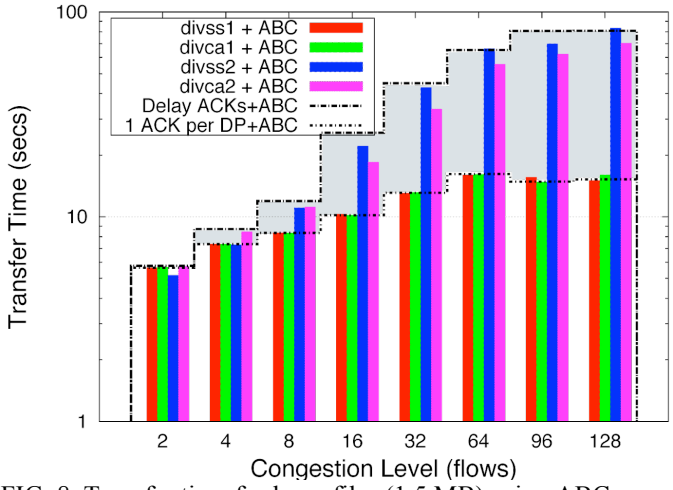

FIG. 8. Transfer time for large files (1.5 MB) using ABC.

As show in Fig. 8, with ABC the sender is able to proportion- ally increase the cwnd to the total acknowledged bytes. Note how for both methods, the performance of a single transfer approximates the base frequency. That is, div{ss,ca}1 behaves as 1-apdp and div{ss,ca}2 behave as Delayed ACKs. The difference in performance of both approximations is due to the in- creased burstiness produced by the delayed divacks being compensated by ABC.

Note that since ABC counts the total acknowledged bytes, a burst of divacks (acknowledging a single packet) is likely to produce a similar effect when in presence of network asymmetry [19]. In other words, in order to slide the cwnd by 1 segment, m divacks have to arrive at the sender as if the full-ACK lasts (at least, not accounting congestion) m times in the return path.

V. CONCLUSIONS

In this article we have analyzed in a broad sense the impact of divacks in data transmission. We have found that, as shown by [3] and suggested by [20], divacks accelerates the ACK clocking improving the performance if a transfer concentrates mostly on the appropriate TCP phase. That is, for long transfers we observe

that sending divacks in congestion avoidance noticeable improves the performance of the transfer. Interestingly, the burstiness induced by divacks does not seem to harm the misbehaving client when congestion increases.

We have also noticed that to make divacks attractive, there should be large buffers at the bottleneck (to deal with burstiness) and the receiver should be aware of the current senders phase. Both are well known issues in transport protocols, and so they require special attention.

Although divacks helps in wireless access networks (e.g., in vertical handovers or in recovering from wireless losses), it can be a harm in wired networks as a TCP client can obtain an unfair share of the bottleneck. This contrasting application of divacks can be easily corrected with Appropriate Byte Counting as it effectively regulates the ACK clocking.

Through the different scenarios we have seen that divacks accelerate the data transmission. However, an unduly acceleration on the ACK clocking may lead to a decrease in TCP performance. We believe that an anticipated congestion control, such as the determination the available bandwidth, may help in a appropriate use of divacks in special cases.

REFERENCES

[1] S. Floyd, A. Arcia, D. Ros, and J.R. Iyengar. Adding Acknowledgement Congestion Control to TCP. RFC 5690, 2010.

[2] J. Postel (ed.). Transmission Control Protocol. Internet Standards Track RFC 793, IETF, September 1981.

[3] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. 29(5), October 1999.

[4] M. Nakata. Receiver-based ACK splitting mechanism for TCP over wired/wireless heterogeneous networks. Master's thesis, Graduate School of Information Science and Technology, Osaka University, Japan, 2006.

[5] A.Arcia, D.Ros, and N.Montavont. Auto-protection of 802.11 networks from TCP ACK division. In CONEXT '08: Proceedings of the 2008 ACM CoNEXT Conference, pages 1–2, Madrid, Spain, 2008. ACM.

[6] M. Allman. TCP byte counting refinements. 29(3), June1999.

[7] M.Allman, V.Paxson, and E.Blanton. TCP Congestion Control. Internet Standards Track RFC 5681, IETF, 2009.

[8] A. Kuznetsov. Re: high latency with TCP connections. Email to the linux-netdev mailing list, Message ID: "http://marc.info/?l=linux-netdev&m=115706704719579&w=2", August 2006.

[9] A. Vodomerov. Re: high latency with TCP connections. Email to the linux-netdev mailing list, Message ID: "http://marc.info/?t=115693258400001 &r=1&w=2", August 2006.

[10] K. Jin, K. Kim, and J. Lee. SPACK: Rapid recovery of the TCP per- formance using split-ACK in mobile communication environments. In Proceedings of IEEE TENCON, pages 761–764, Cheju, South Korea, September 1999.

[11] Y. Matsushita, T. Matsuda, and M. Yamamoto. TCP congestion control with ACK-pacing for vertical handover. In Proceedings of IEEE WCNC, pages 1497–1502, New Orleans, March 2005.

[12] G. Hasegawa, M. Nakata, and H. Nakano. Receiver-based ACK splitting mechanism for TCP over wired/wireless heterogeneous networks. IEICE Transactions on Communications, E90-B(5):1132–1141, May 2007.

[13] Cisco Systems. Cisco IOS Release 12.3(7)T, New Feature Documentation—Rate Based Satellite Control Protocol.

[14] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. IEEE Journal on Selected Areas in Communications, 18(3):535–547, 2000.

[15] J.Nagle. Congestion control in IP/TCP internetworks. Internet Standards Track RFC 896, IETF, January 1984.

[16] W. Richard Stevens. TCP/IP illustrated, volume I. Addison-Wesley Pub. Co., Reading, Mass., 1996.

[17] S. Cheshire. TCP performance problems caused by interaction between nagle's algorithm and delayed ack,?

[18] M. Allman. On the generation and use of TCP acknowledgements. ACM Computer Communications Review, 1998.

[19] H. Balakrishnan and V. N. Padmanabhan. How network asymmetry affects TCP. pages 60–67, April 2001.

[20] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). Experimental RFC 3465, IETF, February 2003