Loss Differentiation and Recovery in TCP over Wireless Wide-Area Networks

Detlef Bosau
detlef.bosau@web.de

Herwig Unger, Lada-On Lertsuwanakul
Fernuni Hagen
{herwig.unger,lada-on.lertsuwanakul}@fernuni-hagen.de

Dominik Kaspar
Simula Research Laboratory, Norway
kaspar@simula.no

***Abstract*—The increasing speed and coverage of wireless wide-area networks (WWAN) has made technologies such as GPRS, UMTS, or HSDPA a popular way to access the Internet for both mobile and stationary users. However, depending on the scenario, WWAN can suffer from severe IP packet loss due to corruption, which is mistaken by TCP as an indication of path congestion. This paper presents an approach to solve the *loss differentiation problem* for the widespread scenario of wireless networks being used as access networks to the Internet. Our solution allows TCP to distinguish between congestion and corruption loss and to properly react to both phenomena. Loss differentiation is achieved by placing an assisting agent on the WWAN's base station, which replies a TCP sender in the wired Internet with a new type of acknowledgement. Without harming TCP's end-to-end semantics, these acknowledgements provide feedback about congestion on the wired path and corruption on the wireless path and support the sender in taking remedial action. Results from simulations indicate that our proposed corruption recovery algorithm significantly improves the TCP goodput. In addition, excessive RTO growth and pauses in the TCP flow that result from repeated packet corruption are considerably reduced.**

I. Introduction

Accessing the Internet and Internet-based services over wireless wide-area networks (WWAN) such as GPRS, UMTS, and HSDPA has become part of our everyday life. A typical scenario is given in Figure 1, which illustrates a mobile host (MH) attached to a wireless access network and communicating with a fixed host (FH) in the Internet. The wired Internet and the wireless access network are interconnected by a base station (BS).

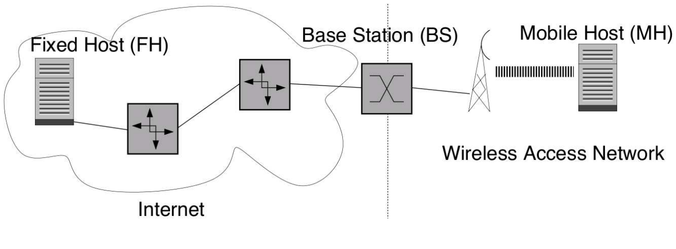

Fig. 1. Internet and wireless access network.

Due to the lossy nature of wireless links, TCP connections between FH and MH are faced with several difficulties.

- The *loss differentiation problem*: TCP implicitly assumes corruption-free links along the entire path, therefore packet corruption is mistaken as an indication of path congestion. As a consequence, TCP may not be able to fully utilize its fair share of the available path capacity.
- The proper *retransmission timeout (RTO) estimation* is known to be difficult [1][2]. The main challenge is caused by TCP's assumption of quasi-stationary round-trip times (RTT) [3][4]. A too large RTO causes inefficient usage of the available capacity, while a too small RTO may cause unnecessary retransmissions and congestion handling. The latter problem is also known as the "spurious timeout problem". Existing algorithms, such as TCP Eifel [5], detect spurious timeouts to correct an erroneously reduced congestion window, but they do not resolve corruption loss.
- TCP employs a Go-Back-N strategy for loss recovery, which is unsuitable when a packet requires more than one transmission attempt for successful delivery. TCP assumes corruption-free links, thus a packet retransmission is expected to be successful when a (perhaps erroneously!) recognized congestion is overcome. Further transmission attempts lead to increasing timeouts and unwanted pauses (refer to Section II-B for more details).

Particularly, the assumption of corruption-free links and quasi-stationary RTT is severely violated in WWANs. Depending on the actual scenario, packet corruption rates in WWANs can be arbitrarily close to 1. In addition, the service time for a packet on a WWAN link, and therefore the average RTT, may vary for several reasons, including varying cell load in cellular networks and therefore varying MAC delays, and varying channel or line coding if the network technology is able to adapt to different signal-to-noise ratios. As a motivating example, Figure 2 illustrates extreme RTT values (431 seconds at the maximum!) measured over HSDPA on a moving train.

In this paper, we decouple congestion control and pacing from reliable delivery by introducing an *assisting agent* on the base station, the "CLACK agent". This agent addresses the problems of loss differentiation and proper RTO estimation by providing the FH with a new type of acknowledgment. Hence, proper loss differentiation and clocking is achieved without harming the TCP connection's end-to-end semantics by ACK spoofing, connection splitting or introducing hard state into the network. The proposed CLACK agent can be considered as a "flow middlebox" in the sense of [6].

In addition, we propose a loss recovery strategy suitable for lossy networks, in which any packet corruption is signaled to

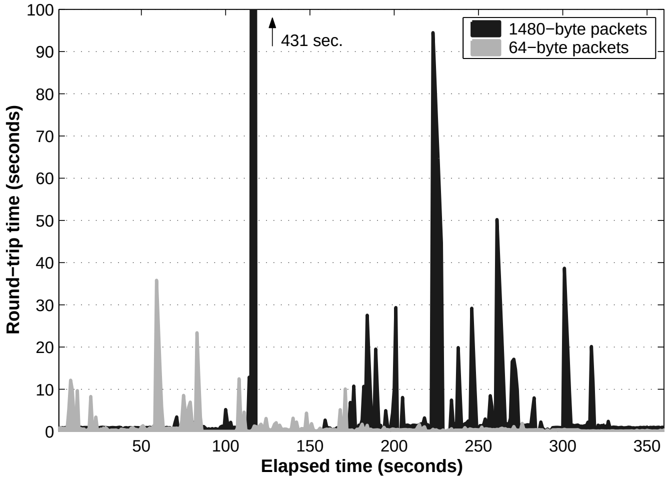

Fig. 2. Extreme round-trip times over an HSDPA access network during a train trip through the Norwegian countryside. The two experiments of small and large packet sizes were conducted at different times and locations.

the communication endpoints as soon as possible. Therefore, our mechanism provides a fast end-to-end retransmission of corrupted packets *before* the expiration of the RTO timer. In other words, our mechanism complements the fast retransmit of packets lost due to congestion by a fast retransmit of packets lost due to corruption.

The remainder of this paper continues with a discussion of related work in Section II. In Section III, a description of our CLACK approach follows. Section IV describes the details of our system model and our simulation environment. Simulation results are presented in Section V. Finally, Section VI concludes the paper and outlines our future research activities.

II. Related Work

The last two decades have seen a vast amount of related work in the area of wireless access to the Internet. Thus, only a selection of relevant papers can be addressed here for space limitations. For a more complete overview, the reader is referred to the work by Sonia Fahmy [7].

A. Local Recovery

In WWANs, the corruption probability for a given packet can be arbitrarily close to 1 and does not primarily depend on the technology in use but on the actual scenario. Packet corruption cannot be completely overcome by local recovery approaches, such as Snoop [8][9][10], which uses buffering and retransmission of lost packets at BS. Local recovery can increase the probability for a packet to be correctly delivered at the expense of increased delivery times. However, analogous to the “Two Army Problem”, it is impossible to *reliably* deliver a packet over a lossy channel in *limited time*, or in a limited number of sending attempts.

A fundamental problem with all kinds of local recovery approaches and performance enhancing proxies (PEP)[11] is the eventual deletion of transient data from buffers and queues. Particularly in case of link outages and high corruption rates, packets buffered in some local recovery mechanism or PEP may stay forever and lead to unlimited head-of-line blocking, if they are not forcefully deleted (e.g., by a reasonably limited persistence). In this paper, we take the position that packet corruption should be handled by the connection endpoints and only limited effort should be spent on local recovery [12].

B. End–to–End Recovery

A second problem of lossy channels in TCP is the reliable end-to-end delivery of data when a lost packet cannot be recovered by a single retransmission. If a corrupted packet needs more than one transmission attempt for successful delivery, the individual retransmissions are started by retransmission timeouts, even if packet loss detection using duplicate and selective ACKs is employed. Because TCP employs cumulative ACKs and a Go–Back–N strategy for loss recovery, some of the yet unacknowledged data may have been successfully received and hence causes DupACKs during the retransmission process. Thus, DupACKs are not suitable for loss detection during the retransmission process. Refer to [13], Section 3.2 (“Fast Retransmit and Recovery”) for details. Due to the retransmission timeout (RTO) backoff, which is used to rapidly increase the RTO to a sufficiently large value in case of a too small or not yet existing RTT estimate [14] repeated timeouts may lead to unwanted and annoying RTO growth and therefore pauses in the TCP transmission process.

To our knowledge, the problem of end–to–end loss recovery has received only little attention in the past. Approaches published so far consider loss recovery in Delay Tolerant Networks, e.g., [15] which reduce the number of retransmissions by adding redundant information to the data stream but do not primarily target recovery of actual packet loss.

C. Loss Differentiation

For the loss differentiation problem, we can identify three main approaches.

1) Some methods try to identify the reason for a packet loss by statistical means, e.g., [16]. The basic idea is to monitor a path’s RTT and to infer from individual RTT observations, whether a “short time RTT average” exceeds a “long time RTT average”, which is taken as an indication of congestion. Therefore, packet loss is recognized as congestion loss, while in the opposite case, packet loss is seen as corruption. Besides the end-to-end RTT, there are other statistics proposed for the same purpose of loss differentiation, e.g., interarrival times, interarrival time jitter etc.

   All of these approaches attempt to do an “ex post reasoning” in order to identify the actual reason for an individual observation, which may be caused by different reasons. For example, an increasing RTT can be due to a noisy channel and increasing *recovery delays*, or to a crowded cell with increasing *MAC delays*. This kind of ex post reasoning is never a proper application of mathematical statistics, particularly it does not yield the *true* reason for an *individual* packet loss.

2) The *CETEN* approach by Eddy, Ostermann and Allman [17] does not focus on the individual packet loss but attempts to modify TCP's *Additive Increase Multiplicative Decrease* mechanism in order to accommodate the packet corruption rate along the path. As long as the packet corruption rate for the individual links is known, TCP is modified so that on average, the congestion window is halved when one congestion drop is observed. However, this approach is not feasible in a WWAN, where the actual packet corruption rate is unknown.[1]

3) There exist several flavors of rate-controlled TCP, which try to estimate the correct rate for a TCP flow and to accommodate accordingly. An example is TCP Westwood [18].

   The basic problem with rate-controlled approaches in wireless networks is that the service time for a Transport Block (TB) is neither known in advance nor does a sufficiently stable estimate exist. Therefore, these approaches are not suitable for WWAN.

III. The CLACK Agent / Approach

The basic idea of our approach is to place an assisting agent (the CLACK Agent) on BS in order to provide FH with proper pacing using Clock Acknowledgements (CLACKs), in addition to unaltered ACKs from the mobile endpoint MH. Figure 3 illustrates the components and nodes used in our approach. These components and their functionality are discussed in the following subsections.

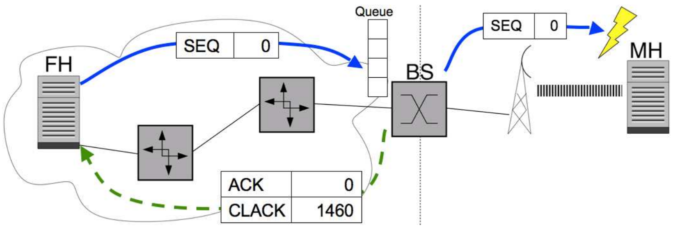

Fig. 3. For each TCP segment that left the wired path towards MH, the BS issues a CLACK (Clock Acknowledgement) back to FH. This example shows a *delivery failure* on the wireless link. Hence, the segment with sequence number 1460 is CLACKed together with the correct ACK number.

A. The Components on FH, BS, MH

1) The Receiver on MH: The receiver socket on the mobile host runs an ordinary TCP Reno without modifications.

2) The CLACK Agent on BS: The assisting agent on BS reacts to each TCP packet from FH in a similar way as an ordinary TCP receiver. However, acknowledgements issued by BS to FH consist of two values: the *CLACK* number, which indicates the next expected sequence number seen by the CLACK Agent *and* the *ACK* number, which is the next sequence number expected by FH. Please note, that TCP packets sent by MH are not simply forwarded by BS, but the acknowledgement information is conveyed to FH in the ACK field in packets sent by BS. Hence, proper pacing is achieved solely by the interaction of FH and BS. It must be pointed out that only unidirectional communication is discussed in this paper, bidirectional communication is left to future work, refer to Section VI.

Every packet received from BS is enqueued at BS or dropped in case of queue overflow. Any packet taken from the queue is forwarded to MH and acknowledged *regardless* whether the delivery attempt to MH is successful or not. This way, the queue from BS to MH appears as a part of the "Internet part" of the path. Queue overflow is handled on FH by the well known TCP congestion control [4].

For proper pacing, acknowledgements to be sent by BS are postponed until the acknowledged segment has left the outgoing queue of BS. While being busy with the delivery attempt, the outgoing interface of BS cannot accept new work. When the delivery attempt finishes, be it successful or not, the packet is released from the wired path (between FH and BS) and a new packet may be injected. Thus, FH correctly adapts to the wireless link's speed.

Another function of the BS is to inspect the regular ACKs returning from MH and to track the cumulative sequence number, which is inserted as the ACK number into the packets from BS to FH, hence the *unaltered end-to-end ACK information* from MH is conveyed to FH correctly. The CLACK agent does not harm TCP's end–to–end semantics and does not introduce any "hard state" into the system, because a TCP flow can recover from a CLACK agent failure without problems.

3) The Sender on FH: The sender is based on TCP Reno with the following extensions. *First*, the sender uses the CLACKs from BS as an indication that a packet has past BS, left the channel and new data may be sent. A missing CLACK indicates path congestion from FH to BS and leads to congestion recovery. *Second*, the sender uses the ACKs as an indication that a packet has been successfully read by MH.

The highest seen CLACK number which exceeds the highest seen ACK number (referred to as "CLACK" and "ACK" for convenience) indicates a corrupted packet to FH: the packet has past BS but is not read by MH and must be recovered. For this purpose, the sender is complemented with a "Corruption Recovery algorithm" in the same way the fast "fast recovery algorithm" is added in TCP Reno, refer to [13].

For several reasons, one may tolerate a certain difference between CLACK and ACK values. However this is left to future work, see the remark on flow regulation in Section VI. In the simulations used for this paper, we enter Corruption Recovery when the CLACK exceeds ACK.

B. Corruption Recovery

The Corruption Recovery algorithm is entered when the sender detects that $CLACK > ACK$. During Corruption Recovery, the FH is paced by the CLACKs sent by BS and missing data is retransmitted by FH until it is acknowledged by MH (which is seen in the ACK value received at BS). After

[1] The *average* Transport Block (TB) corruption rate may be an adaptation goal in some WWAN technologies. Although the *average* TB corruption rate may be known, the TB size is subject to change with the path conditions, hence it is unknown from how many TBs an IP packet is formed.

the data is successfully delivered to MH, FH leaves Corruption Recovery and returns to normal TCP Reno operation. In other words, Corruption Recovery ends when the ACK and CLACK values are again identical.

In our implementation, only the first missing packet is retransmitted, causing the sender to enter Corruption Recovery several times when more than a single packet is missing. Although we look forward to finding better strategies, even this simple approach yields promising results, as shown in Section V.

C. Congestion Control during Corruption Recovery

The number of retransmissions necessary to recover from packet corruption is unknown in advance. In case of a low packet corruption ratio, a single retransmission may be sufficient, but in case of severe packet corruption, the number of required transmission attempts until eventual packet delivery may be arbitrarily high. Nevertheless, the following principles should be obeyed in packet retransmission.

1) The FH must not sent packets at higher speed than these can be conveyed to MH.
2) Packets sent by FH should not lead to path congestion.
3) Data retransmission must continue until the missing data is eventually delivered and acknowledged, unless otherwise agreed by the user or the application.

The fundamental idea of our approach is to employ the well-proven self clocking and congestion control strategy from the "congavoid paper" [4]. During Corruption Recovery, the retransmitted data and the corresponding CLACKs make up the "data in transit" in the sense of [4]. As long as the path from FH to BS remains uncongested, FH will be correctly paced by CLACKs received from BS and continue necessary retransmissions until FH receives the missing ACKs.

Unfortunately, we cannot rely upon the original TCP sliding window scheme because CLACK numbers sent by BS are not advanced by retransmissions. However, we can adapt the congestion control from [4] if we count *packets* instead of *bytes*. Without loss of generality, we can assume equally sized recovery packets.[2] Hence, the sending window's size corresponds to a maximum number of unacknowledged packets in the path. Let's denote the number of allowed packets in the path with as N.

In addition, we introduce a concept of recovery *rounds*. During a round, up to N packets can be injected into the path. When N packets are injected into the network, a new round is started. All packets belonging to the same round are marked by a common *timestamp* (refer to [19]), which is reflected by BS in the CLACKs. The timestamp can be the sending time of a round's first packet.

All packets sent in a round must be acknowledged within the RTO period. In other terms: For every round, the number of CLACKs with the according timestamp is counted. When the last packet was injected into the path, the last CLACK must reach FH within the RTO period. When the number of seen CLACKs is less than N, this is recognized as an indication of path congestion between FH and BS.

IV. Simulation Environment

For testing the performance of the CLACK agent and its impact on TCP, we have designed a Java-based discrete event simulator[3] which implements a simplified TCP Reno according to RFC 5681 [13]. Without loss of generality, we make the following assumptions and simplifications:

- We focus on simplex data flows with FH as a TCP sender and MH as the receiver.
- *Each* TCP packet is acknowledged by a "pure ACK", i.e., an empty ACK packet that contains no data.
- We focus on the "ESTABLISHED" state of a TCP flow, hence the startup and termination phases are omitted.
- A fixed TCP packet size is used in order to avoid a possible silly window syndrome and to facilitate a simple version of Nagle's algorithm.
- In the simulation setup for this paper, there is no congestion between FH and BS.

As shown in Figure 1, our simulation setup consists of an FH that is connected to a router by a full duplex 10 MBit/s link with a propagation delay of 3 ns. The router is connected to BS with a full duplex 10 MBit/s link and a propagation delay of *intentionally* 200 ms, because we are particularly interested in how the recovery algorithm works in presence of an unusual large propagation delay, which is rarely seen in terrestrial networks unless it is caused by WWANs with unreasonably high persistence in packet retransmission.

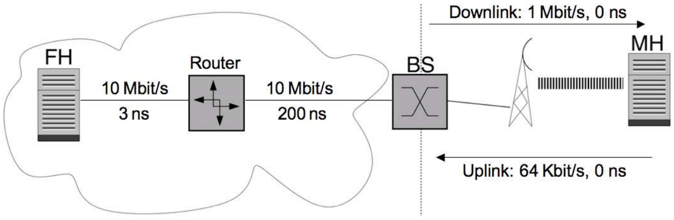

Fig. 4. Simulation setup.

The link between BS and MH is assumed to be a packet-switched WWAN, such as UMTS, GPRS or HSDPA. We assume a "Stop-and-Wait" protocol for the wireless link [10] and a reasonable limit for the number of transmission attempts per packet. As a "best current practice" recommendation, this limit is set to a maximum of *three* transmission attempts. The link between BS and MH roughly follows the HSDPA model: The *downlink* throughput is fixed at 1 MBit/s, the *uplink* throughput is 64000 Bit/s.[4] We performed simulations for several packet corruption rates in downlink direction. The

[2] In a very simple way, this can be achieved by retransmitting only the first missing packet as mentioned in Section III-B.

[3] Available at: http://detlef-bosau.de/index.php?select=tinysim

[4] We *do not* consider any propagation delay but used a fixed throughput, which leads to an appropriate serialization delay. Ideally, a WWAN interface should be modeled by its service time and SDU corruption rate. In our simulation, the packet sizes are fixed, so fixed service times and fixed throughput are equivalent in our case.

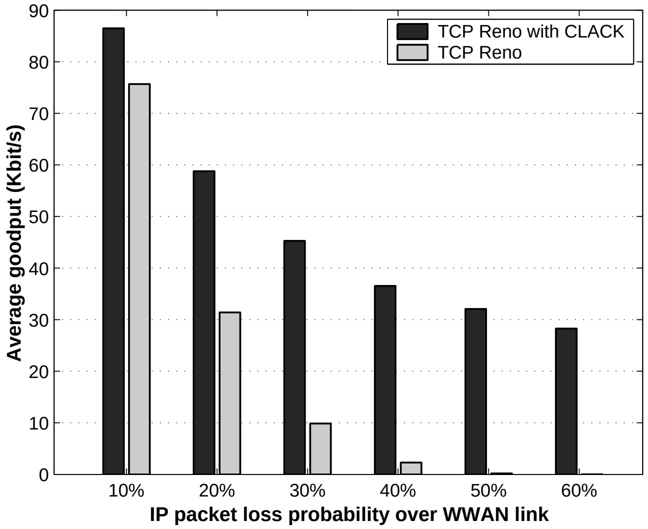

Fig. 5. Average goodput – standard TCP Reno vs. CLACK extensions.

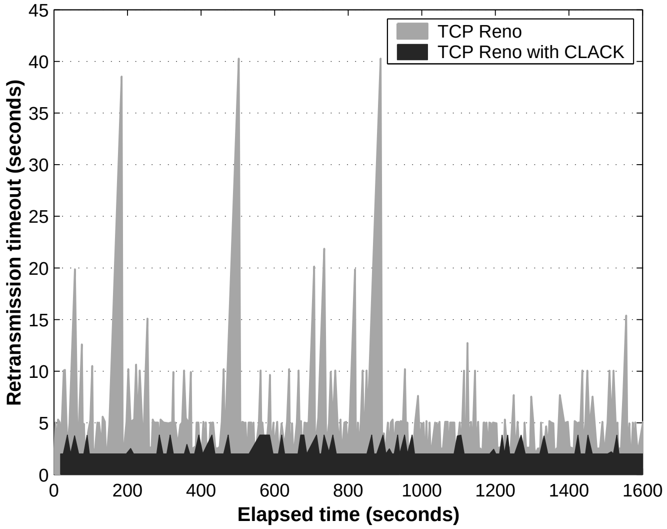

Fig. 6. Retransmission timeouts – standard TCP Reno vs. CLACK extensions (IP packet loss on WWAN link is set to 20%).

packet corruption rate in uplink direction is set to a fix value of 0.0. This simplification reflects the situation in HSDPA networks, where an extremely robust channel coding is used in uplink direction, at the expense of a quite low throughput.

WWANs typically employ some radio link protocol [20][10] which may convey a given packet in a certain *service time* and with a certain *corruption probability*. Actually, WWANs employ Stop-and-Wait protocols (refer to the discussion in [10]), because the wireless path's storage capacity is extremely small and a sliding window protocol would cause unreasonable overhead without providing significant benefit.

Protocols with "selective recovery" or sliding window mechanisms rarely make sense over WWAN links, because the link's storage capacity is typically extremely low and a minor throughput increase will hardly outweigh the overhead caused by a selective recovery protocol[10].

In the proposed protocol, the outgoing interface queue BS to FH is seen as a part of the "Internet part" of the path, hence the wireless network is free of congestion and any packet loss in this area is considered due to corruption. In addition, we assume *packet duplication* to be negligible.

V. Results

In this section, we discuss simulation results that show the performance of the CLACK approach. For the purpose of this paper it is not necessary to introduce a particular WWAN model. A WWAN link can be regarded as a link with constant bit rate and varying packet loss probability. For adaptive technologies, for instance HSDPA, the bit rate will change as well. However, this is beyond the scope of this paper and belongs to the discussion of RTT, refer to Section VI.

A. Average goodput

For the presence of various IP packet loss probabilities on the WWAN link, Figure 5 depicts the average goodput of a pure TCP Reno connection compared to a TCP Reno connection with a CLACK agent installed on the WWAN's base station. While TCP Reno becomes practically useless at error rates above 40%, the CLACK approach is able to maintain a low but steady data flow.

At a corruption error rate of 0% (not shown in Figure 5), both methods achieve an identical average data goodput of 954 Kbit/s, which is close to the 1 Mbit/s throughput of the error-free wireless link. Because our focus is on corruption recovery, we intentionally have no congestion in the wireline part of the path. Hence, there is no "congestion sawtooth" but only the TCP startup phase and the packet headers, which limits the goodput. In the absence of corruption errors, the Corruption Recovery mechanism introduced in Section III-B is never invoked. In other words, the CLACK agent has no negative impact on standard TCP Reno behavior.

B. Retransmission timeouts

The reason why the CLACK agent achieves an improved goodput at extreme corruption rates is because it avoids exponential RTO backoff to happen for packet loss that is caused by corruption on the wireless link. An exponential backoff should only be triggered due to congestion, not due to corruption. Without our proposed mechanism, TCP has to detect packet corruption by timeouts. In case of several sending attempts being necessary, these timeouts may grow significantly large due to the backoff algorithm, leading to phases without any data being delivered.

Figure 6 shows a timeline of two TCP Reno connections between FH and MH, one with a CLACK agent on the base station. In this experiment, the corruption probability was set to 20%, which may already cause timeouts of up to 40 seconds; leading to certainly noticeable pauses on the application layer. At the same time, the RTO values of the CLACK-enabled TCP Reno connection never exceeds 3.8 seconds.

For wireless corruption rates larger than 20%, the CLACK

TABLE I
MAXIMUM RTO VALUES OBSERVED [SECONDS]

Corruption rate	10%	20%	30%	40%	50%	60%
TCP Reno	10	40	1310	10300	166000	$2 * 10^{12}$
CLACK	3.86	3.82	3.82	3.82	3.82	3.82

approach manages to maintain a stable, low RTO value, while a pure TCP Reno connection may experience RTOs in the order of hours if the loss rate exceeds 30%. Table I shows how immensely the RTO grows due to corruption under ordinary TCP Reno operation and how the CLACK approach achieves very stable values.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the CLACK approach to solve the problems of loss differentiation and corruption recovery for the widespread scenario of wireless networks being used as access networks to the Internet. Our scheme builds on the fact that standard TCP senders rely on positive acknowledgements from the receiver for proper pacing and injection of new data onto the end-to-end path. However, high packet loss that is not caused by congestion on the wireline path, but by corruption on the wireless access link, causes TCP to wrongly adapt to the available path capacity.

The central idea of our approach consists in pacing the TCP sender with “clock acknowledgements” (CLACKs) from the wireless base station, where the end-to-end path can be logically split into a wired part that can suffer from congestion (but not from corruption), and a wireless part that can suffer from corruption (but not from congestion). With the introduction of a CLACK agent on the base station, TCP is prevented from misinterpreting packet loss due to corruption as an indication of path congestion, which improves the utilization of the available capacity. In addition, our scheme allows a very fast recovery from packet corruption and therefore avoids extensive RTO growth and annoying pauses in TCP connections.

Although being a “middlebox approach”, the CLACK agent *does not harm TCP’s end-to-end semantics* but offers a supportive means to accommodate TCP to the challenges of mobile and pervasive networking. Additionally, our solution is incrementally deployable and compatible with standard TCP endpoints.

In our future work, we first will overcome the simplifications made in our system model and simulation setup in Section IV, including the following tasks:

- We will extend our work to bidirectional flows. So far, only a sender on FH can take advantage of the CLACK mechanism. However, the problem of loss differentiation and recovery also exists for a sender on MH. A particular task for bidirectional flows is to achieve proper pacing in *both* directions, allowing the bottleneck to reside in the wireline and the wireless parts of the path.
- A simplified recovery scheme is used in this paper. Particularly, the recovery from several lost packets could be done in a more sophisticated way.
- We consider a CLACK-based flow regulation mechanism which makes RTT appear quasi-stationary to the sender as a remedy for spurious timeouts.
- The current solution does not yet support delayed acknowledgments [21].

REFERENCES

[1] L. Zhang, “Why tcp timers don’t work well,” in *Proceedings of SIGCOMM*, 1986.

[2] R. Jain, “Divergenge of timeout algorithms for packet retransmissions,” in *Proceedings of the Fifth Annual International Phoenix Conference on Computers and Communications, Scottsdale, AZ (USA)*, March 1986, pp. 174–179.

[3] S. W. Edge, “An adaptive timeout algorithm for retransmission across a packet switching network,” in *Proceedings of the ACM SIGCOMM*, 1984, pp. 248–255.

[4] V. Jacobson and M. J. Karels, “Congestion Avoidance and Control,” *ACM Computer Communication Review; Proceedings of the Sigcomm ’88 Symposium in Stanford, CA, August, 1988*, vol. 18, 4, pp. 314–329, 1988.

[5] R. Ludwig, “Eliminating Inefficient Cross–Layer Interactions in Wireless Networking,” Ph.D. dissertation, Aachen University of Technology, Aachen, Germany, April 2000.

[6] B. Ford and J. Iyengar, “Breaking Up the Transport Logjam,” in *Proceedings of 7th Workshop on Hot Topics in Networks (HotNets-VII)*, October 2008.

[7] S. Fahmy, V. Prabhakar, S. R. Avasarala, and O. Younis, “TCP over wireless links: Mechanisms and implications,” Purdue University, Tech. Rep. Technical Report CSD-TR-03-004, 2003.

[8] Y. Bai, A. T. Ogielski, and G. Wu, “Interactions of tcp and radio link arq protocol,” in *In Proceedings of the IEEE VTC-Fall*, Amsterdam, The Netherlands, September 1999, pp. 1710–1714.

[9] H. Balakrishnan, “Challenges to reliable data transport over heterogeneous wireless networks,” Ph.D. dissertation, University of California at Berkeley Department of Electrical Engineering and Computer Sciences, 1998.

[10] G. Fairhurst and L. Wood, “Advice to link designers on link Automatic Repeat reQuest (ARQ),” IETF RFC 3366, August 2002.

[11] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, “Performance enhancing proxies intended to mitigate link-related degradations,” IETF RFC 3135, June 2001.

[12] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End–to–end arguments in system design,” *ACM Transactions in Computer Systems*, vol. 2, no. 4, pp. 277–288, November 1984.

[13] M. Allman, V. Paxson, and E. Blanton, “TCP Congestion Control,” IETF RFC 5681, September 2009.

[14] V. Paxson and M. Allman, “Computing TCP’s Retransmission Timer,” IETF RFC 2988, November 2000.

[15] J. Lacan and E. Lochin, “On-the-Fly Coding to Enable Full Reliability Without Retransmission,” *Technical Report, ISAE, LAAS-CNRS, France*, 2008.

[16] J. Liu, I. Matta, and M. Crovella, “End–to–end inference of loss nature in a hybrid wired/wireless environment,” in *Proceedings of WiOpt’03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003. [Online]. Available: citeseer.nj.nec.com/589180.html

[17] W. M. Eddy, S. Ostermann, and M. Allman, “New techniques for making transport protocols robust to corruption-based loss,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 5, pp. 75–88, 2004.

[18] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, “Tcp westwood: Bandwidth estimation for enhanced transport over wireless links,” in *Proceedings of ACM Mobicom 2001*, Rome, Italy, July 16–21 2001, pp. 287–297.

[19] V. Jacobson, R. Braden, and D. Bormann, “TCP Extensions for High Performance,” IETF RFC 1323, May 1992.

[20] 3rd Generation Partnership Project 2 (3GPP2), “Data Service Options for Spread Spectrum Systems: Radio Link Protocol Type 3,” 3GPP2 Document 3GPP2 C.S0017-010-A, June 2004.

[21] R. Braden, “Requirements for internet hosts – communication layers,” IETF RFC 1122, October 1989.