Cross-Layer Optimization of the Link-Layer based on the Detected TCP Flavor

Toktam Mahmoodi, Oliver Holland, Vasilis Friderikos, Hamid Aghvami
Centre for Telecommunication Research, King's College London,
Strand, London WC2R 2LS, UK
{toktam.mahmoodi, oliver.holland, vasilis.friderikos, hamid.aghvami}@kcl.ac.uk

***Abstract*—Numerous flavors of TCP are already in existence, and further variations on TCP mechanisms are frequently being introduced in order to, for example, cope with the packet loss characteristics of wireless links. Moreover, the proliferation new wireless standards and the relative performance differences among them have been mushrooming in recent years. Given the increasingly heterogeneous nature of the Internet, mechanisms do not usually exist for a server to specifically select an appropriate TCP flavor for each individual download. In this paper, we therefore present and assess a cross-layer solution for a node (e.g. a base-station) to rapidly adapt lower layer characteristics (the coding rate and local ARQ retransmissions threshold) based on the detected TCP flavor, in order to optimize the end-to-end performance of the download for that utilized flavor of TCP. We demonstrate that the proposed scheme has considerable potential to improve the overall download throughput, while placing no burden on the server and requiring no changes to existing TCP flavors.**

I. Introduction

Transmission Control Protocol (TCP) [1] is the predominantly utilized transport protocol to achieve reliable data transfer from a server to a client over the Internet. TCP is, however, continually being remolded, initially to optimize its performance over wired networks [2][3][4], but more recently also to improve its performance over large bandwidth-delay links and wireless links [5]. This latter consideration is particularly challenging, as the random losses of wireless links typically fool TCP congestion control into thinking that these losses are due to congestion, causing TCP to unnecessarily reduce its transmission rate.

The Internet as an entity is becoming more and more heterogeneous. With the advent and increasing proliferation of wireless means to support packet data transfer, in addition to new wired transport means such as ADSL, the performance characteristics of the 'black box' end-to-end link over the Internet have become increasingly difficult to predict. Moreover, the means commonly do not exist for a server, which might be serving a vast number of different types of clients with each particular file for download, to be able to detect the lower-layer performance characteristics of each end-to-end connection to each individual client. For this among other reasons, the TCP flavor used by the server for an end-to-end download is likely to be fixed beforehand, and not modifiable on a case-by-case basis to optimize performance based on the lower-layer characteristics experienced by each flow.

In this work, based on the prior observations, we assume a top-down approach to cross-layer optimization. At a base station for example, we show that it is possible to, based on the quick determination of the utilized TCP flavor for each connection, use this information to optimize lower layer parameters (such as coding and local ARQ retransmission thresholds) for that flow in order to achieve an improved overall end-to-end TCP performance. Our solution is particularly pertinent given that the ways in which different flavors of TCP might react to lower-layer performances vary greatly. TCP Reno (TCPR) for example might be fooled by random losses hence unnecessarily reduce its transmission rate, whereas TCP Westwood (TCPW) might be negligibly affected by random losses. Hence in the TCPR case it could be beneficial to provide a small amount of additional lower-layer Forward Error Correction (FEC) over the wireless link to reduce random losses, whereas for TCPW this additional FEC might represent a waste of wireless capacity hence a reduction in goodput. This suggested algorithm builds on our past work on top-down cross-layer TCP optimization [6], and is compatible with active solutions to improve end-to-end performances of reliable services, such as [7].

This paper is structured as follows. In the next section, we quickly introduce common TCP flavors and discuss their performances and procedures under packet losses. Afterwards, the TCP throughput modeling is described and different TCP flavors throughput is compared, in this process also verifying aspects of our simulation model. In section III we investigate levels of local link ARQ persistence and FEC, thereby also arguing the chosen parameterizations for our cross-layer scheme. In section IV we prove the performance benefit of our scheme, before concluding in section V.

II. TCP Flavors and Packet Losses

For the vast majority of the time, TCP connections are likely to be in one of two phases: slow start or congestion avoidance. In the slow start phase, TCP increases its congestion window (cwnd) exponentially, leading to a doubling of the cwnd per Round Trip Time (RTT). In the congestion avoidance phase, initiated upon the cwnd reaching the slow-start threshold (ssthresh), the cwnd is increased linearly by one packet per RTT.

Packet losses in a TCP connection can be detected by Duplicate ACKnowledgements (DupACKs), or by retransmission

timer expirations. DupACKs (i.e., acknowledgements where the sequence number has not been incremented) are returned by the TCP receiver as an immediate response to receiving an out-of-order segment. From the sender's perspective however, DupACKs might be caused by a number of other issues, such as re-routing and traffic shaping, in addition to packet loss. Hence, to be conservative, loss detection is trigged only upon receiving three consecutive DupACKs.

A. TCP Congestion Control in the Presence of Losses

In the presence of losses, the behavior of TCP congestion control varies dependent on the TCP flavor. In this paper, we concentrate on the TCP Reno (TCPR), TCP NewReno (TCPNR), and TCP Westwood (TCPW) flavors, as well as the presence of the SACK option in TCPR and TCPNR. These characteristics are therefore summarized as follows.

TCPR congestion control [3], which can be considered as the baseline for modern TCP implementations, supports fast retransmit and fast recovery upon segment losses. In TCPR, when a sender detects a segment loss through a retransmission timer expiration, the cwnd is set to one segment and the ssthresh is set to half of the FlightSize, where the FlightSize is the amount of outstanding data in flight within the network. If it detects packet loss through incoming DupACKs, the TCP sender invokes fast retransmit, which performs a retransmission of the lost segment immediately without having to wait for timer expiry. Fast recovery, which is used in conjunction with fast retransmit in TCPR and later flavors, sets the ssthresh to half of the FlightSize and the cwnd to the ssthresh plus 3 segments; this is deemed appropriate because although a loss has happened, packets are still getting through hence any reversion to slow-start would be far too severe. Upon receipt of the next ACK for new data, the cwnd is set to ssthresh, and congestion avoidance phase resumes. This received ACK therefore acknowledges all segments sent between the initial lost segment and its retransmission (including segments that triggered DupACKs, as well as those transmitted since and that were already in flight).

TCPR is known to generally not recover efficiently if there are multiple losses in a single flight of packets. TCPNR on the other hand presents a modification to the fast recovery algorithm of TCPR to improve recovery from multiple packet losses per window [4]. In the case of TCPNR, the ACK for new data is a partial ACK. The algorithm then retransmits the first unacknowledged segment, and deflates the congestion window by the amount of new data acknowledged by the cumulative acknowledgement field. Upon receipt of an ACK which acknowledges all segments, fast recovery phase exits.

The Selective ACKnowledgement (SACK) option can significantly further improve performance if there are a large number of packet losses per transmission window (e.g., if there are burst-losses). The SACK option allows a receiver to specify, in acknowledgements, whole blocks of packets which have been received successfully. Generally however, the options part of a TCP header is only be large enough to allow for a maximum of three SACK blocks.

B. Shared Medium Wireless Access

The mechanisms described above can handle competition fairly well through cwnd and ssthresh changes in response congestion-related losses. However, in shared medium access networks the available bandwidth for a TCP flow is highly variable dependent on channel utilization and medium access protocol dynamics. If a sudden change in available bandwidth occurs, TCP may be too slow to converge to this bandwidth. Moreover, TCPR/TCPNR and TCP SACK are usually not robust when random (e.g., wireless) losses occur, as they misinterpret these losses as being caused by congestion. Alternatively, TCPW [5], which only requires modifications to the server-side TCP, has been proposed to solve these problems. In response to a packet loss as detected by DupACKs, TCPW sets the cwnd and ssthresh to an estimated eligible bandwidth, which is calculated by low-pass filtering the rate of incoming ACKs (i.e., if ACKs are being returned at a certain rate, then packets are getting to the receiver at that same rate hence the network can support that rate). Describing this mathematically, if a loss is the result of DupACKs, the values for cwnd and ssthresh are set to

$ssthresh = BWE \cdot RTT_{min}/SegmentSize,$

$cwnd = \min(cwnd, ssthresh).$

In the case of a retransmission timer expiring under TCPW, these values are set to

$ssthresh = \max(BWE \cdot RTT_{min}/SegmentSize, 2),$

$cwnd = 1.$

It will be seen later that TCPW performs far better in cases of random losses (or even in congestion-related losses), as the cwnd and ssthresh are far more appropriately set after the random loss, instead of blindly being halved.

C. TCP Throughput Modeling

TCP throughput models are proposed in the literature to analytically describe throughput dynamics as affected by the RTT and Packet Error Rate (PER). Under the assumption of independence of packet losses between rounds, and that the send rate is not limited by the receiver's advertised window, a closed form for TCPR throughput is proposed in [8] and revised in [9].

An analytical model for TCPNR throughput is proposed in [10], where the same assumptions as above apply. A TCPW analytical throughput model is proposed in [11]. Here it is assumed that the system is always in congestion avoidance phase, and that only a single packet loss occurs in each cycle.

To graphically investigate the effect of random packet losses on TCP throughput, and to verify our simulation platform (the simulation code for TCPW in the utilized OPNET platform was created by ourselves, based on the UCLA model for ns-2), each TCP flavor is simulated in a single-flow scenario with a wireless link at the client. The same conditions of 100ms RTT, 2Mbps bottleneck link, and a random packet loss rate varied between $\sim 10^{-5}$ and $\sim$0.15 are applied to the analytical model and OPNET simulations; moreover, the simulations are all performed over a download file size of 16MB ($\sim$11,000

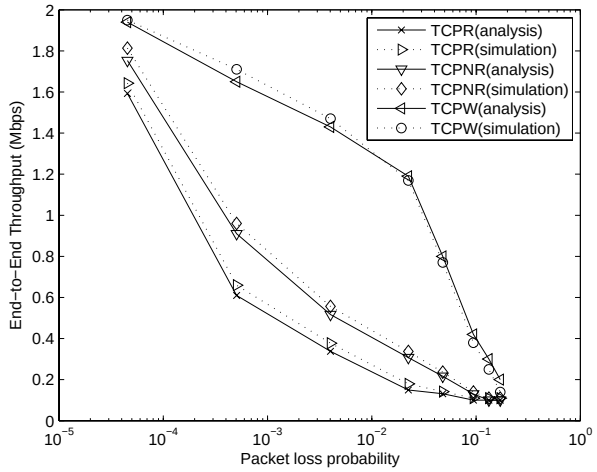

Figure 1. Single-flow analytical model and simulation results comparisons for TCPR, TCPNR, and TCPW

packets). Referring to Figure 1, the simulations compare well with the analysis.

III. ARQ Persistence and FEC

In wireless networks, in order to counter the effects of random losses, some degree of additional reliability is often provided at the Link-Layer (LL). This reliability is usually achieved through local link Automatic Repeat Requests (ARQ) in conjunction with some form of FEC.

ARQ over a single wireless link is more rapidly reactive than TCP's acknowledgment control loop. This is because TCP operates over a much longer delay path than a single wireless link, and because packets (and retransmissions, or coded FEC packets) at the LL are usually much smaller than at the network-layer due to LL fragmentation. ARQ protocols are characterized by their persistency, which affects the length of time the link is allowed to delay a packet [12]. Setting a lower LL persistency reduces the potential to accidentally cause TCP retransmission timeouts, and may therefore reduce the probability of duplicate copies of the same packet being sent by TCP and the connection incorrectly entering slow start. Alternatively, setting a higher persistency increases the wireless link reliability.

FEC coding also improves reliability by sending redundant data coded from the original data sequence prior. This redundant data allows the receiving system to correct a proportion of errors caused by channel corruption. The reliability of FEC is increased by increasing the redundancy rate; this will also increase the channel load and possibly the processing/reception delay. The relationship between the bandwidth utilized by FEC, the extra delay caused by ARQ, and the throughput gained by a TCP flow, is examined thoroughly in the literature (see, e.g., [13]).

A. Assumed Parameters

Among the TCP flavors investigated in this work, TCPR degrades most dramatically in the event of random losses, and the least degradation occurs with TCPW. Thus the degree of reliability necessary for TCPR flows is high, although this is decreased for TCPNR and much more so for TCPW. Furthermore, the SACK option can yield significant improvements for TCP and TCPNR, hence the required FEC rate can be decreased if the SACK option is advertised.

Given the above observations, for TCPR flows, the set of $\frac{1}{3}$ code rate and a maximum of 4 LL ARQ retransmission attempts (denoted in this paper as ($\frac{1}{3}$, 4) is chosen. Given the use of the SACK option with TCPR, these values are changed to ($\frac{1}{2}$, 4). The TCPNR flows are given the set ($\frac{1}{3}$, 2), which is decreased to ($\frac{1}{2}$, 2) given the use of SACK. Lastly, for TCPW, these values are set to ($\frac{2}{3}$, 4). In all cases, we assume a convolutional FEC code implemented in the code rates: $\frac{1}{3}$, $\frac{1}{2}$, $\frac{2}{3}$, or 1 (i.e., no coding).

We assume the NASA standard convolutional encoder/decoder which is well implemented e.g. in Actel enc/dec core [14], and can support selectable code rates of 1/3, 1/2, and 2/3. The constraint length K is equal to 7 and the polynomials are $g_0 = 171$, $g_1 = 133$. The minimum distance of the code, d_{free}, values for the described enc/dec are calculated from the convolutional Trellis diagram with K = 7 and are equal to 15, 10, and 6 respectively for the four mentioned code rates [14].

Identification of the TCP flavor is performed via a mechanism presented in [15]. Here, the TCP flavor and state is determined by monitoring changes in the estimated cwnd, where, using this approach, the cwnd can be estimated passively at any point within the network (e.g., at access points). Of the TCP flavors, TCPR, TCPNR, as well as the use of SACK options, are covered by [15], but TCPW is not considered. We therefore use our own mechanism to identify TCPW, whereby if a DupACK-triggered loss indication does not result in an approximate halving of the cwnd, the flavor is assumed to be TCPW by deduction. Further work on the detection of TCPW may be pursued by us in the future.

IV. Performance Investigation

To validate the proposed cross-layer scheme, a Wireless Local Area Network (WLAN) within the OPNET platform is simulated. TCPR, TCPNR, as well as the use of SACK options, were all supported already within OPNET; however, we had to implement TCPW within OPNET ourselves based on the UCLA model which was already available for Network Simulator 2 [16]. We have verified, through comparison with published simulation results and by comparison with an analytical model (see Figure 1), that our implementation is correct. We have also liaising with the designers of TCPW.

In our simulation scenarios, wireless clients set up connections with wired servers via a single WLAN Access Point (AP) at the wireless side. Each wireless client connects to a unique server, whereby the bottleneck is assumed to be at the wireless link. LL retransmissions are performed via the stop-and-wait ARQ algorithm, and the channel is modeled by the free space path loss, Rayleigh fading (exponential random variable, β=1), and Lognormal shadowing (standard deviation 4dB). Unless otherwise stated, all wireless clients are the same

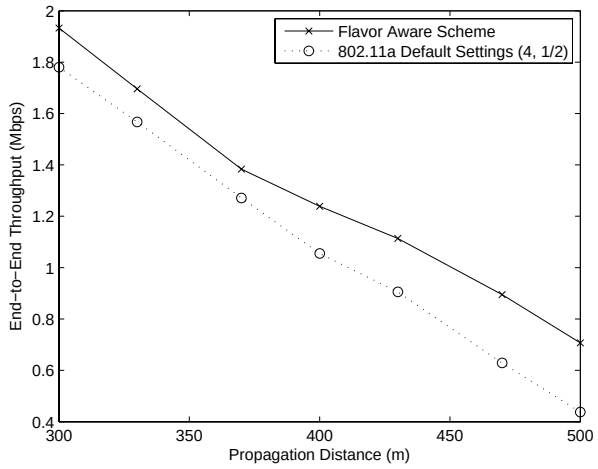

Figure 2. End-to-end throughput vs. wireless propagation distance for the second simulated scenario

TABLE I
APPLICATIONS AND TCP FLAVORS FOR SCENARIOS TWO, THREE AND FOUR

TCP Flavor/Traffic	FTP	HTTP	Email
TCPR	4clients	1client	2clients
TCPNR	1client	1client	1client
TCPW	1client	1client	1client
TCPR+SACK	1client	1client	1client
TCPNR+SACK	1client	0clients	1client

TABLE II
END-TO-END AGGREGATED THROUGHPUT FOR THE NORMAL, EXPONENTIAL AND UNIFORMLY DISTRIBUTED RTTS

RTT Distribution	Normal	Exponential	Uniform
Throughput(Mbps) flavor aware scheme	1.180	1.159	1.098
Throughput(Mbps) ($\frac{1}{2}$, 4) scheme	0.918	0.987	0.944

distance from the AP in order to ensure that the end-to-end loss characteristics are similar for the competing flows in the simulation; the wireless link propagation distance is 400m, and RTTs are all set to 100ms. Fading and shadowing attenuations are updated on a per-packet basis according to the assumed channel model.

Other specific simulation characteristics are as follows:

- Simulation duration: 600s
- FTP servers: 16 MB file download size ($\sim$11,000 packets)
- HTTP servers: HTTP1.1, page interarrival time = exponential (mean 60s), html size = 1kB, images per page = 5, image size = uniform (500B, 2kB)
- Email servers: Email size = 2kB, interarrival 360s
- TCP Maximum Segment Size (MSS): 1,460B
- MAC Buffer size: 32kB
- Physical-layer characteristic: OFDM (802.11a)
- Operating Frequency: 5.4GHz

It is noted that we have concentrated on 802.11a to ensure maximum relevance to novel and future RATs.

The first simulated scenario is a relatively simple case where there are 4 wireless clients and 4 FTP servers, respectively operating with TCPR, TCPNR, TCPW, and TCPR with the SACK option enabled. The performance of our cross-layer optimization scheme is compared with the performance of the 802.11a default settings, specifying a maximum of 4 LL retransmission attempts and a code rate of $\frac{1}{2}$. Under this scenario, aggregated end-to-end throughput shows an average improvement of approximately 11% for our proposed cross-layer scheme, as compared with the use of the default 802.11a LL settings.

In a second more complicated simulation scenario using 18 wireless clients, a range of applications and flavors of TCP are assumed, as conveyed in Table I. Throughput results for this scenario, versus the wireless link propagation distance, are plotted in Figure 2.

In a third simulated scenario, using an otherwise identical configuration to the prior scenario, the RTTs of end-to-end paths are set according to a random variable. There are three chosen RTT distributions: Normal (μ = 100ms, σ = 10ms), Exponential (μ = 100ms), and Uniform (0ms lower bound, 200ms upper bound). Results for this scenario are presented in Table II. Again, significant performance improvements are achieved by our cross-layer scheme, for all random RTT distributions.

In the fourth and final simulated scenario, using the same 18-flow configuration, wireless clients are placed a normally distributed random distance from the AP as opposed to their distances being fixed. The results for this scenario again show an approximate 11% improvement in aggregated end-to-end throughput as achieved by our cross-layer optimization scheme.

V. CONCLUSION

In this paper, we have presented a cross-layer mechanism to optimally set the ARQ retransmission threshold and coding rate over the wireless link, based on the end-to-end TCP flavor for each flow as detected at that wireless link. Our solution imposes no requirements on servers and implies no adaptations to current TCP designs. We have highlighted the means for achieving our mechanism, and have simulated its performance over an OFDM wireless network. Simulation results clearly demonstrate a significant increase in transport-layer throughput that can be achieved by our mechanism. These performance increases are typically 11% as compared with the use of the default 802.11a coding rate and ARQ retransmissions threshold.

VI. ACKNOWLEDGMENT

The work reported in this paper has formed part of the Delivery Efficiency Core Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.com. This research has been funded by EPSRC and by the Industrial Companies who are

Members of Mobile VCE. Fully detailed technical reports on this research are available to Industrial Members of Mobile VCE.

REFERENCES

[1] Information Science Institute, University of Southern California, CA, USA, "Transmission Control Protocol," *IETF RFC 793*, Sep. 1981.

[2] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM '88*, Stanford, CA, USA, Aug. 1988.

[3] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, Apr. 1999.

[4] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," *IETF RFC 3782*, April 2004.

[5] R. Wang, K. Yamada, M. Y. Sanadidi, and M. Gerla, "TCP with Sender-side Intelligence to handle Dynamic, Large, Leaky Pipes," *IEEE JSAC*, vol. 23, no. 3, pp. 235–248, 2005.

[6] T. Mahmoodi, V. Friderikos, O. Holland, and A. H. Aghvami, "Cross-Layer Design to Improve Wireless TCP Performance with Link-Layer Adaptation," *Proc. IEEE VTC Fall*, pp. 1504 – 1508, Oct. 2007.

[7] O. Holland, T. Mahmoodi, and A. H. Aghvami, "A Software Download Management Module," *Proc. IEEE VTC Fall*, pp. 1984 – 1989, Oct. 2007.

[8] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," *IEEE/ACM Tran. Net.*, vol. 8, pp. 133–145, Apr. 2000.

[9] Z. Chen, T. Bu, M. Ammar, and D. F. Towsley, "Comments on Modeling TCP Reno Performance: A simple model and its Empirical Validation," *IEEE/ACM Tran. Net.*, vol. 14, pp. 451–453, Apr. 2006.

[10] R. Dunaytsev, Y. Koucheryavy, and J. Harju, "TCP NewReno Throughput in the Presence of Correlated Losses: The Slow-but-Steady Variant," *Proc. IEEE INFOCOM'06; Global Internet Workshop*, pp. 115–120, Apr. 2006.

[11] A. Zanella, G. Procissi, M. Gerla, and M. Y. Sanadidi, "TCP Westwood: Analytic Model and Performance Evaluation," *Proc. IEEE GLOBECOM'01*, vol. 3, pp. 1703–1707, Nov. 2001.

[12] G. Fairhurst and L. Wood, "Advice to Link Designers on Link Automatic Repeat Request (ARQ)," *IETF RFC 3366*, Aug. 2002.

[13] D. Barman, I. Matta, E. Altman, and R. Azouzi, "TCP Optimization through FEC, ARQ and Transmission Power Tradeoffs," *The 2 nd International Conference on Wired/Wireless Internet Communications*, 2004.

[14] 4i2i Communication Ltd., *Convolutional Encoder and Viterbi Decoder*, 3.1 ed., Dec. 2001.

[15] S. Jaiswal, *Measurement in the Middle: Inferring End-End Path Properties and Characteristics of TCP Connections through Passive Measurement*. PhD thesis, University of Massachusetts Amherst, Sep. 2005.

[16] "The TCP Westwood Model for the ns2 Network Simulator: www.cs.ucla.edu/nrl/hpi/tcpw,"