Evaluating Cross-Layer Cooperation of Congestion and Flow Control in IEEE 802.11s Networks

Michael Rethfeldt, Peter Danielis, Benjamin Beichler, Björn Konieczek, Felix Uster, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany, Tel.: +49 381 498-7269
Email: michael.rethfeldt@uni-rostock.de

***Abstract*—The new standard IEEE 802.11s enables vendor-independent wireless mesh networks based on the 802.11 WLAN technology. Transmission Control Protocol (TCP) is the most widespread transport protocol for reliable data delivery and still the basis for many network applications. TCP supports different mechanisms for flow and congestion control. However, designed for wired networks, it does not consider the dynamics of wireless networks and especially multi-hop wireless mesh networks. In addition, 802.11s provides own mechanisms such as Automatic Repeat Request (ARQ) for frame retransmissions to hide wireless loss from the upper layers. Being transparent to each other, retransmission schemes on both layers may interfere and operate redundantly, if not properly adjusted. We study the effects of ARQ retry limit variation on TCP throughput in a real-world multi-hop 802.11s test bed. As a result, we suggest ARQ adaptation based on the 802.11s standard's Airtime Link Metric (ALM) for path selection, serving as indicator for overall frame travel time. Our proposed approach solely relies on standard features and imposes no modifications to 802.11s or TCP.**

I. Introduction

Envisaged to realize future community networks, information services in public facilities, wireless last-mile internet access, or spontaneous communication in disaster scenarios, WLAN mesh networks have emerged as an attractive technology [1]. Ratified in 2011 as first industry WLAN mesh standard, *IEEE 802.11s* introduces mesh extensions to the existing 802.11 MAC layer specification [2]. By that, it inherits and integrates most existing MAC layer mechanisms, such as *Automatic Repeat Request* (ARQ) for flow control and shielding of frame transmission errors to upper layers. Frames are retransmitted for a certain number of retries until successful reception is confirmed by an ACK.

Above the WLAN MAC layer, *Transmission Control Protocol* (TCP) is still the de facto standard transport protocol to enable reliable data delivery for an abundance of network applications. Originally designed for wired networks, its flow and congestion control mechanisms expect low bit error rates and small delays, contrary to the indeterministic behavior of wireless environments [3]. Over time, a variety of different TCP congestion control algorithms have been proposed, extending or modifying standard TCP [4]–[9]. Some of them even focus on lossy network conditions [4]–[6].

Nevertheless, corresponding mechanisms for flow control at both TCP and 802.11 level need to be tuned carefully to prevent interference [10], [11]. On the one hand, ARQ is beneficial to hide wireless loss from TCP and avoid unnecessary segment retransmissions, which are costly due to inherent reduction of the sending window. On the other hand, TCP congestion control is tightly coupled with flow control timing parameters, such as round trip time (RTT) and retransmission timeout (RTO). These are affected by channel contention and delays imposed by ARQ frame retransmissions and do not consider the underlying MAC layer parameters. Especially in WLAN mesh networks, ARQ is active at all hops along a multi-hop TCP flow. On the occurrence of transient errors and wireless contention, the number of ARQ frame retransmissions may touch a defined *retry limit* per hop, considerably increasing overall delay and causing TCP timeouts.

Consequently, solutions must be pursued that improve cross-layer cooperation between 802.11 and TCP flow and congestion control. Therefore, we have evaluated TCP performance in a real-world 802.11s mesh test bed by varying path length, ARQ retry limit, and TCP variant. Further, we have identified the 802.11s standard's path cost metric as suitable indicator to improve cross-layer cooperation.

The remainder of this paper is organized as follows: Section II outlines the basic principles of IEEE 802.11s and TCP. Section III illustrates the layer mismatch between both protocols. In Section IV, we describe several measurements conducted in a real-world test bed to explore the cooperation between 802.11s and TCP. Discussing our results, we introduce a cross-layer approach to improve MAC-layer ARQ when applying TCP over 802.11s networks. In Section V, we give an overview of related work in optimizing TCP performance over WLAN and WLAN mesh networks. Finally, we conclude our results in Section VI and briefly state next steps for future research.

II. Technological basis

A. IEEE 802.11s WLAN Mesh Networks

The WLAN mesh standard IEEE 802.11s comes as amendment to the 802.11 MAC-layer specification and thus inherits most of its mechanisms. Additionally, it introduces mesh extensions, such as automatic link establishment (peering), frame forwarding, and path selection [2].

Channel Access and ARQ: The majority of 802.11 deployments today relies on the Distributed Coordination Function

(DCF) to perform CSMA/CA-based channel access. However, the Hybrid Coordination Function (HCF) with Enhanced Distributed Channel Access (EDCA), originating from QoS amendment 802.11e, is defined as default mechanism for 802.11s mesh networks [2]. EDCA extends the common DCF mainly by the use of Access Categories (AC) for traffic prioritization. If not specified otherwise, the default "Best Effort" (AC_{BE}) category is assumed. Like in DCF, channel access is regulated based on inter frame spaces (IFS) and an exponential back-off procedure. The latter is determined by a contention window (CW) that gives the number of slot times to wait before a WLAN station attempts medium access [12]. The CW value is chosen from an uniformly distributed integer interval that ranges from 0 to an upper limit. For the initial CW, this upper limit is CW_{init}. Otherwise, the upper limit CW_{cur} of the interval is doubled on consecutive transmission retries until a maximum value CW_{max}. Values for CW_{init} and CW_{max} depend on the applied 802.11 PHY layer.

The slot time (ST) and short inter frame space (SIFS) are likewise technology dependent parameters. Based on both, the other IFS are calculated [12]. The distributed IFS (DIFS) is only used in DCF and given as $SIFS + 2 \cdot ST$. In EDCA, the access category dependent arbitrary IFS (AIFS[AC]) replaces DIFS and results from $SIFS + AIFSN[AC] \cdot ST$, where AIFSN for default category AC_{BE} is 3. All stations in radio range wait for an access category dependent extended IFS (EIFS[AC]) instead of AIFS[AC] after reception of an erroneous frame to not interfere with subsequent retransmissions and ACKs. EIFS[AC] is given as $SIFS + AIFS[AC] + T_{tx_{ACK}}$. The retransmission of frames after ACK timeouts is controlled by stop-and-wait Automatic Repeat Request (ARQ). To have an administrative control over ARQ, an upper ***retry limit*** can be set for a WLAN network interface.

Based on the parameters described, maximum duration of a successful EDCA-based single-hop single-frame transmission, needing a certain number of ARQ retries R, can be approximated as the sum off time (T_{Sum}) needed for successful transmission of the frame (T_F) and its ACK (T_{ACK}), and the delay imposed by ARQ retransmissions (T_{ARQ}) [12]:

$$\begin{aligned} T_F &= AIFS[AC] + CW_{init} \cdot ST + T_{tx_F} \\ T_{ACK} &= SIFS + T_{tx_{ACK}} \\ T_{ARQ[i]} &= T_{ACK} + AIFS[AC] + CW[i] \cdot ST + T_{tx_F} \\ T_{ARQ} &= \sum_{i=0}^{R-1} T_{ARQ[i]} \\ T_{Sum} &= T_F + T_{ACK} + T_{ARQ} \end{aligned} \tag{1}$$

$T_{tx_{ACK}}$ denotes the time needed for ACK transmission at the lowest mandatory rate defined by the 802.11 PHY mode. T_{tx_F} denotes the time needed for frame transmission at a rate given by automatic rate control or a fixed setting. $CW[i]$ is the back-off value chosen from the current CW interval depending on i, the number of ARQ retransmissions already needed. T_{ARQ} describes the worst case where all R retransmissions are caused by errors during data frame delivery and not during ACK delivery. Here, a frame with lost ACK would still be forwarded along the path, leading to a smaller overall latency in spite of higher redundant MAC-layer traffic.

Mesh Path Selection: The most important 802.11s mesh extension is path selection using the Hybrid Wireless Mesh Protocol (HWMP) and Airtime Link Metric (ALM). HWMP is based on the reactive distance vector routing protocol AODV that determines a path as soon as it is needed (on-demand). ALM allows for path cost estimation, as it represents the cumulative frame transmission time over all peer links forming a mesh path in µs [13]. The so-called *airtime cost* (c_a) for a certain peer link is calculated by each node as follows:

$$c_a = \left[O_{ca} + O_p + \frac{B_t}{r} \right] \cdot \frac{1}{1 - e_{fr}} \tag{2}$$

O_{ca} and O_p are constants for the channel access and MAC protocol overhead. B_t is the test frame size and given as 8182 bits (1 kB) by default. r denotes the test frame data rate, given in Mbit/s, whereas e_{fr} denotes the expected frame error rate. The estimation of e_{fr} as well as the values of the overhead constants are left open to vendor implementations. The 802.11s reference implementation *open80211s* for Linux systems provides own parameter variants [14], [15]. While O_{ca} and O_p are summarized to 1, data rate r of the last unicast frame transmission depends on the rate control algorithm (RCA). In current Linux kernels, *minstrel* is used as default RCA [16]. Error rate e_f is updated on every frame transmission and calculated by a moving average filter in current versions of Linux [15].

During HWMP path selection, e.g., being triggered at the application level, the airtime costs calculated by each node are accumulated and disseminated within path request (PREQ) and response (PREP) frames. Thus, a requesting node is made aware of the overall cost associated with a path towards a target node via a certain neighbor ("next hop" peer). Aside from link information and path costs to certain targets, nodes only have a limited network view, which is typical when applying distance vector routing protocols like HWMP.

B. Transmission Control Protocol (TCP)

TCP provides a reliable connection oriented full duplex service between two hosts that exchange data in the form of segments. Two of the most important fields in the header of TCP segments are the sequence number and acknowledgment (ACK) number fields. TCP considers data as an unstructured but ordered stream of bytes. The sequence number points to the first byte of a segment in the byte stream. The ACK number indicates the sequence number of the next byte, which is expected. With the help of these numbers, missing and out-of-order segments can be identified.

Round trip time and timeout: TCP retransmits lost segments after a timeout. In order to avoid unnecessary retransmissions, the timeout is greater than the round trip time (RTT) of the connection. For this purpose, the sender stores the so-called *SampleRTT (SRTT)* of a segment. This is the period between the transmission of the segment and the receipt of

its ACK. Due to changing network load situations, the SRTT values can vary from segment to segment. Therefore, the average of the SRTT values called *EstimatedRTT (ERTT)* is determined. In addition, the deviation *DRTT* indicates how much the SRTT typically differs from the ERTT. Finally, the retransmission timeout (RTO) denotes the time interval after which a TCP retransmission is triggered. Equation 3 shows RTT estimation and RTO calculation. Choice of $\alpha = 0.125$ and $\beta = 0.25$ is recommended according to [17].

$$\begin{aligned} ERTT &= (1-\alpha) \cdot ERTT + \alpha \cdot SRTT \\ DRTT &= (1-\beta) \cdot DRTT + \beta \cdot |SRTT - ERTT| \\ RTO &= ERTT + 4 \cdot DRTT \end{aligned} \quad (3)$$

When a timeout occurs, the RTO is doubled. An initial RTO of 1 s with a fallback to 3 s is recommended in theory [17]. Practical implementations, such as current Linux TCP, set the initial RTO to 3 s. Further, a fixed minimum RTO of 200 ms and maximum RTO of 120 s is hard-coded [18]. Recent practical studies using Linux and WLAN environments have shown that RTO calculation mostly results in values between 500 ms and minimum RTO [19].

Duplicate ACKs and fast retransmit: If a segment is received out of order the receiver immediately sends a *duplicate ACK (DupACK)* to the sender, in which it indicates the sequence number of the next expected byte again. After the sender has received three DupACKs, it performs a *fast retransit*, immediately sending the missing segment again.

Flow control: Hosts on each side of a TCP connection reserve an input buffer for the connection. If a host receives data that is accurate and in the right order it puts it in the input buffer. Thereby, TCP provides flow control as the sender is notified by the receiver how much free buffer space the receiver has available. This variable stored at the sender is denoted as *receive window* $rwnd$. Thus, the flow control ensures that the input buffer of a receiver does not overflow by restricting the transmission rate of the sender.

Basic congestion control: In addition, the sender adapts its transmission rate to the traffic load situation in order to avoid congestion. For this purpose, the sender maintains another variable called *congestion window* $cwnd$. Overall, the effective window size of unacknowledged data in flight may not exceed $min(cwnd, rwnd)$. For the congestion control (CC), an algorithm is used comprising three main phases: (1) slow start (SS), (2) congestion avoidance (CA), and (3) fast recovery (FR) [20], [21].

(1) Slow start (SS): At the beginning of a TCP connection, $cwnd$ is usually set to the maximum segment size (MSS), which severely limits the transmission rate. Therefore, the transmission rate is doubled per RTT if the transmitted segments were confirmed. If an RTO occurs, another variable called $ssthresh$ is set to $cwnd/2$. Subsequently, $cwnd$ is reset to 1 MSS and slow start begins again. If three DupACKs are received, a fast retransmit is performed and $cwnd$ is set to $ssthresh$. If the value of $cwnd$ equals $ssthresh$, SS ends and the CA phase begins.

(2) Congestion avoidance (CA): If $cwnd = ssthresh$, TCP only increases the value of $cwnd$ by 1 MSS per RTT (called additive-increase) [20]. If three DupACKs are received, $cwnd$ is halved (called multiplicative-decrease), fast retransmit is performed as described above, and the FR phase begins. Otherwise, the termination conditions are the same as in the SS phase.

(3) Fast recovery (FR): For each received DupACK, the value of $cwnd$ is increased by 1 MSS. If the ACK for the missing segment finally arrives, TCP enters the CA phase. In case of an RTO, TCP changes to the SS phase as described previously. As TCP CC basically adapts $cwnd$ by additive-increase (AI) and multiplicative-decrease (MD), it is often referred to as AIMD algorithm.

Congestion Control (CC) algorithms: FR is an optional TCP mechanism [20]. It was introduced in the CC algorithm *Reno*. It is the theoretical standard, superseding *Tahoe*, and implements the behavior as described above. Meanwhile, several variations like *NewReno* exist [22], [23]. In the following, some recent TCP CC algorithms, also available in current Linux kernels, are described.

BIC is a complex algorithm and a detailed description can be found in [7]. If $cwnd$ is greater than a constant threshold then BIC is used. Otherwise, Reno variants are used. After a segment loss, BIC enters the CA phase and adapts $cwnd$ by a binary search between a dynamic upper and lower limit. Linux used BIC as default in kernel versions 2.6.8 to 2.6.19, superseding NewReno [24].

CUBIC represents an enhanced version of BIC [8]. It simplifies the $cwnd$ adaptation of BIC by using a single cubic function in terms of the elapsed time since the last lost segment, so it does not require different phases. Linux uses CUBIC as default since kernel version 2.6.19 [24].

Vegas emphasizes packet delay rather than packet loss as a signal to determine the transmission rate [4]. $cwnd$ is adapted depending on the difference D between expected and current transmission rates during the CA phase. Thereby, RTT_{min} is the lowest RTT of the connection. Vegas further defines two thresholds a and b, for linearly increasing or decreasing $cwnd$ during the next RTT. Vegas' SS and FR phases are similar to that of Reno variants.

Veno integrates the proactive congestion detection of Vegas into the CA and FR phases of Reno [6]. To proactively avoid congestion, Veno adapts $cwnd$ less aggressively, if RTT difference exceeds a certain threshold.

Westwood introduces a "faster" recovery mechanism to avoid excessive reduction of $cwnd$ after three DupACKs [5]. It does so by adapting $cwnd$ according to an estimated available bandwidth BW_E.

Illinois implements all functions of the Reno variant NewReno but complements it by a Concave-AIMD (C-AIMD) algorithm [9]. C-AIMD uses segment loss as primary and queuing delay as secondary congestion signal to adapt $cwnd$.

See Table I for an overview of the different TCP congestion control algorithms and their $cwnd$ adaptation.

TABLE I: Comparison of TCP congestion control algorithms (CA=congestion avoidance, FR=fast recovery)

NAME	NETWORK CONDITIONS	MOST IMPORTANT CWND ADAPTATION MECHANISM
BIC	high bandwidth	Binary search increase algorithm
CUBIC	high bandwidth	$cwnd = C\ (t-K)^3 + cwnd_{max}$ $K = (cwnd_{max} \cdot b/C)^{1/3}$
Vegas	lossy medium	CA: $(D < a)$? $cwnd$++, $(D > b)$? $cwnd$-- $D = (exp - cur)RTT_{min}$ $exp = cwnd/RTT_{min}$ $cur = cwnd/RTT_{cur}$
Veno	lossy medium	CA: $(D < b)$? $cwnd$++ every RTT : $cwnd$++ every second RTT FR: $(D < b)$? $ssthresh = cwnd \cdot \frac{4}{5}$: $sstresh = cwnd/2$
Westwood	lossy medium	FR: (DupACKs received) ? $ssthresh = BW_E \cdot RTT_{min}/MSS$ $(cwnd > sstresh)$? $cwnd = ssthresh$
Illinois	high bandwidth low latency	FR: $cwnd$ -= $\beta \cdot cwnd$ CA: $cwnd$ += $\alpha \cdot cwnd$ $\alpha \propto$ RTT and $\beta \propto$ RTT

III. MISMATCH BETWEEN 802.11S MAC LAYER AND TCP

The 802.11 ARQ mechanism for automatic frame retransmissions after ACK timeout is used to effectively hide link-layer loss from TCP, as it operates invisible to the transport layer [10]. In an 802.11s network, ARQ is performed on a per-hop basis along a multi-hop mesh path. Figure 1 shows a TCP transmission with sender and receiver being connected by a multi-hop 802.11s network, i.e., intermediary nodes that each perform frame forwarding and ARQ at the MAC layer.

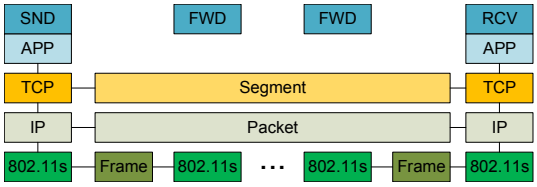


Fig. 1: TCP connection over multi-hop 802.11s path

TCP retransmissions are costly due to inherent $cwnd$ reduction (see Section II-B). Thus, transient link loss should be hidden as best as possible by 802.11 ARQ. However, if overall frame travel time along a mesh path exceeds the TCP connection's RTO, the current segment is retransmitted by the TCP sender at the transport layer and, again, needs to be forwarded along the whole path at the MAC layer. Now assume that the initial segment transmission arrives late at the TCP receiver due to per-hop ARQ and RTO occurs although the frame has not been dropped at the MAC layer yet. Then, ARQ induced by the TCP segment retransmission adds up to ARQ still performed for the initial transmission attempt. Moreover, this also accounts for the flow from receiver to sender in opposite direction, where ACK segments likewise cause ARQ and may contribute to exceeding RTO.

As described in Section II-B, RTT estimation and RTO calculation are updated only after TCP ACK reception or timeouts. Thus, a new RTO value will apply only for an upcoming segment transmission while ARQ of the previous attempt is still active in the path. In this scenario, ARQ of the segment transmission is not capable to achieve successful frame transmission within TCP RTO. It is performed redundantly to the TCP retransmission and, moreover, leads to additional contention on the wireless channel.

As a consequence, duration of consecutive ARQ retransmissions along a mesh path must not exceed the RTO of a TCP connection. Since RTO is dynamically calculated and not known on intermediary nodes in a mesh path, a first approach to adapt ARQ retry limits is to consider the fixed RTO_{min} of TCP implementations. RTO_{min} is the smallest possible RTO calculated for a TCP connection and represents the upper bound for link-layer frame travel time between a TCP sender and receiver before a TCP retransmission of the same segment can be triggered. Practical TCP implementations allow RTO values as low as 200 ms [18]. Recent studies with Linux TCP in WLAN setups have shown that RTO calculation mostly results in values between 500 ms and RTO_{min} [19]. By limiting the time of overall ARQ retries to stay within RTO_{min}, retransmissions will not be performed redundantly for the same segment even in a worst case. Nevertheless, following this constraint, ARQ retry limit should be kept as high as possible to hide link-layer loss from TCP and avoid unnecessary $cwnd$ reduction.

By an introductory thought experiment, we want to illustrate the effects of path length and ARQ retry limit on overall frame travel time in a simplified scenario. To approximate frame travel time along a multi-hop mesh path, including all forwarding steps by intermediary nodes, we multiply the duration of a single frame transmission (see Equation 1) by the hop count between TCP sender and receiver:

$$T_{MultiHop} = Hops \cdot (T_F + T_{ACK} + T_{ARQ}) \qquad (4)$$

Table II lists the EDCA parameters for default access category AC_{BE} and 802.11b/g mixed mode PHY [12], complying to the hardware we used to conduct real-world measurements, as described later in Section IV.

We distinguish three cases with different data frame sending rate and choice of the contention window CW, implying different channel conditions. Table III lists the parameters

TABLE II: EDCA parameters for 802.11b/g mixed mode PHY

Parameter	Value
$Slot\,Time\,(ST)$	20 µs
$SIFS$	10 µs
$AIFS[AC_{BE}]$	70 µs
CW_{min}	$15 \cdot ST$
CW_{max}	$1023 \cdot ST$

TABLE III: Parameters of thought experiment

Parameter	Best Case	Worst Case	Normal Case
$ACK\ Rate$ [Mbit/s]	6	6	6
$T_{tx_{ACK}}$ [µs]	50	50	50
$Data\ Frame\ Rate$ [Mbit/s]	54	6	24
T_{tx_F} [µs]	250	2070	530
CW	0	CW_{cur}	$CW_{cur}/2$

for a "best", "worst" ,and "normal" case. For all cases, we assume an 802.11 ACK frame size of 20 bytes and a data frame size of 1500 bytes. The given transmission rates result in different frame travel times ($T_{tx_{ACK}}$ and T_{tx_F}), according to [25]. In the "best" case, a transmitting node is instantly granted channel access. In the "worst" and "normal" case, CW is set to CW_{cur} and $CW_{cur}/2$, respectively, which depends on the ARQ retry count, as described in Section II-A. Although practically impossible, best and worst case denote lower and upper limits for overall frame travel time. With the "normal" case we want to approximate more realistic scenarios in between these limits.

We assume successful forwarding of a single frame belonging to a TCP data segment along a mesh path where a certain number of ARQ retries is required and performed equally per hop. Because of a lower probability for errors in TCP ACK transmission caused by their smaller size, and due to the existence of different TCP ACK transmission strategies, e.g., cumulative ACKs, we do not model the proportion of time needed for ACK delivery from TCP receiver to sender. Moreover, our simplified scenario does not yet account for interference, node mobility, 802.11 control messages, or competing application traffic. However, these effects only amplify the problem we want to illustrate.

Based on the assumptions described and using equations 1 and 4, we calculate overall frame travel time depending on hop count (1 to 5 hops) and ARQ retries (1 to 31, like the range configurable under Linux) for the three cases introduced, as shown in Figure 2. In all three diagrams, RTO_{min} is highlighted by a red line. Only an extract of the calculated results for 5 to 15 ARQ retransmissions is shown, as RTO_{min} is exceeded in cases (b) and (c) for all multi-hop paths within this retry span.

As shown in Figure 2 (c), a path length of 5 hops with 9 ARQ retries needed per hop already leads to an overall frame travel time of 236 ms in the "normal" case, exceeding TCP's RTO_{min} (200 ms). In the best case (Figure 2 (a)), an overall frame travel time of only 60 ms is needed even on a 5-hop path with 31 retries per hop (upper practical limit), still far below RTO_{min}. However, here we assume the highest frame sending rate and neglect exponential back-off completely. It becomes obvious that rate control and random CW selection have a major impact on overall frame transmission time. In reality, upper CW is doubled on ARQ retries until CW_{max}, increasing probability for selecting higher back-off times. Always selecting a CW of 0 is as unlikely as always taking the current upper limit like in the worst case, where RTO_{min} is exceeded already for 7 retries per hop after 4- and 5-hops.

Nevertheless, note that the difference between the worst case (6 Mbit/s, always CW_{cur}) and the normal case (24 Mbit/s, always CW_{cur}/2) is rather small, especially for paths with 3 and more hops. Here, only 2 to 3 additional retransmissions per hop are sufficient to exceed RTO_{min} in the normal case.

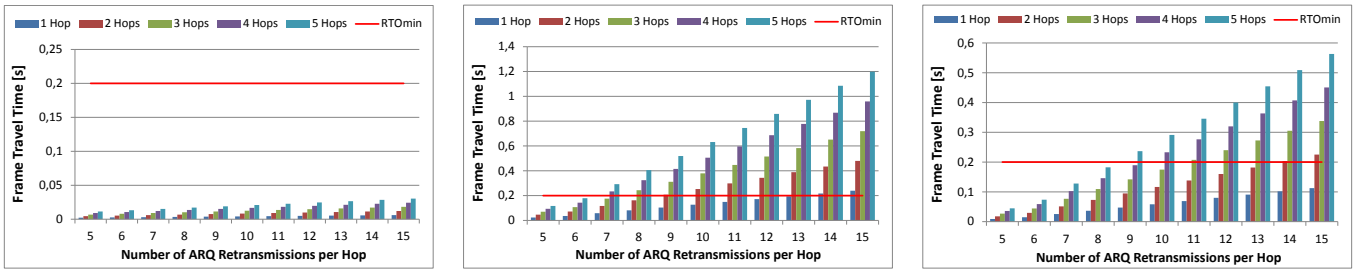
TCP tightly couples its flow and congestion control (CC), adapting its sending rate upon flow control events (RTO, DupACKs). Although many different TCP congestion control variants exist (see Section II-B), all are affected by underlying 802.11 flow control. We identified ARQ to have an important influence on overall TCP performance, even if CC variants designed for wireless environments are used. If executed within TCP RTO, ARQ effectively hides transient loss on the wireless channel. Otherwise, it involves the risk to interfere with TCP retransmissions especially in multi-hop flows, while being performed redundantly. This illustrates that careful TCP-aware tuning of ARQ retry limits should be pursued in practical WLAN mesh networks.

IV. Real-World Test Bed Measurements

A. Experimental Setup

To validate our assumptions, we conducted TCP throughput measurements in a real-world 802.11s mesh test bed for path lengths of 3 and 5 hops and different ARQ retry limits. To show that the problem described can be observed independently of the TCP congestion control (CC) variant in use, we compared multiple algorithms. For the different path lengths, we used 4 to 6 Raspberry Pi model B single-board computers [26] with (700 MHz ARMv6 CPU, 512 MB RAM, and Linux Kernel v3.12.43), integrating 802.11s reference implementation *open80211s* [14]. Every node was equipped with a WLAN USB adapter (Buffalo WLI-UC-GNM, Ralink chipset driver *rt2800usb*) configured to operate as 802.11s mesh point in 802.11g mode (54 Mbit/s PHY rate). We used default 802.11s parameters for peering and path selection. Further, 802.11 automatic rate control (ARC) was activated using the default algorithm *minstrel* [16]. We varied ARQ retry limit per node via user space tool *iw*. The Linux default setting allows for 7 frame retransmissions. Additionally, we evaluated retry limits 1 (practical minimum), 3, 5, 9, 16, and 31 (practical maximum). Considering the "normal" case of our thought experiment (see Section III), we chose a finer granularity for lower values of the retry limit to approach the theoretical threshold from which on ARQ may not be beneficial anymore. All nodes were located in the same room within radio range. Thus, RTS/CTS was left deactivated. We chose a channel not overlapping with other networks in our institute building.

As the applied WLAN adapters did not support transmission power reduction and a deployment range leading to a path length of 5 hops was not feasible, path enforcement was used to create a reproducible multi-hop environment. For this, userspace tool *iw* enables manual setup of multi-hop paths via *peer link block* functionality. On each node, we forbid link establishment to all peers except predecessor and successor in the desired multi-hop chain. An enforced path represents a worst case scenario, as the possibility for frame collisions is increased if all nodes operate within interference range. Furthermore, no parallel transmissions by non-adjacent nodes are possible, compared to a multi-hop topology caused by distance. However, this complies to our introductory scenario described in Section III more closely, as it leverages the chance that the allowed number of ARQ retries is actually performed.



(a) Best case (54 Mbit/s rate, $CW = 0$)

(b) Worst case (6 Mbit/s rate, $CW = CW_{cur}$)

(c) Normal case (24 Mbit/s rate, $CW = CW_{cur}/2$)

Fig. 2: Frame travel time depending on path length and ARQ retry limit

In both the 3- and 5-hop chain we generated a single TCP flow consisting of random data (1500 bytes MTU, 1460 bytes MSS) using performance measurement tool *iperf* (v2.0.5) [27]. For client and server, i.e., TCP source and destination, we always chose the nodes at both ends of the mesh path. In addition to CUBIC, being the default TCP CC algorithm, we evaluated BIC, Vegas, Veno, Westwood, and Illinois, as presented in Section II-B. We used default settings for iperf and Linux TCP.

For each combination of path length, ARQ retry limit, and CC algorithm, we conducted 10 consecutive measurement runs with a duration of 100 s each. The instantaneous TCP throughput measured by the iperf client was sampled every 5 s, resulting in 20 samples per run and 200 samples in total for each parameter set.

B. Results

Figure 3 shows the average throughput for each TCP variant in Mbit/s, depending on mesh path length and ARQ retry limit.

In contrast to AP-based infrastructure 802.11g networks, achieved data rates in the mesh test bed were considerably lower due to required frame forwarding, a common effect in wireless multi-hop networks. Since all nodes were within radio range and paths were enforced manually in our setup, as described in Section IV-A, channel contention increased with node count and path length. More specifically, in deployments with high contention it is likely that the number of ARQ frame retransmissions actually performed comes close or equal to the configured retry limit, like in our scenario given in Section III.

For both the 3- and 5-hop case, as shown in Figure 3, the TCP throughput curves exhibit a concave shape and thus a similar dependence on the configured ARQ retry limit. All TCP variants achieved highest average throughput for retry limits between 5 and 9, which includes the Linux default setting of 7 allowed retransmissions. When setting retry limits to less than 5 in both multi-hop setups, transient frame loss could not be hidden effectively from TCP and led to a higher number of TCP retransmissions and reduced sending window size caused by congestion control.

For retry limits higher than 9, the 3-hop flow throughput dropped below its peak for all TCP variants. In the 5-hop flow, the drop in TCP throughput caused by a too high retry limit per node could be observed earlier, i.e., after 5 to 7 allowed frame retransmissions. In our suggested adjustment range of 5 to 9 allowed ARQ retransmissions, throughput improvement for the best algorithm in the 3-hop setup was 11 % (Veno). The highest throughput difference amounted to 38 % (Westwood). For the 5-hop setup, the best algorithm revealed a throughput improvement of 8 % (CUBIC), whereas the highest difference was measured with 35 % (Vegas). Thus, the expected dependence between path length and ARQ retry limit can be demonstrated, as described in Section III.

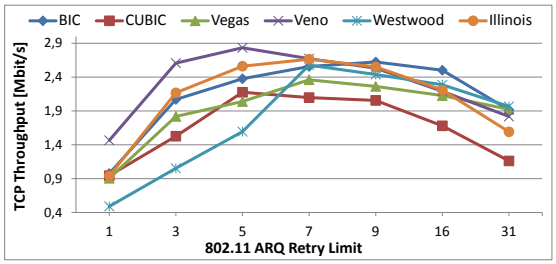
In the 5-hop case, the observed throughput for CUBIC deviates considerably from the other TCP variants. We checked, that these results were neither caused by a faulty mesh path setup nor significant differences in effective TCP window size or wireless channel conditions. For that, we compared TCP traffic captures and 802.11s log files, including physical link rates and mesh path metrics. Nevertheless, the same dependence on ARQ retry limit as for the other TCP algorithms is clearly visible.

C. Discussion

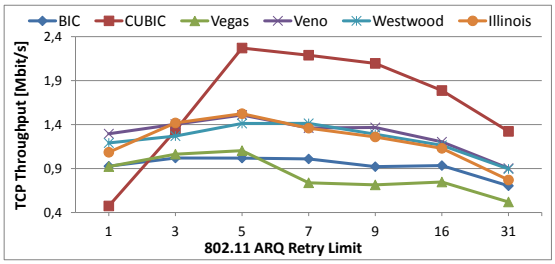
Regarding our results, limiting retries below a certain number is not beneficial. As a general conclusion, ARQ should be adapted considering a mandatory minimum of retransmissions to be allowed. Since 802.11 back-off for channel access increases exponentially on retries, a small frame retransmission number still imposes a tolerable delay but already contributes considerably to hiding transient errors. Summarized, a too small retry limit per node leads to frame loss not being hidden effectively from TCP whereas too high limits lead to overhead, depending on path length, which reduces overall throughput. Thus, ARQ retry limit should be tuned at run-time, keeping it as high as possible without exceeding the retransmission timeout (RTO) of possible TCP flows in the network.

A mesh node can be either TCP end point or serve as forwarding node for other TCP flows. While the first case can be determined easily by accessing local socket information (i.e., using tool *ss* under Linux), identifying the latter would require deep packet inspection, as frames are directly forwarded at the MAC layer. Instead, we want to use existing user interfaces to local mesh information and adapt the retry limit for the worst case TCP flow possibly passing a node.

The 802.11s standard's path selection metric ALM expresses the cumulative airtime cost for frame delivery over a mesh path when choosing a certain next hop node (see Equation 2). Due to its cumulative nature, ALM is proportional to the theoretical multi-hop frame travel time, as given by Equations



(a) 3-Hop TCP Flow



(b) 5-Hop TCP Flow

Fig. 3: TCP throughput depending on path length, congestion control variant, and ARQ retry limit

4 and 1, respectively. Thus, it provides a suitable MAC layer RTT estimation for our ARQ tuning approach.

Retrieval of ALM values and changing the global ARQ retry limit on a node is possible by using existing interfaces of the network stack, without modifying MAC or TCP implementations. Each 802.11s node maintains links to its direct neighbors in a *peer list* and paths to certain target nodes in a *path table*. Linux systems include userspace interfaces, such as *netlink*, to provide easy access these data structures and other functions of the MAC layer within the kernel [14].

For each mesh path, the address of the next hop neighbor and the resulting cumulative ALM cost is stored. From a node's perspective, the highest frame travel duration will occur on the most costly global path it is intermediary node of, effectively representing the network diameter. Adapting ARQ retry limit to this path will also prevent redundant frame retransmissions on other possible TCP flows forwarded by the node. We propose a heuristic to estimate the associated path cost, by which a node combines the two highest ALM values towards target nodes, that are not reached via the same next hop node.

To effectively prevent redundant retransmissions, duration of frame forwarding along the assumed network diameter should be kept smaller than RTO_{min}, which is as low as 200 ms in practical TCP implementations [18]. Based on the associated multi-hop frame travel time $T_{MultiHop_{dia}}$, being proportional to its airtime cost ALM_{dia}, and awareness of practical RTO_{min}, the need for ARQ retry limit increase or decrease can be derived, as shown in Equation 5:

$$RTO_{min} > T_{MultiHop_{dia}} \propto ALM_{dia} \tag{5}$$

As next steps, we will evaluate the practical correlation of cumulative airtime cost, represented by ALM, and actual multi-hop frame travel time, measured in real-world setups. Using these insights, we will implement our proposed TCP-aware ARQ tuning approach that solely relies on standard ALM as MAC-layer RTT indicator.

V. Related Work

Numerous works exists in the field of optimizing TCP for WLAN infrastructure and mesh networks. The survey in [3] categorizes existing TCP optimizations into sender-side, sender- and receiver-side, proxy-based and MAC/routing protocol changes, with most being sender-side solutions. Conclusions are, that control traffic must be kept low and TCP RTO events must be minimized in wireless environments. Especially the latter is a main focus of our work.

Authors in [11] conduct model-based studies on the 802.11 DCF. They confirm that packet drop decreases with a higher number of frame retransmissions whereas delay increases. Thus, ARQ retry limits have to be tuned carefully.

In [10] authors alleviate the challenges of cross-layer collaboration between 802.11 ARQ link-layer retransmissions and TCP congestion control. Simulations are performed for an AP-based WLAN with different error rates, ARQ retry limits and multiple TCP flows. They conclude that, on the one hand, ARQ effectively shields wireless loss from TCP up to a certain error degree. On the other hand, ARQ increases RTT and thus may interfere with TCP congestion control variants that are based on RTT estimates. Furthermore, ARQ may fail to shield wireless losses in extreme network scenarios with congestion and then imposes overhead. All in all, to prevent redundant retransmissions on both layers and maximize ARQ efficiency, cross-layer mechanisms should be pursued.

Subramanian et al. [28] propose a MAC-layer modification and a loss-tolerant variant of TCP. Claiming that ARQ may lead to delay variations and reduced TCP throughput, they severely limit ARQ retries while introducing forward error correction (FEC), leading to good results in combination with their TCP variant. In contrast, we want to leave TCP unmodified and instead tune ARQ retry limit to be as high as possible without causing RTO of TCP flows.

In [29] authors investigate the effects of payload length and ARQ retry limit variation on achievable TCP throughput. The coexistence of voice and data traffic in a common infrastructure WLAN is evaluated for different noise levels via Qualnet simulator. Authors conclude that loss shielding by means of a higher ARQ retry limit comes at the expense of reduced throughput. In case of lossy channels or networks with high node count and contention, ARQ retransmissions are vital for many loss-intolerant applications whereas multimedia applications can often tolerate higher loss rates.

Research in [30] describes a phase type discrete time Markov chain to derive a probability density function for the number of ARQ retries required for successful TCP segment delivery over multi-hop wireless networks. Authors investigate success probability for different ARQ policies and link error rates.

Barman et al. [31] analyze the tuning of transmission power, FEC, and ARQ retry limit. They conclude that all mechanisms decrease loss rate but a compromise between their costs and benefits must be found.

To the best of our knowledge, no work exists considering an adaptive, TCP-aware tuning of ARQ retry limits in a multi-hop WLAN mesh network. Especially, no solution has been presented that utilizes IEEE 802.11s standard features.

VI. Conclusion

In this paper we illustrate the effects of 802.11 ARQ retry limit variation on TCP performance in 802.11s multi-hop WLAN mesh networks. To outline the mismatch between both layers, we show in an introductory thought experiment that redundant retransmissions may lead to undesirable overhead, if ARQ and TCP flow control are not coordinated. To validate our assumptions, we conduct several measurements in a real-world test bed. Results show that there is a significant dependence between mesh path length and ARQ retry limit, regardless of the TCP congestion control variant in use. We investigate throughput of different TCP variants using path lengths of 3 and 5 hops and retry limits between 1 and 31.

When setting retry limits to less than 5 in both multi-hop setups, transient frame loss could not be hidden effectively from TCP and led to a higher number of TCP retransmissions and reduced sending window size caused by congestion control. For retry limits higher than 9, the 3-hop flow throughput dropped below its peak for all TCP variants. In the 5-hop flow, a drop in TCP throughput could be observed already for limits higher than 5. In our suggested adjustment range of 5 to 9 allowed ARQ retransmissions, throughput improvement for the best algorithm in the 3-hop setup was 11 % (Veno). The highest throughput difference amounted to 38 % (Westwood). For the 5-hop setup, the best algorithm revealed a throughput improvement of 8 % (CUBIC), whereas the highest difference was measured with 35 % (Vegas).

Moreover, we suggest a distributed TCP-aware ARQ retry limit adaptation, solely based on the 802.11s standard's default path selection metric ALM serving as MAC-layer RTT indicator. To implement our approach, we will assess the practical correlation of ALM and actual multi-hop frame travel time, measured in an extended real-world setup.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG), Research Training Group 1424 (MuSAMA) for their financial support.

References

[1] M. Portmann and A. A. Pirzada, "Wireless mesh networks for public safety and crisis management applications," *Internet Computing, IEEE*, vol. 12, no. 1, 2008.

[2] "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, 2012.

[3] B. Francis, V. Narasimhan, A. Nayak, and I. Stojmenovic, "Techniques for enhancing TCP performance in wireless networks," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*. IEEE, 2012.

[4] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, 1995.

[5] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM, 2001.

[6] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 2, 2003.

[7] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004.

[8] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS OS Review*, vol. 42, no. 5, 2008.

[9] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6, 2008.

[10] P. Dalal, M. Sarkar, K. Dasgupta, and N. Kothari, "Link Layer Correction Techniques and Impact on TCP's Performance in IEEE 802.11 Wireless Networks," *Communications and Network*, vol. 2014, 2014.

[11] H. M. Alsabbagh, C. Jianping, and X. Youyun, "Influence of the limited retransmission on the performance of WLANs using error-prone channel," *Int'l J. of Communications, Network and System Sciences*, vol. 1, no. 01, 2008.

[12] M. Burton and G. Hill, "802.11 Arbitration," *White Paper, Certified Wireless Network Professional Inc., Durham, NC*, 2009.

[13] R. C. Carrano, L. C. S. Magalhães, D. C. M. Saade, and C. V. N. Albuquerque, "IEEE 802.11s multihop MAC: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 1, 2011.

[14] "open80211s." [Online]. Available: http://open80211s.org/open80211s/

[15] R. Garroppo, S. Giordano, and L. Tavanti, "A joint experimental and simulation study of the IEEE 802.11s HWMP protocol and airtime link metric," *Int. Journal of Communication Systems*, vol. 25, no. 2, 2012.

[16] "Minstrel RCA." [Online]. Available: http://wireless.kernel.org/en/developers/Documentation/mac80211/RateControl/minstrel

[17] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298 (Proposed Standard), Internet Engineering Task Force, Jun. 2011.

[18] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP," in *In Proceedings of USENIX*. Springer, 2002.

[19] S. Fuicu and A. Harman, "An Experimental Approach into TCP Congestion Mechanism over a WLAN Network," in *Advances in Wireless Sensor Networks 2013, Conference Proceedings*, 2013.

[20] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681 (Draft Standard), Internet Engineering Task Force, Sep. 2009.

[21] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001 (Proposed Standard), Internet Engineering Task Force, Jan. 1997, obsoleted by RFC 2581.

[22] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582 (Proposed Standard), Internet Engineering Task Force, Apr. 2012.

[23] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018 (Proposed Standard), Internet Engineering Task Force, Oct. 1996.

[24] S. Arianfar, "TCP's Congestion Control Implementation in Linux Kernel," in *Proceedings of Seminar on Network Protocols in Operating Systems*, 2012.

[25] "Packet transmission time in 802.11." [Online]. Available: https://sarwiki.informatik.hu-berlin.de/Packet_transmission_time_in_802.11

[26] "Raspberry Pi." [Online]. Available: http://www.raspberrypi.org/

[27] "iperf." [Online]. Available: http://sourceforge.net/projects/iperf/

[28] V. Subramanian, K. Ramakrishnan, and S. Kalyanaraman, "Experimental study of link and transport protocols in interference-prone wireless LAN environments," in *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*. IEEE, 2009.

[29] S. Choudhury, I. Sheriff, J. D. Gibson, and E. M. Belding-Royer, "Effect of payload length variation and retransmissions on multimedia in 802.11a WLANs," in *Proceedings of the 2006 international conference on Wireless communications and mobile computing*. ACM, 2006.

[30] T. Issariyakul and E. Hossain, "Analysis of end-to-end performance in a multi-hop wireless network for different hop-level ARQ policies," in *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, vol. 5. IEEE, 2004.

[31] D. Barman, I. Matta, E. Altman, and R. El Azouzi, "TCP optimization through FEC, ARQ, and transmission power tradeoffs," in *Wired/Wireless Internet Communications*. Springer, 2004.