A Practical Adaptive Pacing Scheme for TCP in Multihop Wireless Networks

Sherif M. ElRakabawy, *Member, IEEE*, and Christoph Lindemann, *Senior Member, IEEE*

Abstract*—We introduce and evaluate a feasible end-to-end congestion control algorithm for overcoming the severe deficiencies of TCP in IEEE 802.11 multihop wireless networks. Our approach, which we denote as TCP with Adaptive Pacing, implements rate-based scheduling of transmissions within the TCP congestion window. The TCP source adaptively sets its transmission rate using an estimate of the current out-of-interference delay and the coefficient of variation of recently measured round-trip times. TCP-AP retains the end-to-end semantics of TCP and does neither rely on modifications at the routing or the link layer, nor requires cross-layer information from intermediate nodes along the path. Opposed to previous proposals that build on network simulators, we implement and evaluate our approach in a real wireless mesh testbed comprising 20 nodes. In a comprehensive comparative performance study using our testbed we show that, depending on the current network state and traffic patterns, TCP-AP achieves up to ten times more goodput than TCP NewReno, provides excellent fairness, and is highly responsive to changing network traffic conditions.

Index Terms*— Analysis and design of transport protocols, end-to-end congestion control, IEEE 802.11 wireless mesh testbeds, performance evaluation.

I. Introduction

For several years now, multihop wireless networks have been within the focus of research in the networking community. While most of the research in this area is still conducted in network simulators such as [12] and [27], the trend is increasingly moving towards deploying such networks in reality. Examples include MIT Roofnet [3], TFA-Rice ([4], [14]), All-wireless Office [11], and Freifunk [26], which have proven the feasibility of multihop wireless networks.

Within several research topics in multihop wireless networks, TCP performance has acquired great attention. Multihop wireless networks using IEEE 802.11 namely possess several properties, which are different to the wired Internet for which the widely deployed TCP NewReno implementation has been optimized. Opposed to wired networks, in IEEE 802.11 networks, the wireless channel is a scarce resource shared among nodes within their radio range. Furthermore, channel capture, hidden and exposed terminal effects, and the IEEE 802.11 medium access control constitute features of multihop wireless networks not present in a wired IP network. In fact, for multihop wireless networks, most losses experienced by TCP are due to packet drops at link layer and not due to buffer overflow [13]. Furthermore, since the congestion control of TCP NewReno is based on lost data packets, the size of its congestion window is overshooting rather than proactively sensing incipient congestion by monitoring the network traffic. Because of all these features, TCP NewReno possesses quite poor performance in multihop wireless networks, as well as exhibits severe unfairness among competing TCP flows.

Several approaches (e.g. [10], [13], [22] and [24]) have been proposed for improving TCP performance in multihop wireless networks. Unfortunately, such approaches have been only designed and evaluated in network simulators. Simulators often rely on optimistic assumptions compared to the real world and, thus, do not always deliver accurate results. Moreover, many physical measures in reality, such as the distance between nodes in a network, can be simply inquired in simulations, but are not available at nodes in reality due to the absence of global knowledge.

For the best of our knowledge, we are the first to introduce and evaluate a feasible, and TCP-compatible, end-to-end approach for improving TCP goodput and fairness in a real wireless mesh testbed rather than in simulators. We build on previous work [10] and introduce a novel congestion control algorithm for TCP over real multihop IEEE 802.11 networks, implementing rate-based scheduling of transmissions within the TCP congestion window. In our approach, a TCP sender adaptively sets its transmission rate using an estimate of the current out-of-interference delay and the coefficient of variation of recently measured round-trip times. The out-of-interference delay describes the time elapsed between transmitting a TCP packet by the TCP source node i and receiving the packet at the node which lies beyond the range in which collisions with node i may occur. The novel TCP variant, which is fully TCP-compatible, is denoted as TCP with Adaptive Pacing (TCP-AP). We incorporate an advanced end-to-end algorithm in TCP-AP, which estimates the current out-of-interference delay and derives a suitable adaptive pacing rate accordingly.

Opposed to previous proposals for improving TCP over multihop IEEE 802.11 networks, TCP-AP retains the end-to-end semantics of TCP and does neither rely on modifications at the routing or the link layer nor requires cross-layer information from intermediate nodes along the path. In order to evaluate the performance of TCP-AP versus the widely deployed TCP NewReno, we build a miniaturized wireless mesh testbed comprising 20 nodes and variable signal attenuators. A comprehensive performance study using the miniaturized wireless mesh testbed shows that, depending on the current network state and traffic patterns,

Manuscript received July 11, 2009; accepted November 1, 2010. This work was supported in part by the German Research Council (DFG) under grant Li 645/18-2.

Sherif M. ElRakabawy, the corresponding author, is with the University of Leipzig, Department of Computer Science, Johannisgasse 26, D-04103, Germany (email: sme@eecs.berkeley.edu). Part of the research in this paper was conducted while author was visiting researcher with the EECS Department at the University of California, Berkeley.

Christoph Lindemann is with the University of Leipzig, Department of Computer Science, Johannisgasse 26, D-04103, Germany (email: cl@rvs.informatik.uni-leipzig.de).

TCP-AP achieves up to ten times more goodput than TCP NewReno, provides excellent fairness, and is highly responsive to changing network traffic conditions.

The remainder of this paper is organized as follows. Section II summarizes related work on TCP for multihop wireless networks and Section III describes the miniaturized wireless mesh testbed used for evaluating our approach. In Section IV, we introduce the novel TCP congestion control algorithm and describe its implementation in the Linux operating system. A comprehensive performance study of TCP-AP versus TCP NewReno using our wireless mesh testbed is presented in Section V. Finally, concluding remarks are given.

II. Related Work

Wei et al. [23] and ElRakabawy et al. [10] showed that pacing for TCP can improve goodput and fairness both for wired as well as for multihop wireless networks. The authors in [23] found out that pacing yields reduced burstiness of traffic, increased synchronization among flows as well as fragmented SACK blocks in a flow. Although the authors point out to a possible decrease in performance when competing with aggressive flows, they conclude that pacing brings significant benefits for many applications. In [10], we introduced a preliminary version of TCP with Adaptive Pacing and evaluated it using ns-2 [12]. The results showed that adaptive pacing yields significant performance improvement with respect to standard TCP.

As opposed to [23], our approach is tailored for multihop wireless networks and not for the Internet, which possesses fundamentally different characteristics. We build our approach on [10]. Since the algorithm in [10] is optimized for simulations, we introduce an improved algorithm which exploits end-to-end information to derive a suitable adaptive pacing rate. Furthermore, we evaluate our approach in a real mesh testbed rather than in simulations.

Fu et al. [13] pointed out the hidden terminal problem in multihop wireless networks and experimentally showed that for a chain topology the optimal windows size, for which TCP achieves best throughput, is roughly given by 1/4 of the hop count of the path. Xu et al. [24] proposed the neighborhood RED (NRED) scheme at routing layer to throttle TCP senders when incipient congestion is detected, by purposely dropping TCP packets at intermediate nodes. In [20], Nahm et al. analyzed the performance of TCP in multihop wireless networks by investigating the interaction of TCP with the routing and link layers. They proposed a fractional window increment scheme to prevent TCP's congestion window from overshooting to improve goodput.

Our approach differs fundamentally from [13], [20], and [24]. TCP-AP just requires slight modifications at the transport layer and does neither require modifications at the routing layer as [24] and link layer as [13], nor extra communication between neighboring nodes. As a consequence, TCP-AP can be incrementally deployed. Furthermore, beyond [20], TCP-AP integrally improves both fairness and goodput.

Sundaresam et al. [22] introduced ATP, and Anastasi et al. proposed TPA [1], which are both novel transport protocols for multihop wireless networks. ATP employs pure rate-based transmission of packets, where the transmission rate is determined using feedback from intermediate nodes along the path. The authors propose to dynamically adjust the transmission rate according to the maximum packet queuing delay on intermediate nodes along the network path. TPA uses a similar congestion control algorithm like TCP, in such that packets are transmitted window-based. By limiting the congestion window size and thinning the ACK stream using delayed ACK mechanisms, a less aggressive data transfer is achieved.

In contrast to [22], TCP-AP retains the end-to-end semantics of TCP without relying on any cross-layer information from intermediate nodes along the path. Furthermore, opposed to [1] and [22], TCP-AP is fully TCP compatible and thus can be incrementally deployed.

Opposed to [13], [22] and [24], our approach is tailored and evaluated in a real mesh testbed rather than simulations. This makes our approach feasible and improves the reliability of the acquired results.

Several studies proposed clamping the TCP congestion window in order to reduce packet bursts from congesting the wireless channel. Casetti et al. proposed Westwood+ [5], a sender-side modification of TCP's congestion control algorithm over lossy wireless links. Westwood+ monitors the arrival rate of TCP ACKs and adjusts the congestion window and slow start threshold accordingly. In [19], Kawadia et al. investigated the performance of TCP in an indoor testbed consisting of 7 off-the-shelf laptops. Consistent with [13], the authors found out that clamping the size of the congestion window can improve TCP throughput in multihop wireless networks. Koutsonikolas et al. reported similar results in [9], where they investigated the impact of the congestion window size on TCP throughput. In [2], the authors provided a receiver-side control algorithm denoted as CLAMP to control the TCP receiver's advertised window limit. They showed that CLAMP can provide a fairer allocation of the bandwidth than TCP NewReno.

In contrast to [5], [9], and [19], we do not only address TCP throughput maximization, but also fairness between competing flows. While Westwood+ [5] and CLAMP [2] are mainly tailored for one-hop wireless communication with lossy links, TCP-AP addresses the specific problems of TCP over IEEE 802.11 multihop networks, e.g. hidden terminal effects. Clamping the window size like in [9] and [19] does not adapt the transmission rate of TCP according to the current network state, but rather sets an upper bound for the congestion window. Such an upper bound does not prevent packet bursts from being transmitted back-to-back, which is the main reason for the unfairness problem of TCP in multihop networks.

III. The Miniaturized Wireless Mesh Testbed

To study the performance of the enhanced TCP-AP algorithm in reality and compare it to the widely deployed TCP NewReno, we built up a miniaturized wireless mesh testbed. The testbed, which is depicted in Fig. 1, comprises 20 wireless mesh nodes. Each node consists of a low-cost PC with an Intel Celeron 3.2 GHz processor and an IEEE 802.11b wireless PCI card, which is connected to a variable signal attenuator and a 2.1dBi low-gain antenna. Using the variable attenuators, the signal power of the wireless PCI cards can be adaptively shrunk in order to limit the

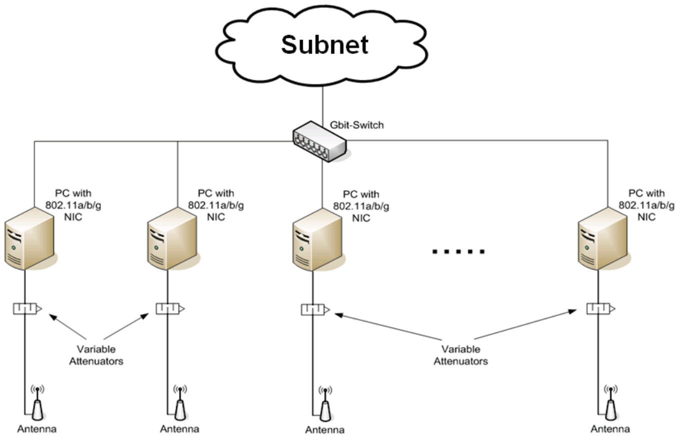

Fig. 1: Miniaturized Wireless Mesh Testbed

TABLE I: HARDWARE AND SOFTWARE COMPONENTS OF THE MINIATURIZED TESTBED

Hardware	
Component	Description
PC	Siemens ESPRIMO P2510 Celeron 3,2 GHz, 512 Mbytes RAM, 80 Gbyte HDD
Wireless NIC	Netgear IEEE 802.11b wireless PCI card WG311T with Atheros chipset
Variable attenuator	Broadwave 751-002-030 variable attenuator, attenuation range 0-30dB in 1 dB steps
Coaxial cable	7m aircell5 coaxial-cable, 50 Ohm with SMA / RPSMA connectors, attenuation: -0.53 dB/m
Antenna	Maldol mini 2.1 dBi antenna with magnetic mount and 3m SMA cable
Software	
Component	Description
Operating System	SuSE Linux 10.2 with custom kernel version 2.6.18 with high resolution subsystem patch
Wireless NIC driver	Madwifi Linux kernel device driver for Atheros chipsets version 0.9.2
Multihop routing protocol	OLSR for Linux version 0.4.10 with ETX support

maximum transmission range of each node. Thus, similar to [8], large wireless mesh networks spanning a few kilometers can be scaled down to a few meters, making quick topology and parameter modifications for efficient evaluation of network protocols possible.

Testbed nodes run a SuSE Linux 10.2 operating system with a custom-compiled kernel version 2.6.18 with the high-resolution timer subsystem patch [15]. As driver for the wireless PCI cards, we employ the Linux Madwifi kernel device driver version 0.9.2 for Atheros chipsets. We employ the Optimized Link State Routing Protocol (OLSR) version 0.4.10 for Linux [6] [25] for multihop routing, which incorporates the ETX metric [7] for selecting routes based on the current loss probability of the links.

All wireless nodes further possess a Gigabit Ethernet NIC which are connected to the subnet of the department through a Gigabit switch. This allows a remote management of the wireless nodes from any wired host in the subnet. Hence, wireless experiments can be started and stopped from a remote computer and traces can be copied and evaluated through the wired network. Table I shows a detailed description of hardware and software components of the miniaturized testbed.

IV. PUTTING TCP-AP INTO PRACTICE

A Discrepancy between Simulation and Reality

Simulation is still the most common way for designing and evaluating new protocols for research in multihop wireless networks. Due to the higher complexity and expenses of real multihop wireless networks, many researchers choose to implement and evaluate newly designed protocols in simulators such as ns-2 [12] and Qualnet [27]. However, simulators often rely on optimistic assumptions compared to the real world and, thus, do not always deliver accurate results. Moreover, many physical measures in reality, such as the distance between nodes in a network, can be simply inquired in simulations, but are not available at nodes in reality due to the absence of global knowledge. Thus, previous approaches such as [10], [13] and [22], which rely on such measures in order to be feasible, cannot be simply deployed in real multihop wireless networks.

The current adaptive pacing approach, as given in [10], is optimized for a specific, simulation-based ratio between the transmission range and carrier sensing/interference range of the nodes. Specifically, the 4-hop propagation delay [10] [13] only corresponds to a ratio between the transmission range and the carrier sensing/interference range which is equal to 250m/550m. In real life, such ranges strongly depend on a number of variable physical parameters such as current transmission power and channel interference, and thus typically fluctuate over time. Consequently, in a real multihop wireless environment, the suitable number of hops required as a delay for the adaptive pacing rate is variable and is not bound to a specific number, such as 4, as given in [10] and [13]. In fact, the suitable propagation delay required by TCP to consider the spatial reuse constraint of IEEE 802.11, which we denote as out-of-interference delay (*OID*), changes over time depending on current network conditions. Thus, the main challenge for adopting the adaptive pacing approach in [10] into practice is to develop an algorithm for adaptively determining the out-of-interference delay. The complexity lies in the fact that, other than in simulation, parameters needed for determining such delay cannot be simply inquired by the TCP source node. Instead, the delay must be approximated by means of measures which are available at the TCP source node.

Subsequently, we discuss the main deficiencies of standard TCP variants in multihop wireless networks and how our approach overcomes such deficiencies. Furthermore, we give insight into the interaction between the carrier sensing range and hidden terminals in a chain of nodes. We show how to exploit information about the carrier sensing range to derive a suitable estimation of the out-of-interference delay and how to use such a delay for a computation of an enhanced adaptive pacing rate.

B Rate-based Congestion Control

Several researchers identified the interaction of TCP with the underlying routing and link layers as the key factor for the poor performance of TCP in IEEE 802.11 multihop wireless networks. If we neglect mobility-related problems of TCP in such networks [16], most important deficiencies of TCP arise from TCP's congestion control algorithm. First, TCP's window-based congestion control leads to packet bursts when received acknowledgments trigger the transmission of several data packets, e.g., when receiving a cumulative ACK. Due to the spatial reuse constraint of the wireless channel in IEEE 802.11 multihop wireless networks, concurrent nodes in a chain cannot transmit simultaneously without causing collisions. Thus, packet bursts result in increased contention on the wireless channel. This link layer contention may lead to packet drops due to the hidden and exposed terminal problems [10] [13]. Second, TCP's congestion control algorithm relies on packet losses as indication of congestion and, thus, provokes losses in order to identify spare bandwidth. In IEEE 802.11 multihop wireless networks, this behavior results in increased congestion, causing significant performance degradation for TCP [13]. Recall that network congestion often triggers (false) route failures, even in static wireless networks, since the routing protocol cannot distinguish between a packet loss due to congestion and a packet loss due to a broken route.

To overcome both deficiencies stated above while preserving the TCP compatibility, our protocol TCP-AP incorporates a rate-based transmission algorithm into TCP's window-based congestion control. The problem of packet bursts is solved by spreading the transmission of successive data packets according to the computed transmission rate, which accounts for the spatial reuse constraint in IEEE 802.11 multihop wireless networks. Furthermore, by proactively identifying incipient congestion, i.e. before congestion-related losses actually occur, TCP-AP is able to adjust the transmission rate and, hence, reduce contention at link layer. In contrast to TCP Pacing for the Internet [23], where the transmission of a window of packets is evenly spread over the duration of a round trip time (RTT), our approach schedules the transmission of packets based on both the size of the congestion window and the computed transmission rate. As long as the size of the congestion window is larger than the number of packets in flight, new packets are scheduled for transmission according to the current transmission rate.

C Identification of Incipient Congestion

Due to its end-to-end semantics, TCP's congestion control algorithm is based on the measurement of round trip times (RTT) and packet loss. In fact, in current TCP variants such as Reno and NewReno, the actual identification of congestion is solely laid upon the observation of packet loss. Therefore, standard TCP increases the load issued into the network until a packet loss is detected, where such a packet loss identifies congestion.

TCP-AP incorporates a congestion control algorithm which identifies high contention on the network path of the TCP connection, and proactively throttles the transmission rate before losses occur. That is, as congestion increases, the variance of round trip times increases correspondently, indicating high link contention. Hence, TCP-AP uses the coefficient of variation of recently measured round trip times, cov_{RTT}, as key measure for the degree of the contention on the network path. This measure is given by:

$$cov_{RTT} = \frac{\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(RTT_i - \overline{RTT})^2}}{\overline{RTT}} \tag{1}$$

Here, N is the number of considered RTT samples, $\overline{RTT}$ is the mean of the samples, and RTT_i denotes the value of the i-th RTT sample. The coefficient of variation cov_{RTT} can be obtained purely end-to-end without provoking congestion or packet losses.

D The Spatial Reuse Constraint

Besides the measure of contention on the network path, TCP-AP also accounts for the spatial reuse constraint of IEEE 802.11 multihop wireless networks. That is, due to the hidden terminal effect and the absence of perfect scheduling at link layer, concurrent nodes in a chain cannot transmit simultaneously without causing collisions. A crucial factor that has a significant impact on the spatial reuse constraint of a multihop wireless network is the carrier sensing range of wireless nodes. Physical carrier sensing is a mechanism incorporated in IEEE 802.11 [28], by which a wireless node senses the medium before it transmits a packet. Only if the sensed signal power is below a certain threshold, denoted as carrier sense threshold T_{cs}, does the node initiate a transmission. As the radio signal of a node attenuates with the distance, the range in which the node can sense the transmission of another node is limited. The carrier sensing range defines the range in which the current transmission of a node can be sensed by other nodes. The key role of the carrier sensing range lies in determining which hops on a chain of nodes are prone to be potential hidden terminals. That is, nodes which operate beyond each other's carrier sensing range on a chain comprise mutual hidden terminals. Thus, the transmission of each of the nodes cannot be sensed by the other node, respectively, resulting increased collision at link layer. Fig. 2 shows a chain of 8 nodes. Assume a TCP connection is running between node 1 as a TCP source and node 8 as a TCP destination. In this chain, nodes 1 and 4 comprise mutual hidden terminals, since both nodes operate beyond each other's carrier sensing ranges. In this case, node 4 cannot sense the transmission from node 1 to node 2 and thus may transmit packets to node 5, resulting collisions with the ongoing transmission between nodes 1 and 2.

From the point of view of the TCP source, i.e. node 1, the first node which is positioned right at the border of its carrier sensing range, node 4 in this case, is the first node that comprises a potential hidden terminal. This means that collisions can be avoided if node 1 defers its transmission until node 4 finishes its transmission to node 5. Note that which node comprises the hidden terminal is mainly determined by the carrier sensing range and does not have to be the 4th node on the chain as given in Fig. 2. This means that the hidden terminal varies with varying carrier sensing range. Let node i be the TCP source node and node (i+x), $x \geq 2$, be the hidden terminal to node i. We refer to the time elapsed between transmitting a TCP packet by the TCP source node i and receiving the packet at node (i+x+1) as the out-of-interference delay (*OID*).

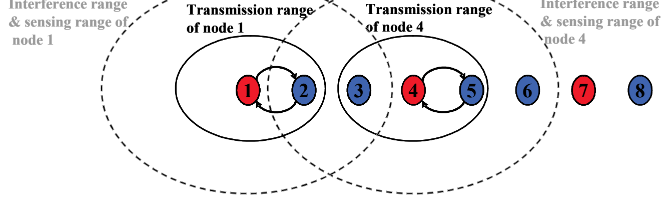
Interference range & sensing range of node 1
Transmission range of node 1
Transmission range of node 4
Interference range & sensing range of node 4
1
2
3
4
5
6
7
8

Fig. 2: Spatial reuse constraint: Hidden terminals in a chain are dependent on current carrier sensing range

The challenge is to approximate *OID* by determining the hidden terminal for the TCP source node. In order to identify the hidden terminal for the TCP source node, we have to determine the carrier sensing range in terms of number of hops. The next node right at the border of the carrier sensing range comprises the potential hidden terminal. Subsequently, we introduce the Adaptive Out-of-Interference Delay approach, which incorporates an effective way for estimating the carrier sensing range of the TCP source node and approximating the out-of-interference delay accordingly.

E The Adaptive Out-of-Interference Delay Approach

The main challenge in approximating the carrier sensing range of the TCP source node lies in the lack of fundamental information such as transmission range and distances between nodes. Such information can be easily inquired in simulations, but are very hard, sometimes even impossible, to determine in real life. As we set the preservation of the end-to-end semantics of TCP as a strict design goal, we introduce an approach for approximating the carrier sensing range purely end-to-end without any support from intermediate nodes. All parameters needed for estimating the carrier sensing range are available at the TCP source node and can be inquired from the IEEE 802.11 driver.

We approximate the carrier sensing range in terms of number of hops, not in meters, by estimating how many hops it takes for the transmission signal of the TCP source node to get attenuated such that it falls below the carrier sensing threshold T_{cs}. That is, the first hop that comes after the threshold T_{cs} is undercut is a potential hidden terminal for the TCP source node.

The first step towards estimating the carrier sensing range in terms of number of hops is to estimate the signal attenuation for the first hop on the path from TCP source to TCP destination. Let P_{out} be the actual outgoing signal power of the TCP source node. Following the Equivalent Isotropically Radiated Power (EIRP) [21] equation we get:

$$P_{out} = P_{tx} + G_{ant} - A_{cab} - A_v \tag{2}$$

where P_{tx} denotes the transmission power of the wireless NIC at the source node, G_{ant} denotes the signal gain of the mini antenna, and A_{cab} and A_v describe the signal attenuation caused by the coaxial cable and the variable attenuator, respectively. The parameters A_{cab} and A_v only correspond to the deployed testbed and are set to zero if no cables and/or no hardware attenuators are used in the multihop wireless network. The signal attenuation for the first hop, L_1, is given by the difference between the received power P_{rx} at the second node in the chain and the outgoing signal power from the TCP source node, i.e. first node in the chain, P_{out}:

$$L_1 = P_{out} - P_{rx} \tag{3}$$

The received power P_{rx} can be easily inquired from the IEEE 802.11 driver at the source node using the Received Signal Strength Indication mechanism (RSSI) [28]. The next step is to derive an equation for estimating the signal attenuation for an arbitrary number of hops, n. Such an equation shall approximate the signal attenuation of the TCP source node at nodes which are n hops away from the source node. The signal attenuation equation as described by the ITU-R indoor propagation model [21] is given by

$$L = 20\log_{10}(f_c) + 10p\log_{10}(d) \tag{4}$$

where f_c denotes the frequency of the transmitted signal, i.e. a channel in the 2.4 GHz band in our case, p denotes the path loss exponent, and d describes the distance between transmitter and receiver in meters. The path loss exponent p depends on the operating environment of the wireless nodes and ranges from 2 for propagation in free space up to 5 in dense indoor environments. Due to findings from extensive measurements in our testbed and following [21], we set $p = 3$.

Let d_1 be the distance of the first hop in the chain, i.e. between the TCP source node and the second node, then we get according to Eq. 4:

$$L_1 = 20\log_{10}(f_c) + 10p\log_{10}(d_1) \tag{5}$$

It holds that $\sum_{i=1}^{n} d_i = d_1 n + \delta$, where d determines the deviation between the distance $d_1 n$ and the actual distance of the first n hops. For the signal attenuation of the TCP source node after n hops, L_n, we get:

$$\begin{aligned}
L_n &= 20\log_{10}(f_c) + 10p\log_{10}\left(\textstyle\sum_{i=1}^{n} d_i\right) \\
&= 20\log_{10}(f_c) + 10p\log_{10}(d_1 n + \delta) \\
&= 20\log_{10}(f_c) + 10p\left(\log_{10}(d_1) + \log_{10}(n) + \log_{10}\left(1 + \frac{\delta}{d_1 n}\right)\right) \\
&= L_1 + 10p\log_{10}(n) + 10p\log_{10}\left(1 + \frac{\delta}{d_1 n}\right) \\
&= L_1 + 10p\log_{10}(n) + \varepsilon
\end{aligned} \tag{6}$$

where ε describes the approximation error, which is determined by d. Since ε is a logarithmic factor, its ratio to the overall attenuation L_n diminishes with increasing distance.

Nevertheless, in a real large-scale mesh network, $\sum_{i=1}^{n} d_i$ may well be determined more accurately, either by deploying localization techniques in IEEE 802.11 [28], or by using GPS localization. In case such localization information are available at the TCP source, an even more accurate approximation of L_n may be achieved.

Finally, we can derive the carrier sensing range H_{cs} in terms of number of hops for an h-hop chain:

$$H_{cs} = \min\{k \mid k \in \{1, 2, ..., h\} \wedge P_{out} - L_k < T_{cs}\} \tag{7}$$

In other words, H_{cs} is the smallest number of hops k for which the actually sensed power of the TCP source node (i.e. P_{out} - L_k) is below the carrier sensing threshold T_{cs}. This implies that (H_{cs} + 1) is the first node in the chain which cannot sense the transmission of the TCP source and thus comprises a potential hidden terminal.

By means of the estimated carrier sensing range H_{cs} as well as RTT measurements at the TCP source, the out-of-interference delay *OID* of TCP data packets can be derived. The RTT is composed of the sum of the delay experienced by the data packet on the way from TCP source to TCP destination and the delay experienced by the ACK packet sent from the TCP destination to the TCP source. Each of these delays comprises the time to forward the packet over h hops, where each forwarding requires a queuing delay t_q and transmission delays t_{data}, and t_{ACK}, respectively. The parameters involved in the estimation of the out-of-interference delay are given in Table II. Using the measured RTT, we get:

$$RTT = h\left(t_q + t_{data} + t_{LLD}\right) + h\left(t_q + t_{ACK} + t_{LLA}\right) \quad (8)$$

Here, t_{LLD} and t_{LLA} denote the average link layer delay required for transmitting the TCP data packet and the TCP ACK packet, respectively. This delay comprises the transmission time of IEEE 802.11 control packets, link layer backoff, and potential retransmissions at link layer. Since information on link layer backoff and retransmissions on the path are not available at the TCP source node, we approximate t_{LLD} and t_{LLA} by defining the corresponding upper and lower bounds:

$$\frac{ACK_{LL}}{b_{base}} \le t_{LLD} \le \frac{ACK_{LL}}{b_{base}} + 3\left(\frac{s_{data}}{b} + cw_{cur} \cdot t_{slot}\right) \quad (9)$$

and

$$\frac{ACK_{LL}}{b_{base}} \le t_{LLA} \le \frac{ACK_{LL}}{b_{base}} + 3\left(\frac{s_{ACK}}{b} + cw_{cur} \cdot t_{slot}\right) \quad (10)$$

The lower bounds apply when the TCP packet (data or ACK) can be delivered with no retransmissions. The upper bounds correspond to the case when it takes the maximum number of retransmissions to deliver the TCP packet. According to the IEEE specifications [28], by default, a total of 4 attempts (i.e. 3 retransmissions) are distinguished at link layer before the packet is dropped (short retry limit). We omit the DIFS and SIFS intervals [28] due to their negligible sizes. We consider the case with RTS/CTS deactivated. In case RTS/CTS is activated, the corresponding transmission times of the RTS and CTS packets at a bandwidth of b_{base} are considered in Eqs. 9 and 10. Assuming a data bandwidth b of 11Mbit/s and considering the corresponding values for the link layer parameters in Eqs. 9 and 10 as given by the IEEE specifications [28], we get:

$$0.11ms \le t_{LLD} \le 7.72ms \quad (11)$$

and

$$0.11ms \le t_{LLA} \le 4.60ms \quad (12)$$

By setting t_{LLD} = 3.9ms and t_{LLA} = 2.4ms as average values we get an approximation error of at maximum 3.8ms and 2.2ms, respectively. For a typical wireless RTT of 60ms to 70ms at 11Mbit/s, this makes up an approximation error of 3%-6% at maximum. Note that the approximation can be easily computed for arbitrary values for b, which can be inquired directly from the wireless NIC driver.

Solving for t_q in Eq. 8 while using $t_{data} = s_{data}/b$ and $t_{ACK} = s_{ACK}/b$, we derive the average queuing delay as:

$$t_q = \frac{1}{2}\left(\frac{RTT}{h} - \frac{s_{data} + s_{ACK}}{b} - t_{LLD} - t_{LLA}\right) \quad (13)$$

Subsequently, we can estimate the out-of-interference delay of the TCP data packet:

$$OID = \left(H_{cs} + 1\right)\left(t_q + \frac{s_{data}}{b}\right) \quad (14)$$

The number of hops h on the network path to the receiver and the bandwidth b of the wireless network interface can be easily inquired without extra overhead from the kernel routing table and the IEEE 802.11 driver, respectively.

In theory, the maximum spatial reuse with minimum collisions can be achieved with a transmission rate R_{max}=1/OID. In fact, this transmission rate reflects the upper bound of the bandwidth-delay product for IEEE 802.11 multihop wireless networks. Following [10], an upper bound for the capacity of a path with h hops in an IEEE 802.11 multihop wireless network is given by $h/(H_{cs} + 1)$ packets. Let $T_{one\text{-}way}$ denote the time a packet traverses from the sender to the receiver. This quantity can be computed as $T_{one-way} = OID \cdot h / \left(H_{cs} + 1\right)$. Subsequently, the number of packets in flight on the way from the TCP source to the TCP destination with a transmission rate of R_{max} is given by:

$$\tilde{P} = R_{max} \cdot T_{one-way} = \frac{1}{\left(H_{cs} + 1\right)} h \quad (15)$$

Thus, the number of packets in flight $\tilde{P}$ transmitted with the maximum transmission rate R_{max} reflects the maximum capacity of the network path.

TABLE II: PARAMETERS FOR THE ADAPTIVE COMPUTATION OF THE TRANSMISSION RATE

Parameter	Meaning
L_1	Signal attenuation of the TCP source node after 1 hop
L_n	Signal attenuation of the TCP source node after n hops
f_c	Frequency of the transmitted signal
H_{cs}	The carrier sensing range in terms of number of hops
h	Number of hops from sender to receiver
b	Bandwidth for transmission of data packets
b_{base}	Base bandwidth for transmission of IEEE 802.11 control packets (1 Mbit/s)
t_q	Average packet queuing delay per node
t_{slot}	IEEE 802.11 slot time (20 microseconds)
s_{data}	Size of TCP data packet (incl. link layer overhead)
s_{ACK}	Size of TCP ACK packet (incl. link layer overhead)
t_{data}	Transmission time for TCP data packet
t_{ACK}	Transmission time for TCP ACK packet
t_{LLD}	Average link layer delay for transmitting a TCP data packet
t_{LLA}	Average link layer delay for transmitting a TCP ACK packet
ACK_{LL}	Size of link layer ACK (14 bytes)
cw_{cur}	Current size of IEEE 802.11 contention window
RTT	Current round trip time of TCP packets
cov_{RTT}	Coefficient of variation of RTT samples
OID	Current sample of out-of-interference delay
$\widehat{OID}$	EWMA of out-of-interference delay

F Deriving the Adaptive Pacing Rate

Since the computation of the adaptive transmission rate should account for both the current contention on the network path and the spatial reuse constraint, TCP-AP incorporates cov_{RTT} and OID in the transmission rate formula. Recall that a rate of R_{max}=1/OID specifies an upper bound for the achievable goodput under theoretically optimal conditions, i.e. with perfect scheduling and no contention. In order to adaptively throttle the transmission rate R according to the current degree of contention, we use cov_{RTT} as additional decay factor:

$$R = \frac{1}{\widehat{OID} \cdot (1 + 2cov_{RTT})} \quad (16)$$

The factor $\widehat{OID} \cdot (1{+}2cov_{RTT})$ describes the delay between successive packet transmissions at the TCP source (in milliseconds). Consequently, the transmission rate is $1/\widehat{OID} \cdot (1 + 2cov_{RTT})$ (i.e. one packet each $\widehat{OID} \cdot (1{+}2cov_{RTT})$ milliseconds). The coefficient of variation quantifies the percentage of sample deviation from the mean. However, since we want to quantify the size of the spectrum in which the samples fluctuate around the mean, we double the value cov_{RTT} in the rate formula.

Note that in favor of a stable transmission rate, we have to average the measured out-of-interference delay samples and employ a reasonable history size N for the computation of the coefficient of variation. Recall that N denotes the number of the most recent samples used for determining cov_{RTT}. For averaging the out-of-interference delay samples, we use the exponentially weighted moving average (EWMA) with averaging weight α.

That is:

$$\widehat{OID} = \alpha \cdot \widehat{OID}_{old} + (1-\alpha) \cdot OID \qquad (17)$$

As given in [10], suitable values for the EWMA weight α and the history size N are 0.7 and 50, respectively.

It is noteworthy that signal attenuation on some links in a chain may be either higher or lower than other intermediate links. This is due to sporadic, spatial-dependent interference that may occur on certain links as a result of nodes in the vicinity or other disturbing devices operating in the 2.4 GHz ISM band. Such sporadic interference may decrease the estimation accuracy of L_n described in Eq. 6, since the signal attenuation is not similar for all links. Such inaccuracy may result in either a more conservative estimation or a more optimistic estimation of *OID*. In case of a conservative estimation of *OID*, there would be no increased collisions due to hidden terminals since the TCP source node defers its transmission until previously sent packets are forwarded by its potential hidden terminal. However, due to the conservative estimation of *OID*, and according to Eq. 16, a potential slight decrease in the pacing rate may occur. On the other hand, in case of a more optimistic estimation of *OID*, collisions with hidden terminals may occur. However, such collisions would be implicitly reflected in the derived coefficient of variation cov_{RTT}, which would increase due to the increased RTT fluctuation. Thus, according to Eq. 16, the increased collisions would be compensated by an automatic adjustment of the pacing rate, which would consequently decrease collisions at link layer. In Section V we show that such sporadic effects have a negligible impact on the performance of TCP-AP.

G The refined TCP-AP Algorithm

In order to give intuition on how to implement the refined TCP-AP algorithm, we present the pseudo code in Fig. 3.

```
Key variables:
H_cs:                 Carrier sensing range in terms of # hops
InterPacketDelay:     Time between successive packet transmissions
OID:                  Out-of-Interference Delay
seqno:                Current TCP sequence number
highestACK:           Sequence number of last ACK received
awnd:                 Receiver advertised window size
cwnd:                 Congestion window size
cov_RTT               Coefficient of variation of RTT

Utility functions:
recv():                              Function called upon ACK receipt
pacing_timeout(InterPacketDelay):    Function called every InterPacketDelay time
estimate_H_cs:                       Function called to estimate the current
                                     carrier sensing range in
                                     terms of # hops

Algorithm:
function recv() {
        for each received ACK do
                $H_{cs}$ = estimate_H_cs()
                $OID := (H_{cs} + 1)\left(t_q + \frac{s_{data}}{b}\right)$
                $\widehat{OID} := \alpha \cdot \widehat{OID}_{old} + (1-\alpha) \cdot OID$
                calculate $cov_{RTT}$ over most recent N RTT
                samples
                $InterPacketDelay := \widehat{OID} \cdot (1 + 2cov_{RTT})$
        done
}

function estimate_H_cs() {
        $P_{out} = P_{tx} + G_{ant} - A_{cab} - A_{v}$
        $L_1 = P_{out} - P_{rx}$
        $L_k = L_1 + 10p\log_{10}(k)$
        $H_{cs} = \min\{k \mid k \in \{1, 2, \ldots, N\} \wedge P_{out} - L_k < T_{cs}\}$
        return $H_{cs}$
}

function pacing_timeout(InterPacketDelay) {
        if $seqno \le highestACK + \min(awnd, cwnd)$ then
                send new packet
        else
                stay idle
        endif
}
```

Fig. 3: Pseudo code of refined TCP-AP algorithm

H The TCP Framework Implementation for Linux

In order to evaluate our adaptive pacing approach, we implemented a user-space framework for TCP in Linux, in which we incorporated TCP-AP as well as the widely deployed variant TCP NewReno. The framework communicates directly with the network layer in Linux. This is made possible by using raw IP sockets, which only add IP and MAC headers to the packets by default. The TCP header is constructed and included by the framework. The user interacts with the traffic generator and provides numerous parameters such as TCP destination, number of bytes to be transmitted, and employed TCP variant. The implementation of the TCP framework has a modular hierarchy and thus supports expansions with further TCP flavors.

TCP-AP builds on TCP NewReno without changing any of its internal congestion control dynamics. Slow start, congestion avoidance, fast retransmit, and fast recovery are all kept unchanged to ensure the friendliness to other TCP flavors. The main aspect that is modified in the implementation is the transmission of TCP packets, which is performed rate-based within the congestion window. Specifically, TCP packets within the current

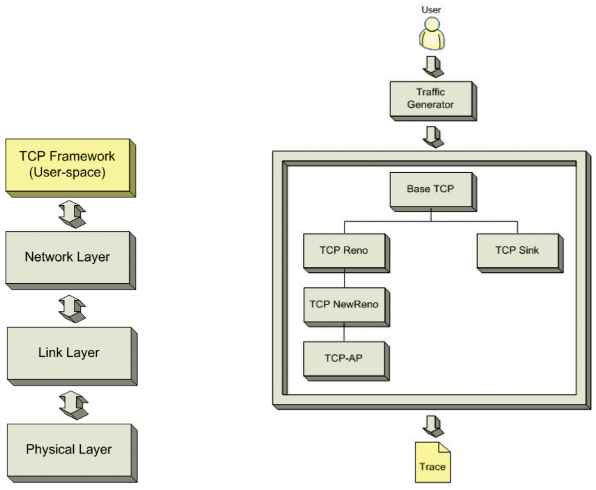

Fig. 4: Interaction between the TCP framework implementation and lower layers as well as the class hierarchy of the TCP framework

congestion window are placed in a queue which schedules its transmission time according to the current adaptive pacing rate.

For computing the adaptive pacing rate accurately, TCP-AP requires fine grained timers. Unfortunately, the standard POSIX timers available in the Linux user space suffer from low accuracy, resulting up to 20 milliseconds timer-jitter. Thus, we integrated the high-resolution timer subsystem [15] into the Linux kernel, which provides a high accuracy in the order of nanoseconds.

The TCP-AP implementation within the framework requires no modifications of underlying layers. The number of hops *h* required for the calculation of the pacing rate is acquired from the kernel routing table using default Linux APIs, whereas the link-layer parameters needed are acquired from the IEEE 802.11 driver using default I/O control requests. This makes TCP-AP incrementally deployable, since it works directly with off-the-shelf multi-hop routing protocols as well as IEEE 802.11 link-layer drivers.

V. Performance Evaluation

We present a comprehensive performance study of TCP-AP versus the widely deployed TCP NewReno by means of our miniaturized wireless mesh testbed. In all experiments, except for experiments showing transient behavior, we conduct steady-state simulations starting with an initially idle system. In each run, we activate TCP connections until 55,000 packets are successfully transmitted, and split the output of the experiment in 11 batches, each 5,000 packets in size. The first batch is discarded as initial transient. The considered performance measures are derived from the remaining 10 batches with 95% confidence intervals by the batch means method.

For all experiments, we set the TCP packet size to 1,460 bytes and the TCP receiver's advertised window to 64 packets. We set the IEEE 802.11 data rate to 11 Mbit/s and the attenuation level of the variable attenuators to 16 dB, unless otherwise stated. This provides a transmission range of roughly 0.5m.

A FTP-like Traffic

In the first set of experiments, we consider scenarios with FTP-like, bulky data transfer in different network topologies. That is, the TCP source transmits packets continuously, representing a large file transfer.

Chain Topology

The first topology we consider is an equally spaced chain comprising *h*+1 nodes (*h* hops) with a single flow, as depicted in Fig. 5. TCP packets traverse along the chain from the leftmost node (i.e., the source) to the rightmost node (i.e., the destination). Nodes in the chain are positioned such that only direct neighbors can communicate with each other over one hop.

First, we validate the TCP NewReno implementation of our Linux TCP framework. We do so by comparing the goodput achieved by our user-space TCP NewReno to the Linux kernel TCP using the Iperf measurement tool [29]. Table III shows the goodput of each variant for varying number of hops between TCP source and TCP destination. The goodput is averaged over 40 runs, where each run comprises 10 batches. We omit the confidence intervals since they only make up 2% of the corresponding goodput value, at maximum.

In Table III we see that the values of both user-space TCP NewReno and Linux kernel TCP are almost identical. The Linux kernel TCP slightly outperforms user-space TCP by about 1% due to the typical lower jitter and CPU time of kernel-space implementations compared to user-space implementations.

In order to evaluate TCP-AP versus TCP NewReno in a variety of different network conditions, we vary network-related parameters to reflect typical real world settings. For one, we consider the goodput of TCP-AP and TCP NewReno with and without the RTS/CTS handshake. Furthermore, we set the attenuation level of the variable attenuators such that the signal between nodes is either optimal (at low attenuation level) or very weak (at high attenuation level). Such a high attenuation level and/or weak inter-node signal often occurs in real multihop wireless networks, in cases where either links between nodes suffer from high external interference or the distance between nodes is considerably large.

1
2
3
4
5
6
7
8
9
10

Fig. 5: 9-hop chain topology with a single flow

Table III: Average Goodput of user-space TCP NewReno vs. Linux kernel TCP

Number of Hops	Avg. Goodput of User-space TCP NewReno [Kbit/s]	Avg. Goodput of Linux Kernel TCP [Kbit/s]
1	5588	5671
2	2750	2843
3	962	981
4	709	722
5	748	761
6	903	919
7	773	783
8	722	731
9	657	672

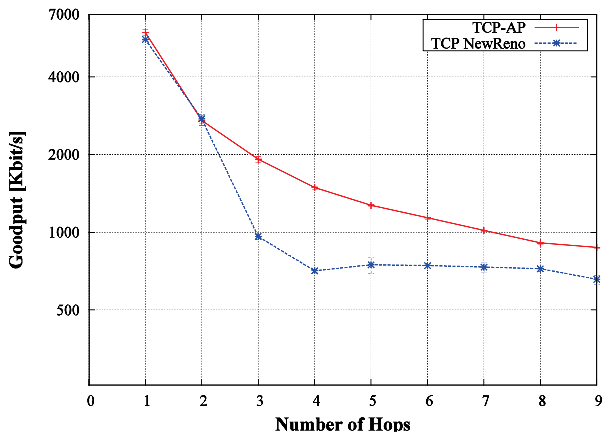

Fig. 6: Goodput vs. number of hops without RTS/CTS and with low signal attenuation (16 dB)

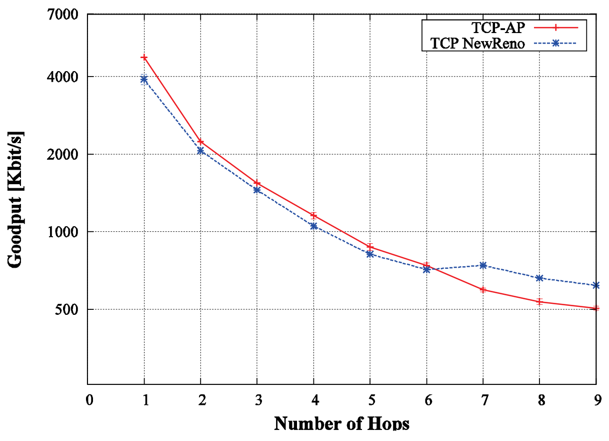

Fig. 7: Goodput vs. number of hops with RTS/CTS and with low signal attenuation (16 dB)

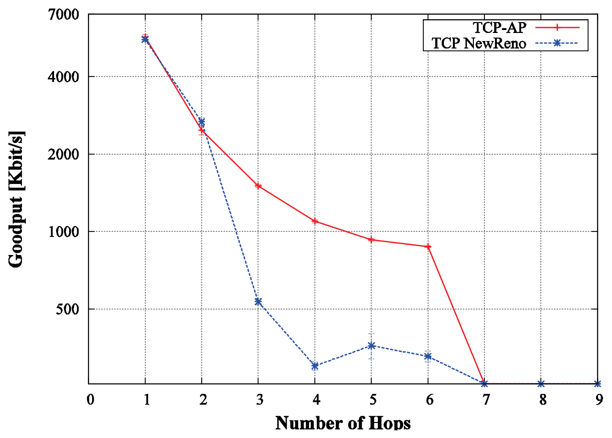

Fig. 8: Goodput vs. number of hops without RTS/CTS and with high signal attenuation (29 dB)

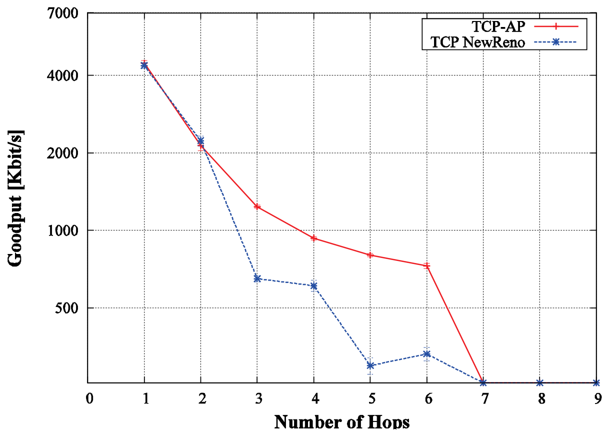

Fig. 9: Goodput vs. number of hops with RTS/CTS and with high signal attenuation (29 dB)

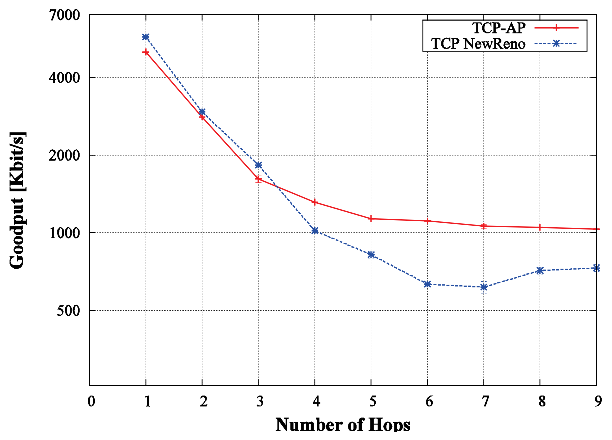

Fig. 10: NS-2: Goodput vs. number of hops without RTS/CTS

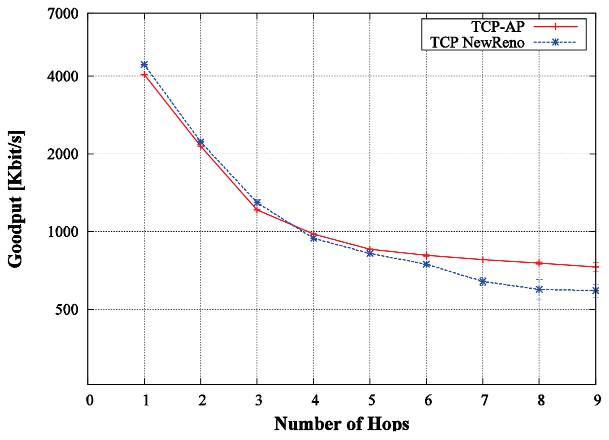

Fig. 11: NS-2: Goodput vs. number of hops with RTS/CTS

Figures 6 to 9 show the results of this experiment, plotted as goodput versus number of hops between TCP source and TCP destination. In Fig. 6, where RTS/CTS is disabled and the signal between nodes is optimal, TCP-AP outperforms TCP NewReno by up to 113%. In case RTS/CTS is enabled, as shown in Fig. 7, TCP-AP has only a slight increase in goodput up to 6 hops. At a chain length of 7 to 9 hops, TCP NewReno achieves more goodput than TCP-AP. This is due to that fact that with RTS/CTS activated, the number of hidden terminals in a chain significantly decreases, resulting in an increased goodput for TCP NewReno, which suffers at most from hidden terminal effects. However, when comparing Fig. 6 with Fig. 7, we can state that the RTS/CTS overhead reduces the goodput achieved by TCP-AP up to almost 50% as well as the goodput of TCP NewReno for 1 to 3 hops. This is consistent with previous studies such as [3] and [14], which have shown that RTS/CTS degrades goodput significantly.

The previous findings are consistent with the results of Figures 8 and 9, in which the signal between nodes is very weak due to the high attenuation level. Comparing both figures, we observe that RTS/CTS also results in a significant degradation in goodput. Furthermore, we see that TCP-AP achieves up to ten times more goodput than TCP NewReno. That is, in such an environment where the signal between nodes is not optimal, the aggressive transmission of TCP NewReno greatly overwhelms the channel, resulting in a severe packet loss rate. On the other hand, the adaptive pacing approach of TCP-AP adjusts the transmission rate according to the current state of the channel, reducing packet loss, and thus achieving more goodput. As the signal between nodes is at its lower limit, it does not suffice for delivering packets successfully for a chain of 7 to 9 hops.

To compare the results acquired from the TCP implementations in the testbed with the TCP implementations in the network simulator ns-2 [12], we conduct a simulation study in ns-2, where we examine the goodput of TCP NewReno versus TCP-AP for varying number of hops. Figures 10 and 11 show the results of the simulation with and without RTS/CTS, respectively. When comparing Fig. 6 with Fig. 10, we notice that although the improvement of TCP-AP with respect to TCP NewReno is very similar, the number of hops at which such improvement is at its maximum varies. While the largest gap between TCP-AP and TCP NewReno lies at 4 hops for the testbed experiment (Fig. 6), it lies at 7 hops in the simulation (Fig. 10). The reason for such shift in the number of hops is that in ns-2, the out-of-interference delay only corresponds to a fixed ratio between transmission range and carrier sensing/interference range, which is equal to 250m/550m. In the testbed, such a delay is variable and is not bound to a specific number, which is considered by the refined implementation of TCP-AP in the testbed. Thus, as positions of hidden terminals vary in the simulation with respect to the testbed, the number of hops at which TCP-AP achieves its best goodput compared to TCP NewReno also varies. Consistent with Fig. 7, Fig. 11 shows that the RTS/CTS handshake decreases TCP goodput due to the higher overhead associated with it.

In a further experiment, we evaluate the fairness between competing flows when employing TCP-AP versus TCP NewReno. As depicted in Fig. 12, two competing TCP flows run from both ends of a 6-hop chain, i.e. nodes 1 and 6, to the middle node 4. This emulates the case in which some node in a mesh network wishes to download two files from two different nodes simultaneously.

Figures 13 and 14 show the goodput of each of the two flows evolving over time for TCP-AP and TCP NewReno, respectively. In Fig. 13, we observe how both flows share the available bandwidth equally when employing TCP-AP. In contrast, in Fig. 14, we see that TCP NewReno results in almost a complete starvation of flow 2 due to its aggressive transmission. That is, since IEEE 802.11 favors aggressive flows over less aggressive ones, the flow which succeeds to acquire the channel first, i.e. flow 1 in this case, also succeeds to take control of the channel until the end of its transmission. On the contrary, TCP-AP adjusts its adaptive pacing rate such that multiple flows share the bandwidth equally. The component of the adaptive pacing rate of TCP-AP which is responsible for maintaining inter-flow fairness is the coefficient of variation cov_{RTT}. As given in Eq. 16, this factor throttles the adaptive pacing rate proportional to the current interference caused by other flows, and thus prevents one or more flows from starving. As flows in this experiment mutually impose a nearly equal degree of interference, their corresponding adaptive pacing rate is also nearly equal. Hence, the bandwidth is distributed fairly among both flows.

Fig. 15 shows the goodput of both TCP flows as well as the aggregate goodput achieved throughout the experiment. Consistent with the previous results, TCP-AP achieves almost optimal fairness between the flows, while TCP NewReno favors flow 1 at cost of flow 2. We notice that TCP NewReno achieves slightly more aggregate goodput than TCP-AP, which is due to the known trade-off between aggregate goodput and fairness caused by the absence of optimal scheduling of the IEEE 802.11 link layer protocol [10] [24].

In a third experiment, we evaluate the responsiveness of TCP-AP versus TCP NewReno. As responsiveness we denote how quickly the congestion control algorithm adapts to changing network conditions such as additional flows competing for the bandwidth. We re-conduct the previous experiment with the two competing flows, however, with different start and stop times of the considered flows. Specifically, we let flow 1 run from the beginning to the end of the experiment, i.e. until the 40th second. Flow 2 starts at the 10th second of the experiment and stops at the 30th second. We then investigate how flow 1 reacts upon the activation of flow 2. Figures 16

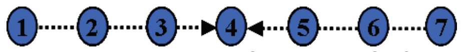

Fig. 12: Parallel download: Competing TCP flows

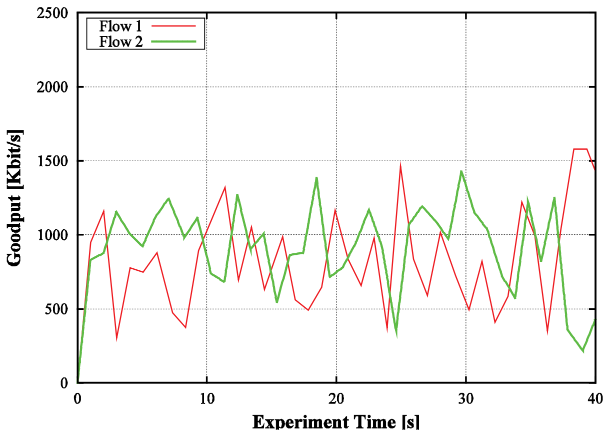

Fig. 13: Fairness of TCP-AP: Goodput vs. time

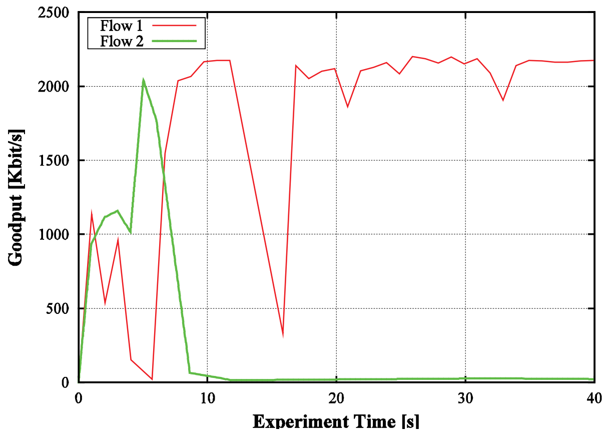

Fig. 14: Fairness of TCP NewReno: Goodput vs. time

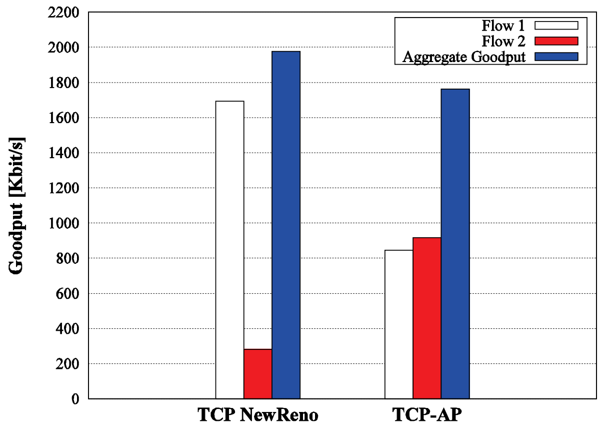

Fig. 15: Goodput and fairness of the competing TCP flows

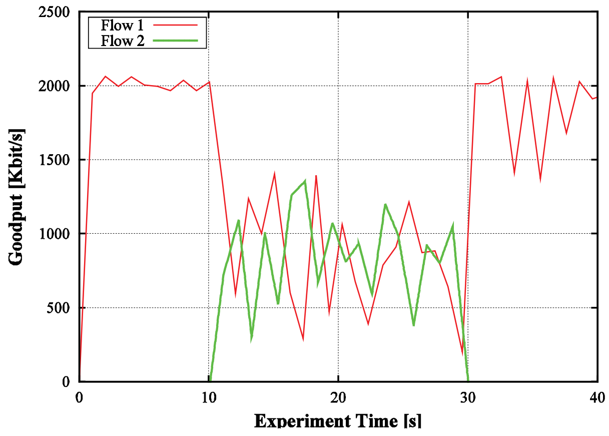

Fig. 16: Responsiveness of TCP-AP: Goodput vs. time

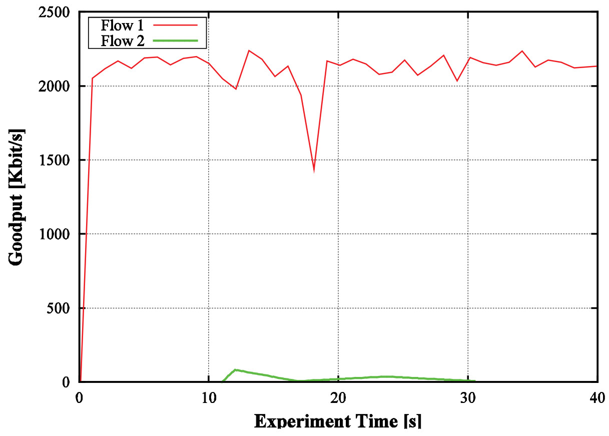

Fig. 17: Responsiveness of TCP NewReno: Goodput vs. time

and 17 show the results of this experiment. We notice that, employing TCP-AP, flow 1 utilizes the available bandwidth when there is no competing flow and shares the bandwidth fairly when both flows compete for the channel. On the contrary, due to the aggressiveness of TCP NewReno, flow 1 acquires the entire bandwidth available throughout the experiment time, resulting in a complete starvation of flow 2.

Parallel Chains Topology

In this experiment, we consider a topology of two 4-hop symmetric parallel chains, as depicted in Fig. 18. The chains lie beyond each other's transmission range, but within each other's interference range. We run one TCP flow on each chain and investigate the achieved goodput of each flow as well as the aggregate goodput, i.e. the sum of the goodput achieved by both flows.

Figures 19 to 22 show the results of this experiment. In Fig. 19, we observe how TCP-AP achieves to share the goodput equally among both flows, whereas TCP NewReno penalizes flow 1 in favor of flow 2. Moreover, TCP-AP achieves around 75% more aggregate goodput than TCP NewReno. Figures 20 and 21 show the transient behavior of both flows during the first 20 seconds of the experiment. The figures show the sequence numbers of TCP packets received at the TCP destination for TCP-AP and TCP NewReno, respectively. In Fig. 20 we notice that the growth in the sequence numbers has a similar slope for both flows, indicating that the bandwidth is shared equally among them. In contrast, in case of TCP NewReno, flow 2 acquires most of the bandwidth as its sequence numbers increase steeply, while the sequence number growth of flow 1 stagnates. This indicates a high TCP packet loss rate and multiple TCP timeouts for flow 1 in favor of flow 2.

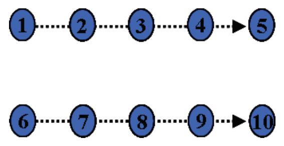

Fig. 18: Parallel chains topology

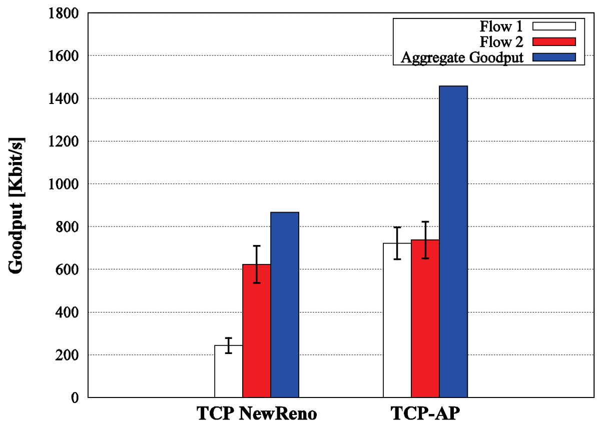

Fig. 19: Goodput and fairness for the parallel chains topology

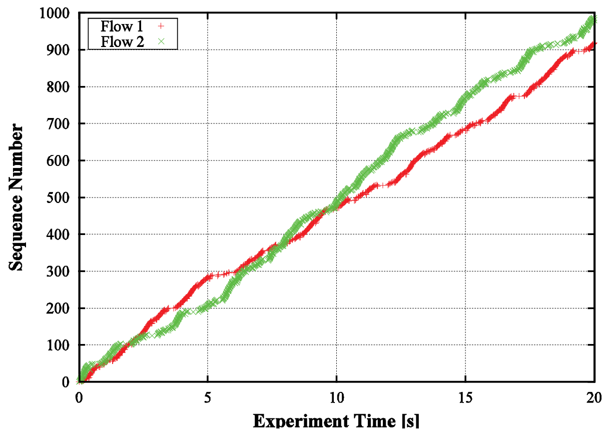

Fig. 20: TCP-AP: Sequence numbers of TCP packets received at TCP destination

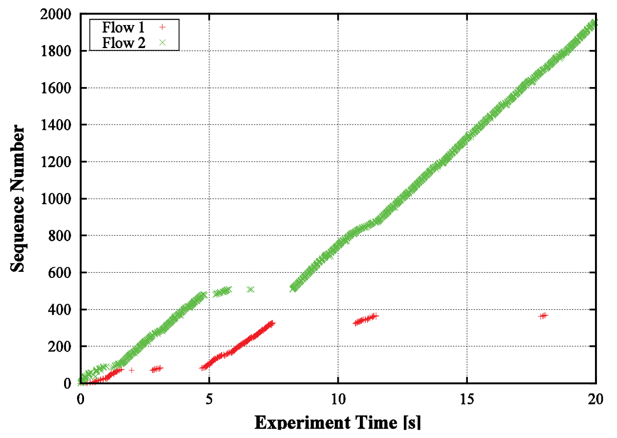

Fig. 21: TCP NewReno: Sequence numbers of TCP packets received at TCP destination

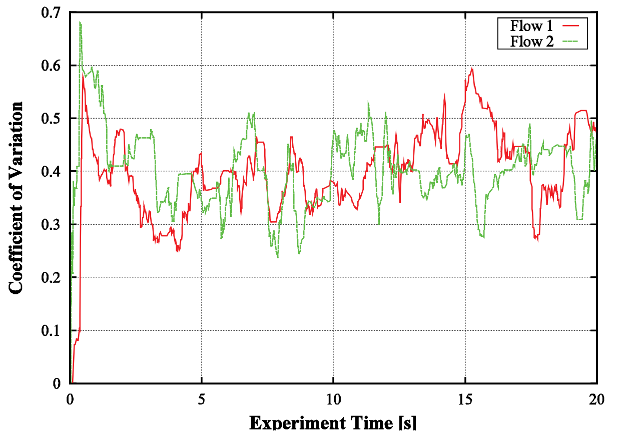

Fig. 22: TCP-AP: Coefficient of Variation of RTT

Fig. 22 shows the coefficient of variation of RTTs, plotted over time for both TCP-AP flows. It is obvious that the gradient of the coefficient of variation is similar for both flows. This implies that both flows experience similar RTT fluctuation, and thus transmit at a similar pacing rate, which is consistent with the fairness results shown in Fig. 21.

Random Topologies

Random node topologies are typically found in community mesh networks such as [3] and are widely deployed in reality. We evaluate TCP-AP and TCP NewReno in such topologies by considering random placements of the testbed's 20 antenna-stations. The 20 antenna-stations are distributed uniformly on a flat area of 2m x 3m such that full connectivity between each pair in the network over one or more hops is granted. In addition to the batch means method described earlier, and in order to achieve optimal results in terms of representativeness, we consider 20 replicates when deriving performance measures. Each replicate corresponds to a different random placement of the nodes.

Fig. 23 shows the cumulative distribution function (CDF) of goodput between each pair in the network for TCP-AP versus TCP NewReno over all considered 20 random node placements. The higher slope of NewReno's CDF for low goodput values (i.e. 0 to 2000 Kbit/s) and lower slope for high goodput values (i.e. 4000 Kbit/s to 6000 Kbit/s) indicate that employing TCP-AP yields more goodput than TCP NewReno. Specifically,

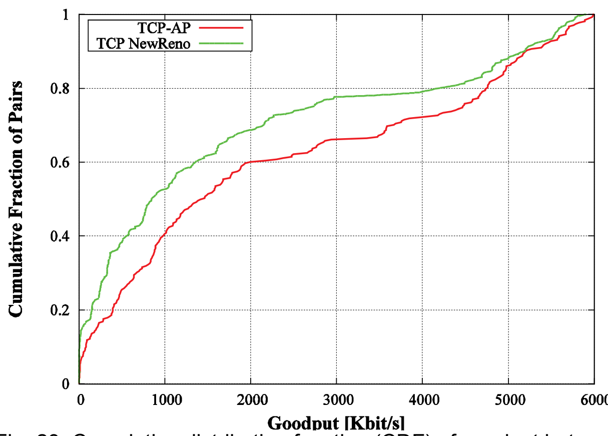

Fig. 23: Cumulative distribution function (CDF) of goodput between each pair in the network (Average goodput: TCP NewReno: **875 Kbit/s**, TCP-AP: **1123 Kbit/s.** Median goodput: TCP NewReno: **847 Kbit/s**, TCP-AP: **1413 Kbit/s**)

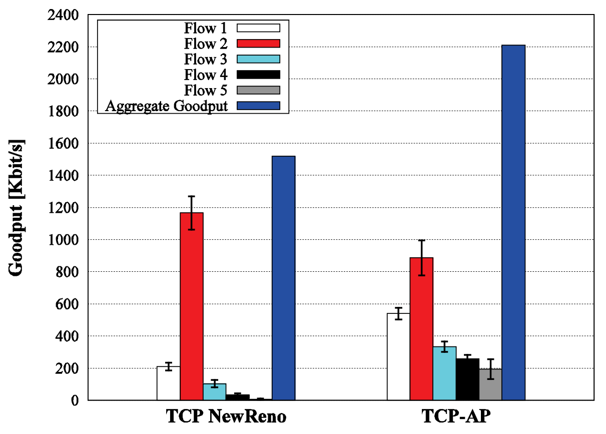

Fig. 24: Goodput of five parallel flows running between five randomly chosen pairs

the average goodput of TCP NewReno constitutes 875 Kbit/s, whereas the average goodput of TCP-AP makes up 1123 Kbit/s, i.e. around 28% more goodput than TCP NewReno. As for the median goodput, TCP NewReno achieves only 847 Kbit/s while TCP-AP achieves 1413 Kbit/s, i.e. around 67% improvement in goodput.

In Fig. 24 we consider a representative sample of the 20 replicate-topologies, in which we run five parallel TCP flows between five randomly chosen pairs. We plot the individual goodput of each flow as well as the aggregate goodput both for TCP-AP and TCP NewReno accordingly. We observe how TCP NewReno lets flow 2 acquire most of the available bandwidth at cost of the other flows, while letting flows 4 and 5 almost completely starve. In contrast, TCP-AP prevents the starving of any flow by dividing the available bandwidth more fairly among flows, while achieving about 46% more aggregate goodput than TCP NewReno.

The reason why flow 2 acquires most of the available bandwidth is that it runs on a 2-hop chain, while other flows run on chains with hops greater or equal 4. As a result of the IEEE 802.11 spatial reuse capacity limits, and as validated in Figures 6 to 11, the TCP goodput decreases with increasing number of hops. This affects TCP NewReno more than TCP-AP since, as discussed above, TCP NewReno does not throttle its rate according to other flows in the vicinity. Thus, as the most aggressive flow, flow 2 acquires most of the available bandwidth at the cost of other flows.

B HTTP-like Traffic

In the second set of experiments, we consider variable length flows, i.e. HTTP-like data transfer, where the TCP sender transmits small files with variable pause times between successive file transfers. Following [18], we assume that file sizes are Pareto distributed with mean 30 Kbytes and shape factor $\beta = 1.5$, whereas pause times between successive file transfers are exponentially distributed. Figures in this section describe the behavior of the considered TCP variants during HTTP ON-phases, in which the contents of a webpage are typically downloaded.

We reconsider the random topologies in Section V.A and adopt the same settings given, except using HTTP-like traffic instead of FTP-like traffic to derive performance measures. Performance measures in Figures 25 and 26 are derived as an average over each five-pair set in the network, where each set contains five simultaneously active HTTP connections. In Fig. 27 we consider five concurrent HTTP connections running between five randomly chosen pairs, whereas in Fig. 28 we consider one HTTP flow between each pair in the network.

Fig. 25 shows the average download delay of the downloaded files for varying mean of exponentially distributed pause times for TCP NewReno and TCP-AP, respectively. The download delay denotes the time needed for downloading the desired file. We observe that for small pause times of 0.1s and 0.2s, TCP-AP requires up to 51% less download delay than TCP NewReno. As pause times between successive file transfers increase, the difference in download delay between TCP NewReno and TCP-AP vanishes. That is, for pause times above 0.2s, the download delay of TCP NewReno and TCP-AP is almost identical. Such behavior is due to the traffic load in the network, which is high for small pause times and decreases gradually as pause times increase. Thus, since TCP-AP is superior to TCP NewReno at high network load, which causes increased contention at link layer, TCP-AP outperforms TCP NewReno for small pause times. As pause times increase, contention in the network decreases and the difference in download delay between TCP-AP and TCP NewReno becomes smaller.

We evaluate the fairness in this experiment by computing Jain's fairness index [17], which is defined as:

$$F(x) = \left[\sum_{i=1}^{n} x_i\right]^2 \Big/ n\sum_{i=1}^{n} x_i^2 \qquad (18)$$

where n is the number of flows and x_i denotes the goodput achieved by flow i. The index ranges between 0 and 1. An index of 1 corresponds to the best fairness achievable between competing flows whereas and index of $1/n$ denotes the case where a single flow acquires the entire bandwidth available.

Fig. 26 shows Jain's fairness index of TCP NewReno and TCP-AP for increasing pause times. Again, for small pause times, i.e. 0.1s and 0.2s, TCP-AP achieves considerably better fairness than TCP NewReno. As pause times increase, the fairness of TCP NewReno improves until reaching the same level like TCP-AP. Note that in such random topologies, an optimal fairness index of 1 is typically not achieved, since Jain's fairness index is only optimal when all considered flows have similar physical characteristics. Such similarities include for instance running over the same number of hops, which is typically not the case in random environments.

Fig. 27 shows the average download delay of five concurrent HTTP connections running between five randomly chosen pairs with a mean for pause times between successive file transfers of 0.1s. Consistent with the previous results, TCP-AP achieves around 25% less aggregate download delay than TCP NewReno, while achieving better fairness between the competing HTTP flows.

In Fig. 28, we plot the cumulative distribution function (CDF) of download delay between each pair in the network. The higher slope of TCP-AP's CDF for low download delays points out

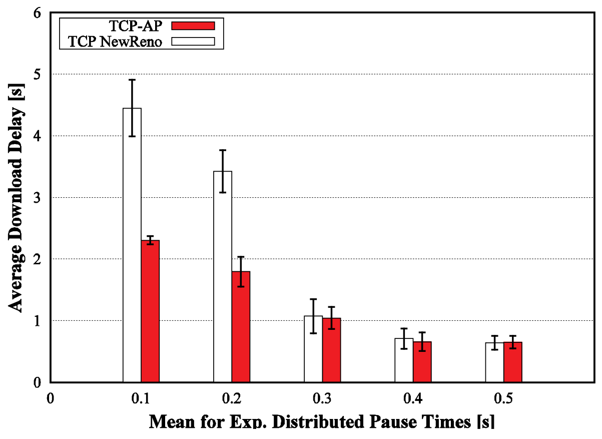

Fig. 25: Average download delay vs. mean for pause times

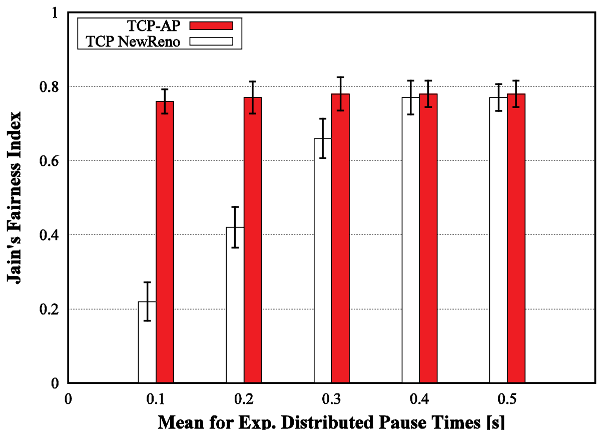

Fig. 26: Jain's fairness index vs. mean for pause times

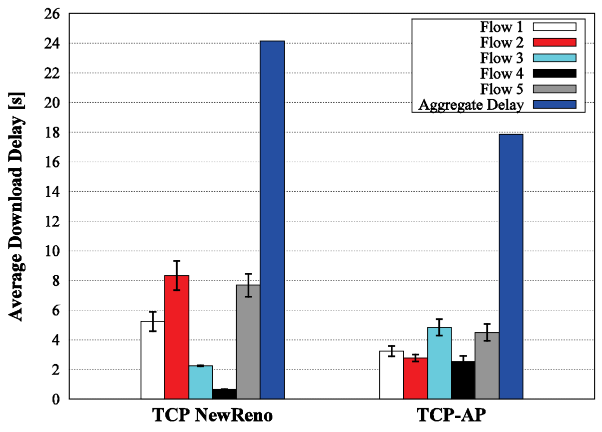

Fig. 27: Average download delay for five concurrent HTTP flows

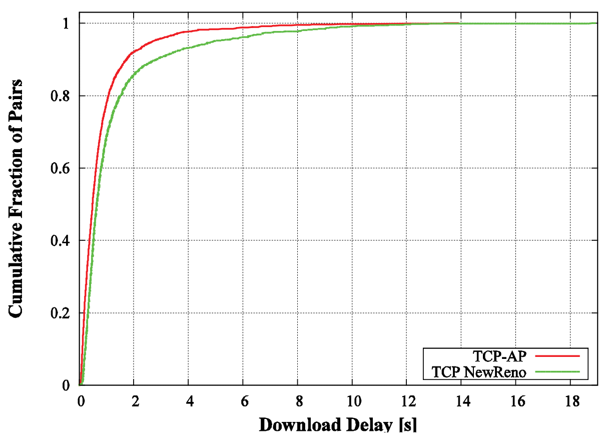

Fig. 28: Cumulative distribution function (CDF) of download delay between each pair in the network (Average download delay: TCP NewReno: **1.26s**, TCP-AP: **0.82s.** Median download delay: TCP NewReno: **0.64s,** TCP-AP: **0.5s**)

that files are downloaded faster using TCP-AP than the case when using TCP NewReno. Specifically, the average download delay of TCP NewReno is 1.26s versus 0.82s for TCP-AP, whereas the median download delay of TCP NewReno makes up 0.64s compared to 0.5s for TCP-AP. This makes up 35% less average download delay and 22% less median download delay for TCP-AP compared to TCP NewReno.

VI. Conclusion

We introduced an effective end-to-end congestion control algorithm for TCP in real multihop wireless networks. Our approach, denoted as TCP with Adaptive Pacing (TCP-AP), employs rate-based transmission within the current congestion window. By adapting the transmission rate to the current network state, TCP-AP reduces collisions at link layer, and thus achieves improved goodput and fairness. Due to the TCP-compatibility of TCP-AP, and since it relies solely on end-to-end measurements of round trip times and requires no modifications at the routing layer or the link layer, TCP-AP is easily deployable.

We implemented TCP-AP in Linux and validated its feasibility in a specially built 20-node wireless mesh testbed. A comprehensive performance study showed that, depending on the current level of interference and signal attenuation, TCP-AP yields up to ten times more goodput than TCP NewReno, while achieving excellent fairness results. The acquired results indicate that the gain of TCP-AP is particularly high for FTP-like traffic, where contention at link layer is considerably heavy.

Currently, we are investigating the suitability of TCP-AP for MIMO mesh networks, in particular for indoor mesh networks adopting IEEE 802.11n technology. The acquired findings will then be used to further calibrate the adaptive pacing algorithm for the deployment in future large-scale outdoor mesh network scenarios. Such networks may require considering different signal attenuation equations due to the different propagation characteristics of the radio signal. Due to the modular design of the adaptive pacing approach introduced in this paper, different path loss formulas can be easily employed in the calculation of the out-of-interference delay (*OID*).

References

[1] G Anastasi, E. Ancillotti, M. Conti and A. Passarella, Experimental Analysis of a Transport Protocol for Ad hoc Networks (TPA), *Proc. ACM PE-WASUN Workshop*, Terromolinos, Spain, 2006.

[2] L. Andrew, S. Hanly, and R. Mukhtar, Active Queue Management for Fair Resource Allocation in Wireless Networks, *IEEE Transactions on Mobile Computing*, Vol. 7 Issue 2, February 2008.

[3] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, Architecture and Evaluation of an Unplanned 802.11b Mesh Network, *Proc. ACM MOBICOM*, Cologne, Germany, 2005.

[4] J. Camp, J. Robinson, C. Steger, and E. Knightly, Measurement Driven Deployment of a Two-Tier Urban Mesh Access Network, *Proc. ACM MobiSys*, Uppsala, Sweden, 2006.

[5] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links, *Proc. ACM MOBICOM*, Rome, Italy, 2001.

[6] T. Clausen and P. Jacquet, Optimized Link State Routing Protocol, RFC 3626, *http://www.ietf.org/rfc/rfc3626.txt*, October 2003.

[7] D. De Couto, D. Aguayo, J. Bicket, and R. Morris, A High-Throughput Path Metric for Multi-Hop Wireless Routing, *Proc. ACM MOBICOM*, San Diego, CA, 2003.

[8] P. De, A. Raniwala, R. Krishnan, K. Tatavarthi, J. Modi, N. Syed, S. Sharma, and T. Chiueh, MiNT-m: an autonomous mobile wireless experimentation platform, *Proc. ACM MobiSys*, Uppsala, Sweden, 2006.

[9] D. Koutsonikolas, J. Dyaberi, P. Garimella, S. Fahmy, and Y. Hu, On TCP Throughput and Window Size in a Multihop Wireless Network Testbed, *Proc. ACM WinTech*, Motreal, Canada, 2007.

[10] S. ElRakabawy, A. Klemm, and C. Lindemann, TCP with Adaptive Pacing for Multihop Wireless Networks, *Proc. ACM MobiHoc*, Urbana-Champaign, IL, 2005.

[11] J. Eriksson, S. Agarwal, P. Bahl, and J. Padhye, Feasibility Study of Mesh Networks for All-wireless Offices, *Proc. ACM MobiSys*, 2006, Uppsala, Sweden

[12] K. Fall and K. Varadhan (Ed.), The ns-2 Manual, Technical Report, The VINT Project, UC Berkeley, LBL, and Xerox PARC, 2007.

[13] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, The Impact of Multihop Wireless Channel on TCP Performance, *IEEE Transactions on Mobile Computing*, Vol. 4, Issue 2, March 2005.

[14] V. Gambiroza, B. Sadeghi, and E. Knightly, End-to-End Performance and Fairness in Multihop Wireless Backhaul Networks, *Proc. ACM MOBICOM*, Philadelphia, PA, 2004.

[15] T. Gleixner and D. Niehaus, Hrtimers and Beyond: Transforming the Linux Time Subsystems, *Proc. 8th OLS Linux Symposium*, Ottawa, Canada, 2006 (Source code available at http://www.tglx.de/projects/ktimers/).

[16] G. Holland and N. Vaidya: Analysis of TCP Performance over Mobile Ad Hoc Networks, *Wireless Networks,* Vol. 8, Issue 2, 2002.

[17] R. Jain, D. Chiu, and W. Hawe, A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems, DEC Technical Report DEC-TR-301, 1984.

[18] Y. Joo, V. Ribeiro, A. Feldmann, A.C. Gilbert, and W. Willinger, TCP Traffic Dynamics and Networks Performance: A Lesson in Workload Modeling, Flow Control, and Trace-driven Simulation, *ACM SIGCOMM Computer Communication Review*, 31, 2001.

[19] V. Kawadia and P. R. Kumar, Experimental Investigations into TCP Performance over Wireless Multihop Networks, *Proc. ACM E-WIND*, Philadelphia, PA, 2005.

[20] K. Nahm, A. Helmy, C. C. Kuo, TCP over Multihop 802.11 Networks: Issues and Performance Enhancement, , *Proc. ACM MobiHoc*, Urbana-Champaign, IL, 2005.

[21] S. Saunders, Antennas and Propagation for Wireless Communication Systems, *Wiley & Sons*, May 2007

[22] K. Sundaresan, V. Anantharaman, H-Y. Hsieh, R. Sivakumar, ATP: A Reliable Transport Protocol for Ad Hoc Networks, *Transactions on Mobile Computing*, Vol. 4, Issue 6, November 2005

[23] D. Wai, P. Cao, and S. Low, TCP Pacing Revisited, *Proc. IEEE INFOCOM*, Anchorage, AK, USA, 2007

[24] K. Xu, M. Gerla, L. Qi, and Y. Shu, TCP Unfairness in Ad Hoc Wireless Networks and a Neighborhood RED Solution, *Wireless Networks*, Vol. 11, Issue 4, 2005.

[25] OLSR.ORG Implementation for Linux, *http://www.olsr.org*.

[26] Freifunk Mesh Community, *http://start.freifunk.net/*.

[27] The Qualnet Simulator, *http://www.scalable-networks.com/*.

[28] IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ISO/IEC 8802-11, August 1999.

[29] Iperf, the TCP/UDP Bandwidth Measurement Tool, *http://dast.nlanr.net/Projects/Iperf/*.

Sherif M. ElRakabawy received his PhD in computer science from the University of Leipzig, Germany in 2009, and his Diplom (MSc) in computer science from the University of Bonn, Germany in 2003. In the summer of 2009, he was a visiting researcher at the EECS department of the University of California at Berkeley, where he worked with Professor Eric Brewer and the TIER group on long-distance wireless technologies. His research interests include mobile ad-hoc and mesh networks, as well as mobile peer-to-peer systems.

Christoph Lindemann holds the Chair of Computer Networks and Distributed Systems in the Department of Computer Science at the University of Leipzig. He received the degree Diplom-Informatiker (MSc in computer science) from the University of Karlsruhe, Germany in 1988 and the degree Doktor-Ingenieur (PhD in Engineering) from the Technische Universität Berlin, Germany in 1992. His current research interests lie in mobile computing systems, especially mobile ad hoc networks and peer-to-peer systems as well as modeling and performance evaluation as an umbrella topic.
Christoph Lindemann is a member of the IFIP working group 7.3. He was on the editorial board of the international journal Performance Evaluation from 2005 to 2010. In 2005, he served as general co-chair for the 11th International Conference on Mobile Computing and Networking, ACM MobiCom. He served as general chair of the 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation, Performance 2007. In 2010, he was program co-chair for the 11th ACM International Symposium on Mobile Ad Hoc Networking and Computing, ACM MobiHoc.