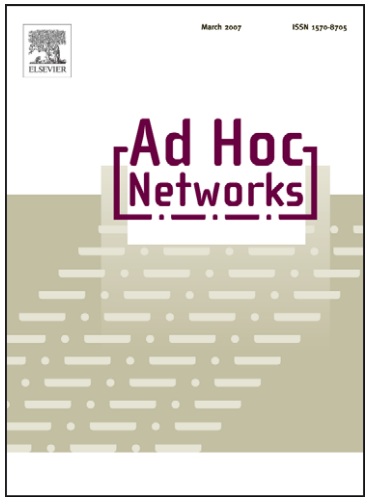


Implicit Hop-by-Hop Congestion Control in Wireless Multihop Networks

Björn Scheuermann, Christian Lochert, Martin Mauve

PII: S1570-8705(07)00015-7
DOI: 10.1016/j.adhoc.2007.01.001
Reference: ADHOC 223

To appear in: *Ad Hoc Networks*

Received Date: 22 March 2006
Revised Date: 21 December 2006
Accepted Date: 3 January 2007

Please cite this article as: B. Scheuermann, C. Lochert, M. Mauve, Implicit Hop-by-Hop Congestion Control in Wireless Multihop Networks, *Ad Hoc Networks* (2007), doi: 10.1016/j.adhoc.2007.01.001

Implicit Hop-by-Hop Congestion Control in Wireless Multihop Networks

Björn Scheuermann [1], Christian Lochert, Martin Mauve

Heinrich Heine University Düsseldorf, Computer Science Institute,
Universitätsstr. 1, D-40225 Düsseldorf, Germany
Phone: +49 211 81-11692, Fax: +49 211 81-11638
{scheuermann,lochert,mauve}@cs.uni-duesseldorf.de

Abstract

It has been shown that TCP and TCP-like congestion control are highly problematic in wireless multihop networks. In this paper we present a novel hop-by-hop congestion control protocol that has been tailored to the specific properties of the shared medium. In the proposed scheme, backpressure towards the source node is established implicitly, by passively observing the medium. A lightweight error detection and correction mechanism guarantees a fast reaction to changing medium conditions and low overhead. Our approach is equally applicable to TCP- and UDP-like data streams. We demonstrate the performance of our approach by an in-depth simulation study. These findings are underlined by testbed results obtained using an implementation of our protocol on real hardware.

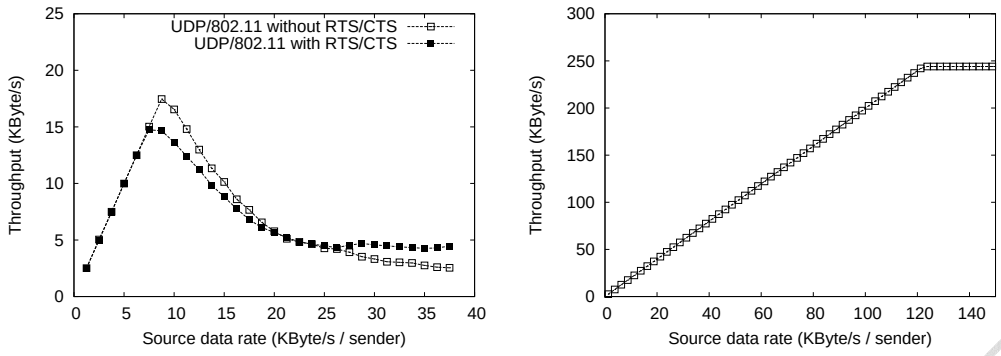
Key words: Congestion control; Wireless multihop networks; Shared medium

1 Introduction

It has become more and more apparent that wireless multihop networks are much more prone to overload-related problems than traditional wireline networks like the Internet. Appropriate congestion control is thus vital to ensure network stability and acceptable performance.

TCP congestion control, which is one of the major foundations of today's Internet, has proven to be highly problematic in wireless multihop networks [1–5]. Severe fairness problems, suboptimal throughput and throughput stability issues have

[1] Corresponding author.



(a) Wireless network. (b) Wired network.

Figure 1. Obtained throughput in bidirectional chain topology.

been reported. Such effects have also been observed experimentally in real multihop wireless networks, e. g. in [6]. Recently it has even been shown that TCP can generally not work as well in those networks as it does in common wired networks, because the rates of multiple TCP flows do not necessarily converge to a fair sharing of the bandwidth due to the shared medium [7].

All this is not too surprising, considering the fundamentally different properties of wired networks and multihop wireless networks. In a wireless network, the medium is a locally shared broadcast medium. Thus, congestion is a spatial phenomenon: neither nodes nor links, but geographical regions of the network are overloaded. The impact of this can be demonstrated by a very simple simulation experiment. In Figure 1(a), ns-2 simulation results of a bidirectional 10-hop chain topology are shown. In the simulation, static routing and the IEEE 802.11 MAC protocol are used. UDP constant bitrate traffic is injected with increasing data rate at both ends of the chain, traveling towards the opposite end. It can clearly be seen that the obtained total throughput drops rapidly once an optimal load is exceeded. This is due to an increasing number of collisions, leading to more and more packet drops. In a wireline network, throughput degrading effects for a too high UDP load are also well-known; they led to the development of TCP congestion control, which is able to deal with these problems very well. However, the problem observed here is of a completely different nature, which becomes immediately clear if we set up an equivalent wired topology in ns-2 and measure the throughput in the same scenario, as done in Figure 1(b)—due to the duplex connections used between the routers on the Internet, the two packet streams do never even share a link or queue, and thus of course maximum throughput is achieved and maintained.

Note that the observed throughput drop is not a routing effect, since we use static routing without routing overhead and with no link breaks. Enabling RTS/CTS does not improve the situation significantly, and is thus not of help here. It is also not a problem of TCP congestion control, since TCP is not used. But TCP is also not an appropriate solution to the problem since, as explicated at the beginning of this

section: TCP-like congestion control doesn't behave well in wireless multihop networks. The results of the experiment demonstrate the spatial nature of congestion in wireless multihop networks. The problem is fundamental, and needs to be taken into consideration by any wireless multihop network design. Thus, the congestion problem in wireless multihop networks deserves to be reconsidered, and there is a need to search for better suited ways to perform congestion control.

Here, we propose a novel congestion control concept for wireless multihop networks, and a concrete protocol design realizing this concept. Our approach takes the local broadcast property of the wireless medium into account. It considers radio interference and can deal with an interference range that is—as it is the fact in real networks—much larger than the possible transmission range. It applies equally well to both TCP- and UDP-like traffic. It is a cross-layer approach, recombining functionality from different traditional layers, while still preserving an overall separation of functionality.

Our congestion control mechanism is a hop-by-hop approach. Traditionally, hop-by-hop congestion control means that local feedback on the sustainable rate for each node is transmitted to the respective upstream node, in order to establish some kind of backpressure towards the source. Here, we go one step further and actively exploit wireless multihop medium properties. We do not just try to cure the symptoms the traditional protocols exhibit when used in a environment they have not been designed for. In fact, we perceive the properties of the medium not primarily as a handicap, but instead also as a chance—a chance to solve problems like congestion control in new, different ways.

We call our congestion control approach implicit hop-by-hop congestion control, because its foundations are the hop-by-hop nature and implicit feedback, i. e., information gained by observing the transmissions of other nodes in the neighborhood. Even more central, the rate regulation at the source also happens implicitly, just by obeying some simple packet forwarding rules. The protocol design proposed here that realizes the concept of implicit hop-by-hop congestion control is called *cooperative cross-layer congestion control* (CXCC), because it relies on the cooperation of multiple layers.

Various factors can influence the congestion situation in a network. However, the most fundamental ones stem from the medium properties. The effects that we have shown above are not caused by, e. g., routing or mobility. There is a great variety of different routing paradigms that have been proposed for wireless multihop networks. By using static routing tables for the evaluations presented here, we stay neutral with regard to the routing protocol, and we avoid effects that might occur only in conjunction with a specific protocol, and thus cannot be unrestrictedly generalized.

The remainder of the paper is structured as follows. In Section 2 we will review

some related work. Thereafter, in Section 3, the behavior of wireless multihop networks in congestion situations will be analyzed and the principles of implicit hop-by-hop congestion control will be deduced from these insights. Following that, we will apply the developed principles in a basis variant of a concrete protocol design, the CXCC protocol, in Section 4. In Section 5, we will analyze the performance of this basic approach, point out some remaining problems and show how they can be overcome. How we envision the integration of our approach into the protocol stack is detailed in Section 6. We will then assess the performance of our approach in detail by a simulation study in Section 7. To show that our concepts also work on real hardware, we present a real-world implementation and some experimental measurements in Section 8. Finally, we conclude our paper in Section 9.

2 Related Work

In recent publications several approaches for congestion control in wireless multihop networks have been proposed. Most of them can be classified based on whether they seek to improve TCP or propose alternative approaches. *TCP improvements* deal with adaptions of TCP to the special wireless characteristics. *Alternative approaches* are specifically designed to the wireless multihop medium and are mostly independent of the mechanisms of TCP. We only describe approaches that are relevant to our work here, a more complete overview of congestion control proposals for mobile ad-hoc networks can be found in [8].

2.1 TCP Improvements

A lot of work has been done in the area of TCP improvements that take the characteristics of wireless networks into account, examples are [9–14]. In the following, we discuss only two representatives of this class. For both approaches simulator implementations are publicly available. We use them for comparison purposes in the evaluation of our own protocol.

Fu et al. present ADTCP in [13]. ADTCP is motivated by a key problem of end-to-end transport protocols in mobile ad-hoc networks: the noisiness of the measurements of indicators for certain network events. To overcome this, different metrics are used: the inter-arrival time of two successive packet, out-of order packet arrivals, the current packet loss ratio, and the short-term throughput in the immediate past. ADTCP combines these metrics in order to obtain a more accurate and robust estimate of the network situation, which then helps to react more appropriately. In ADTCP, the receiver detects the most probable current network state and includes this information into its feedback to the sender.

ElRakabawy et al. propose TCP with Adaptive Pacing (TCP-AP) [14], also an end-to-end approach. In TCP-AP, the focus is on avoiding large packet bursts. For this purpose, the packets that are allowed to be sent out by the TCP congestion window are paced adaptively. The authors define the 4-hop propagation delay as the time between the transmission of a packet by the TCP source node and its reception by the node four hops downstream. TCP-AP estimates this value from the round-trip time (RTT) of the packets. In combination with the coefficient of variation of the RTT samples the estimate is used to establish a minimum time between two successive packet transmissions.

2.2 Alternative Approaches

Our own approach clearly falls into the category of alternative approaches. Therefore, we investigate the work in this area more closely.

The Ad-hoc Transport Protocol (ATP) by Sundaresan et al. [15] is a rate-based, network supported transport protocol for mobile ad-hoc networks with end-to-end congestion control. The authors consider TCP's mechanisms as inappropriate for ad-hoc networks. Thus, ATP is designed as an "antithesis" to TCP. ATP strictly separates congestion control from reliability mechanisms and requires only limited feedback from the receiver. The intermediate nodes piggyback the maximum queuing delay along the route on the packets passing by. This information is then used to determine the appropriate rate at the source node. A similar approach to ATP is EXACT by Chen et al. [16]. In EXACT, not delays are transmitted, but the intermediate nodes calculate sustainable rates for each flow directly; these are then piggybacked. To accomplish this, EXACT, in contrast to ATP, requires state information for each flow in the intermediate nodes. Both approaches show that effective end-to-end congestion control can be performed in different ways than with TCP, and that it can be tailored for mobile ad-hoc networks. However, both approaches control the rate only at the source node, based on feedback from within the network. The time until the feedback eventually arrives at the source node is relatively long, considering the rapidly varying medium conditions. By using in-network reaction to congestion, our scheme guarantees an immediate, localized adaptation to a changing environment.

Yi and Shakkottai [17] discuss the usage of hop-by-hop congestion control for wireless multihop networks. Their theoretic approach shows the feasibility of hop-by-hop congestion control in these networks. It provides a good basis for the theoretical understanding of the behavior of congestion control feedback schemes. From a more practical point of view, however, it has to be considered that the assumptions they make are not fulfilled by today's common wireless hardware. The proposition that in a wireless network any two links not sharing a common node are completely independent from each other cannot be met. Additionally, their approach uses ex-

plicit feedback to the upstream nodes, and therefore imposes load for the feedback traffic on the network. In contrast our scheme avoids additional load during congestion situations.

A concept that is in some aspects similar to ours has been used in the DARPA packet radio network with the "adaptive pacing" protocol [18]. Packet radios using this protocol wait for a "passive acknowledgment" before the next packet is transmitted along a link. This is combined with rate throttling based on delay measurements. For these measurements their protocol requires to maintain state information for each neighbor of a packet radio, which will quickly become outdated. Thus, we assume that the central aspect of their work—the measurement and calculation of inter-packet delays (or rates, accordingly)—cannot be immediately adopted for wireless multihop networks with IEEE 802.11 like physical layers. Our simulations underline this assumption.

"Passive" or "implicit" acknowledgments obtained by overhearing have been mentioned in several publications since being introduced for packet radio, e. g. with the DSR routing protocol [19]. In our approach, implicit acknowledgments are one central element of the congestion control mechanism.

Zhai at al. propose an approach which they call Optimum Packet scheduling for Each Traffic flow (OPET) [20,21]. It consists of four mechanisms to reduce the impact that wireless medium contention has on throughput and fairness in MANETs. One is a hop-by-hop backpressure scheme, similar to both packet radio adaptive pacing and our own approach. Their mechanism is used in combination with standard TCP, and the backpressure mechanism is tightly coupled to the RTS/CTS mechanism in 802.11, using explicit "Negative CTS" packets. While both design decisions are very reasonable when compatibility must be preserved, we chose a more radical approach. We replace TCP altogether and thus avoid any problems of possible feedback loops between congestion control on multiple layers of the protocol stack. Furthermore, our scheme is able to send fewer explicit messages and thus minimizes the load on the network.

Hop-by-hop congestion control has been a basis of several proposals in the context of wireless sensor networks (WSNs). In WSNs, typical traffic consists of relatively small packets and is directed to or from a small number of special nodes, the sinks. Thus, the sensor network approaches have in common that they consider packet flows that are directed to or from such a sink, whereas we primarily consider unicast traffic among arbitrary pairs of nodes. A survey on congestion control in sensor networks is provided in [22].

Implicit acknowledgments have also been used in the WSN context, e. g., by Woo and Culler [23]. There, they are primarily used to reduce the control packet overhead, by saving the bandwidth for the acknowledgment in the RTS/CTS/DATA/ACK handshake. In the same paper, an AIMD-based backpressure rate control approach

is also proposed. In their scheme, the nodes maintain a probability with which a packet is allowed to be sent out or forwarded, essentially determining the aggressiveness of the medium access. The probability is increased or decreased based on the observation of the downstream node's forwarding behavior. This yields congestion feedback that propagates backwards towards the sources. In our scheme, no windows, rates or forwarding probabilities need to be maintained.

The Congestion Detection and Avoidance scheme (CODA) [24] was also one of the first sensor network approaches. In CODA, the intermediate nodes broadcast explicit backpressure messages as long as they detect congestion. These are then propagated towards the sources. Our scheme does not use explicit congestion messaging, to avoid the generation of additional network load in a congestion situation.

In [25] the authors adopt hop-by-hop backpressure for wireless sensor networks as one of several mechanisms to improve performance. In their scheme, if a node detects congestion, a "congestion bit" is set in its outgoing packets. When the children of the node in the routing tree towards the source overhear packets with this flag being set, they stop sending packets.

Since no dedicated congestion messages are used, schemes like [23] and [25] are also often called "implicit". Note, however, that our notion of implicit here is slightly different: in our work, "implicit congestion control" means that the congestion control itself works without explicitly maintaining window sizes or packet rates, without any node explicitly deciding whether there currently is congestion or not, and without any congestion feedback, be it explicit or piggybacked.

A general problem with piggybacked congestion feedback is that it requires packets to be sent in order to inform the neighbors of the congestion. If a node cannot send a packet due to an occupied medium, it will not be able to signal congestion. This problem does not exist in our scheme.

SenTCP is another hop-by-hop congestion control protocol for wireless sensor networks [26]. As in some of the previously described approaches, explicit feedback is used in this scheme, whereas our scheme carefully avoids explicit feedback.

In the Reliable Bursty Convergecast protocol (RBC) [27], the authors specifically focus on the equally named traffic pattern in sensor networks, where many nodes—triggered by a commonly observed event—transmit a lot of small packets in parallel to a single sink. In RBC, a sophisticated queue management scheme is combined with piggybacked feedback. This feedback is on the one hand used as a replacement for link layer acknowledgments, on the other hand nodes use overheard feedback from peers in the vicinity to determine channel access priorities. Multihop backpressure in order to regulate source nodes' data rates, as it is provided by our approach, is not considered. In their setting this is not crucial, since there is no continuous data stream, but single packets from many sources.

3 Algorithmic Idea

3.1 Shared Medium Model

To motivate the implicit hop-by-hop congestion control approach we introduce a very simple model for the effects of the shared medium. It is easy to see that in any part of a network—be it wired or wireless—, on a sufficiently long time scale to avoid short-term effects, the output rate of traffic forwarded through this area (OUT) cannot exceed the forwarded traffic input rate (IN). We denote this fact by

$$\mathrm{OUT} \leq \mathrm{IN}. \tag{1}$$

In common wireline networks, there are also separate, independent upper bounds for the input and output data rates, given by the bandwidth of the respective links. If we denote the total ingress link bandwidth into the area under consideration by BWIN, and similarly the total outgoing bandwidth by BWOUT, we have

$$\mathrm{IN} \leq \mathrm{BWIN} \tag{2}$$
$$\mathrm{OUT} \leq \mathrm{BWOUT}. \tag{3}$$

The important point is that, in this constellation, it is not of immediate disadvantage for the throughput if the input rate exceeds the output rate. Of course this will lead to dropped packets, but these drops will occur *before* the bottleneck—in this case the output—, and maximum throughput will still be obtained.

This is the foundation of any common end-to-end congestion control. TCP, but also, e. g., the TCP-Friendly Rate Control (TFRC) protocol [28], utilize packet drops to determine the bottleneck bandwidth. In order to enable this, drops are actually provoked: TCP congestion control aims to keep the buffers in the network full, causing strict inequality in (1).

If we consider the situation in a wireless multihop network, the shared medium adds a new aspect to the above considerations. Instead of independent bounds on the input and output rate as in (2) and (3), ingoing and outgoing links within a collision domain share the available bandwidth. This leads to a constraint of the form

$$\mathrm{IN} + \mathrm{OUT} \leq \mathrm{BW}, \tag{4}$$

where BW is the abovementioned shared total bandwidth.

This has an important implication: here, the output rate cannot be optimal in the case of strict inequality in (1). Therefore, the situation for the congestion control is completely different. Due to (4), increasing the input rate beyond the output rate will result in a sub-optimal overall throughput. This is one key reason for TCP's fundamental problems in wireless multihop networks.

These observations can also serve as an explanation for the previously described throughput breakdown in case of network load beyond the optimal point. Excessive network input will yield a more and more extreme inequality in (1), an increasingly dominant role of the input in (4), and consequently leads to decreasing throughput.

The implicit hop-by-hop congestion control proposed here is based on a completely different feedback paradigm than TCP. Our approach aims to achieve strict equality in (1) even on a short-term time scale. The key observation is that this is actually possible in wireless multihop networks, by actively exploiting the local broadcast property.

3.2 Implicit Hop-by-Hop Congestion Control

We define a flow as a directed pair of communicating nodes, i. e., all the packets traveling from a node *A* to another node *B* belong to the same flow. Other definitions are possible, but since neither the idea of implicit hop-by-hop congestion control nor the CXCC protocol necessarily depend on this definition this is only a matter of form. With implicit hop-by-hop congestion control, the protocol enforces that the input rate for a given flow does not exceed the output rate at any intermediate node. This is accomplished by preventing the transfer of a second packet to a node until this node has forwarded the previous one. Every node along the route thus queues at most one packet of a flow, and no further packet is forwarded until the queue space at the next node is free again. We call a flow *blocked* in a node when there is no space for the next packet available at the downstream node.

The mechanism of waiting for the next hop to forward one packet before passing on the next one leads to a fast and efficient implicit backpressure mechanism towards the packet source. If a node is not able to forward a packet immediately, it will thereby implicitly stop the input flow from its predecessor, and so on, until the packet source itself will not be allowed to send the next packet. The concept ensures that the input rate cannot exceed the output rate, and it yields a very fast reaction to changing medium condition. If the forwarding is delayed, the backpressure is established immediately. Furthermore, such a mechanism works for both TCP-like and UDP-like traffic, whereas TCP congestion control as well as most other approaches depend on reliable transmission: in our approach, the backpressure is independent from reliability mechanisms.

Another possible interpretation of implicit hop-by-hop congestion control is that a node is only allowed to transmit if there is a "hole" at the downstream node. Thus, these holes propagate along the route in reverse direction. Before a network bottleneck, where backpressure builds up, holes are rare. The nodes there are not allowed to transmit until a hole has propagated through the bottleneck area; therefore, the input into the congested area is limited. Behind the bottleneck, the holes dominate, meaning that the output of the congested area is not constrained.

For the backpressure mechanism to work, a node has to know whether the downstream node still has a packet of the same flow in its queue. This information could be sent explicitly, but such a mechanism would induce additional traffic and increase the congestion of the network. In our approach, the shared nature of the medium is exploited to gain the necessary information at no additional cost. In most wireless technologies that are considered as a basis for multihop communication, transmissions are de-facto broadcasts: the radio waves propagate to all nodes in the vicinity of the sender, in particular also to the upstream node. Then the forwarding of a packet by the downstream node can be overheard. For our approach, we assume that this is possible. The central rule of implicit hop-by-hop congestion control is then very simple: if the forwarding of the previously sent packet is overheard, this indicates that the next one may be transmitted (Figure 2(a)).

At the same time this implicit notification about a free queue space can—as a side effect—serve as an implicit acknowledgment, indicating successful packet delivery to the next hop. Note that this requires no additional, piggybacked header fields. If it has a unique ID, the packet is sufficient as-is. Common MAC layer acknowledgments are therefore no longer required. This is also called "passive acknowledgment" in the literature.

The ability to overhear transmission exists in nearly all multihop-capable wireless interfaces that are in use today, for some others it can be achieved by appropriate modifications to the lower layers. E. g., if power control is used, it would be necessary to ensure a transmission power that allows both the next hop node and the previous node to receive the packet. With directional antennas, however, one might have to resort to sending a separate, explicit ACK frame to the previous node after the transmission.

Due to backpressure or medium contention, the forwarding can be delayed. Thus, the nodes have to tolerate a significant delay before the implicit acknowledgment. We don't see this as a problem; instead, we actually realize this kind of delay tolerance as a necessity for robust multihop wireless protocols. This leads to another key concept of our approach, which we call *soft timing*. Due to the local broadcast property of the wireless medium it is very hard to predict when a chance to transmit a packet will arise. Thus, it is also not unrestrictedly possible to guarantee a collision-free answer from a specific node within a certain, tight time frame. Afterall, the medium around this node might be busy, preventing it from sending.

802.11's way to deal with this issue is to ignore carrier sensing when transmitting MAC layer acknowledgments; however, this can cause additional collisions and therefore wastes bandwidth. We thus advocate to use asynchronous answers within a comparatively long period of time—potentially only after several other transmissions have happened.

Obviously, when a packet reaches its final destination node, there will be no further forwarding. Thus, the packet cannot be implicitly acknowledged. The destination node therefore needs to send an explicit acknowledgment. By delaying this acknowledgment if the packet cannot be immediately passed up the protocol stack, an integration with flow control is possible—that this is tolerable for the protocol can be seen as a consequent generalization of the soft timing principle.

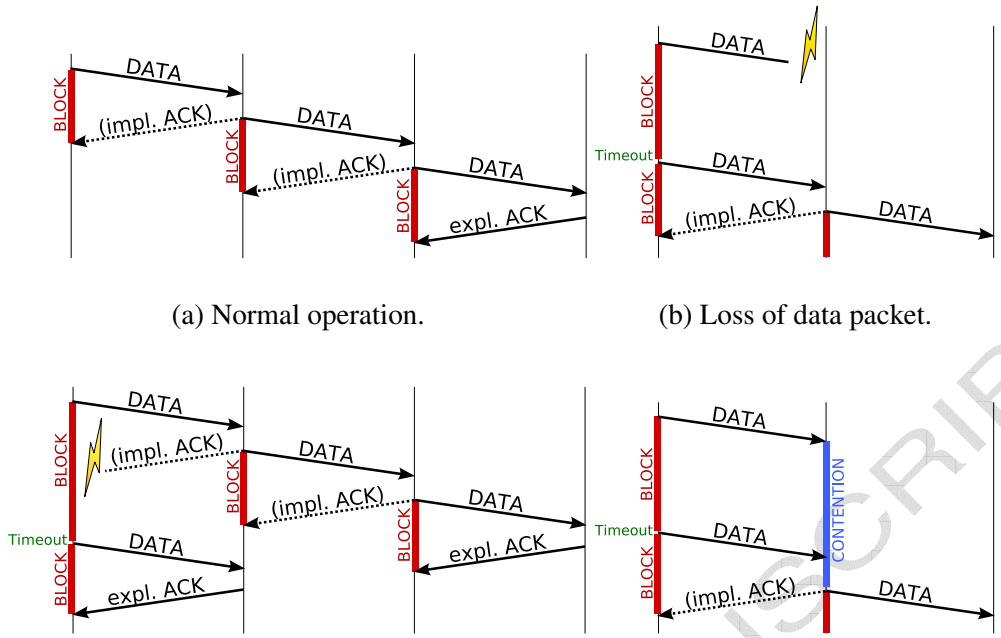
It is conceivable to allow a higher number of unacknowledged packets to be transmitted before a node has to wait for an acknowledgment and backpressure builds up. This, however, not only increases the complexity of the protocol, it also means that typically a smaller number of queues is blocked. A higher number of nodes in the same area would be allowed to transmit, therefore more stations are potentially contending for medium access. The latter is detrimental for the network's performance, due to the reasons discussed above. Furthermore, the ability to transmit multiple unacknowledged packets would occur primarily in front of a bottleneck, but not behind the congested area, where packets can be forwarded away quickly. This increases the backlog in front of the bottleneck, the queuing delay and, consequentially, the packet latency. These substantial drawbacks outweigh potential benefits by far, and thus we consider using one single unacknowledged packet to be the best option.

3.3 Deadlock Freeness

In implicit hop-by-hop congestion control, flows are stopped completely when backpressure occurs. Thus, it is crucial to verify that this blocking will eventually be released, i. e., that no deadlocks can occur where flows are blocked indefinitely. However, it is easy to see that our scheme will not run into such a deadlock situation.

When a queue is blocked in a node, then this is because it waits for the next node downstream to forward another packet of the same flow. This node might in turn wait for the next node downstream, and so on. Eventually, the queues will, directly or indirectly, all wait for the destination node of the flow. Thus, as long as the destination node is accepting packets, the blocking will be released.

In order to formalize and generalize this statement, we define the wait graph of a network. The wait graph is the directed graph where the vertices are the queue instances in all nodes, and where a pair of queues (q_1, q_2) is in the set of edges if



(a) Normal operation.

(b) Loss of data packet.

(c) Loss of implicit ACK.

(d) Early timeout.

Figure 2. Basic CXCC protocol operation.

and only if q_1 is currently waiting for q_2. If the wait graph is loop-free, then there is no deadlock situation: all queues will eventually wait either for a non-blocked queue or for the destination node. In our scheme, the wait graph is always loop-free if the routing is loop-free. This holds because a queue q_1 of flow f in node n_1 can wait—directly or indirectly—only for another queue q_2 if q_2 also belongs to f and is located at a node n_2 such that n_2 is further downstream on f's route.

This is the reason why implicit hop-by-hop congestion control works on a per-flow basis. One could also think of a scheme where a node generally waits for the next node to forward the previously sent packet before the next one is transmitted, regardless of the flow it belongs to. In this case, however, the loop-freeness of the wait graph would not be ensured.

4 Basic CXCC

In the previous section, the general idea of implicit hop-by-hop congestion control has been reasoned and explained. Now we focus on how to realize this idealized, abstract idea in a real network, where packet losses due to collisions and other adversities are common. In order to do so, we will describe a basic version of our CXCC protocol, using the previously introduced concepts.

4.1 Dealing with Lost Packets

In wireless multihop networks it is common that a packet transmission is unsuccessful. For implicit hop-by-hop congestion control packet loss poses a serious threat: a packet loss will block the transmission of data for the flow that the lost packet belonged to. Thus, CXCC needs to be able to recover from packet loss efficiently.

The most basic loss situation is that the next hop node does not receive a data packet transmission (Figure 2(b)). No more packets would then be forwarded: since the next hop will of course not forward a packet it has never received, no implicit acknowledgment will arrive. The simple solution to this problem in the basic CXCC protocol is to repeat the packet transmission if, after a certain timeout, no acknowledgment has been received.

A data packet transmission is not only used to communicate the data in downstream direction, but also to acknowledge the reception of the packet at the same time implicitly. The situation that the upstream node, waiting for the acknowledgment, does not receive the packet is also possible (Figure 2(c)). The upstream node will then stop sending any further packets because it has missed the implicit ACK. Note that the upstream node is not able to distinguish this situation from the case above: it cannot tell if the next hop has not received the packet, or if the implicit acknowledgment has not been received. It will thus conduct a packet retransmission as described above.

This could lead to packet duplication because this packet has already been forwarded. Therefore, the proposed solution has to be augmented by a duplicate detection mechanism. This is easy to accomplish, since only a duplicate of the last received packet is possible. When a second copy of a packet is received, the next hop will drop the duplicate and will not forward the packet again. However, then the previous hop will again not receive an implicit acknowledgment. To overcome this, we propose the following behavior: a node sends an explicit acknowledgment when it receives a duplicate of a packet it has already forwarded and for which it has already received an (implicit or explicit) acknowledgment. If these conditions do not apply it silently ignores the duplicate. This ensures that an ACK is only sent when there will definitely be no further chance to acknowledge the packet implicitly. This avoids unnecessary transmissions.

There is a third situation that can also not be distinguished from the ones above by the upstream node: the timer of the upstream node could expire before the next hop has been able to forward the packet. This is depicted in Figure 2(d). In this case, the next hop node will have the packet in its queue and it will ignore the received duplicate. It will not transmit any explicit information to the upstream node. The reason for this behavior is twofold: firstly, an explicit acknowledgment would not be of any direct use for the upstream node, since its reaction would be the same

as if nothing at all is received: it would wait. Secondly, if the packet is still in the queue this probably means that there is network congestion. Thus, if there is a chance to transmit *anything* at all, the precious medium time should be used to transmit data—which will also serve as an implicit acknowledgment—rather than an explicit control packet.

4.2 Queuing in CXCC Nodes

The forwarding rules established above define how a basic CXCC node works. An important implication of these rules is that queuing in each node requires some attention. For each flow passing through a node a queue has to be maintained. These queues can be created on-demand when the first packet of a stream arrives. Because of the one-packet-per-hop restriction the queue has to provide space for at most two packets: one that has already been forwarded but is not yet acknowledged, and therefore needs to be cached in order to be retransmitted if necessary. The other one that has been received from the upstream node but must not be forwarded until an acknowledgment for the preceding packet arrives from the next hop node.

As one node can handle queues for different streams, it is necessary to decide out of which queue a packet shall be forwarded. Normally, one of the queues that are not empty and not blocked is chosen randomly. Retransmissions after a timeout are handled like the first transmission of the respective packet. If an explicit acknowledgment is waiting to be sent, it is given priority over data packets. More sophisticated schemes could be integrated to enforce, e. g., certain quality of service or fairness metrics.

From the discussion above it can be seen that CXCC requires to keep per-flow state in the intermediate nodes, and that it also implies a certain computational overhead. There are per-flow queues, and the duplicate detection mechanism needs to remember the last packet of each queue for some time. However, we do not consider this to be a problem in practice. For a node in, e. g., a mobile ad-hoc network, one can expect a relatively high computational power and many resources in relation to the small effective bandwidth of the shared multihop medium. Because of this small bandwidth the number of flows crossing one node is also quite limited, compared to, e. g., typical Internet routers.

Furthermore, the information does not need to be kept for a long time. When no more packets of a stream are queued in a node, the corresponding queue can be removed immediately. The information that is needed for the duplicate detection can also be removed quickly, since it can reasonably be expected that a duplicate can only be received within a relatively short time. So, the small additional overhead is perfectly reasonable and does not significantly limit the scalability of our approach.

4.3 Retransmission Timeouts

The choice of an appropriate retransmission timeout is a crucial factor for CXCC to work properly. In our implementation, the timeout TO before a packet transmission is scheduled depends only on the packet's size and thus the medium time needed for the transmission of a packet. The packet transmission time T_P can be easily calculated in the case of a constant medium bandwidth B, since the packet size s_P is known:

$$T_P = \frac{s_P}{B}.$$

We chose a value of three times the packet transmission time with a small jitter j for the timeout, so $TO = 3 \cdot T_P \pm j$. This choice might seem to be a very short time. But an elapsing packet retransmission timeout does not imply an immediate retransmission. It just *allows* the retransmission of the packet. It still has to wait until the medium is free and a packet can actually be sent, and until the respective queue is chosen out of potentially several alternatives. Thus, the actual delay until a retransmission is performed depends largely on the network conditions and current media utilization; the retransmission timeout just establishes a lower bound on this time. Therefore, some adaptivity to different medium conditions is already achieved by this simple scheme. Note that it also remains possible for a packet to be acknowledged after the timeout has elapsed, while it is waiting for its retransmission.

One could imagine more complex schemes to find a good retransmission timeout. We have conducted many experiments with different approaches. Amongst them were, for example, sliding averages over a measurement of the average delay until packet forwarding in neighbored nodes, or exponential backoff schemes for repeated retransmissions. But none of them performed better than the simple variant with a packet size based timeout. Our conclusion is that the simple scheme described above already achieves the flexibility and adaptivity that is needed to deal well with the medium properties.

5 Request for ACK

5.1 First Simulation Results with Basic CXCC

As shown in Figure 1 the throughput of an 802.11-based multihop wireless network goes up with increasing data rates, but beyond some optimal point it cannot maintain this throughput. Instead, due to collisions and retransmissions, the throughput decreases rapidly. Basic CXCC in the same topology on the other hand exhibits one

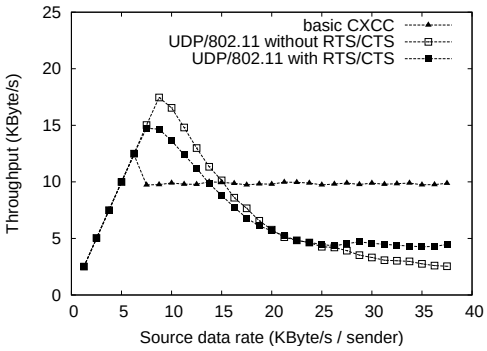


Figure 3. Obtained throughput in a bidirectional chain topology.

important characteristic: it is able to stabilize the throughput if too high input data rates are offered. Figure 3 shows this trait of CXCC.

These results are on the one hand very promising, as the basic CXCC protocol is able to handle the congestion situation and to guarantee a balanced throughput over multiple hops. However, the achieved throughput is significantly lower than the maximum achieved by UDP at the optimal input data rate. Thus, basic CXCC achieves stability, but not optimality. In the following, we will introduce an improved version of the CXCC protocol, that addresses this problem of the simple basic CXCC scheme.

The reason for the suboptimal performance lies in the packet retransmissions as they are performed in basic CXCC. Whenever a node does not receive an acknowledgment and a timeout expires, the whole data packet is retransmitted. However, if only the implicit acknowledgment has been lost, or if the next hop node has simply not yet been able to forward the packet due to contention or backpressure, it is unnecessary to retransmit the whole packet: it has already been successfully transmitted before, the payload has already arrived at the next node.

5.2 RFA Mechanism

In Section 4.1, three situations have been distinguished where basic CXCC performs a packet retransmission. It has also been reasoned that the retransmitting node is not able to tell these situations apart. But only in one situation it is actually necessary to retransmit the payload. Therefore, a strategy is missing that helps to avoid unnecessary transmissions. For the development of such a strategy it is important to know which of the three loss situations is the dominating one. We conducted simulations to obtain this information. Figure 4 shows, which of the three loss situations causes how many retransmissions in the nodes of a simulated bidirectional 10-hop chain.

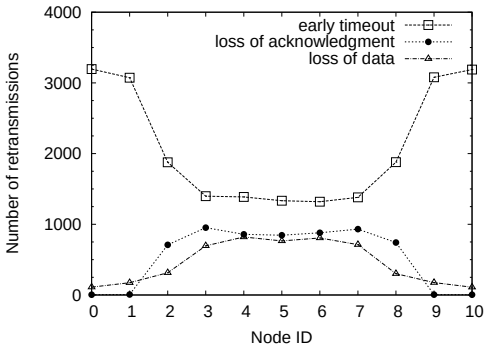


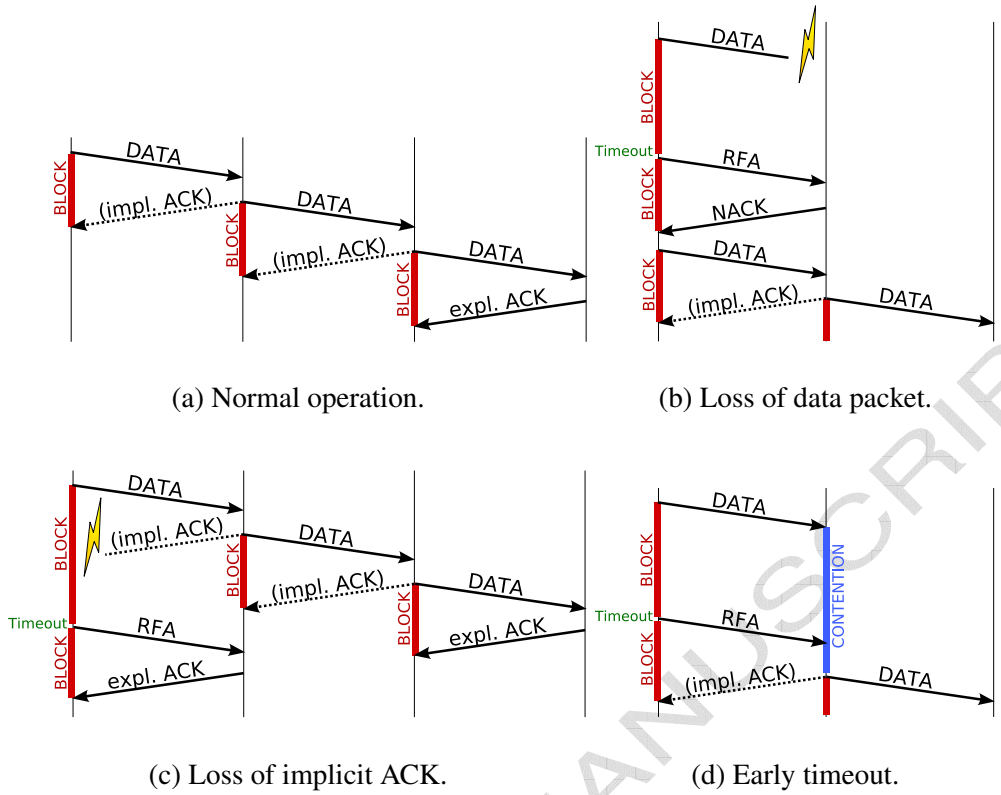
Figure 4. Reasons for packet retransmissions in bidirectional chain topology.

From this figure it becomes clear that, at least in this topology, the cases where a retransmission is performed because the timeout expires before the packet could be forwarded by the next node clearly outnumber the other two situations. Results from other topologies and the performance figures presented later in this paper point in the same direction. One possible interpretation of the distribution of the three retransmission reasons is that the timeouts introduced in the previous section are too short, resulting in too many or too fast retransmissions. However, our simulations also show that longer retransmission timeouts do not improve the performance of basic CXCC, because then the recovery from the two true loss situations happens with a too long delay. Therefrom arises the necessity for a scheme that is able to recover fast enough from a real loss situation, but at the same time avoids the large number of unnecessary packet retransmissions in the case of delay through backpressure.

Our solution to this problem is to not retransmit the whole data packet after an expired timeout, but instead just a small control packet. We call these control packets *Request For Acknowledgment* (RFA). The RFA contains all the important header information from the data packet, but not the payload. It thus provides the downstream node with all the information that is necessary to react appropriately.

We now look at the reaction of the downstream node upon reception of an RFA packet. The simplest case is when the transmission of the data packet had not reached the next hop, i. e., the RFA refers to an unknown packet. Then, a retransmission including the payload is necessary, and should happen as soon as possible. If the downstream node detects this situation, it thus tells the upstream node to retransmit the full packet by sending an explicit negative acknowledgment (NACK) frame. This is depicted in Figure 5(b). Because of the additional RFA-NACK-handshake the overhead in this case is actually higher than with basic CXCC. However, as the previously stated simulations indicate, this occurs rather seldom. In the other two cases, a retransmission of the data packet is not necessary, and thus a lot of otherwise unnecessarily occupied medium time can be saved with the RFA scheme.

When just the implicit acknowledgment has been lost although the packet had actually been correctly forwarded, the downstream node sends an explicit acknowledg-



(a) Normal operation.

(b) Loss of data packet.

(c) Loss of implicit ACK.

(d) Early timeout.

Figure 5. Protocol operation with RFA control packets.

ment if the packet has been forwarded further and acknowledged by *its* downstream node, as explicated above. This behavior, shown in Figure 5(c), is the same as for basic CXCC.

In case of an early timeout, the reaction is also identical to that of basic CXCC: no action is performed, as shown in Figure 5(d). Therefore, the only significant difference in this as well as in the previous case is that the payload has not been retransmitted and thereby medium time has been saved.

It should be noted that for an upstream node an overheard RFA packet can serve as an implicit acknowledgment for the packet it refers to.

6 System Architecture

While the previous sections dealt with the CXCC protocol itself and its functions, we now discuss how it can be integrated into a protocol stack. Since CXCC is a cross-layer protocol, there are some interesting aspects that deserve attention.

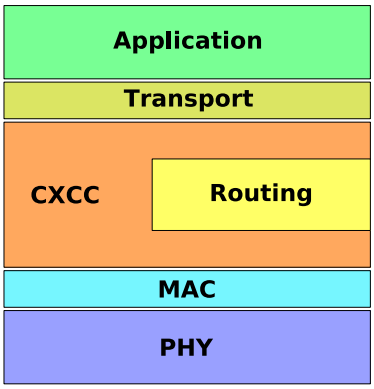


Figure 6. CXCC's position in the protocol stack.

We are aware that cross-layer designs require significant control over the network participants' protocol stack. However, our point is that the cost of preserving a strictly layered architecture is too high, considering all the fundamental negative results concerning the common protocol stack. Since wireless multihop networks can often be expected to be closed systems with sufficiently homogeneous devices, potentially even tailored for one specific application, we consider the required modifications feasible.

An overview of the CXCC protocol stack is shown in Figure 6. At the transport layer, the congestion control function is moved to the CXCC module. Therefore, the transport layer protocol can focus on its main tasks: providing process-to-process communication (i.e., ports), and, optionally, byte-stream-based, connection-oriented communication primitives, preservation of packet ordering and/or reliability. Note that a different transport layer protocol does not necessarily mean a different interface to the application: a TCP/UDP-compatible socket interface to the application layer can be provided with CXCC, meaning that existing applications do not need to be changed.

CXCC does not interfere with the core functionality of the routing layer. This is important in order to stay independent from a specific routing approach. CXCC essentially embraces the routing protocol. This is necessary since CXCC components need to reside both above and below the routing protocol. They communicate, bypassing the routing protocol. In a practical implementation CXCC can take over the responsibility for layer 2 and 3 packet queuing completely. This simplifies the maintenance of the per-flow queues, and the routing protocol can concentrate on its most basic functionality: providing information on the next hop towards a given destination node. This implementation aspect is, however, not mandatory, and it is independent from CXCC's congestion control functionality.

Finally, the MAC layer is also reduced to its core responsibility: observing the medium and deciding when a transmission is allowed to take place. There is no need for link layer retransmissions if CXCC is used. Such mechanisms have been introduced in wireless MAC protocols to overcome the inherent unreliability of the

medium. However, CXCC is aware of the medium properties and does not need this support. Retransmissions and acknowledgments are handled by CXCC itself in a very efficient manner. Likewise, CXCC does not use the RTS/CTS mechanism for virtual carrier sensing. Our simulation results show clearly that RTS/CTS has in fact a mostly negative performance impact in wireless multihop networks. This observation is in accordance with many previous results, e. g., in [2,29]. RTS/CTS has been designed for single-hop wireless networks, and does not fully solve the hidden terminal problem in a multihop environment, but instead causes new problems, like false blocking due to overheard, but failed RTS/CTS handshakes [30].

One additional difference between the protocol stack of a CXCC node and the common one is that there exists no interface queue between CXCC and the MAC. Instead, the queuing is handled by CXCC. The MAC provides feedback on when a packet may be sent. The reason why we modify the common interface to the MAC layer is easy to see: with the common interface queue concept, it would be possible that a packet is enqueued, but can not be sent immediately. It thus might happen that before the transmission can be started, other messages are received that redundantize the waiting transmission—like an acknowledgment for this packet. In this case an unnecessary transmission would be performed, even though the transmitting node actually has the knowledge that it is of no benefit. To avoid this effect, we have removed the interface queue. Another approach would have been to provide means of altering the interface queue upon demand. However, this would require a much more complex interface between the layers, and thus contradicts our intention to keep the functional separation clean.

Due to the MAC modifications CXCC can not immediately be used with most commodity wireless hardware available today. This is because typically significant parts of the MAC implementation are located in the firmware and cannot easily be changed. However, the firmware modifications that are necessary in order to allow using CXCC on existing hardware are limited. And, as our results show, the possible gain is significant. More importantly, many, if not most, wireless multihop networks will be application-specific, and will often be based on application-specific hardware.

7 Simulations

In order to examine the performance of CXCC we have performed extensive simulation studies. For the simulations in this paper the ns-2 network simulator [31], version 2.29 has been used. In this section, we present the results of the simulations.

The evaluation is based on different scenarios. First, we compare the behavior of CXCC and that of UDP traffic over IEEE 802.11 in four different topologies, where packets are produced by constant bit rate (CBR) traffic sources with increasing

frequency. This provides us with some general insights on how well CXCC is able to adjust a source's rate in order to utilize the capacity of the network.

Thereafter, we compare CXCC to three other congestion control protocols: TCP NewReno [32], ADTCP [13], and TCP-AP [14]. The latter ones are end-to-end approaches that aim to improve TCP performance in mobile ad-hoc networks. For our comparisons, we use the respective protocol implementations that have been made available by the authors.

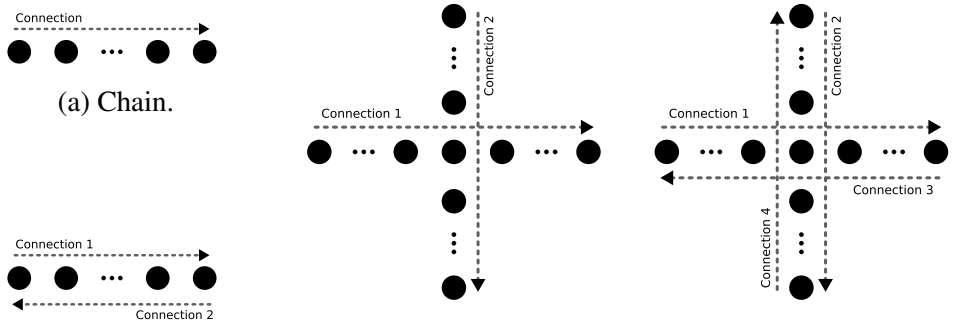
In all simulations presented here the packet size is set to 512 bytes. We have also performed simulation runs with smaller and larger packets. Although the absolute results were of course different, the overall relative outcome remained the same. The network bandwidth for all simulations is fixed to one megabit per second. We use two different propagation models in our simulations, the two-ray ground model and the log-normal shadowing model. The two-ray ground model is the most widely used simulation model, and therefore guarantees some degree of comparability of our results to previous work. The two-ray ground model is used with the typical settings of 250 meters radio range and 550 meters interference range. All simulation results presented in Sections 1 and 5.1 have been generated using the two-ray ground model.

The log-normal shadowing model can generally be considered to be a little closer to reality. This model also includes a random component that makes transmissions nondeterministic. For the simulations with the shadowing model, we set the model parameters corresponding to a shadowed urban area setting (loss exponent $\beta = 4$, shadowing deviation $\sigma_{\mathrm{dB}} = 4$). The reception and carrier sense thresholds have been adjusted to yield a 95 % probability to receive a transmission over a distance of 250 m, and to sense it at a distance of 550 m. This allows us to use the same topologies for simulations with both radio models, and to compare the obtained results directly.

We use static routing with optimal routes (with regard to the hop count), in order to remove any influence of a routing protocol and to focus solely on the inherent difficulties of congestion control in a multihop wireless network. In this static setting, route breaks do not occur, and CXCC's retransmissions and RFAs can guarantee single-hop reliability. Therefore, end-to-end reliability is ensured in our CXCC simulations, without additional measures.

7.1 Deterministic Topologies

In our simulations with deterministic topologies and fixed packet streams we consider the four different scenarios shown in Figure 7. All of them are based on 10-hop equidistant chains, with neighbored nodes placed 200 meters apart. We have also performed simulations with different chain lengths, obtaining very similar results.



(a) Chain.

(b) Bidirectional chain.

(c) Cross.

(d) Bidirectional cross.

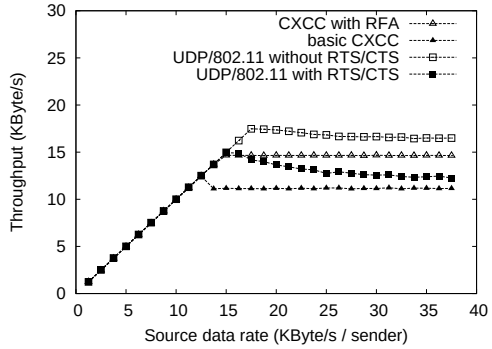
Figure 7. Deterministic Simulation Topologies.

The first and second topology are based on one of these chains. Along this chain, we use one single connection from the first to the last node in the chain as our first scenario ("chain"). In the second scenario, we added a second stream of packets to the same topology, running from the last towards the first node, that is, in the opposite direction ("bidirectional chain").

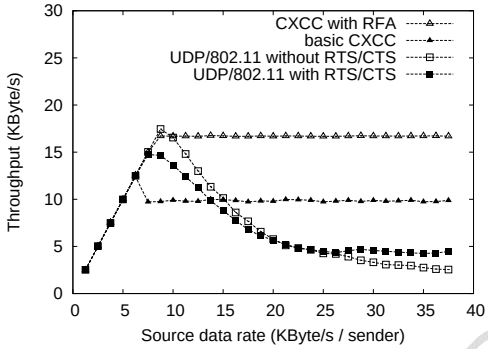
The other two scenarios are based on two chains, one in x, the other one in y direction of a plane. They cross each other in the middle of the chains, where they share one common node. In this "cross" topology we use two streams, one along each of the two chains. "Bidirectional cross" has four data streams altogether, one starting at each chain end, with their destinations at the opposite end of the respective chain.

Like in the bidirectional chain simulations described before in Sections 1 and 5.1, the CBR sources at the end of the chains were configured to produce packets at a fixed data rate, which we varied in a broad range. Here, we give results for these topologies with UDP traffic over IEEE 802.11 with RTS/CTS enabled and disabled, and for the CXCC protocol in the variants with and without RFA. Since we can adjust the amount of data that is injected in the network freely for UDP, this allows us to draw conclusions on the performance that some *arbitrary* 802.11-based protocol would achieve, if it chose to adjust its output to a certain rate. The comparison with CXCC then shows how the rather different way of packet forwarding with implicit congestion control deals with the same data rate being generated by the application.

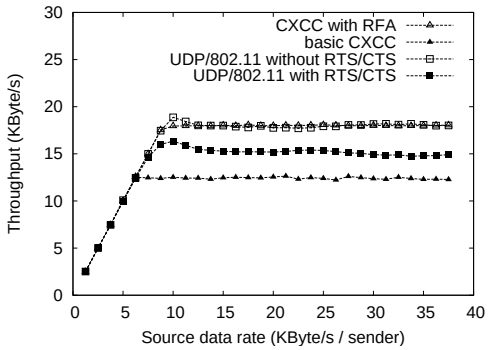
Figure 8 shows how for each of these four protocol variants the obtained throughput develops with increasing source data rate. The results are averaged over 10 simulation runs with different random seeds for backoff and jitter. It can be seen that the UDP traffic is able to sustain a good throughput for the topologies without two-way data transport. There, 802.11-like packet forwarding seems to be self-regulating. However, as discussed before, for bidirectional traffic the throughput suffers substantially if the input data rate exceeds a very small range. This implies that a congestion control approach on top of 802.11 needs to adjust its output rate quite exactly.



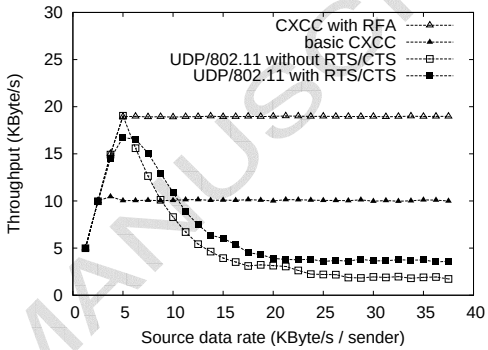
(a) Chain topology.



(b) Bidirectional chain topology.



(c) Cross topology.



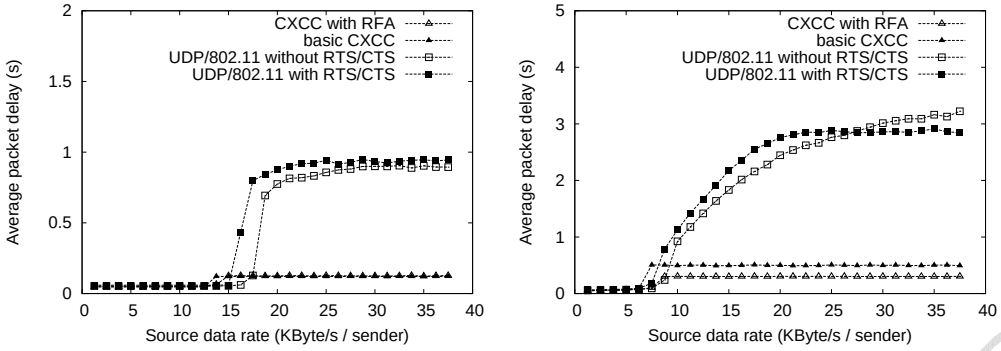
(d) Bidirectional cross topology.

Figure 8. Obtained throughput (two-ray ground).

CXCC with the RFA extension is able to achieve and maintain very good throughput in all four scenarios. While basic CXCC without RFA suffers from a large number of spurious packet retransmissions, the RFA mechanism allows the protocol to use network resources well and provides the sender effectively with feedback on the best sending rate. By comparing the number of packets delivered for each single connection in the topologies with more than one sender-receiver-pair we found that at least in these simple, symmetric topologies all protocols considered here share the bandwidth fair among the flows.

The optima of the performance of UDP over 802.11 also mark the optimal throughput that can be achieved by any protocol that uses the common protocol stack. An ideal protocol would be able to find the optimal sending rate for UDP without inducing additional traffic. Thus, it can be seen that CXCC with RFA performs extremely well in comparison to *any* possible 802.11-based protocol, at least in those simple scenarios, by achieving a throughput very close to the optimum and maintaining it for any higher source rate.

The good performance of CXCC is further confirmed by other metrics obtained



(a) Chain topology.

(b) Bidirectional chain topology.

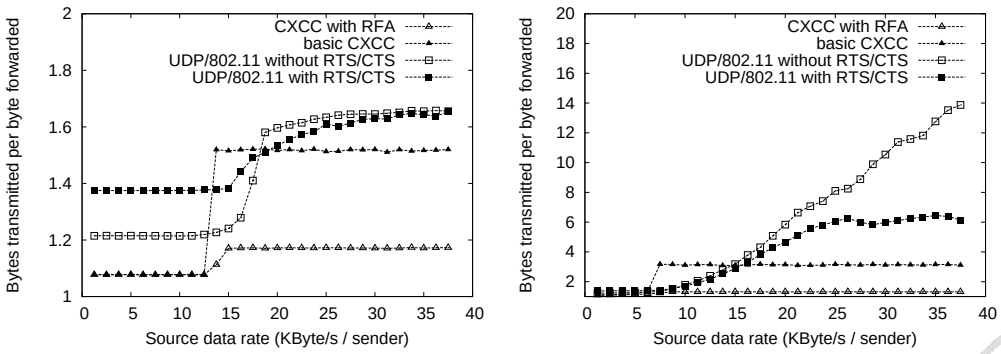
Figure 9. Average packet delay (two-ray ground).

from the same simulations. We show only the results for the chain and the bidirectional chain topologies here, since the cross topology and the bidirectional cross are generally very similar to their respective counterparts, with further amplified differences.

Figure 9 shows the average packet delay for two of the topologies. We define it as the time between the start of the MAC layer transmission of the packet at its source node and the completion of its reception at the destination node. It can be seen that, for UDP over 802.11, the performance deteriorates rapidly with regard to packet delay once the sender's rate exceeds the optimum. The degradation is eminent in all four topologies. Depending on the scenario, the average packet delay is four to six times higher for UDP than it is for CXCC with RFA for higher data rates. For data rates lower than the optimal sending rate, UDP and CXCC with RFA perform comparably well. This again demonstrates the stabilizing properties of CXCC.

In some cases there is a small interval of source rates in which CXCC's packet delay times already grow, while UDP is still able to maintain the very good values of an underutilized network. This is the range in which collisions already occur, but not at a frequency that is high enough to actually prevent packets from being delivered. While UDP over 802.11 is able to immediately retransmit a packet if no link layer acknowledgment is received and thus does not loose much time in such cases, CXCC will wait for at least one packet retransmission delay (set to three packet transmission times) before the transmission is repeated. This is not optimal, but the losses are small in comparison to those that result from a just slightly too high packet injection rate, as can be seen from the UDP results.

A last evaluation of the performance in the deterministic topology simulations deals with the induced overhead of the protocol. The overhead is defined as the number of bytes that have to be transmitted on the wireless medium in order to deliver one byte of payload data. The sum of all bytes transmitted on the MAC layer is divided by the route length to get the average per hop effort. In order to determine



(a) Chain topology.

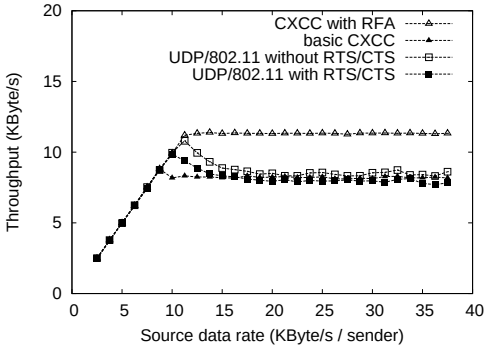
(b) Bidirectional chain topology.

Figure 10. Overhead and energy consumption (two-ray ground).

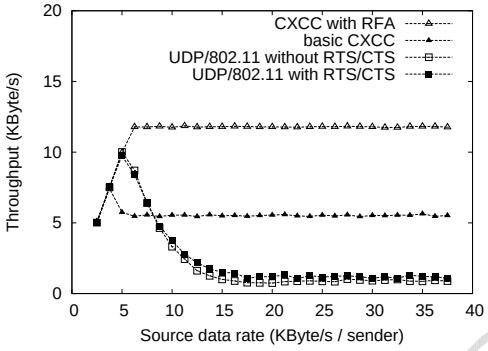
the overhead, the per hop effort was divided by the number of bytes successfully delivered to the final destinations. This measure is closely related to the energy efficiency of the protocol: a higher number of bytes to be transmitted corresponds to a higher power consumption.

Figure 10 shows the results of this evaluation. It can be seen that again CXCC outperforms standard 802.11 packet forwarding with UDP largely at almost all source data rates. The reason for this is that CXCC ensures that only packets may enter the network which will be able to reach their destination. It therefore does not waste resources on packets that are dropped later on. In the unidirectional topology, a higher number of unnecessary payload data retransmissions increases basic CXCC's overhead up to a region similar to UDP/802.11. In the variant with RFA there are no unnecessary retransmissions of complete data packets, but at most small RFA packets. This yields an extremely low overhead in comparison to any of the simulated alternatives, which underlines the appropriateness of our design principle to avoid unnecessary transmissions.

All results presented so far have been obtained using the two-ray ground radio model. In order to assess the influence of the radio propagation model on our results, we have repeated the simulations using log-normal shadowing. The outcome of these simulations further supports our conclusion that the CXCC protocol, in particular CXCC with RFA, performs its task well. The overall results are similar to the ones presented above, but the advantage of CXCC increases. With the shadowing model, the performance of all the considered protocols decreases in comparison to the two-ray ground simulations. This is not surprising, since the simulated wireless medium is generally less reliable with the shadowing model. Due to the random component, packet transmissions may fail even within the potential communication range, and interference over long ranges becomes possible, leading to more collisions. Figures 11, 12, and 13 show the obtained simulation results for the throughput, the average packet delay, and the protocol overhead, respectively, for the unidirectional and bidirectional chain topologies. Again, the results for the

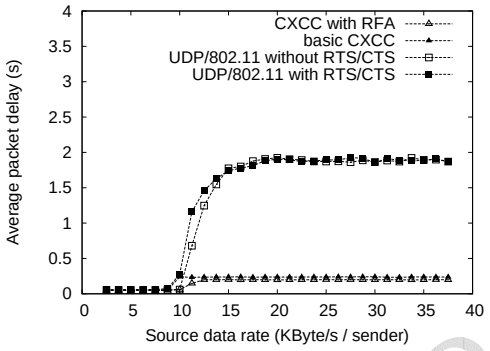


(a) Chain topology.

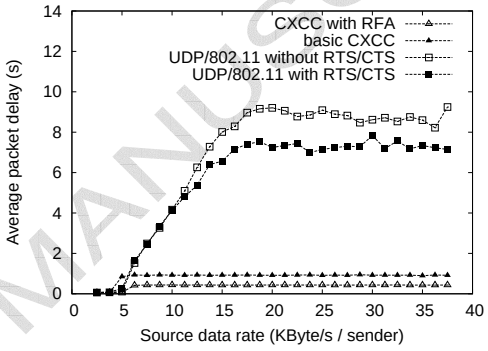


(b) Bidirectional chain topology.

Figure 11. Obtained throughput (shadowing).



(a) Chain topology.



(b) Bidirectional chain topology.

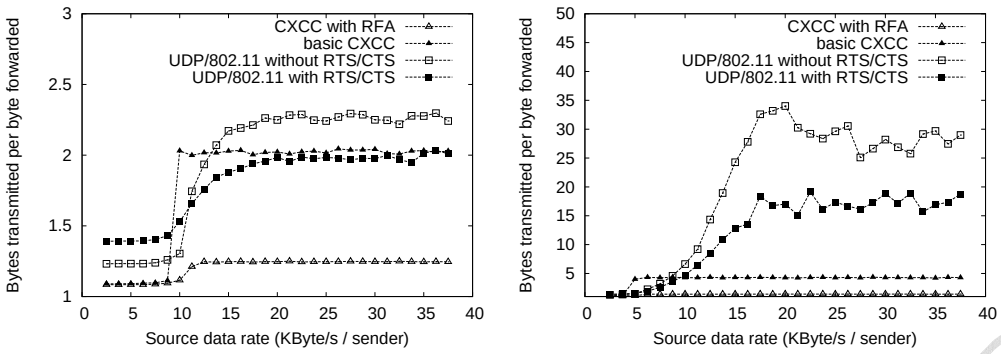
Figure 12. Average packet delay (shadowing).

cross and bidirectional cross are similar, with further pronounced differences.

7.2 Error Resilience in Bidirectional Chain Topologies

Up to now we have seen that CXCC is able to adjust the rate well and to avoid overload problems very efficiently at least in simple topologies. This, however, only shows that congestion control is performed, and that it is effective, but it does not tell us how it compares to other congestion control approaches. From now on, we compare CXCC's congestion control abilities to those of TCP NewReno, ADTCP, and TCP-AP. We have simulated all three TCP variants with and without 802.11's RTS/CTS mechanism enabled.

Since it became clear that CXCC with RFA is far superior to the basic CXCC variant, we concentrate purely on CXCC with RFA from now on. Therefore, if we refer to "CXCC" in the following, we mean CXCC with RFA.



(a) Chain topology. (b) Bidirectional chain topology.

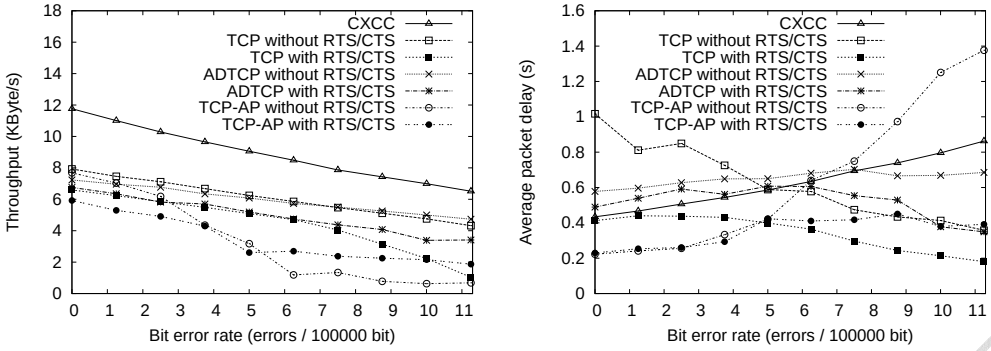
Figure 13. Overhead and energy consumption (shadowing).

For the moment, we stay with the bidirectional 10-hop chain topology that has already been introduced above. We use it to test how well the considered protocols can deal with an unreliable medium. For this purpose, we employ the shadowing propagation model, and add more and more random packet losses, i. e., an increasing bit error rate (BER). We then assess how this influences throughput and latency. Like above, we have also simulated different chain lengths, which again did not qualitatively change the results.

Figure 14 shows the results of the simulations for a wide range of bit error rates on the x-axis. On the right hand side of the chart, the probability that a transmission of a typical data packet with 512 bytes payload plus all headers succeeds is only approximately 60 %. Since only 60 % of the bandwidth is available for successful transmissions, one would expect a relative decrease in throughput by a factor of $3/5$ in the ideal case. It could be further decreased by a higher coordination effort. Similarly, a corresponding increase in packet delay by a factor of $5/3$ plus the coordination effort's impact can be expected. The charts show that this in fact fits quite well for CXCC, where the respective factors are 0.56 for the throughput and 1.95 for the latency.

Some of the TCP variants, however, exhibit a largely different behavior. In particular the packet delay of TCP NewReno actually decreases with increasing BER, instead of growing as expected. Without RTS/CTS it dropped by as much as 60 %. Our explanation for this surprising trait of TCP is that the higher packet loss rate leads to a smaller TCP congestion window size, and this in turn reduces the medium contention so much that the negative effect of the necessary link layer retransmissions is overcompensated.

In summary, the presented results show that CXCC deals well with random packet losses over a wide range of loss rates, and that it performs very well especially in terms of throughput. The packet delay of CXCC is higher than it is for some of the TCP variants. However, this should be seen in relation to the higher throughput of



(a) Throughput. (b) Average packet delay.

Figure 14. Bidirectional chain topology with increasing BER (shadowing).

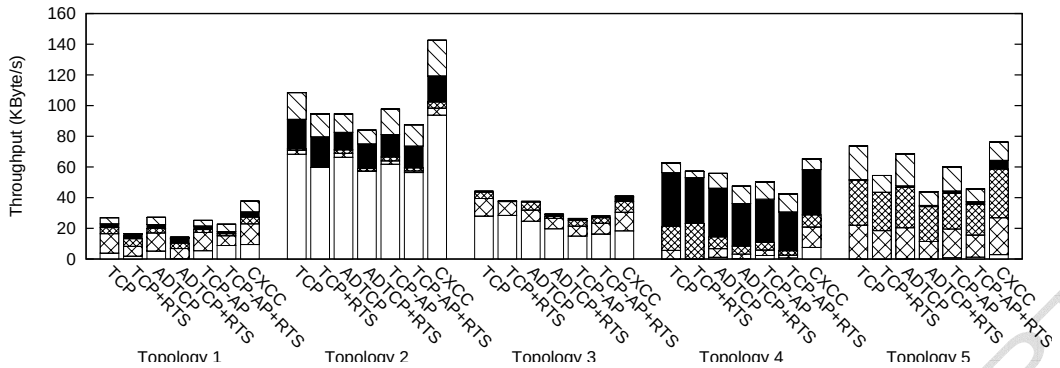
CXCC that corresponds to a much better media utilization.

7.3 Random Topologies with Long Connections

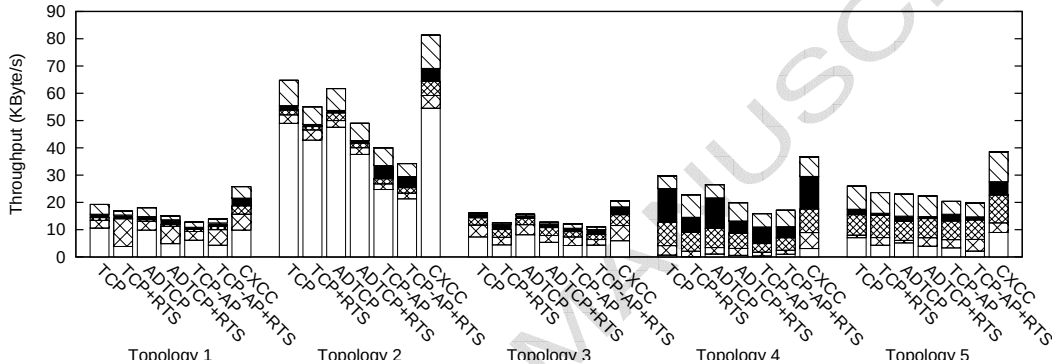
In the next set of simulations, we examine the steady-state throughput of CXCC in comparison to the three TCP variants in more realistic topologies. The simulated networks cover an area of 1500×1500 square meters, where 150 nodes are placed randomly. Five random connections are set up in each scenario, that continuously try to deliver as much data as possible. The same scenarios are simulated with all considered protocols.

The throughput of each connection with each of the protocols is measured after an equilibrium had been reached. Figure 15 shows the results of these measurements for five random network setups for both the two-ray ground and the log-normal shadowing model. In the charts, each segment of a bar stands for the throughput of one stream. Within each topology, an identical fill pattern indicates that the respective segment belongs to the same pair of communicating nodes. The chosen representation thus allows not only a comparison of the total throughput, but also of its distribution to the five streams.

As can be seen from these results, some connections are starved completely or almost completely by TCP NewReno. This problem aggravates with RTS/CTS enabled, and it is less pronounced in the simulations with the shadowing model. Severe fairness problems with TCP have been reported many times in the literature, and have been traced back to medium capture problems. A primary motivation of TCP modifications and alternatives for wireless multihop networks has always been fairness improvement. While ADTCP and in particular TCP-AP are able to improve the fairness, this comes at the cost of throughput. This is interrelated, because TCP often starves in particular those flows that traverse many hops. But a



(a) Two-ray ground.



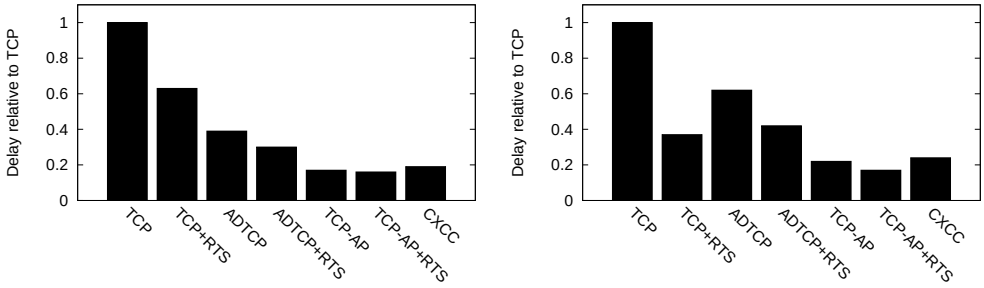
(b) Log-normal shadowing.

Figure 15. Total network goodput in random topologies with five long-lasting streams, broken down into single streams' throughputs.

higher throughput for these long flows comes at a high cost, since more medium capacity is necessary to deliver a packet, compared to a flow with few hops.

CXCC, however, behaves significantly more fair than TCP, without a loss in throughput. Complete flow starvation did never occur in the simulations. A significant throughput surplus over the TCP modifications and, in particular when the log-normal shadowing model is considered, also over TCP NewReno can be seen. Thus, once again we see the general observation confirmed that the more realistic shadowing propagation model amplifies the advantage of CXCC.

A look at the packet latencies exposes further interesting aspects. However, a comparative evaluation for more than single streams is not straightforward: the latencies of connections over different hop counts and in different topologies are not directly comparable. We thus consider relative differences instead of absolute ones. For this evaluation, we take only those connections into account for which each approach managed to deliver at least 50 packets, in order to guarantee a solid statistical basis.



(a) Two-ray ground.

(b) Log-normal shadowing.

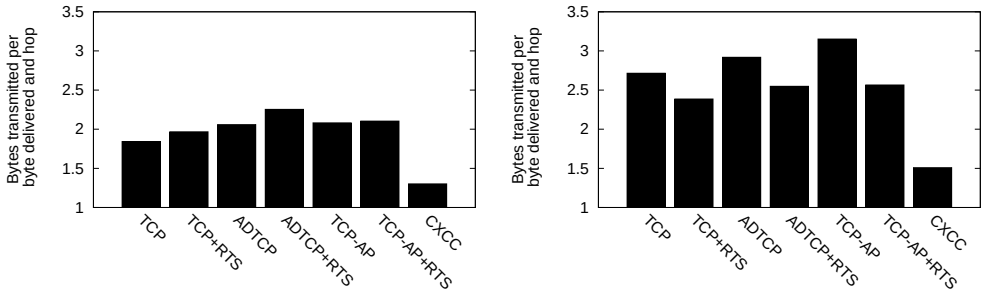
Figure 16. Packet delay in relation to TCP, in random topologies with five long-lasting connections.

This criterion applies to 60 % of the connections in the two-ray ground simulations, and to 92 % for the shadowing model. For each connection, we use the packet delay of TCP NewReno without RTS/CTS as a reference, and calculate the factor by which the latency of the other approaches differs. In Figure 16, the geometric mean [2] of these relative differences for all connections is shown. Obviously, the factor for the reference TCP NewReno without RTS/CTS is one. A factor of 0.2 for CXCC means that the mean packet latency for CXCC was $1/5$ of that of TCP. It is evident that CXCC as well as the TCP modifications achieve a, sometimes very significantly, lowered packet delay. While for the TCP variants this comes, as seen before, at the cost of throughput, CXCC combines both low latency and high throughput. The main reasons for the low delays of CXCC are the—by design—extremely short queues, and the resulting short queuing delays. Interestingly, while RTS/CTS generally seems to have a negative impact on throughput and fairness, it can considerably decrease the latency for TCP. This might stem from the lower media utilization with RTS/CTS, resulting in a lower contention level.

The protocol overhead, just like the packet delay, cannot be directly compared between different connections due to the differing hop counts. As a metric that takes this into account we calculate the number of transmitted bytes on the medium per payload byte and hop, i. e., how many bytes need at an average to be transmitted on the medium in order to bring one byte of payload one hop further. Figure 17 shows the results of this evaluation. The significantly higher overhead of all TCP variants results from a higher number of retransmissions. CXCC's RFA mechanism reduces the number of retransmissions of packets with payload to the absolutely necessary minimum.

The hop-count weighted approach also leads to a different throughput measure: TCP's unfairness mainly comes at the cost of streams with a higher number of

[2] The geometric mean is more suitable to averaging relative ratios than the arithmetic mean.



(a) Two-ray ground.

(b) Log-normal shadowing.

Figure 17. Overhead and energy consumption in random topologies with five long-lasting connections.

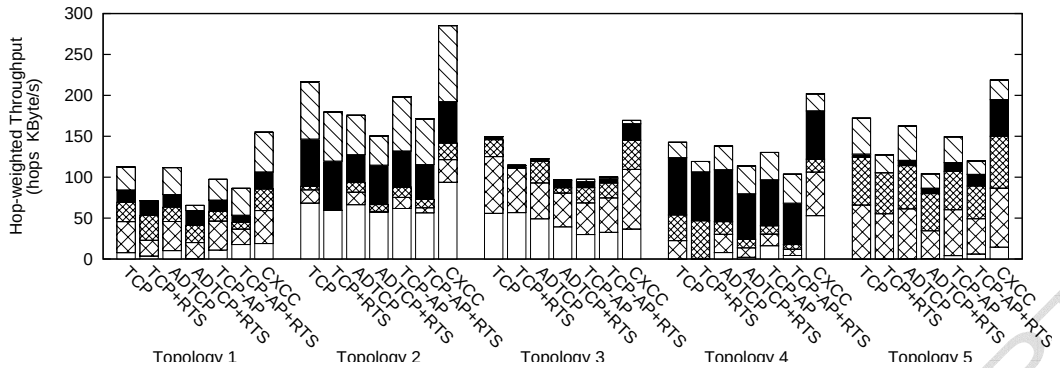
hops. But, as already mentioned, a high throughput for a stream with few hops can be achieved at a lower media utilization. Thus, in Figure 18, we provide the same results as before in Figure 15, but with each connection's throughput weighted by the number of hops along its route. The significant difference shows that CXCC is in fact able to utilize the network best, since long connections gain more throughput and thus the per-hop throughput is higher.

7.4 Random Topologies with Fixed Amounts of Data

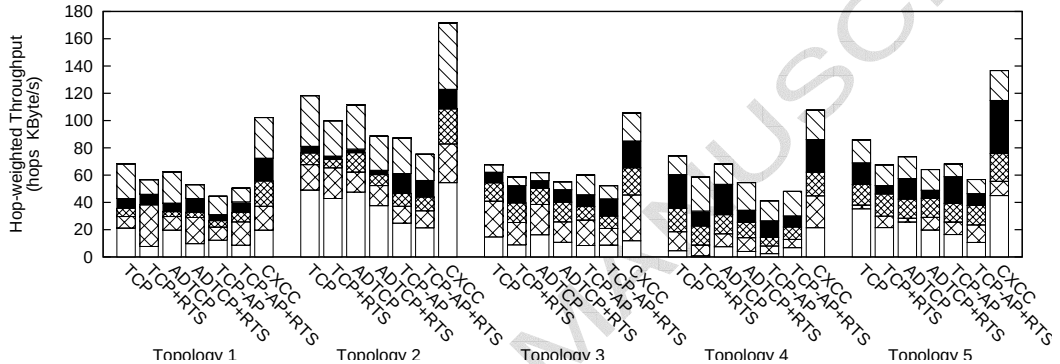
Through the last type of simulation that we have performed we examine how well CXCC behaves in a more realistic and dynamic scenario. We consider random topologies of the same dimensions and node counts as above. Now, many short data transmissions are scheduled between random pairs of nodes, each starting at a random time between 0 and 120 simulation seconds. Each of these transmissions has a random, equidistributed amount of data in the range between 5 and 50 kilobytes to deliver. For the two-ray ground simulations, we use 120 transmissions. For the shadowing propagation model, we account for the, as previously seen, generally lower throughput by reducing this number to 90.

Again, we have performed all simulations of all TCP variants with and without RTS/CTS enabled. Like for most aspects before, the performance with RTS/CTS enabled is generally worse. Therefore, for space and readability reasons, we show only the results without RTS/CTS here.

Figures 19 and 20 depict the packet reception times of some randomly selected streams in one of the simulation runs for the two-ray ground model and for the shadowing model respectively. Each point denotes one packet reception. The y-coordinate of each point denotes the stream that it belongs to, while on the x-axis the packet arrival time is shown. All four sub-figures are based on the same streams



(a) Two-ray ground.

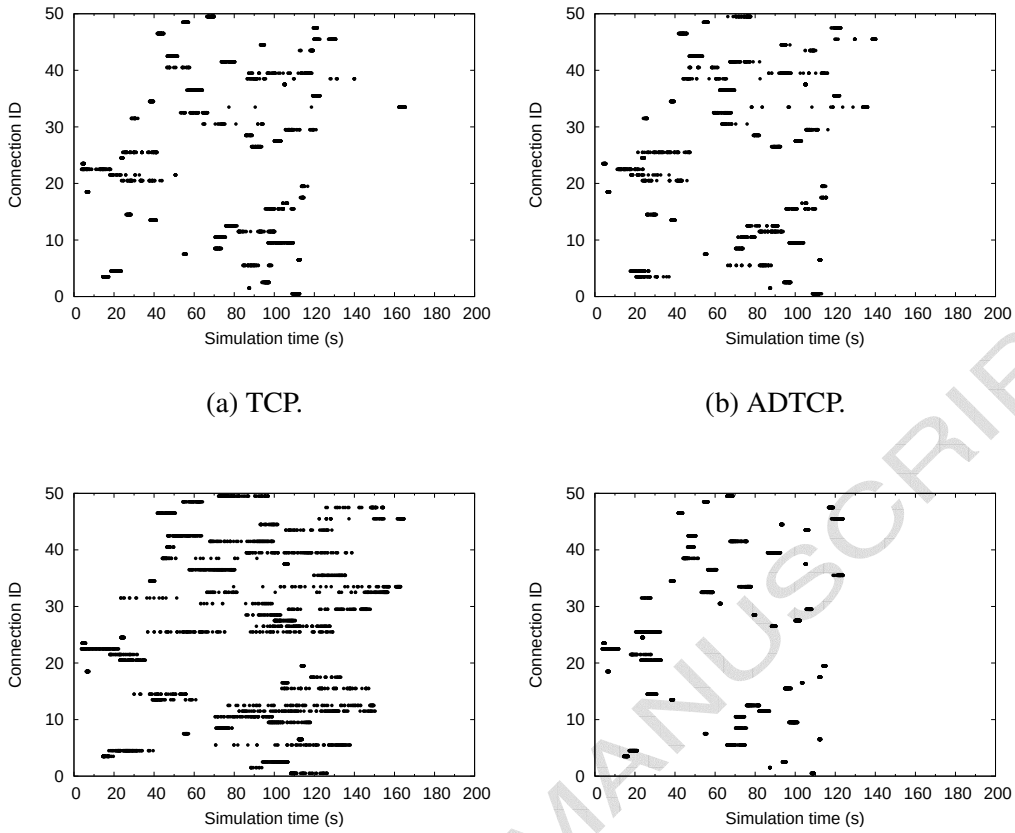


(b) Log-normal shadowing.

Figure 18. Total network goodput in random topologies with five long-lasting streams, weighted by hop count.

from the same scenario, and thus the node positions and communication partners are identical, equal amounts of data are to be transmitted and the transmissions start at the same times. To avoid misinterpretations, only the first successful reception of a segment by the receiver is shown upon duplicate deliveries.

It is visible that, for the vast majority of transmissions, CXCC not only delivers the last segment much earlier than all TCP variants, but it is also able to sustain a smoother rate. We attribute this to a faster, since implicit and thus practically immediate, adjustment to the optimal rate in case of changing network conditions. Our interpretation of the bad results for all TCP versions is that the feedback path is too long to yield accurate information on the network state if the traffic pattern is as dynamic as in these simulations. During some of the transmissions, multiple losses of data segments and acknowledgments and the resulting long retransmission timeouts cause long periods of inactivity. Therefore, a number of transmissions that in fact start quite early are not able to complete for a long time, even when the network is idle. The charts show only the first 200 seconds.



(a) TCP.

(b) ADTCP.

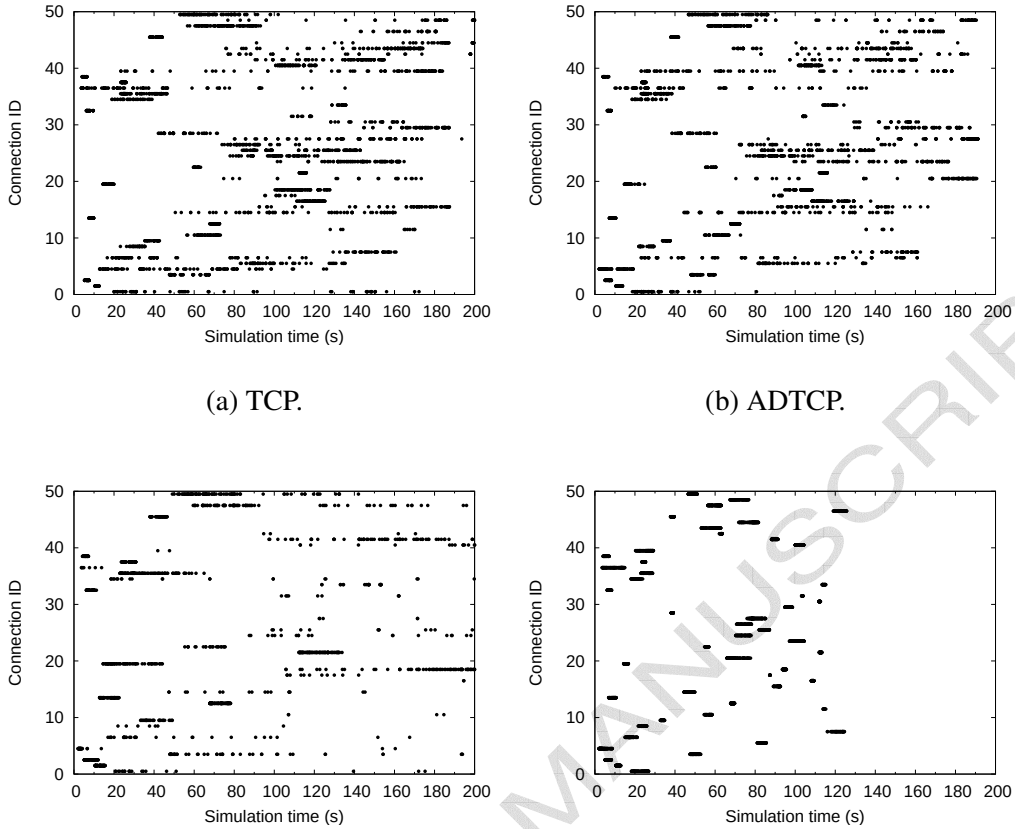
(c) TCP-AP.

(d) CXCC.

Figure 19. Times of packet receptions in random topology (two-ray ground).

Finally, we now consider the per-stream throughput and the fairness of the different protocols under rapidly changing traffic patterns. To examine this aspect, we performed more simulations in different random topologies, all with their key parameters chosen as described above. Figure 21 shows the cumulative distribution functions of the throughputs of all the streams. Note the logarithmic x-axis in the charts. We calculate the throughput by dividing the amount of data by the transmission duration. The transmission duration is defined as the time between the start of the transmission, when the packets are enqueued at the source node, and the point in time when each segment has been received at least once by the destination node.

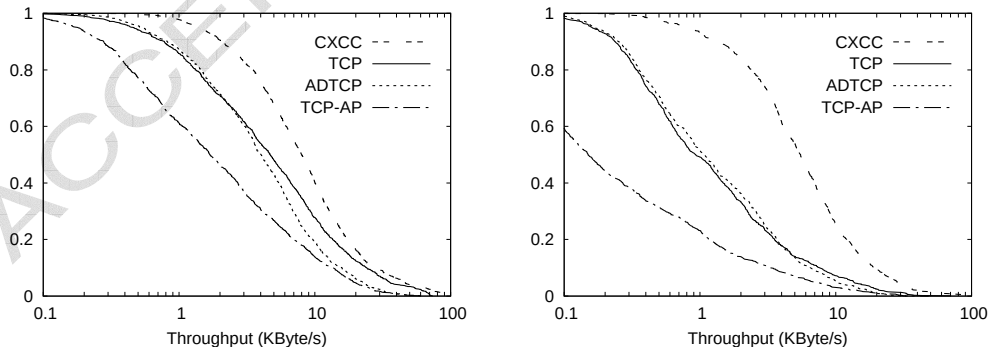
Apart from a general trend to higher throughputs, a significantly better fairness of CXCC is evident: while for the TCP variants many streams have a low or very low throughput, the CXCC connections all obtain at least some minimum share of the bandwidth. This can be seen from CXCC's curve, which starts decreasing comparatively late. The key reason for the better fairness of CXCC is that the nodes refrain from capturing the medium for a long time. After a packet transmission a node is forced to stop sending more data from the same stream. It has to wait for the reception of an implicit or explicit acknowledgment, or until the retransmission delay



(a) TCP. (b) ADTCP.

(c) TCP-AP. (d) CXCC.

Figure 20. Times of packet receptions in random topology (shadowing).



(a) Two-ray ground. (b) Log-normal shadowing.

Figure 21. Cumulative distribution functions of stream throughputs in random topologies with short connections.

elapses. This gives other nodes the opportunity to start or continue transmissions.

8 Real-World Testbed Results

In the previous section we have presented simulation results, indicating that our implicit approach of performing congestion control is in fact able to provide an efficient way of protecting the network from overload. However, we are aware that simulations in general—and particularly for wireless multihop networks—are not able to model all factors that might influence a protocol in the real world, even if they are based on an elaborated propagation model. Therefore, to complement our simulations, we have also implemented CXCC with RFA in a real hardware testbed, and conducted measurements with this implementation.

As stated before, CXCC cannot be implemented on today's commodity 802.11 wireless hardware. There, a large part of the MAC functionality is realized very close to the hardware, in the (proprietary) firmware, which is not accessible for modifications. When looking for a way to overcome these difficulties we came across the ESB sensor nodes. These relatively inexpensive devices were developed at the Freie Universität Berlin as part of the ScatterWeb project [33]. They are intended to serve as a testbed platform for wireless sensor networks. ESB nodes are battery-powered and equipped with a collection of sensors and a wireless interface. For our purposes, however, their main advantage is the open firmware, which allows modifications to every part of the software, down to the manipulation of each single bit transmitted on the wireless medium. Here, they are used as devices in a MANET testbed instead of their original purpose of being used in sensor networks.

Of course the non-802.11 compatible physical layer of the ESB nodes, operating at only 19.2 KBit/s in the 868 MHz band, does not allow for a direct performance comparison to 802.11-based networks. But our main intention is to show that CXCC works in practice and exhibits a behavior similar to that in the simulations.

Since on the ESB nodes there is not as much software "infrastructure" available as it can taken for granted on, e. g., PDAs or a PC, it was not sufficient to implement only the CXCC protocol. We also created the rest of the necessary testbed infrastructure. This comprises, for example, a logging facility that is able to log a large enough number of MAC layer events in the limited storage space available (64 KB in each node), a static routing module, and traffic generators. Additionally, some convenience tools for routing table generation and distribution, for the verification of the topology and for the collection of log data have been created. More details on this experimental framework can be found in [34].

In addition it was necessary to make modifications to the standard ESB firmware's MAC and link layer to resemble 802.11 more closely. For example, the number of

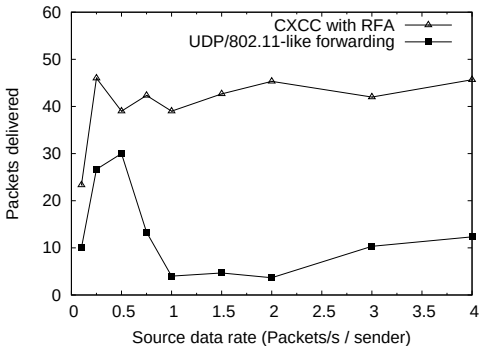


Figure 22. Measured throughput in bidirectional chain topology experiments.

packet retransmission attempts has been set to seven for those experiments where CXCC was not used. Also the ESB nodes are—due to their very limited hardware resources—not able to handle packets of the size that is common for IEEE 802.11. Since there are only 2 KB of RAM available in total, there is not enough space for queues containing long packets. In order to be able to use a reasonable data packet size, we prepended additional 200 bytes to the data packets, as some kind of additional preamble. We used 32 bytes of payload per packet in our experiments, a size that is easily feasible with the ESB nodes. With the prepended 200 bytes, a 32 byte data packet transmission results in the same medium occupancy as a "real" 232 byte packet would. Control packets such as ACKs or RFA packets are transmitted without the artificially increased preamble, so they occupy the medium with their regular size.

In our experiments we used six ESB nodes, set up in a bidirectional chain topology. Each node was placed in a different room. This was sufficient to prevent a reliable direct communication between nodes that are not neighbors. In each of three separate experiment runs we slowly increased the offered load at the nodes for both CXCC with RFA and 802.11-like packet queuing and forwarding. At each examined data rate, traffic was generated for two minutes. Then the successfully delivered packets were counted. Figure 22 shows the outcome of our throughput measurements, averaged over the three experimental runs.

The results show clearly that the real-world behavior of both protocols matches the simulations very closely. Of course the absolute values are very different—but this is hardly surprising given the vastly different radio layers. However, much more importantly, on a qualitative level, the 802.11-like approach's performance drops to a very low level after some optimal input rate is exceeded, while CXCC's throughput remains stable at a comparably high level. We consider this as a confirmation that the throughput-stabilizing properties of CXCC are also present in real networks.

9 Conclusion

In this paper we have proposed a novel way of accomplishing congestion control in wireless multihop networks: implicit hop-by-hop congestion control. It is based on the insight that an input rate exceeding the optimal output rate of a node or network area even on a short-term will be detrimental for the performance of a wireless multihop network. Our mechanism exploits the wireless broadcast medium in order to gain the necessary information for a backpressure mechanism that reliably limits the number of packets to one per flow and hop, and thereby implicitly avoids network congestion. We have presented a protocol, CXCC, that builds upon the idea of implicit hop-by-hop congestion control. An improvement of the CXCC protocol, the Request For ACK (RFA) mechanism, avoids unnecessary data packet retransmissions.

Our simulation results demonstrate that in simple and deterministic scenarios as well as in more realistic ones CXCC is able to effectively adjust the packet sources' rates and to utilize the network capacity well. In comparison to TCP and two other transport protocols for mobile ad-hoc networks, good fairness properties and a very competitive throughput can be observed. This altogether shows that implicit hop-by-hop congestion control as a new congestion control paradigm not just works well, but also exhibits some remarkable advantages over common transport layer end-to-end mechanisms. In particular these are the ability deal with UDP- as well as with TCP-like traffic, very fast reaction times, a low packet delay, very good energy efficiency, and a simple protocol design, which greatly eases the adaption of the protocol to new usage scenarios and environments. Since wireless multihop network applications span a broad range of traffic types, we consider this a very central benefit.

The simulations with both a deterministic and a probabilistic radio propagation model are accompanied by an implementation on real hardware. We have shown that the behavior of CXCC in a real network matches the expectations from the simulations. The chosen hardware platform allowed us to do such an implementation by avoiding the constraints imposed by commodity 802.11 hardware.

Acknowledgments

We wish to thank Markus Koegel, Yves Jerschow, and Alfonso Cervantes for their great work on the CXCC protocol implementations. We also want to express our gratitude to the authors of ADTCP and TCP-AP, for making their ns-2 implementations available. In particular we thank Sherif ElRakabawy from the University of Leipzig, Germany, for helpful discussions on the TCP-AP simulations.

References

[1] M. Gerla, K. Tang, R. Bagrodia, TCP Performance in Wireless Multi-hop Networks, in: WMCSA '99: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Los Alamitos, CA, USA, 1999, p. 41.

[2] S. Xu, T. Saadawi, Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc network, IEEE Communications Magazine 39 (6) (2001) 130–137.

[3] R. de Oliveira, T. Braun, TCP in Wireless Mobile Ad Hoc Networks, Tech. Rep. IAM-02-003, Institute of Computer Science and Applied Mathematics, University of Berne (Jul. 2002).

[4] Z. Fu, X. Meng, S. Lu, How Bad TCP Can Perform In Mobile Ad Hoc Networks, in: ISCC '02: Proceedings of the 7th IEEE International Symposium on Computers and Communication, 2002, pp. 298–303.

[5] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, M. Gerla, The Impact of Multihop Wireless Channel on TCP Throughput and Loss, in: INFOCOM '03: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, 2003, pp. 1744–1753.

[6] T. Plesse, C. Adjih, P. Minet, A. Laouiti, A. Plakoo, M. Badel, P. Mühlethaler, P. Jacquet, J. Lecomte, OLSR performance measurement in a military mobile ad hoc network, Elsevier Ad Hoc Networks 3 (5) (2005) 575–588.

[7] V. Raghunathan, P. R. Kumar, A Counterexample in Congestion Control of Wireless Networks, in: MSWiM '05: Proceedings of the 8th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2005, pp. 290–297.

[8] C. Lochert, B. Scheuermann, M. Mauve, A Survey on Congestion Control for Mobile Ad-Hoc Networks, Wiley Wireless Communications and Mobile Computing, to appear.

[9] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks, in: ICDCS '98: Proceedings of the 18th International Conference on Distributed Computing Systems, IEEE Computer Society, 1998, pp. 472–479.

[10] G. Holland, N. H. Vaidya, Analysis of TCP Performance Over Mobile Ad Hoc Networks, in: MobiCom '99: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999, pp. 219–230.

[11] F. Wang, Y. Zhang, Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response, in: MobiHoc '02: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing, ACM Press, 2002, pp. 217–225.

[12] R. de Oliveira, T. Braun, A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Wireless Networks, in: INFOCOM '05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005, pp. 1863–1874.

[13] Z. Fu, B. Greenstein, X. Meng, S. Lu, Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks, in: ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols, Washington D.C., USA, 2002, pp. 216–225.

[14] S. M. ElRakabawy, A. Klemm, C. Lindemann, TCP with Adaptive Pacing for Multihop Wireless Networks, in: MobiHoc '05: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, ACM Press, New York, NY, USA, 2005, pp. 288–299.

[15] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, R. Sivakumar, ATP: A Reliable Transport Protocol for Ad-hoc Networks, in: MobiHoc '03: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing, ACM Press, 2003, pp. 64–75.

[16] K. Chen, K. Nahrstedt, N. Vaidya, The Utility of Explicit Rate-based Flow Control in Mobile Ad Hoc Networks, in: WCNC '04: Proceedings of the IEEE Wireless Communications and Networking Conference, Vol. 3, 2004, pp. 1921–1926.

[17] Y. Yi, S. Shakkottai, Hop-by-hop Congestion Control over a Wireless Multi-hop Network, in: INFOCOM '04: Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, 2004, pp. 2548–2558.

[18] N. Gower, J. Jubin, Congestion Control Using Pacing in a Packet Radio Network, in: MILCOM '82: Proceedings of the IEEE Military Communications Conference 'Progress in Spread Spectrum Communications', 1982, pp. 23.1.1–23.1.6.

[19] D. B. Johnson, D. A. Maltz, Dynamic Source Routing in Ad Hoc Wireless Networks, in: T. Imielinski, H. Korth (Eds.), Mobile Computing, Vol. 353, Kluwer Academic Publishers, 1996, Ch. 5, pp. 153–181.

[20] H. Zhai, J. Wang, Y. Fang, Distributed Packet Scheduling for Multihop Flows in Ad Hoc Networks, in: WCNC '04: Proceedings of the IEEE Wireless Communications and Networking Conference, Vol. 2, 2004, pp. 1081–1086.

[21] H. Zhai, X. Chen, Y. Fang, Alleviating Intra-Flow and Inter-Flow Contentions for Reliable Service in Mobile Ad Hoc Networks, in: MILCOM '04: Proceedings of the IEEE Military Communications Conference, Vol. 3, 2004, pp. 1640–1646.

[22] C. Wang, K. Sohraby, B. Li, M. Daneshmand, Y. Hu, A Survey of Transport Protocols for Wireless Sensor Networks, Network, IEEE 20 (3) (2006) 34–40.

[23] A. Woo, D. E. Culler, A Transmission Control Scheme for Media Access in Sensor Networks, in: MobiCom '01: Proceedings of the 7th Annual ACM International Conference on Mobile Computing and Networking, ACM Press, New York, NY, USA, 2001, pp. 221–235.

[24] C.-Y. Wan, S. B. Eisenman, A. T. Campbell, CODA: Congestion Detection and Avoidance in Sensor Networks, in: SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems, 2003, pp. 266–279.

[25] B. Hull, K. Jamieson, H. Balakrishnan, Mitigating Congestion in Wireless Sensor Networks, in: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM Press, New York, NY, USA, 2004, pp. 134–147.

[26] C. Wang, K. Sohraby, B. Li, SenTCP: A Hop-by-Hop Congestion Control Protocol for Wireless Sensor Networks, IEEE INFOCOM 2005 (Poster Paper) (Mar. 2005).

[27] H. Zhang, A. Arora, Y. Choi, M. G. Gouda, Reliably Bursty Convergecast in Wireless Sensor Networks, in: MobiHoc '05: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2005, pp. 266–276.

[28] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control for Unicast Applications, ACM SIGCOMM Computer Communication Review 30 (4) (2000) 43–56.

[29] K. Xu, M. Gerla, S. Bae, How Effective is the IEEE 802.11 RTS/CTS Handshake in Ad Hoc Networks?, in: GLOBECOM '02: Proceedings of the IEEE Global Telecommunications Conference, 2002, pp. 72–76.

[30] S. Ray, J. B. Carruthers, D. Starobinski, RTS/CTS-Induced Congestion in Ad Hoc Wireless LANs, in: WCNC '03: Proceedings of the IEEE Wireless Communications and Networking Conference, 2003, pp. 1516–1521.

[31] The ns-2 network simulator, http://www.isi.edu/nsnam/ns/.

[32] S. Floyd, T. Henderson, A. Gurtov, The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 3782 (Proposed Standard) (Apr. 2004).

[33] Freie Universität Berlin, Computer Systems Telematics, ScatterWeb Project, http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net.

[34] Y. I. Jerschow, B. Scheuermann, C. Lochert, M. Mauve, A Real-World Framework to Evaluate Cross-Layer Protocols for Wireless Multihop Networks, in: REALMAN '06: Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality, 2006, pp. 1–6.