Near-Optimal Co-ordinated Coding in Wireless Multihop Networks

Björn Scheuermann*
Heinrich Heine University
Düsseldorf, Germany
scheuermann@cs.uni-duesseldorf.de

Wenjun Hu
University of Cambridge
Cambridge, UK
wenjun.hu@cl.cam.ac.uk

Jon Crowcroft
University of Cambridge
Cambridge, UK
jon.crowcroft@cl.cam.ac.uk

ABSTRACT

The recent work on COPE by Katti *et al.* demonstrates a practical application of network coding to wireless multihop networks. We note, however, that the opportunistic nature of COPE leaves it at the mercy of higher and lower layer protocols to create coding opportunities spontaneously. In this paper, we go one step beyond COPE's opportunism and study how to create coding opportunities in a more deterministic, yet still practical way. We start from the insight that in two-way traffic the existence of coding opportunities can be guaranteed through carefully co-ordinated packet scheduling, and establish general properties of protocols that are able to achieve this. We then propose Near-Optimal Co-ordinated Coding (noCoCo), a cross-layer scheme that integrates per-hop packet scheduling, network coding, and congestion control in a novel way. Extensive simulations show that noCoCo significantly outperforms standard non-coding approaches as well as COPE in terms of network throughput, delay and transmission overhead.

1. INTRODUCTION

With network coding, a router transmits multiple packets within a single "coded" packet, and can thereby make more efficient use of the available bandwidth. The recent work COPE [4] applies this technique to wireless multihop networks. Considering a simple three-node, two-hop setting outlined in [11], the basic principle of network coding is easy to understand: if node A sends a packet p_A to C via B, and C sends a packet p_C to A, also via B, then B may send the XOR of p_A and p_C, $p_A \oplus p_C$, instead of transmitting them separately. Since A and C know p_A and p_C respectively, they can extract the data intended for them with another XOR operation: $p_A \oplus (p_A \oplus p_C)$ to retrieve p_C, and similarly $p_C \oplus (p_A \oplus p_C)$ to retrieve p_A. The concept can be transferred to more complex scenarios; COPE generalises it to using overheard packets to decode.

In COPE, coding is performed when opportunities arise spontaneously. Such an opportunistic approach may work very well if, in the above example, indeed both A and C make their transmissions before B accesses the medium. If, however, A transmits first and B forwards the packet further on immediately, no coding is possible. Consequently, the order of transmissions can significantly impact the availability of coding opportunities, and hence the coding benefits.

This prompts us to investigate more deterministic alternatives to opportunistic coding. In this work we start by analysing packet flows in two-way data traffic in detail, and conclude that it is possible to guarantee coding opportunities through carefully co-ordinating packet transmissions. After studying the theoretical limits, we translate the understanding into a practical protocol proposal, Near-Optimal Co-ordinated Coding (noCoCo). noCoCo guarantees achieving the maximum possible coding gain for each single bidirectional connection in isolation. It can be further combined with opportunistic schemes to identify and exploit additional coding opportunities with, e. g., cross-traffic or unidirectional traffic components.

noCoCo is built around a few simple rules for packet forwarding. These rules, though designed to ensure the existence of coding opportunities, also perform very effective congestion control. The forwarding rules can be seen as an extension of the approach in CXCC [9], a recent cross-layer proposal that performs congestion control implicitly hop by hop. Noting these similarities, we also adopt mechanisms for signalling and single-hop reliability from CXCC, and obtain a coding-aware congestion control scheme. We are thus not primarily interested in the information theoretic limits of network coding, but rather in how to schedule packet trans-

*Part of the work on this paper was done while Björn Scheuermann was visiting the Computer Laboratory at the University of Cambridge, UK.

*CoNEXT'*07, December 10-13, 2007, New York, NY, U.S.A.

missions in order to achieve high coding gains on a COPE-like basis.

In the following sections, we first review related work. Introducing a centrally scheduled coding scheme in Section 3, we derive general properties of schedules that can maximise the coding gain in two-way traffic flows. With this understanding, we then describe and discuss noCoCo in Section 4. We analyse its performance in a comparative simulation study in Section 5, and finally summarise the paper in Section 6.

2. RELATED WORK

Network coding permits a router to transmit packets derived from mixing information from different packets. This technique was initially proposed in a seminal study of multicast communications in wireline networks [1], and shown to substantially increase the network capacity. Subsequent theoretical analysis has studied network coding for unicast flows, with an emphasis on modelling and theoretical bounds, although few in a wireless context [3, 6, 11]. In contrast, we are concerned with implementing network coding as a distributed protocol for unicast in wireless multihop networks.

The work most relevant to ours is COPE [4], where the authors proposed an opportunistic coding protocol and implemented it as a coding shim between the MAC and routing/IP layers. In addition to combining packets that traverse the same relay in opposite directions, COPE uses overheard packets for additional coding opportunities. For a coded transmission to be successful, all intended nexthop receivers must be able to decode it. This requires a receiver to have all other component packets except the one directed to itself. Consequently, the task of identifying coding opportunities boils down to obtaining information about which packets are known by which neighbouring node. COPE employs three mechanisms. First, a node knows all packets it has sent out. Therefore the node from which a packet has been received will know that packet—this is the case discussed previously. Secondly, COPE nodes piggyback "reception reports" onto their transmissions, to explicitly notify neighbours of the packets received/overheard. Finally, COPE also "guesses" coding opportunities: based on the link quality information from the routing protocol, a node estimates the overhearing probabilities of coding candidates at specific neighbours. This yields the probabilities of successful decoding on all addressees for a particular combination of packets. If this success probability is above a threshold, the packets are combined into one packet—at the risk of occasional failed decoding on some neighbour. In this work, we mostly study coding in two-way traffic as proposed in [11], but follow COPE's general protocol architecture.

Some recent efforts considered cross-layer approaches in the context of coding-aware routing [10]. Chaporkar and Proutiere also studied the issue of joint scheduling and COPE-like coding, focusing on characterising the capacity region of a simplified version of COPE combined with scheduling according to backpressure [2]. In contrast, we explore the joint design of MAC scheduling, coding, and congestion control protocols, and propose a practical protocol, which also uses backpressure.

The single-hop reliability features of noCoCo are closely based on CXCC [9], which employs a very similar concept for congestion control, but without using network coding. We explain CXCC in detail in Section 4.3.

3. MAXIMISING THE CODING GAIN

3.1 A centralised scheduler

We concentrate on the non-opportunistic coding of packets belonging to the same bidirectional connection, consisting of the two flows from endpoint A to B, and from endpoint B to A. We consider a single bidirectional connection in isolation. In practice, and in our noCoCo implementation, opportunistic coding is used to exploit additional coding opportunities between different connections. For the discussion below, we term the two endpoints A and B the "left hand side node" and the "right hand side node".

Note that two-way traffic is the typical situation for possible continuous coding gain. Such traffic exists in many applications, including any form of real-time communications. Generally, almost all protocols generate at least acknowledgement traffic in the opposite direction to original data traffic. It has even been argued that symmetry of outgoing and incoming packets counts should be a design criterion for good protocols [5].

While we concentrate on traffic that is bidirectional and symmetric between a pair of end nodes, it is also conceivable to apply the presented ideas for traffic belonging to different end-to-end connections, but sharing parts of the route in opposite directions. In fact, the question how such an "aggregation" of oppositely directed traffic on partial routes can be accomplished might constitute an interesting future research question.

To obtain a maximum number of coding opportunities, we look at the scheduling of transmissions: in which order should which nodes transmit which data packets in which coded combinations? The first question that arises in this context is what the maximum coding gain is, after all. The follow-up question whether it can be achieved is intimately related: is there a schedule that can always guarantee maximum coding gain, over an arbitrarily long timespan? In the following, we will first show that optimal scheduling of two-way traffic is indeed possible, by explicitly giving a centrally scheduled solution. We then point out some interesting properties that *any* schedule with maximum coding gain will necessarily possess.

We first observe that the initial transmission of a packet, when it leaves the source node, can not be coded. This is because no other node in the network knows this packet to be able to undo the coding. Within a bidirectional connection,

in the ideal case, all other transmissions combine two oppositely directed packets. One such transmission will then yield two single-hop packet deliveries. At most two (oppositely directed) packets from the same connection can be combined. Therefore, if we manage to combine two packets in each transmission of an intermediate node, we obtain the maximum coding gain.

The theoretical gain is also easy to quantify: for a connection over h hops, $2(h-1)+2$ packets will have been forwarded over one hop each after all $h+1$ nodes have made one transmission each ($h-1$ coded transmissions by the intermediate nodes, and two uncoded transmissions by the end nodes). The theoretical coding gain within the bidirectional connection is therefore

$$\frac{2(h-1)+2}{h+1} = \frac{2h}{h+1}. \quad (1)$$

A close look soon reveals that for a connection over more than two hops, a coding partner for each transmission cannot possibly be available from the beginning. The intermediate nodes first need to have packets in both directions available. Let us thus consider the situation after some initialisation has taken place. Enumerate all intermediate nodes along the route, starting by one. Assume that the initialisation manages to place exactly one packet for either direction in each intermediate node with an odd index. If the total number of hops is odd, thus giving an even number of intermediate nodes, it places an additional left-directed packet in the rightmost node. Any other nodes with even indices hold no packets. Now further assume that we have a global, centralised scheduler available. We may thus arbitrarily decide on the occurring transmissions and their order. Let the scheduler work as follows:

1. First, all intermediate nodes with odd indices makes one coded transmission each, thereby forwarding one packet in each direction. These transmissions may happen in an arbitrary order.

2. If one or both of the end nodes have received a packet during step 1, they "answer" by injecting a new packet into the network.

3. Then, all intermediate nodes with even indices will have a packet in each direction available, and may now make coded transmissions.

4. Again, the end nodes reply to received packets by injecting a new packet.

5. Repeat this sequence of transmissions from step 1.

For the case of a four-node, three-hop scenario, this is schematically visualised in Figure 1.

This scheme can be run indefinitely. All transmissions by intermediate nodes will always forward one packet in either direction, thus realising optimal coding gain. This demonstrates that optimal coding gain is possible, though for now we do not know whether and how it can be achieved in a distributed way.

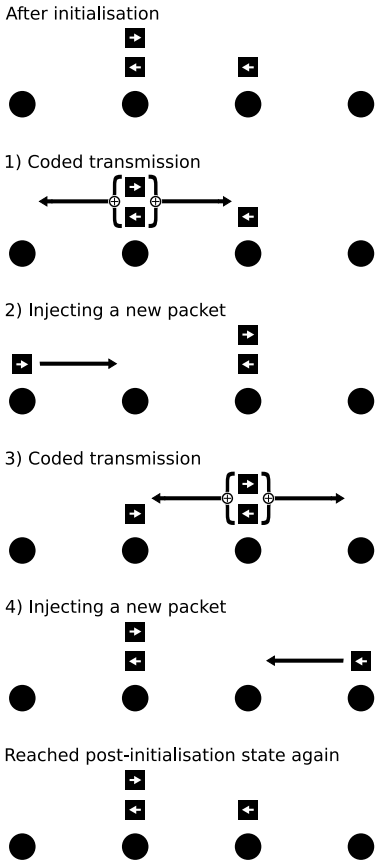

Figure 1: Operation of the centralised scheduler in a three-hop environment.

Scheduled coding within a bidirectional flow can be combined with any arbitrary opportunistic scheme, thus opening up opportunities for even higher coding gain. Further packets can of course also be opportunistically combined with pairs of coded packets from bidirectional connections.

3.2 Notation

Before further examining the general properties of scheduling schemes that can guarantee successful network coding, we introduce a notation for the state of the network. The state of the queue is denoted in a node as x/y, where x is the number of queued packets directed to the node's left neighbour, and y the number of packets directed to the right neighbour. x, y are non-negative integers. We neglect all packets that do not belong to the connection under consideration.

To denote sets of possible states, we will use three placeholders. "$*$" means that at the respective position there may be any arbitrary non-negative integer, i. e., $*/y = \{x/y \mid x \in \mathbb{N}\}$. "$+$" stands for a positive integer, so $+/y$ denotes the set of states $\{x/y \mid x \in \mathbb{N}^+\}$. Finally, "?" means either 0 or 1, so $?/y = \{x/y \mid x \in \{0,1\}\}$.

If we look at a set of consecutive nodes along the route, we may write their joint queue state as $x_1/y_1,\ x_2/y_2,\ x_3/y_3, \ldots$. Transmissions can then be expressed as transformations of the joint queue state. For example, a right-directed transmission of a single packet is

$$x_k/y_k,\ x_{k+1}/y_{k+1} \quad \rightarrow \quad x_k/y_k - 1,\ x_{k+1}/y_{k+1} + 1. \quad (2)$$

Such a transmission can take place if the transmitting node's queue state is in $*/+$.

Likewise, a coded transmission of two packets by node k would be

$$\begin{aligned} &x_{k-1}/y_{k-1},\ x_k/y_k,\ x_{k+1}/y_{k+1} \\ &\quad \rightarrow \quad x_{k-1}+1/y_{k-1},\ x_k - 1/y_k - 1,\ x_{k+1}/y_{k+1} + 1. \end{aligned} \quad (3)$$

The necessary precondition here is that the sending node's state x_k/y_k is in $+/+$.

3.3 Properties of high coding gain schedules

Ideally, we want all transmissions of intermediate nodes to be coded, so we allow only transmissions as in (3) in the intermediate nodes. As mentioned earlier, no coded transmission can take place at the end nodes. Additional packets are thus inserted via uncoded transmissions. This yields state transitions of the following form at the leftmost intermediate node of the route:

$$x/y \quad \rightarrow \quad x/y+1, \quad (4)$$

and equivalent ones at the rightmost intermediate node.

Packets leave the network by coded transmissions. For the leftmost pair of intermediate nodes, the corresponding transition is

$$x_1/y_1,\ x_2/y_2 \quad \rightarrow \quad x_1 - 1/y_1 - 1,\ x_2/y_2 + 1, \quad (5)$$

with precondition $x_1/y_1 \in +/+$.

The availability of coding partners is likely if there are many packets in both directions available in the intermediate nodes. Conversely, coding is seldom possible if "too few" packets are on their way. Note, however, that it is desirable to keep the queues short, and thus the number of packets low, to bound delays in the network. Therefore, a suitable scheme would need to strike a balance between coding opportunities, throughput, and packet delay. We thus now determine how many packets are necessary in order to obtain high coding gains.

As a step in this direction, consider the allowable transitions (3), (4), and (5) and how they can be applied to "navigate" the global joint queue state space. We assume the use of some arbitrary scheduling scheme that maximises the number of coding opportunities, and derive some properties that such a scheme must exhibit. It turns out that there is a subset of states which can never be left, should they ever be entered. The following lemma points out this "state trap".

LEMMA 1. *The subset of global joint queue states where the joint queue state of two consecutive nodes is in* $0/*,\ */0$ *can never be left.*

PROOF. Consider the case where the joint queue state of a pair A, B of consecutive intermediate nodes is in $0/*,\ */0$. Then, neither A nor B can make a coded transmission. A, however, can increase its number of left-directed packets only if a packet is received from B, and vice versa. Hence, this set of states can never be left by coded transmissions only. □

Once such a state has been reached, it is impossible to maintain network coding with maximum gain – the affected nodes cannot continue to forward data with purely coded transmissions. This permits the reversal conclusion that a state as in Lemma 1 will never be reached by *any* scheme that maintains optimal coding gain.

If the joint queue state of each node pair will never be in $0/*,\ */0$, there always has to be at least one packet in one of the queues of each node pair. The following theorem uses this to establish a lower bound on the number of packets in transit.

THEOREM 2. *In any n consecutive intermediate nodes there are at least $n-1$ queued packets at any point in time.*

PROOF. Let the joint queue state of the n nodes be $x_1/y_1, \ldots, x_n/y_n$. For each pair of consecutive nodes with states $x_i/y_i,\ x_{i+1}/y_{i+1},\ 1 \leq i < n$, there is, as a consequence of Lemma 1, either $x_i > 0$ or $y_{i+1} > 0$. Therefore $\forall i, 1 \leq i < n : x_i + y_{i+1} > 0$. For the total number of queued packets we obtain

$$\begin{aligned} \sum_{i=1}^{n} (x_i + y_i) &= y_1 + \sum_{i=1}^{n-1} (x_i + y_{i+1}) + x_n \\ &\geq \sum_{i=1}^{n-1} (x_i + y_{i+1}) \geq n - 1. \end{aligned} \quad (6)$$

□

This result can now immediately be used to obtain a lower limit on the total number of packets on the route, as follows.

COROLLARY 3. *For a bidirectional connection over h hops, there are at least $h-2$ packets in the network at any point in time. This number must be exceeded temporarily.*

PROOF. The first assertion follows immediately from the previous theorem, since there are $h-1$ intermediate nodes.

For the second part, consider (6) in the proof above in a situation just before a packet leaves the network. For this to happen, one of the outmost intermediate nodes must be in a state in $+/+$. Therefore, $y_1 > 0$ or $x_n > 0$. Then, the first inequality in (6) becomes strict, and thus the total number of packets is at least $h-1$. □

Let us see how close the centralised scheduler approaches this lower bound. For a route over h hops, it will, after the

initialisation, start with h packets in the network. Since a new packet is only injected after another one has left, this number is never exceeded. Therefore, the number of packets will always stay within the range $[h-2,h]$, and is thus nearly optimal.

It is worth mentioning that the bound in Corollary 3 is rather optimistic, and sometimes h packets are needed. Consider a two-hop connection over three nodes, i. e., the case $h = 2$. In order to make coded transmissions, the middle node must have two packets available. Therefore, the number of packets in the network must be up to h in this situation.

In summary, it is indeed possible to schedule the transmissions in a bidirectional connection in a way that allows for the highest possible coding gain. There are global joint queue states which do not allow the intermediate nodes to proceed with only coded transmissions. Thus, an optimal schedule will have to avoid these states. This suggests that, when the number of packets being queued along the route is below a certain threshold, the coding cannot be optimal. Since a large number of queued packets in the network will increase packet delivery latency, there is a potential tradeoff between achieving the maximum possible coding gain and keeping a low number of packets in the network.

4. A PRACTICAL PROTOCOL

4.1 Basic protocol rules and mechanisms

We will now refine the centralised approach from the previous section to obtain a practical and distributed scheduling scheme for network coding with success guarantees, Near-Optimal Co-ordinated Coding (noCoCo). The key idea is to approximate the centrally enforced ordering of the transmissions in a decentralised way. We generate a similar pattern of alternating transmissions of nodes with odd and even positions, while keeping the number of packets in the network as low as possible. We then show how the resulting mechanism can be adapted to deal with the adversitites of real wireless media.

Doing so has an interesting implication. The centralised scheduler does not allow additional packets to enter the network before the previously sent ones have left. It therefore limits the number of packets in transit. A similar property can be carried over to achieve a congestion control effect: packets will not enter the network at a higher rate than they are able to traverse it. This is indeed the case in our proposed scheme.

First, however, we need to initialise the network to a valid starting state. Obviously, we need to begin with transmitting single packets as in (2). In our scheme, a node is allowed to forward single packets until it has "seen" packets going in both directions. Thereafter, only transmissions as in (3) and, at the outmost intermediate nodes, (4) and (5) are permitted.

The number of packets forwarded without coding should clearly be as low as possible. This motivates a *backpressure rule*: a packet may only be transmitted to a node which has currently no packet for the same direction in its queue. This prevents an excessive number of packets from entering the network.

With the addition of the backpressure rule, the allowable state transitions can be refined in the following way. For uncoded transmissions by nodes that have not yet encountered packets in both directions, replacing (2), we get

$$x_k/1,\, x_{k+1}/0 \quad \rightarrow \quad x_k/0,\, x_{k+1}/1 \tag{7}$$

for a right-directed transmission by such a node. A coded transmission may only take place if the target queues in the neighbouring nodes are free. Thus (3) becomes

$$0/y_{k-1},\, 1/1,\, x_{k+1}/0 \rightarrow \quad 1/y_{k-1},\, 0/0,\, x_{k+1}/1. \tag{8}$$

For packets entering and leaving the network, e. g., at the left end of the route, we get

$$x/0 \quad \rightarrow \quad x/1 \tag{9}$$

and

$$1/1,\, x_2/0 \quad \rightarrow \quad 0/0,\, x_2/1, \tag{10}$$

as refinements of (4) and (5), respectively.

It is easy to verify that the packet has left the queue of the successor: only coded packet transmissions are allowed. Thus, the previously transmitted packet has left the neighbouring node if and only if the next packet from this node has been received. During initialisation, information on the forwarding of one packet by the downstream node can be obtained similarly: the forwarding of the next hop node can be overheard, which allows to infer its current queue state.

At the beginning, packets from both sources will travel into the network. At most one packet from each source may be queued in each intermediate node. Eventually, there will be one node which is the first to hold packets in both directions. Let the position of this node be denoted by k. When such a node has emerged, the joint queue state of all intermediate nodes is in

$$0/?,\ldots,\, 0/?,\, \underbrace{1/1}_{\text{node } k},\, ?/0,\ldots,\, ?/0. \tag{11}$$

The uncoded transmissions according to (7) allow each position with the placeholder ? in (11) to hold a packet. Taking this into account, as well as the fact that node k is allowed to perform a coded transmission, the state will eventually reach

$$0/?,\ldots,\, 0/?,\, 1/1,\, \underbrace{0/0}_{\text{node } k},\, 1/1,\, ?/0,\ldots,\, ?/0. \tag{12}$$

Iterating this, it is easy to see that a situation will emerge, in which each node is allowed to transmit exactly once whenever both its neighbours have performed a transmission. This also holds for the end nodes. Hence, the emerging scheme is in fact very similar to the centrally scheduled scheme discussed before.

4.2 An upper bound on the number of packets

We will now look at the number of packets in transit for the proposed protocol. Note that this analysis shows a very interesting symmetry to the derivation of the respective general properties in Section 3.3.

LEMMA 4. *With the proposed protocol, a state where the joint queue state of two consecutive nodes is in $+/*$, $*/+$ will never be reached.*

PROOF. For some node A with queue state x_A/y_A, due to the backpressure rule, it always holds that $x_A, y_A \leq 1$. Consequently, after A has performed a transmission, $x_A = y_A = 0$. The latter is true both during the initialisation phase and during normal operation.

Assume the joint queue state x_A/y_A, x_B/y_B of a pair A,B of consecutive intermediate nodes is in $+/*$, $*/+$. Then, $x_A > 0$ and $y_B > 0$. $x_A > 0$ means that B's last transmission must have been more recent than A's last transmission. $y_B > 0$, however, requires the opposite, that A has transmitted more recently than B. This is a contradiction, hence the assertion holds. □

This lemma can, very similar to what we did before in Section 3.3, be used to derive a limit for the number of packets in a connected subset of the intermediate nodes, this time an *upper* bound.

THEOREM 5. *In any n consecutive intermediate nodes using our protocol, there are never more then $n+1$ queued packets.*

PROOF. Let the joint queue state of the n nodes be denoted by $x_1/y_1, \ldots, x_n/y_n$. For each pair of consecutive nodes with states x_i/y_i, x_{i+1}/y_{i+1}, $1 \leq i < n$, there is, by the previous lemma, either $x_i = 0$ or $y_{i+1} = 0$. Furthermore, from the backpressure rule, we know that for all i with $1 \leq i \leq n$, both $x_i \leq 1$ and $y_i \leq 1$ hold. Therefore, $\forall i, 1 \leq i < n : x_i + y_{i+1} \leq 1$. For the total number of packets it thus holds that

$$\begin{aligned} \sum_{i=1}^{n}(x_i + y_i) &= y_1 + \sum_{i=1}^{n-1}(x_i + y_{i+1}) + x_n \\ &\leq 2 + \sum_{i=1}^{n-1}(x_i + y_{i+1}) \\ &\leq 2 + n - 1 = n + 1. \end{aligned} \tag{13}$$

□

Again, this can be used to obtain a bound on the total number of packets along the route.

COROLLARY 6. *For a bidirectional connection over h hops using out protocol, there are at most h packets in the network at any point in time. The number is temporarily undercut.*

PROOF. The first assertion follows directly from the previous theorem. The second part is rather obvious: when a packet leaves the network, the number of packets decreases, until a new packet is inserted. □

This demonstrates that the proposed distributed algorithm stays within the same bounds on the number of packets in the network as the centralised scheduler.

4.3 Dealing with real wireless media

The astute reader will surely have noticed that, so far, we have assumed that all transmissions are always successful. In a real wireless network this is for sure not the case. Wireless interference can make practically every transmission fail. For a coded transmission, this may mean that one or both of the intended receivers cannot decode the message. Thus, we now look at how it is possible to recover from such a situation without sacrificing the desirable properties we have pointed out so far.

Let us first take a detour and examine a related protocol, CXCC [9], a congestion control protocol for wireless multihop networks which is founded on implicit feedback and control. The key idea is hop-by-hop backpressure, enforced by a rule very similar to the backpressure rule used in noCoCo: in CXCC, congestion control is performed by only allowing the transmission of a follow-up packet after the forwarding of the previously sent one has been overheard. This has been shown to yield a fast-reacting and efficient mechanism, which does not only achieve very high throughput, but also a low protocol overhead and, due to the very short queues, extremely low packet latency.

CXCC combines the backpressure feedback with implicit acknowledgements: overhearing the forwarding of a packet by the downstream node both acknowledges the reception of that packet and releases the backpressure. For single-hop reliability, CXCC uses small control packets called "Request For Acknowledgement" (RFA). If a node has not received an acknowledgement for a too long time, it issues an RFA. The RFA identifies the packet for which an ACK is missing. Upon reception of such an RFA, a decision can be made whether an acknowledgement has indeed been missed by the RFA's sender. In that case, an explicit acknowledgement can be sent to resolve the situation.

If the RFA's receiver does not know the data packet the RFA refers to, then this transmission must have been lost. A retransmission of the full packet can then be requested by a non-acknowledgement (NACK). The main benefit of the RFA/(N)ACK handshake, compared to an immediate retransmission of the data packet if an ACK is missing for too long, is the substantially lower overhead. It avoids unnecessary payload retransmissions in cases where the (implicit) acknowledgement has not yet been sent or has been lost. More details can be found in [9].

Since CXCC uses a similar backpressure rule as the one introduced above, integrating these approaches becomes a natural solution. It turns out that only one significant modi-

fication to CXCC is necessary to yield the desired behaviour: adding the rule that only coded pairs of oppositely directed packets may be transmitted by intermediate nodes after leaving the initialisation state.

One more aspect, however, also deserves attention. In noCoCo, when a node has made a transmission, it has to wait for feedback from both its neighbours before the next transmission is allowed. When a transmission fails, additional protocol handshakes are required for recovery. It is important to design these in a way that always ensures a consistent view of a neighbour's queue state. For example, it may not happen that a node A learns that its neighbour B has forwarded the previously sent packet (through an implicit or explicit ACK), without at the same time obtaining the information that it had been combined with a transmission in the opposite direction. This is particularly crucial during the initialisation phase: otherwise the backpressure at A is released through the ACK, allowing A to forward another (uncoded) packet to B. It should, however, have sent a *coded* combination with the oppositely directed packet. Therefore, we need to make sure that we both allow recovery from packet losses and ensure consistent state updates at the neighbouring nodes.

Fortunately, this is relatively simple to achieve. Data packets are coded and thus always carry information on both forwarding directions, from which the queue states can be inferred. Control packets can be tailored to contain information on a node's left- and right-directed queue states. In particular, a packet in one direction must never be acknowledged without also communicating the queue state of the opposite direction at the same time. This is achieved by also sending the respective control messages in pairs, one for each direction. In effect, each correctly received packet conveys information on both directions. The resulting protocol is able to effectively recover from packet losses, while retaining all the desirable properties shown previously.

4.4 Handling finite bursts of data

As the final step to a practically usable protocol, we need to take into account that practical data transmissions are of finite length. In the protocol described so far, there is no way to return to a state where an intermediate node is allowed to continue with unidirectional transmissions, if the packet stream from one of the end nodes has ended, temporarily or permanently. Therefore, a mechanism is needed to allow for the completion of a transmission, in one or in both directions.

A viable solution is in fact pretty simple: when the source node sends the last packet of a burst, it may set a special flag in this packet. An intermediate node, after forwarding a packet with this *end-of-burst flag* set, will resume the initialisation state. Unidirectional transmissions in the opposite direction are then allowed again at this node.

On the source nodes, the end of a burst may be indicated by the application itself, if such a tight integration is desired and possible. Otherwise, a source node may simply set the flag if it sends out the last packet in its queue, i. e., if no further packets have so far been generated by the application.

In practice, this approach yields a smooth, load-dependent transition between opportunistic and enforced coding: if both sources produce packets at a rate that fills up the capacity of the route, maximum coding gain will be enforced. If at least one direction does currently not produce packets at a rate that suffices to build up backpressure, the scheme falls back to opportunistic coding. For unidirectional traffic, noCoCo performs COPE-style opportunistic coding, with scheduling that reduces to unmodified CXCC. It therefore also provides efficient congestion control for the unidirectional flow.

5. PERFORMANCE EVALUATION

Since CXCC, and also noCoCo, are cross-layer approaches involving changes to the MAC layer, they cannot be easily implemented on an IEEE 802.11-based wireless testbed. Therefore we have performed simulations using the network simulator ns-2.30 [7] to assess the performance of co-ordinated coding against a purely opportunistic approach. We use the IEEE 802.11 MAC, UDP, and TCP Newreno implementations from the standard ns distribution and the implementation of CXCC from [9]. For performance comparison, we have implemented three additional protocols:

- COPE, as described in [4], operating over the plain 802.11 MAC; just as UDP/TCP over 802.11 and CXCC, it serves as a reference.
- A straightforward combination of CXCC and COPE-style opportunistic network coding; this protocol performs CXCC-style packet forwarding and implicit acknowledgements, but combines multiple packets via XOR into one transmission if coding opportunities arise. The existence of such coding opportunities is, however, not guaranteed.
- The co-ordinated network coding scheme we introduce.

In all but our most simple simulation scenarios there will be more than one connection, and thus additional coding opportunities may arise. The COPE-based protocols rely on recognising these opportunities to maximise the coding benefit. Since the coding component in noCoCo is based on COPE, our noCoCo implementation also makes use of coding opportunities beyond the current two-way connection, as they arise and are identified. In particular this implies that noCoCo does not miss coding opportunities that can be identified by CXCC+COPE, and may thus be expected to perform at least equally well under all circumstances.

However, COPE's reception reports and guessing both incur substantial complexity and add variance to the protocol performance. Given our focus is on the different performance due to co-ordinated versus opportunistic coding, we

establish upper and lower bounds on what *any* mechanism for identifying coding opportunities can possibly achieve, instead of following any of the above mechanisms.

For this purpose, we derive two variants for each coding scheme, termed "conservative coding" and "omniscient coding". Conservative coding combines packets only if decoding is guaranteed successful, without exchanging any additional information. In effect, overheard packets are not used for coding. Conservative coding thus represents a lower bound of what can be achieved by any scheme for identifying coding opportunities. Omniscient coding, on the other hand, employs a central component to provide each node with immediate and exact information on the packets known by every other node. While this can be implemented in a simulator, it would obviously not be possible on real devices. Omniscient coding thus allows for perfect coding decisions, without the delay or overhead of reception reports or the risk of guessing wrong. This gives an upper bound on what an ideal scheme may achieve for identifying coding opportunities. We will show later that the lines representing either bound often overlap for noCoCo in the plots.

Unless otherwise stated, we use the common settings of ns-2.30. This includes a physical layer bit rate of 1 Mbps, the two-ray ground propagation model, 250 m radio range, and 550 m carrier sense radius. Data packets carry 512 bytes of payload. We use static, hop-count minimal routes to avoid possible side-effects introduced by a specific routing approach. The RTS/CTS mechanism of 802.11 was switched off in all our simulations. We have observed performance degradation with RTS/CTS enabled, in line with similar findings in the literature (see, e. g., [8, 12, 13]). In the following, we first investigate the protocols' performance in simple chain and cross topologies, and then consider random scenarios with dynamic traffic patterns.

5.1 Chain topology

For our first set of simulations, we use a chain topology with ten hops. The distance between neighbouring nodes is 150 m. We set up bidirectional UDP traffic, originating from both ends of the chain, and being directed to the respective opposite end. The offered load at the sources is gradually increased.

All intermediate nodes can encode at most two packets together at a time, one from each direction. In such an environment, there is no difference between the coding opportunities that can be identified with conservative coding and with omniscient coding. Therefore we do not need to distinguish between these two.

Figure 2 shows how the total application-layer throughput varies with increasing offered load. Bidirectional UDP throughput over 802.11 drops rapidly once the optimal offered load is exceeded [9]. Opportunistic network coding alone barely alleviates this. Co-ordinated network coding with noCoCo, however, achieves superior throughput. One reason is the implicit backpressure property of the congestion control mechanism, which is also present in CXCC. But noCoCo achieves a substantially higher throughput than CXCC, and this throughput gain far exceeds the one obtained with opportunistic coding in CXCC+COPE. The main reason for the smaller gain with the opportunistic approach is that coding opportunities do not reliably arise. Nodes often have only one packet in the transmission queue, and many transmissions are uncoded.

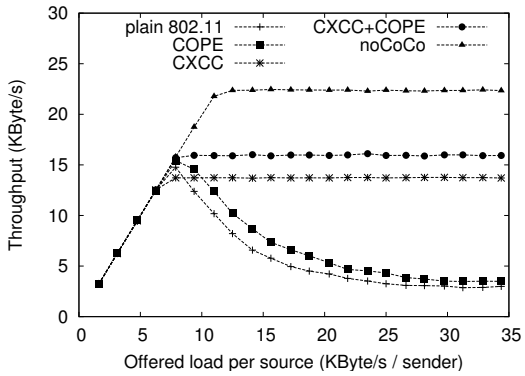

Figure 2: Throughput in chain topology.

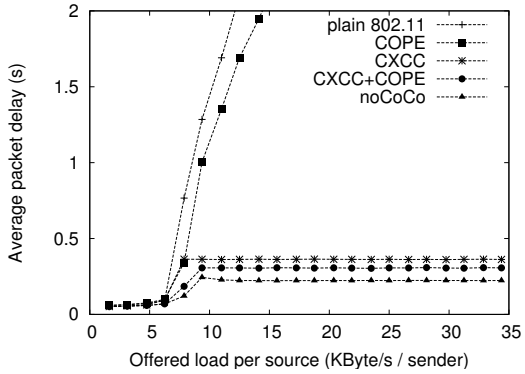

Figure 3: Packet latency in chain topology.

Figure 3 shows the average packet delay, measured from when a packet leaves the source node to its successful reception at the destination. The delay increases very quickly with increasing offered load for plain 802.11 as well as for COPE, because long queues are building up, especially in the nodes close to the ends of the chain. A packet has to wait for a potentially long time in these queues, before being forwarded. Since the backpressure rules in noCoCo and CXCC result in very short queues in the intermediate nodes, the delays are substantially shorter. As a consequence of even better medium utilisation, noCoCo's delays are smaller than those of opportunistic coding schemes.

The efficiency of the medium utilisation is also closely related to the protocol overhead. Here we use an overhead metric that quantifies the average amount of data transmitted on the wireless medium in order to bring one byte of payload one hop further. It is computed by summing up the bytes

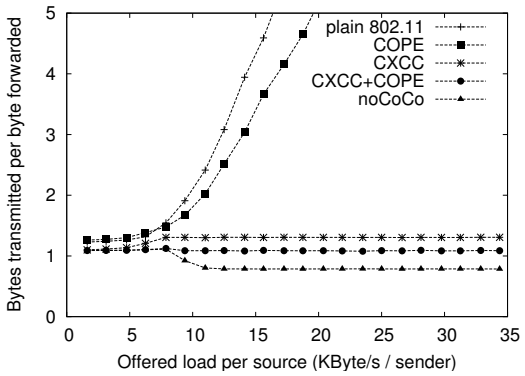

Figure 4: Overhead in chain topology.

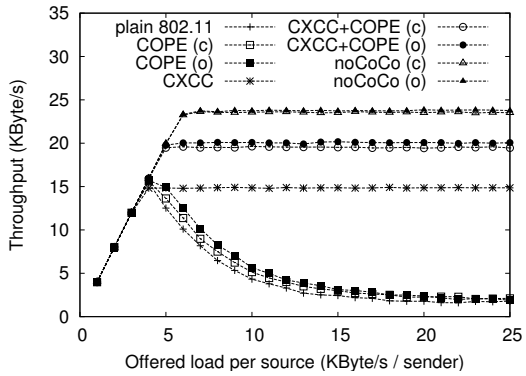

Figure 5: Throughput in cross topology with increasing offered load.

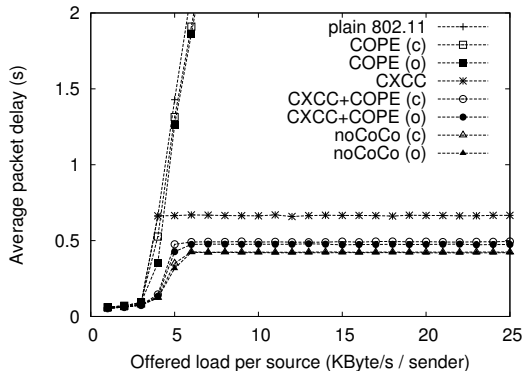

Figure 6: Packet latency in cross topology with increasing offered load.

from all packets transmitted on the MAC layer, divided by the product of the amount of application data successfully delivered and the hop distance between source and destination. Note that all transmissions are included, including control traffic and retransmissions of data packets. Since wireless communication, especially transmitting, is rather expensive in terms of energy consumption, this metric also reflects the energy efficiency of the protocol. Without network coding, the optimal value of our overhead metric is one: it is clearly not possible to forward one byte of payload while transmitting less.

Figure 4 presents the results of this evaluation in the chain topology simulations. Apart from generally confirming the picture gained from the previous metrics, the noCoCo plot impressively demonstrates the benefits of network coding. Around an offered load of 10 KB/s at each source the source data rate approaches the network capacity. At this point, the transition from opportunistic forwarding of single packets to enforced coding happens for noCoCo. Once network coding at each hop is guaranteed, noCoCo underruns the value of one for the overhead metric: on average it transmits about 0.79 bytes to forward one byte of payload over one hop. COPE does not achieve this, again due to the lack of spontaneous coding opportunities.

5.2 Cross topologies

We now turn to cross topologies, consisting of two orthogonally aligned chains like above, sharing one node in the middle. UDP traffic flows from the end of either leg of the cross to the end of the opposite leg. We first increase the offered load in a cross with five hops in each leg, similar to what we did in the above chain simulations. Subsequently we will look at the effects of varying leg lengths.

Due to the node spacing of 150 m and the communication radius of 250 m, the nodes adjacent to the centre node are able to overhear the transmissions of their two counterparts in the other chain. Thus, it is generally possible to combine up to four packets into one transmission at the middle node. While omniscient coding can make optimal use of this, conservative coding will not. We therefore distinguish between these two strategies in our figures, denoting conservative coding by "(c)" and omniscient coding by "(o)".

Figures 5, 6, and 7 show throughput, packet delay, and overhead respectively for the cross with a leg length of five hops. The occurring effects are generally quite similar to those already observed in the chain topology. Remarkably, there is generally very small difference between conservative coding and omniscient coding. This suggests that the benefit from additional coding opportunities is quite limited. For noCoCo in particular, the differences between the simulations with conservative coding and with omniscient coding are so small that the respective lines in the plots are barely distinguishable.

One might argue that these small differences stem mainly from having only 1 out of a total of 21 nodes that is able to make use of the additional opportunities. We thus complement the results with plots showing the effects of varying leg lengths of the cross. In Figures 8, 9, and 10 we use a saturated offered load and gradually increase the size of the cross. From these results, it becomes clear that the previously discussed overall picture of relative performance sets in very quickly, and generally holds for a leg length of two hops already.

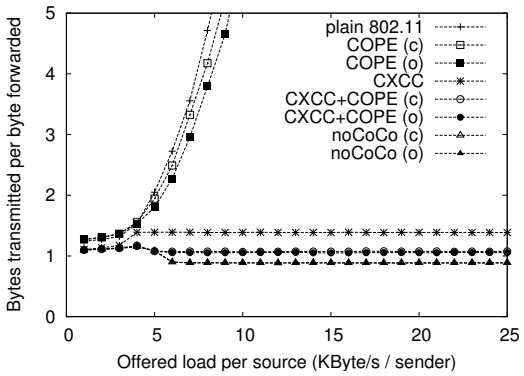

Figure 7: Overhead in cross topology with increasing offered load.

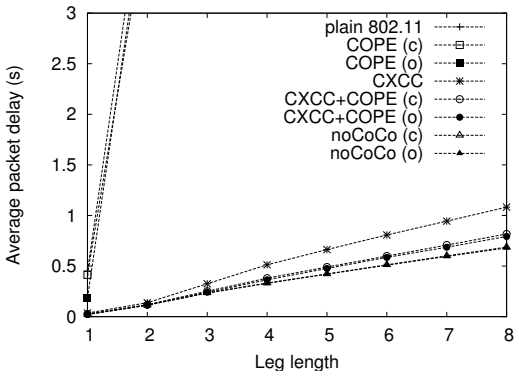

Figure 9: Packet latency in cross topology with increasing leg length.

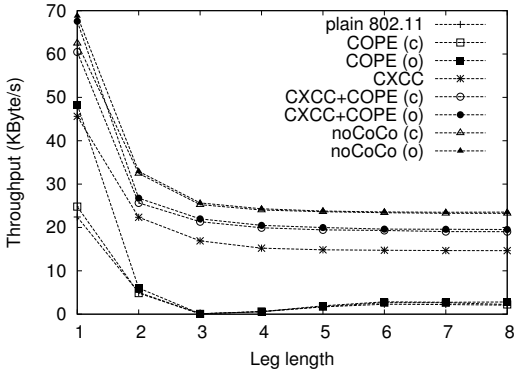

Figure 8: Throughput in cross topology with increasing leg length.

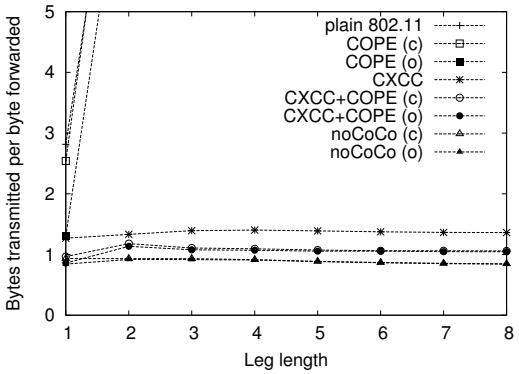

Figure 10: Overhead in cross topology with increasing leg length.

The only significant differences occur for the situation where each leg is only one hop long, i. e., where the source/sink nodes are directly adjacent to the centre node. In this case, the differences between conservative and omniscient coding are indeed significant, particularly for COPE. The effect of a high gain of COPE versus plain 802.11 in this specific setting has been observed and explained as the "coding+MAC gain" in [4]. Without coding the intermediate node needs to transmit four times as often as the other nodes, but the 802.11 MAC does only assign it 1/5 of the medium time. With omniscient coding, this limitation is no longer a bottleneck. The central node will not be able to access the medium more often, but it can transmit up to four packets with one medium access.

The backpressure rules in noCoCo and the CXCC-based schemes make the source nodes refrain from further transmissions until the central node has forwarded the previous one. These protocols therefore inherently avoid the problem of inappropriate assignment of medium access opportunities.

5.3 Random topologies

Finally, we study gains obtainable with noCoCo versus opportunistic schemes in more practical settings, and consider random, static topologies. We intentionally set them up in a way that yields rapidly changing traffic patterns. With few, long-lived connections the availability of coding opportunities essentially depends on the routes of these connections and whether they share many intermediate nodes, whereas many short-lived will result in a large variation in the packets that meet.

Each simulation scenario uses 150 nodes at uniformly random positions on a 1500×1500 m square area. A total of 40 bidirectional connections start at random times between 0 and 120 seconds. Each connection is assigned a random amount of data between 5 and 50 KB, which is to be transmitted in both directions. In the absence of route breaks the single-hop reliability mechanism of CXCC will result in reliable end-to-end delivery; the same holds for noCoCo, which adopts CXCC's respective mechanisms. It does not hold for plain 802.11 and COPE. Thus, in order to ensure reliable delivery of the data we use TCP Newreno as a reliable transport protocol. The generally unsatisfactory performance of TCP congestion control over wireless multihop networks is well-known and has to be taken into account when comparing the results here. However, since we are more interested in the relative improvements with opportunistic and co-ordinated network coding, this is only of secondary relevance here.

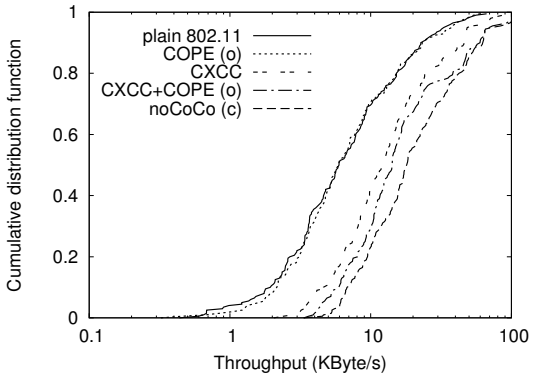

Figure 11: Cumulative distribution function of throughputs in random topologies.

We simulate all protocols using the same set of topologies and traffic patterns. Since the connections run over very different hop counts and transmit different amounts of data, it is hard to define a global throughput metric that would be able to capture all effects appropriately. But it is easily possible to calculate the throughput of each single connection, by dividing the amount of data delivered by the time it took from initiating the connection to the successful delivery of the last data segment. In Figure 11 we show the cumulative distribution functions of these per-flow throughputs. For example, about 25 % of all connections with noCoCo achieve a throughput below 10 KByte/s, whereas the same applies to about 40 % of the connections with CXCC, and to about 70 % of those with TCP Newreno over IEEE 802.11.

The results with conservative coding and omniscient coding are generally close together here. To maintain readability of the figure, we only show the results with omniscient coding for COPE and CXCC+COPE (as an upper bound of these protocols' performance), and with conservative coding for noCoCo (as a lower bound).

The random topology simulations confirm that our findings from the deterministic topology simulations above hold in a similar way also in more complex environments. It was pointed out in [4] that the interaction between COPE and TCP is complex, due to TCP's congestion avoidance rules, timing issues and potential packet reordering. We also make this observation here. Even though the traffic is bidirectional – and coding opportunities of similarly sized packets can thus generally exist at each single intermediate hop – COPE barely improves the throughput. For the protocols with a backpressure rule, the relative performance matches the picture from the previous simulations, with opportunistic coding noticeably improving upon the performance of non-coding CXCC, but in turn being clearly outperformed by coordinated network coding.

The average per-hop delay of the connections is shown in Figure 12. The per-hop delay of a packet is the time from leaving the source node until the arrival at the destination node, divided by the number of hops along the route. The

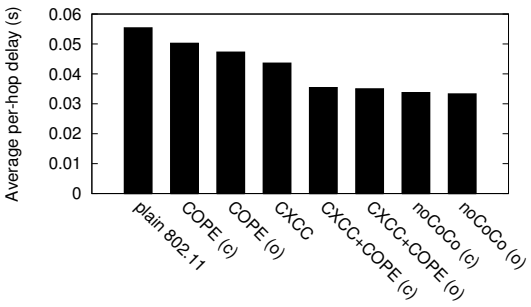

Figure 12: Per-hop delay in random topologies.

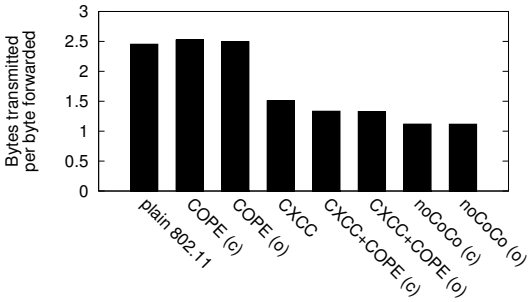

Figure 13: Overhead in random topologies.

relative ordering of the protocols remains the one observed above, though the absolute differences are generally not as grave as those of, e. g., the throughput.

In Figure 13 we have evaluated the overhead in the random topology simulations, and again the overall picture seems familiar. In the presence of a complex traffic pattern and with TCP being used, COPE actually exhibits a minimally higher overhead than plain 802.11. This can be traced back to a higher number of packet retransmissions. Unlike in the chain and cross topologies above, noCoCo does not underrun the non-coding limit of an overhead of one in these simulations, but it comes very close to this value.

On a final note, we emphasise again that the main focus of noCoCo is on scheduling packet transmissions to coordinate coding, and much of the benefit we have seen in this section of noCoCo over COPE with the 802.11 MAC is due to the scheduling. Therefore, we used traffic patterns that would favour coding in the first place to highlight the importance of scheduling. Dealing with coding-adverse traffic patterns, such as asymmetric and more random flows is an interesting research issue but beyond the scope of this paper.

6. CONCLUSION

In this paper, we have introduced a deterministic packet scheduling scheme for network coding within two-way traffic flows in wireless multihop networks. We have derived some general properties of scheduling schemes that achieve maximum coding gain in the given scenario, and introduced a centralised scheduling approach with favourable performance. Developing this further into a practical and dis-

tributed protocol, we proposed Near-Optimal Co-ordinated Coding (noCoCo). noCoCo has been shown to approach the theoretical limits established with the centralised scheduling approach. It also results in a form of congestion control resembling the implicit hop-by-hop congestion control in CXCC.

Our evaluation demonstrates the potential of a co-ordinated approach for network coding, in terms of throughput, packet delay, and protocol efficiency. The results also reveal some interesting insights regarding opportunistic coding schemes. Without appropriate medium allocation, network coding alone may also be at its wit's end for achieving good performance. It is also interesting to notice the small differences between conservative coding and omniscient coding. This suggests that the benefits from identifying additional coding opportunities are limited in this context, which could be used to simplify coding protocols adopting similar scheduling principles.

Note that the current noCoCo design is based on insight from analysing two-way traffic and has an end-to-end control flavour. This opens many avenues to generalise the design and extract further coding benefit. For example, more coding opportunities could arise if the signalling for burst traffic is carried per hop. Furthermore, an analysis of different traffic patterns could bring new insights.

Acknowledgments

The authors are grateful to the German Research Foundation (DFG) and the ITA project for funding this research.

7. REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] P. Chaporkar and A. Proutiere. Adaptive network coding and scheduling for maximizing throughput in wireless networks. In *MobiCom '07: Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, pages 135–146, Sept. 2007.

[3] T. Ho, Y.-H. Chang, and K. Han. On constructive network coding for multiple unicasts. In *Proceedings of 44th Allerton Conference on Communication, Control and Computing*, 2006.

[4] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in the air: Practical wireless network coding. In *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 243–254, Sept. 2006.

[5] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using packet symmetry to curtail malicious traffic. In *HotNets '05: Proceedings of the 4th Workshop on Hot Topics in Networks*, Nov. 2005.

[6] Z. Li and B. Li. Network coding: The case for multiple unicast sessions. In *Allerton '04: Proceedings of the 42nd Annual Allerton Conference*, Sept. 2004.

[7] The ns-2 network simulator. http://www.isi.edu/nsnam/ns. version 2.30.

[8] S. Ray, J. B. Carruthers, and D. Starobinski. RTS/CTS-induced congestion in ad hoc wireless LANs. In *WCNC '03: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 1516–1521, Mar. 2003.

[9] B. Scheuermann, C. Lochert, and M. Mauve. Implicit hop-by-hop congestion control in wireless multihop networks. *Elsevier Ad Hoc Networks*. To appear, available online. DOI 10.1016/j.adhoc.2007.01.001.

[10] S. Sengupta, S. Rayanchu, and S. Banerjee. An analysis of wireless network coding for unicast sessions: The case for coding-aware routing. In *INFOCOM '07: Proceedings of the 26th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1028–1036, May 2007.

[11] Y. Wu, P. A. Chou, and S.-Y. Kung. Information exchange in wireless networks with network coding and physical-layer broadcasts. In *CISS '05: Proceedings of the 39th Annual Conference on Information Sciences and Systems*, Mar. 2005.

[12] K. Xu, M. Gerla, and 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*, pages 72–76, Nov. 2002.

[13] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *IEEE Communications Magazine*, 39(6):130–137, June 2001.