A Modular Framework for Implementing Joint Wireless Network Coding and Scheduling Algorithms

Ioannis Broustis¶, Georgios S. Paschos*¶, Dimitris Syrivelis¶, Leonidas Georgiadis‡¶, Leandros Tassiulas†¶
¶Center for Research and Technology, Hellas, Greece,
*LIDS, Massachusetts Institute of Technology, USA,
‡Aristotle University of Thessaloniki, Greece,
†University of Thessaly, Greece

Email: gpasxos@mit.edu;

*Corresponding author

Abstract

Intersession network coding (NC) can provide significant performance benefits via mixing packets at wireless routers; these benefits are especially pronounced when NC is applied in conjunction with intelligent scheduling. NC however imposes certain processing operations, such as encoding, decoding, copying and storage. When not utilized carefully, all these operations can induce tremendous processing overheads in practical settings. Our testbed measurements suggest that such processing operations can severely degrade the router throughput, especially at high bit rates. Motivated by this, we design **NCRAWL**, a Network Coding framework for Rate Adaptive Wireless Links. The design of NCRAWL facilitates low overhead NC operations, thereby effectively approaching the theoretically expected throughput benefits of joint NC and scheduling. We implement and evaluate NCRAWL on a wireless testbed. Our experiments demonstrate that NCRAWL meets the theoretical predicted throughput gain while requiring much less CPU processing, compared to related frameworks[1]

[1]The work of D. Syrivelis and L. Tassiulas was supported by the European Commision project STREP-FP7-INFSO-ICT-215252:N-CRAVE. The work of G. S. Paschos was supported by the WiNC project of the Action: Supporting Postdoctoral Researchers, funded by national and Community funds (European Social Fund). The work of L. Georgiadis was supported by STREP-FP7-INFO-ICT-224218: OPNEX.

1 Introduction

Intersession Network Coding (NC) enables the local processing and mixing of independent traffic flows. Combining such flows at wireless routers can increase the available capacity [1–3]. However, such increase is evident only when: (a) routers (which perform the encoding operations) are able to quickly identify efficient coding opportunities that increase the NC gain; (b) packet decoders are able to correctly decipher the encoded packets and acknowledge the decoded packets that they receive in diverse channel conditions; and (c) the overheads imposed due to the inclusion of additional packet headers as well as packet processing operations [4] are kept minimal. Moreover, while NC can increase the router throughput in random-access networks [5], prior studies have shown that when the packets are scheduled carelessly, NC may not offer significant benefits [2,6]. With multi-rate links, and when decisions are made based on statistical information, scheduling is necessary to avoid packet losses. All these factors should be taken into account when designing and developing practical, efficient NC algorithms and systems.

Implementing efficient wireless NC in practice is a challenging task: Although intersession NC can theoretically offer unprecedented wireless router capacity benefits, realizing these benefits in practical systems is by no means an easy task. Prior implemented efforts towards practical NC [3, 7–9] have demonstrated throughput benefits at low transmission rates but have also discovered a series of complexity issues arising in such implementations. Our testbed measurements suggest that it is quite challenging to map the theoretically expected benefits offered by wireless NC in practical multi-rate deployments. This is due to two reasons, which motivate our study; we explain them below.

a. The overheads incurred by NC may be excessive: With NC, routers need to be aware of the packets that have been successfully overheard by each neighbor, in order to further decide which packets to encode together and when. One method that has been proposed for addressing this requirement is by enforcing every neighbor into explicitly acknowledging overheard packets (e.g. see [3, 7]). However, unless clever implementation techniques are employed, this approach may not improve the performance; this is because significant additional packet processing needs to be performed at routers, which may intrusively increase the already imposed processing overhead. Thus, although the wireless channel may be conducive to the use of high bit rates, routers may be incapable of transmitting as many packets in order to meet those rates. Our experiments with various NC implementations suggest that with overhead-agnostic design choices, the theoretical benefits offered by NC cannot be mapped in practice, due to the excessive processing and network overheads that are imposed.

b. Scheduling for NC has not been exploited in practice: Previous studies have demonstrated the benefits of jointly applying NC and link scheduling [2, 6]. However, prior practical implementations (i) have not incorporated any such techniques, and (ii) have not been designed to host scheduling algorithms in an efficient manner. This necessitates the design of a broad, although lightweight framework, which can facilitate the efficient coexistence of wireless NC and scheduling.

Our contributions: Towards addressing the above two issues, we design and implement NCRAWL, a Network Coding framework for Rate Adaptive Wireless Links. NCRAWL has been optimized at each stage of NC operations to utilize the router resources efficiently. It is a modular tool, which can host the implementation of intersession NC schemes that are either standalone, or tightly integrated with scheduling algorithms. The NCRAWL modules manage all the NC operations, such as encoding, decoding, storage and routing in a lightweight manner, allowing for overhead-limited network operations. Note that to the best of our knowledge, our framework is the first to facilitate the practical coexistence of NC and scheduling. Theoretically shown throughput benefits can be easily assessed on NCRAWL and adapted for operating on real networks with limited effort. We implement NCRAWL on *Click* as a Linux kernel module [10]; we evaluate our framework on a wireless testbed via measurements, under various indoor and outdoor topological and traffic settings. Our experiments demonstrate that NCRAWL offers significant throughput improvements even at high bit rate regimes, where previously implemented schemes are unable to operate, due to excessive computational and networking overheads.

The scope of our work: Our focus is not on proposing optimal scheduling policies for NC, but on developing an accurate, lightweight and easy-to-deploy experimental tool that can host NC and scheduling schemes. Our system design and implementation decisions could be embedded into previously proposed platforms, such as COPE [3], ER [7], CLONE [8], etc. However, we have chosen to implement NCRAWL from scratch, instead of trying to modify such platforms, only because it has been much easier to implement certain design choices in a particular, modular manner. Moreover, in this paper we focus on local NC, where an encoded packet does not traverse more than one hop. This has direct applications in WLANs, where clients exchange data via their associated access point, thereby enabling high-bandwidth applications such as online gaming and video streaming. Note that NCRAWL can also be applied on top of multi-hop topologies with long routes, where native packets are potentially encoded/decoded multiple times at intermediate routers along their traversed routes.

The rest of the paper is structured as follows. In section 2 we discuss relevant previous studies. In section 3 we provide a high level overview of the considered NC scheme. In section 4 we present the modular design and implementation of NCRAWL. In section 5 we demonstrate the strengths of our design through a scheduling-driven case study. In section 6 we assess the performance of NCRAWL via extensive testbed measurements. Finally, our conclusions form section 7.

2 Related work

In this section, we discuss previous related NC studies and differentiate our work.

Experimental studies on wireless coding: Katti et al. [3] propose COPE, the seminal implementation of wireless NC. With COPE, routers are fully aware of packets that have been overheard by every neighbor. For this, each node is required to inform the router about overheard packets. Experiments with COPE on a wireless testbed show that even with very simple encoding operations, intersession NC can provide significant capacity gains. Rozner et al. in [7] present ER, a scheme that adopts the design of COPE and employs NC to perform efficient packet retransmissions. With ER, packets that need to be retransmitted are coded together, such that one retransmission can recover multiple packet losses. Kim et al. [11] extend the design of COPE to include NC-aware bit rate control and clever selection of nodes that acknowledge the reception of encoded packets. Rayanchu et al. [8] propose CLONE, a suit of algorithms for NC that take into account losses on wireless links. However, [7, 8, 11] all follow COPE's logic regarding the dissemination of information about which packets have been stored as keys. Moreover, these studies do not make online decisions about whether to enable coding or not, based on the link quality. MORE [9] is a routing protocol where routers perform random mixing of packets before forwarding them. MIXIT [12] encodes symbols rather than packets. Similarly to MORE, batches of packets are coded together. However, since a packet is a sequence of symbols, Intermediate relays use hints from the PHY layer in order to infer which symbols within a packet are correctly received with high probability.

The above experimental approaches differ from ours in two main ways: (i) We consider the use of stochastic information for overhearing instead of blindly acknowledging each particular packet. This allows for efficient implementation and avoids computationally expensive packet processing operations. Note that our design choices can also be applied to most of the above approaches. (ii) NCRAWL is a "tool" for studying problems regarding joint NC and scheduling with feedback, and can potentially be intertwined with an optimal algorithm to provide the best solution within the class of implicit ACK schemes. To the best of our knowledge, our work is the first to provide a coherent, lightweight framework for practically assessing joint NC and scheduling schemes.

Analytical and simulation NC studies: In the recent literature there exists a series of astonishing theoretical results regarding NC in wireless networks. Lun et al., in [13], show that the problem of minimizing the communication cost can be formulated as a linear program and solved in a distributed manner. Chaporkar and Proutiere [2] show that unless appropriate scheduling is applied, NC may lead to performance degradation. We support this claim by identifying an additional example when implicit ACKs are used. Liu and Xue in [14] consider NC for two-way relaying in a three-node network. They analytically characterize the achievable rate regions for the traditional Alice-Relay-Bob topology, and they find the theoretically optimal end-to-end sum rates. Scheuermann et al. [6] propose noCoCo, a deterministic packet scheduling scheme for NC within two-way multihop traffic flows. Their scheme involves per-hop packet scheduling, NC and congestion control. Seferoglu and Markopoulou in [15] provide an understanding of the interplay between application data rate control and NC. Finally, Vieira et al. [16] provide observations on how the combination of NC and bit rate diversity affects the performance of practical broadcasting protocols. They show that it is possible for multi-rate link layer broadcasts and NC to jointly increase the network throughput in multicast applications. More theoretical results can be found in [17] with the list being non-exhaustive.

In recent theoretical work [18] it is shown, that for the 2-user case, maximum throughput can be achieved without explicit ACKs, i.e. by utilizing scheduling algorithms that guess, code and then correct the transmissions using feedback. In particular, this scheme has no loss of optimality if either the downlink channel rates are equal for the two receivers or if the overhearing probability at the slow receiver is 1 (perfect overhearing). Moreover, whenever the overhearing probabilities are high enough (> 0.6), the loss of throughput in comparison to the explicit ACKs is very small ($< 5\%$). This motivates further our work, which is in line with such schedulers. In particular, our framework enables the implementation of such algorithms for reasons of performance analysis and prototyping. We believe that the experimental evaluation is very important since the complexity of some of the newly proposed approaches may degrade the performance and result in smaller benefits than compared to the theoretically predicted ones.

3 Network coding scheme

We study topologies that include 2–hop flows crossing a central node, as shown in Figs. 1-2. The central node, called the *relay* (or router), is connected to all other nodes, called *neighbors*. Links between neighbors may exist as well. Each link connecting nodes i and j is characterized by two channel rates r_{ij}, r_{ji} and two probabilities q_{ij}, q_{ji} which correspond to the Packet Delivery Ratios (PRD) in each direction.

In the uplink part (the first hop of these flows), *native* packets are transmitted without NC towards the relay. In the downlink part, the relay selects a number of native packets, applies the XOR operator and

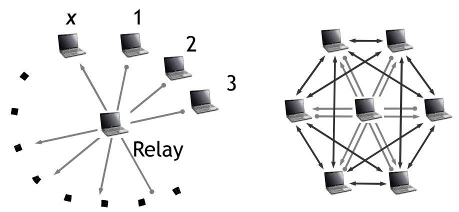

Figure 1: Example topologies supported by NCRAWL: (left) The $\frac{x}{2}$–wheel with $x+1$ nodes and $\frac{x}{2}$ activated flows. (right) The 6–wheel topology, grey arrows represent flows and black arrows represent overhearing links.

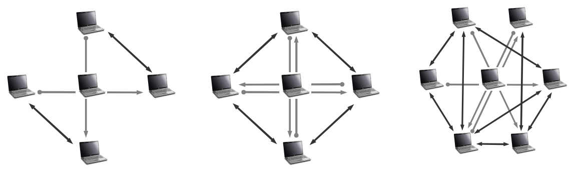

Figure 2: Example topologies supported by NCRAWL: (left) The $\frac{4}{2}$–wheel also called half-cross, (middle) the 4–wheel topology also called cross and (right) an arbitrary subgraph of the 6–wheel topology.

transmits the constructed *encoded* packet. If a receiver recognizes its address in the header, it attempts to perform decoding in order to obtain the intended native packet. To achieve this, the receiver should be in possession of all the other native packets that comprise the encoded packet, in order to perform decoding. These packets are known to the receiver either because they have been generated by it (in case of symmetric flows) or because they have been obtained by means of opportunistic listening, i.e. overhearing of the uplink transmissions [3]. Packets that are successfully obtained are acknowledged to the relay via a separate mechanism that operates in a higher layer (feedback packets are generated and sent to the relay), see §4.2.

In order to experiment on high gain topologies, we consider two particular classes of topologies which we name as $\frac{x}{2}$–wheel and x–wheel; these names are inspired by [3]. Such topologies are ideal for performing wireless NC, since they allow a maximum number of employed 2–hop flows to be mixed due to the existence of appropriate overhearing links. For example, in Fig. 1 (right), the relay may encode 6 packets and send them in one transmission, while the decoding is guaranteed due to the existence of appropriate links (black arrows). A constructive way to build an $\frac{x}{2}$–wheel topology is the following: Split the neighbor nodes in two equal sets (x must be even), the source and the destination set, and select a matching of these two sets which corresponds to $\frac{x}{2}$ flows (pairs of nodes). Then, allow all nodes (the relay and the neighbors) to be connected except those that are paired in the above selected matching. Finally, create $\frac{x}{2}$ more flows by inverting the roles of source and destination. If only the initial $\frac{x}{2}$ flows are enabled, we refer to the topology as "$\frac{x}{2}$–wheel setting"; otherwise, if all x flows are enabled, we refer to it as "x–wheel setting". See Figs. 1-2 for examples of such settings wherein NC can be applied.

Evidently, the 2–wheel setting corresponds to the well known Alice-Relay-Bob topology. Throughout this paper, we also use $\frac{4}{2}$–wheel settings (referred to as "half-cross") and 4–wheel ones (referred to as "cross), as well as a 6–wheel setting, wherein we activate the flows serially, one after the other (again see Figs. 1-2 above). We should point out that NCRAWL supports any random subgraph of the N–wheel topology with any possible set of flows activated on top of it. In addition, it supports settings with any possible combination of link qualities and/or channel rates. The wheel topology is the most general topology to be considered around a single node. Any actual network topology can be reduced to a subgraph of a wheel topology, if the nodes, links and flows irrelevant to NC on any given node are removed. Since NCRAWL runs on all nodes in the network, our scheme works with any arbitrary network topology, providing the maximum possible local NC gain subject to some constraints we explicitly mention below. As we discuss later, the encoding opportunities at each node are automatically discovered by the combination of NCRAWL and SRCR [19]. Hence, NCRAWL operates under any assumed graph providing throughput gains opportunistically. We study the wheel setting where the maximum such gains arise, in order to showcase that NCRAWL can achieve it in many cases despite the overheads and complexity induced by NC.

We have incorporated the following features in order to demonstrate the practicality of NCRAWL in a tractable manner:

- The XOR (binary field) operator is used instead of more general linear coding, similarly to [3].
- We enforce the decoding of encoded packets at the next hop, since this is practically the most possible case in today's wireless access deployments. Multihop flows can still benefit from our scheme by cascading the 2-hop coding operations.
- Only native packets are stored as keys.
- We consider the use of *implicit ACKs* of overheard packets, as we explain below.
- We use equal packet size (maximum MAC PDU); if a packet has smaller number of bits, we pad with zeros.

The decision on imposing these features is justified by the necessity to keep the NC scheme simple, practical, implementable and efficient in terms of processing overhead. Clearly, the modular nature of NCRAWL allows the development of more complex features, taking into account potential application and policy/administration requirements.

4 Architectural Blueprint

In this section, we describe the modular design and implementation of NCRAWL.

Employing *Click* as the basis of our framework: NCRAWL has been developed in the *Click* modular router framework [10]. Click can be used to develop primarily OSI layer 3 packet processors, which can be directly deployed inside the standard Linux network stack. A Click processor is mainly comprised of (a) processing stages which are called *elements* and (b) an element interconnection configuration that indicates the processing flow. Execution in Click is event-driven, with four different types of asynchronous events, namely the *incoming packet* event, the *ready-to-forward packet* event, the *timer expire* event and the *external read or write* events to Click memory. Since all Click events are asynchronous, a Click packet processor leverages internal queues to temporarily store incoming packets, since they are to be forwarded asynchronously.

In what follows, we describe the NCRAWL system design and implementation. We also present the NCRAWL interface that can be used to develop new algorithms, as well as for deploying and managing experiments on wireless testbeds.

4.1 Design preliminaries

NCRAWL realizes an OSI layer-2.5 protocol that lies immediately under the routing layer. More specifically, it can be considered as an extension to the Click modular router implementation of the SRCR protocol [20], which is the heart of the MIT Roofnet wireless network. NCRAWL operates below the routing layer; this simplifies the format of the NCRAWL packet headers, which are now used only to encapsulate encoded packets, and carry the corresponding ACKs for successful decoding back to the sender. The NCRAWL header formats are depicted in Fig. 3. Network-wide unique 32-bit packet identifiers are used by applying the *sdbm* hashing algorithm [21] on data tuples that are comprised of packet source IP, the IP header sequence number, and the respective offset.

4.2 NCRAWL System Description

Next, we discuss the modular design of NCRAWL in detail. Our platform is based on a Click network packet processor that includes the SRCR routing protocol implementation for wireless networks [20]. We have included two additional processing stages: the NCRAWL *decoder* and the NCRAWL *encoder*. We have developed these stages as individual Click elements, and we have placed them before the beginning and after the end of the SRCR processing flow, respectively, as depicted in Fig. 4.

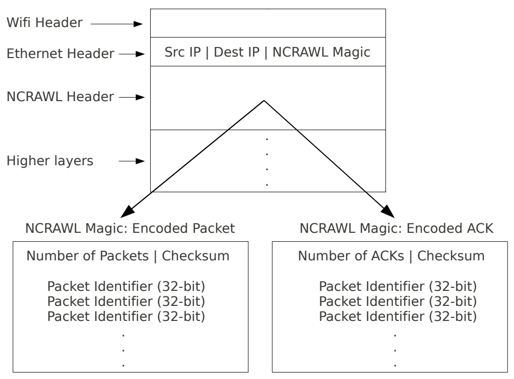
Wifi Header
Ethernet Header
Src IP | Dest IP | NCRAWL Magic
NCRAWL Header
Higher layers
NCRAWL Magic: Encoded Packet
Number of Packets | Checksum
Packet Identifier (32-bit)
Packet Identifier (32-bit)
Packet Identifier (32-bit)
NCRAWL Magic: Encoded ACK
Number of ACKs | Checksum
Packet Identifier (32-bit)
Packet Identifier (32-bit)
Packet Identifier (32-bit)

Figure 3: NCRAWL header format. The Ethernet header magic number distinguishes between encoded packets and ACK headers which are otherwise identical.

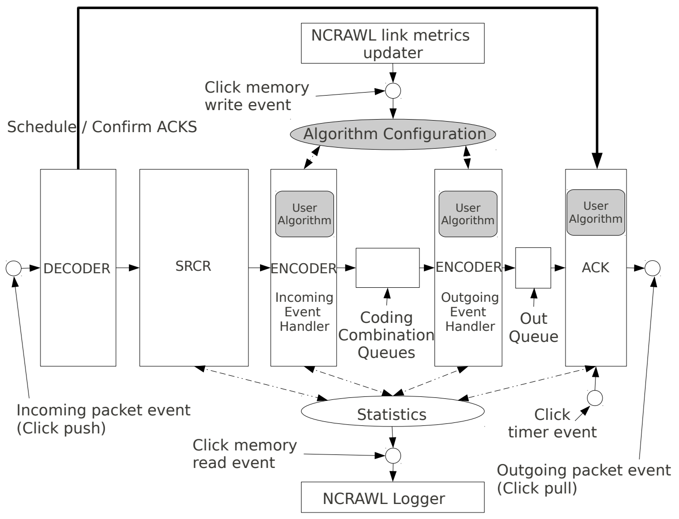

Figure 4: NCRAWL framework system components and their interaction.

a. NCRAWL acknowledgements (implicit acks): As opposed to [3], which uses explicit ACKs, where each successful overhearing of a packet is acknowledged to the router[2], NCRAWL uses ACK packets only for verification of correct decoding. Following the transmission of a coded packet, the router expects layer-3 ACK messages from the intended receivers of the coded packet. This takes place for two reasons:

- The router obtains MAC-level ACK only from one receiver; thus without the layer-3 ACK, the router cannot be notified of a loss that has occured at the other intended receivers.
- Since the overhearing events are unknown to the router, it is impossible to know if the decoding was successful (i.e., if the receivers have the necessary keys in their buffers).

[2]Recall that the role of the router in an ad hoc setting might be played by virtually all nodes in the network

Note, that a NACK (negative ACK) architecture is possible if the broadcast 802.11 mode is used. If NACKs are used, the overhead of ACK messages can be reduced significantly in practical scenarios (95% reduction is possible). Even with the ACK architecture though, ACK messaging is reduced due to the lack of unnecessary overhearing ACKs.

The router holds a buffer for the packets to be acknowledged. When an ACK arrives, the buffer is inspected and the acknowledged packet is removed. The buffer has a periodic timer and each packet is stored in the buffer for at least a specific period. When the timer expires, packets found to have been stored for more than that period, are analyzed and the corresponding non-acknowledged native packets are sent back to the system queues with the information of ”decoding failure”. Note that a decoding failure can be attributed either to MAC failure or to inability of the intended receiver to decode (lack of proper keys). Note also that a timer-expire event triggers the transmission of ACKs in separate packets when there is not enough outgoing traffic to piggyback them (Fig. 4).

b. The packet decoder: The main tasks of the decoder module are the following:

- To use the available (from overhearing or ownership) *key* packets in order to decode the received encoded packet.
- To schedule the transmission of layer-3 acknowledgements (ACKs) for the correctly retrieved native packets, derived by the decoding operation.
- To determine any potential pending ACKs, as well as to verify any received ACKs.
- To tag and store all the correctly overheard data packets as potential keys; as discussed above, these will be potentially used in the near future for decoding the received encoded packets. Moreover, the key repository is used for packet retransmissions, in case an expected ACK never arrives.

The decoder resides at the packet receiving side of the system and is invoked by the corresponding packet arrival event.

c. The NC packet encoder: The NCRAWL encoder element resides at the sending side of the system and is more complicated, since it maintains and manages the processor packet queues. A part of the element handles incoming packet events, another part deals with outgoing packet events, and there is also code that gets invoked upon timer expiry as well as read and write Click configuration events (Fig. 4). It is this element that exports the framework API which can be used to develop NC algorithms. Specifically, the main assigned tasks for this module are the following:

- To process and place incoming native packets into specifically maintained queues. Our system supports a plurality of queueing operations, which can be configured as per the requirements of the NC algorithm under development.

- To identify and combine packets together, towards forming encoded packets. The selection of the appropriate packet set follows the directions of the NC algorithm under consideration, supported by NCRAWL.

- To piggyback any ACKs (through the use of scheduled, upcoming data packet transmissions) that have been scheduled by the decoder element.

- To generate potentially expected ACK tokens for each of the native packets comprising an encoded packet.

d. Maintaining up-to-date topological information: The *link metrics updater* is responsible for collecting information about the transmission rates and PDR values of the links with neighbors. This information is gathered and passed to the rest of the system via the Click memory write event mechanism. Furthermore, the software procedures that set and apply the encoding combination policies are invoked as needed.

Gathering link quality information: The NCRAWL updater relies on the SRCR protocol component [19], which maintains link connectivity information and performs periodic link measurements. SRCR sends probe packets at all rates to determine the PDR for each link and chooses the highest rate that performs well. PDR information is then used by SRCR to calculate the ETX or ETT metric [22, 23], which provides information about entire routing paths (not just 1-hop links). This information is kept in the SRCR link table, and is accessible by the NCRAWL components. The SRCR measurement period can be set as desired (the default value [19] – also used in our work – is 3 sec).

Collecting information about neighbor node links: Based on the information stored in the SRCR link table, the link updater maintains its own so-called Neighbor Table (NT), which includes information for its neighbors. Initially, the NT is empty. The updater periodically reads the SRCR link table and updates NT as needed. The NT contents are updated whenever (i) a new neighbor appears, (ii) an existing neighbor disappears, or (iii) a certain link quality changes. In such cases, the NCRAWL updater broadcasts a packet with the new NT contents and sets a timer. When such a NT packet is received (overheard) by other nodes, their corresponding updater utility replies by broadcasting its NT, provided it has not done so recently. Specifically, in order to avoid excessive network overheads due to such messaging, we leverage a NT *reply suppression threshold*; while the value of this threshold is tunable (and can be configured based on network and application dynamics), we set it to be equal to the SRCR broadcasting interval period. The NT packets are used by the NCRAWL updater to maintain the so-called Received NT Table (RNTT), which complements NT by storing information about the link quality, as measured by neighbors.

Feeding NC algorithms with updated topological information: Each time the updater modifies the contents of the NT or RNTT (i.e., each time it proactively sends or receives a NT packet which leads to an update of the RNTT) a timer is set. Upon timer expiry, the new link qualities are passed to the main NCRAWL system, where they will potentially drive adaptive NC decisions, based on the NC designer's needs. This timeout is (generously) set to 1 sec, providing ample time for any NT reply packets to arrive. The timeout value can be set based on the dynamics of the network on which NCRAWL is applied. Note here that overheads are kept low with NCRAWL, since the updater employs its own threads of execution to perform these information maintenance tasks; the current implementation uses 2 threads, but these remain suspended most of the time, making this component quite lightweight in terms of CPU occupancy. Note that the implementation of the NCRAWL updater can be trivially adjusted to cooperate with other link information gathering protocols as well, i.e., it is not tied to SRCR.

e. NCRAWL logger: The *read* events are used by another utility, the NCRAWL *logger*, which gathers various statistics that are generated online by both the encoder and decoder.

Utilizing resources effectively: Efficient resource utilization is an inherent property of NCRAWL. The repository that stores copies of packets uses a FIFO queue as the main indexing mechanism and can host up to a user-defined quantity. After the storage limit is reached, the oldest packet is removed in order for a new one to get stored. The same packets are also indexed in a hash table based on their network-wide unique identifiers. The hashtable is used to quickly retrieve packets used either as keys for decoding, or for re-sending them in case an expected ACK token expires. The same indexing approach has been used for the ACKs and expected ACK tokens as well.

4.3 Implementing NC algorithms

NCRAWL exports an API (Application Programming Interface) that can be used to implement scheduling algorithms for intersession NC. This API is a library of functions that can be used to carry out NCRAWL common tasks and mandatory function extensions to the handling code of each event. Points of extensibility and/or programmability are denoted in Fig. 4 with shadowed boxes.

Implementing packet handling operations: Regarding the incoming packet event, the designer should account for placing arriving packets into proper queues. In particular, each flow is associated with a queue, and the scheduler checks all available controls, i.e. activates a set of queues by combining one packet from each queue. It maintains a list with the expected score (reward) of all controls and selects one of those controls at each time instance. The controls that activate only one queue correspond to the case of transmitting native packets. It is always possible to deactivate NC by imposing the use of only those latter controls. With this, the developer may implement logic that disables NCRAWL when needed. Furthermore, after the placement of the incoming packet, the developer may: (a) invoke the function that chooses the next encoded packet queue combination according to the NC algorithm, (b) retrieve the packets from the respective queues, (c) encode them and schedule the encoded packet for transmission by placing it into the outgoing queue. This operation may be repeated until the outgoing queue contains a user defined number of packets. Apart from the queue combination retrieval function, which needs to be implemented by the algorithm developer, the rest of the required functionality is already seamlessly provided by NCRAWL.

Implementing the core NC logic: The main algorithm implementation takes place in the context of the Click memory write event, generated by the NCRAWL updater. The latter provides the user with a table of links with entries denoted by the corresponding source and destination IP pairs. Each entry holds the link direction, the PDR and the transmission rate. This information can be used by the NC algorithm to decide upon valid NC combinations by selecting the queues to activate together. Since this part of the code runs periodically, the developer is encouraged to implement any complex algorithm steps here, and thoroughly index the NC available combinations. With such an approach, the overhead of choosing the most beneficial packet combination during the incoming or outgoing packet events is minimized.

Sending data and ACK packets: The outgoing packet event checks the size of the outgoing packet queue. If this is below the defined (configurable) threshold, the functions that choose and encode combinations are called. The developer may also add logic for the handling of ACKs. By default, NCRAWL resends packets that have not been acknowledged, by directly placing them on the outgoing queue. Note that NCRAWL allows the development of algorithms that deal with the failed packets in a different way (see section 5 for an example). Since NCRAWL groups ACKs that belong to the same encoded packet, the scheduling algorithm knows which packets have been decoded successfully at which destination, and hence may further decide whether a packet should be resent directly, or potentially re-considered in encoding combinations.

System monitoring: Our framework allows for user defined timer events. Statistics for incoming/outgoing packet activity as well as queue lengths are all logged using counters. The NCRAWL logger periodically retrieves statistics and notifies the user at runtime about the flow stability and the corresponding queue lengths. The latter are also available for use in the NC algorithm if needed. Finally, the developer may also implement additional debugging support for inspecting the algorithm configuration at runtime, using Click's read handler.

4.4 Deploying NCRAWL Experiments

We have integrated NCRAWL deployment scripts with the OMF framework [24] for wireless testbeds, as an optional feature. OMF is a Control, Management and Measurement Framework that provides users with tools to easily describe, execute and collect testbed results. NCRAWL leverages the following OMF utilities (see [24] for details):

- ***Gridservices:*** This is a set of web services that are used by OMF to fetch information and perform actions remotely on the nodes.
- ***Nodehandler:*** This component resides on the central server that interacts with the user for the experiment submission. Moreover, it provides the necessary applications for node system image loading, experiment execution, image saving and node status check.
- ***Nodeagent:*** This component obtains instructions for the experiment deployment, arriving from the nodehandler.

NCRAWL extensions have been written for both nodehandler and nodeagent. The former performs transfers of the Click executable along with user defined parameters. The latter retrieves local node information (e.g. the network interface name and MAC address) and then parametrizes a generic NCRAWL deployment script. Finally, the nodehandler is notified if the deployment was successful. With OMF, NCRAWL experiments can be deployed with minimal user effort.

5 Case study

This section demonstrates by example how the NCRAWL framework allows for easy implementation of scheduling algorithms. We also discuss a case study which we use for the assessment of NCRAWL in the next section.

At each WiFi transmission opportunity, the scheduler selects a number of packets, encodes them and sends the encoded packet to the MAC layer. The problem is of scheduling nature: *which are the best packets to select considering the queue backlogs, the transmission rates and the overhearing probabilities?* Note that selecting only one packet corresponds to transmission without NC.

A well known family of optimal algorithms for scheduling is the maximum weight (maxweight) algorithms, applied in the stability theory of stochastic networks and input queues switches, see for example [25, 26]. In these algorithms, *control actions* are chosen to maximize a reward that depends on the product of link rates and queue lengths. The application of such algorithms towards solving the joint NC and scheduling problem with arbitrary rates is then promising [27].

For the case of the implicit ACKing scheme used by our framework, one observes that the actual rate of service is random. The randomness comes from the decoding; the scheduler (at the relay node) does not know *a-priori* whether packets needed for decoding are missing or not by some of the receivers. Nevertheless, a NCRAWL equipped relay knows the probabilities of overhearing and is then in position to determine the probability of decoding and thus the *expected service rate*. Let C represent a selected subset of receivers (neighbors of the relay). If the scheduling decision is C, then the reflected action is to combine the first packet from each of the queues with packets destined to nodes C, and send to the MAC layer the coded packet. The expected service rate for receiver $i \in C$ with such an action will be:

$$\mu_i(C) = \min_{j \in C}\{r_j\} \prod_{\substack{j \in C \\ j \neq i}} q_{s_j,d_i} \ , \tag{1}$$

where $\min_{j \in C}\{r_j\}$ determines the actual transmission rate with this action, and $\prod_{\substack{j \in C \\ j \neq i}} q_{s_j,d_i}$ determines the probability of decoding at receiver i. Here we have assumed that the overhearing events are independent. This is a realistic assumption since the time and space for each overhearing event is different. Collisions and Rayleigh fading may be the causes for this randomness. Below we discuss some example algorithms that can be implemented and assessed with NCRAWL.

5.1 Algorithm 1

Let Q_i be the queue backlog at the relay for flow i, and $\mathcal{C}$ the set of all possible control actions. Algorithm 1 searches set $\mathcal{C}$ for the action that maximizes the *total* expected service rate.

Algorithm 1 is throughput-optimal under the condition that the random decoding event distribution does not change during transmissions and thus, the expected service rate computed above is accurate. This happens when (i) the probabilities are 0 or 1, as in Alice-Relay-Bob topology (and any other symmetric flow

```
Algorithm 1: MaxWeight Algorithm without feedback
    Input: $Q_i, \mu_i(C)$
    Output: $C^*$
    $C^* := \arg\max_{C \in \mathcal{C}} \{\sum_{i \in C} Q_i \mu_i(C)\}$;
```

setting), or when (ii) upon a decoding failure we reschedule the uplink transmission for the failed flow. The latter may arise e.g. in a TCP scenario. In the general case, however, whenever a particular encoded packet is not correctly decoded, extra feedback information is obtained and the expected service rate changes. If, for example, $P_1 \oplus P_2$ is not decoded by both receivers, the relay knows that these two packets are not overheard by receiver 2 and 1 respectively, and they should not be coded together again. The impact of feedback clearly biases the probabilities of decoding. Therefore, in the general case, Algorithm 1 is not optimal; also, when the overhearing probabilities are quite small, it might perform quite badly.

5.2 Algorithm 2: the case of two flows

Utilizing feedback information for making scheduling decisions is a notoriously difficult problem. In [28,29] an *optimal* scheduling algorithm is discussed, which however requires a significant number of calculations, since it operates on a virtual network with $O(4^N)$ virtual queues, where N is the number of physical neighbors. Here we discuss a simple myopic scheduling algorithm that is not optimal, but is able to effectively utilize feedback and outperform Algorithm 1. For tractability we discuss the case for two flows; note that it can be extended to the case of arbitrary number of flows and pairwise coding.

In order to cope with feedback, we add two more knowledge states. Apart from *unknown* (newly arrived) packets whose behavior is captured by known probabilities, we have a state for *good* packets (successfully overheard) and one for *bad* packets (those not overheard), where the scheduler has deterministic knowledge of the decoding event. Given this, the system maintains the queues Q_i^s where $i \in \{1, 2\}$ signifies the flow and $s \in \{u, g, b\}$ signifies the state. The new set of controls $\mathcal{C}_2$ contains all controls that activate one or two queues with the constraint that no two queues from the same flow can be activated. The packets are initially injected in the queues corresponding to the unknown state. Once a packet is not decoded properly, the relay classifies it as either good or bad, based on feedback information. If it is bad, it is retransmitted without encoding. If it is deemed as good, it is transferred to the corresponding queue at the good state (Q_1^g or Q_2^g depending on the flow it belongs to). When calculating average service rates, the packets at the good state have probability of overhearing equal to one. Apart from these alterations, algorithm 2 works in the same way as algorithm 1.

```
Algorithm 2: Myopic Algorithm with feedback
Input: $Q_i^s, \mu_i^s(C)$
Output: $C^*$
At feedback time:
    • For each packet that was not correctly decoded, define
      whether it is good or bad using the decoding event of the paired packet.
    • Bad packets are directly sent to the MAC layer for
      transmission without coding.
    • Good packets are sent to the corresponding queue at the good state.
At decision time:
$C^* := \arg\max_{C \in \mathcal{C}_2} \{\sum_{i \in C} Q_i^s \mu_i^s(C)\}$;
```

5.3 Algorithm 3: fixed threshold policy

This algorithm operates only with implicit ACKs and makes decisions based on principles used in the COPE framework. In this sense, it emulates COPE in its probabilistic mode. An important difference is that, instead of calculating average service rates, Algorithm 3 (namely δ*–Fixed Threshold Policy* or δ–FTP) simply compares the overhearing probabilities $q_{i,j}$ with a fixed threshold $\delta \in [0, 1]$, and sets them to 1 if they exceed the threshold or zero otherwise. The algorithm selects at each decision instance the control that maximizes the number of transmitted packets.

5.4 Implementing the three algorithms in NCRAWL

Next, we demonstrate how to implement the three above algorithms on NCRAWL. For all three cases, we configure NCRAWL at each node to maintain one queue per flow for incoming packets.

Implementing algorithm 1: We first describe how one may organize queues in an efficient manner. Subsequently, we show how to utilize the queue information to apply NC.

• ***Organizing packet queues:*** To begin with, we dedicate one vector per control C which contains the identity of the involved queues (e.g. the flow it belongs to and/or the state) and the identities of the packets enqueued at the involved queues. The formed vectors are stored in a double linked list. Each vector is assigned a weight (see the argument of the maximization in the above algorithms); the higher this weight, the higher the preference of the encoder for using the combination. This weight is recalculated every time the backlog size of a member queue changes. The linked list is formed such that the head of the list contains always the current maximum weight. For the sake of low processing overhead, vectors are also directly indexed by their member queues; with this, the weight update process is fast. Vectors as well as their linked list are all constructed during the NCRAWL updater write event.

• ***Applying NC operations:*** Given the construction of the control list, the encoder event examines the head of the list, and further: (a) retrieves packets from their respective queues, (b) updates the vector weights (since the respective backlogs are decremented), and (c) sets the vector with the highest weighted combination as the head of the list. The latter is actually a process with slowly scaling complexity with the number of vectors-combinations, since each updated vector weight is just compared against the weight of the current head, and only takes its place if it is higher. Retrieved packets are subsequently combined using the NCRAWL *encode* library call, and the resulting encoded packet is scheduled for transmission.

Implementation considerations for algorithm 2: This algorithm is similar to algorithm 1, however it involves an additional acknowledgment scheme logic. Therefore, for each flow NCRAWL now maintains two queues: (a) one with new incoming packets, and (b) one with packets that have been successfully logged as keys by fellow nodes, but have not reached their ultimate destinations[3]. Algorithm 2 exploits the NCRAWL acknowledgment scheme facility; this process groups the packet acknowledgment tokens, which have been created for outgoing packets combined together in the same encoded packet. This information is provided by NCRAWL to the developer. Algorithm 2 directly sends packets that have not yet reached their destinations; such packets are not reconsidered for encoding. However, the algorithm considers favorable queues and "unknown" queues for the same flow separately, when forming vectors. Note that the vectors formed with this algorithm scale intrusively, compared to the simple maxweight algorithm described previously. Throughout our measurements we only consider the scenario of two flows and thus avoid the arising complexity. This issue is expected to be resolved in the future using the NCRAWL framework.

Algorithm 3 in NCRAWL: For the implementation of the third algorithm we simply need to create vectors, (i.e. controls or queue combinations) for which the decoding probability is nonzero, according to the user-defined threshold δ and the channel quality. As soon as packets are available in all queues that constitute a vector, they are combined and transmitted at once, without considering or updating the queue backlogs. This algorithm selects controls that mix the largest possible number of packets each time.

Note that as opposed to COPE [3], NCRAWL does not use any time-threshold policy towards increasing the backlog size of the incoming packet queues, before deciding to send outgoing packets. With NCRAWL, queue backlogs will increase when the relay's outgoing packet rate is smaller than the incoming packet rate. In such cases, NC proves to be a panacea for the router stability; if the NC algorithmic operations are supported by a lightweight implementation, the router capacity can be truly increased, as our measurements suggest.

[3]For example this could be due to the fact that the destination failed to decode properly a previously sent encoded packet.

6 Evaluating our Framework

In this section, we evaluate NCRAWL in conjunction with scheduling algorithms (NCRAWL + alg1, NCRAWL + alg2 and δ–FTP) described in section 5, in terms of both throughput and resource utilization. We begin by describing the wireless testbed infrastructure and the configurations that we used to deploy experiments. Next, we quantify the CPU overheard that is introduced by each NCRAWL processing stage, under maximum traffic loads, and we compare total CPU utilization to: (i) the public COPE implementation that uses an explicit acknowledgment scheme and (ii) legacy IEEE 802.11b-g. Following, we demonstrate that NCRAWL can support theoretical gains even when coding opportunities lead to more than 2-packet combinations. Finally, we deploy experiments that demonstrate how the proposed algorithms perform in cases with variable link qualities and different rates.

6.1 Experimental setup

Our testbed is comprised of 20 ORBIT nodes, deployed both indoors and outdoors, in the Department of Computer Engineering and Telecommunications, at UTH. Each node consists of one 1 GHz i386 processor, 512 MB of RAM, two Ethernet ports and two miniPCI slots which are used to host two AR5212 Atheros 802.11a/b/g WiFi cards. UDP and TCP traffic is generated using the *iperf* software tool [30]. For CPU occupancy measurements we appropriately instrument NCRAWL with the Linux *getrusage* system call, which accurately estimates CPU usage time. We place several getrusage calls at the borders of each processing stage, we record the average usage time of each stage and we compare it to the whole NCRAWL system usage time. We have repeatedly performed all of our experiments late at night, in order to avoid interference from collocated networks.

6.2 CPU occupancy measurements

In order to measure the efficiency of our framework in terms of CPU occupancy, we compare it to the case of running COPE [3], as well as the legacy IEEE 802.11 protocol.

NCRAWL is more CPU friendly than previous approaches: We invoke the Alice-Relay-Bob setting (see section 3) and we inject fully saturated traffic in both flows. We compare NCRAWL + alg1, NCRAWL + alg2, COPE and the plain 802.11, for the case of 802.11b; Fig. 5 depicts the results. Note that COPE can support at most the IEEE 802.11b rate set as discussed in section 1; for the sake of a fair comparison here, we use this mode of operation for NCRAWL as well. We observe that NCRAWL makes use of the CPU resources in a very efficient manner: it reduces the CPU utilization by at least 2 and as much

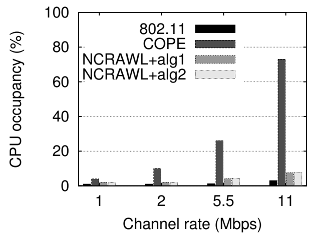

Figure 5: NCRAWL requires much less CPU processing than other network coding approaches, even at low bit rates.

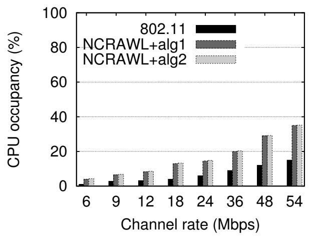

Figure 6: The CPU occupancy with NCRAWL remains at very low levels, even at high transmission rate regimes.

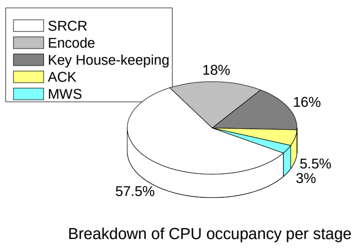

Figure 7: The CPU demands of packet encoding and key housekeeping are kept at low levels with NCRAWL.

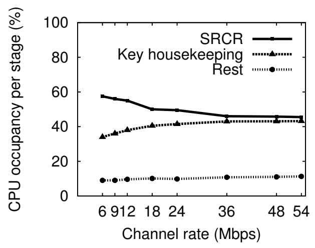

Figure 8: NCRAWL operates efficiently even at high transmission bit rates, in terms of required CPU processing support.

as 7 times compared to COPE (we have validated these observations for the case of ER [7] as well, which is based on COPE). Furthermore, we test NCRAWL for the case of 802.11g. Our measurements (Fig. 6) suggest that NCRAWL does not need to occupy more than 37% of the CPU resources for NC operations at 54 Mbps, with fully saturated UDP traffic. This implies that the design of NCRAWL includes low additional overhead functions (as opposed to legacy 802.11).

Evaluating individual operations of NCRAWL: Next, we deploy getrusage calls and measure the breakdown of CPU occupancy per processing stage (Fig. 7). The most CPU intensive operation is the SRCR stage (it contains legacy IEEE 802.11 operations as well). The most computationally heavy pieces of NCRAWL are the encode stage and the key house-keeping[4]. Note here that these two lie at the heart of any NC system and in a way represent unavoidable costs. It should also be noted that the processing stage of the scheduler remains at very low values and there is a certain percentage dedicated to dealing with ACKs.

[4]Key house-keeping refers to operations over a hash structure maintained for packet identification.

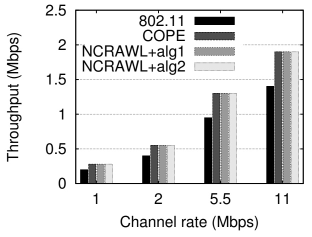

Figure 9: NCRAWL and COPE achieve the theoretically optimal throughput/flow at low bit rates in the Alice-Relay-Bob scenario.

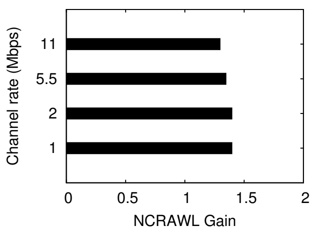

Figure 10: The gain with NCRAWL is equal to the one that is theoretically calculated at low bit rates (case with 802.11b).

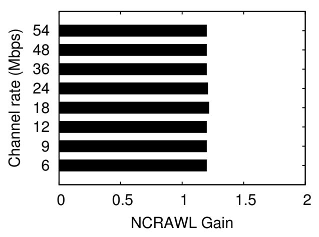

Figure 11: NCRAWL achieves high gain even at higher transmission rates in the Alice-Relay-Bob topology (case with 802.11g).

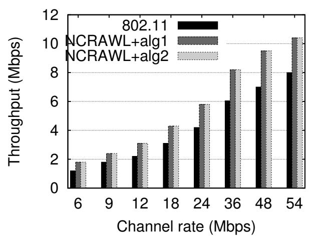

Figure 12: The throughput improvements due to NCRAWL over plain 802.11g are pronounced at all transmission rates.

Furthermore, as depicted in Fig. 8, by increasing the channel rate (and thus the number of packets into the system per unit time), the coding stage increases in complexity disproportionally with the SRCR stage. This implies that the coding complexity increases faster than SRCR as the rate increases. Nevertheless, for high channel rates the differences are reduced. This suggests that NCRAWL could potentially operate efficiently at much higher channel rates, such as with 802.11n systems. We plan to test NCRAWL on MIMO networks in our future work.

6.3 Throughput measurements with UDP

Next, we assess the ability of NCRAWL to approach the theoretically expected benefits of NC.

Experiments with the simple Alice-Relay-Bob topology: We calculate and measure the maximum throughput for both symmetric flows, such that the system remains stable (i.e. the queues do not rise more than a large permissible number). Figs. 9, 10, 11 and 12 show the results. Note that since the receivers

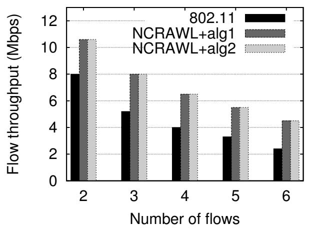

Figure 13: The per flow throughput naturally drops, as the number of flows increases, but the aggregate throughput increases.

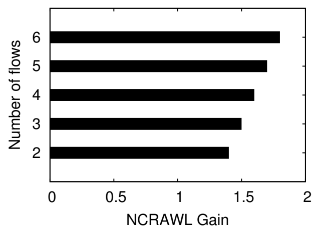

Figure 14: The gain is an increasing function of flows and approaches asymptotically the value of 2.

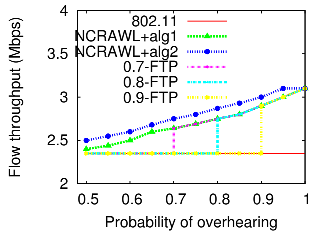

Figure 15: The combination of NCRAWL+alg2 is superior in terms of delivering the maximum throughput.

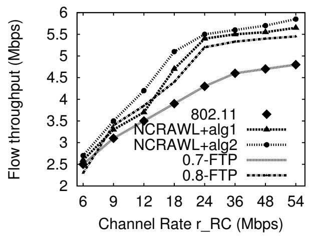

Figure 16: NCRAWL+alg2 always manages to outperform NCRAWL+alg1, since it is able to use feedback information.

always have the proper keys (these are the keys from their own transmitted packets [2]), decoding is always possible and thus algorithm 1, algorithm 2 and COPE are optimal in this setting. In each case, a gain in throughput of $\frac{4}{3}$ is identified, which matches the theoretical for this topology. Our measurements suggest that COPE achieves the theoretical throughput for small rates, but it fails to do so in higher rates. Note that the public COPE code was initially available for 802.11b only; while we carefully modified COPE to operate at 802.11g rates, we observed that such modifications lead to a very unstable system when rates higher than 18 Mbps are used. A closer look at certain individual components of the COPE implementation revealed that the reason for this instability is the excessive overhead induced by the NC system operations (as discussed earlier). For this reason we do not explicitly compare COPE here at these high rates. Nevertheless, from these measurements one can realize that COPE cannot provide benefits at rates higher than 18 Mbps, due to the tremendous CPU processing overheads that its design incurs. In contrast, NCRAWL manages to reach the theoretical gain at high channel rates (e.g. at 54 Mbps), as shown in Figs. 10 and 12.

The case for wheel topologies: Furthermore, we scale the number of flows (see Figs. 13 and 14); the topology is an $\frac{x}{2}$–wheel. The theoretical gain in this case is $\frac{2x}{x+1}$ where x is the number of flows combined at the downlink. Our measurements support the theoretically predicted gain at the channel rate of 54 Mbps. We observe the per flow throughput naturally drops, as the number of flows increases, but the aggregate throughput increases. The gain (Fig. 14) is an increasing function of flows and approaches asymptotically 2; note that this is perfectly aligned with the findings in [2] as well. Note also that in $\frac{x}{2}$–wheel topologies, piggybacking is not available since there is no return flow from the receivers. NCRAWL is able to select the appropriate ACKing method and the results show that the overhead incurred is negligible.

Experiments with cross topologies: We now present two more cases of interest that can appear in realistic environments. We setup various *cross* topologies with nodes in different locations across our testbed; we activate the flows Alice-Relay-Chloe and Bob-Relay-David. The arrivals are again chosen in a symmetric way, i.e. the arrival rate of the one flow is equal to the other.

• In the first case (Fig. 15), David overhears Alice's uplink transmissions with probability 1 and Chloe hears Bob with probability q. The rates of all links are equally set to 12Mbps (the channel rate is not important in this experiment). We measure the highest throughput that guarantees queue stability while varying the probability q, by considering different node locations. We compare NCRAWL+alg1, NCRAWL+alg2 and IEEE 802.11g as well as δ–FTP for $\delta = \{0.7, 0.8, 0.9\}$ (see section 5 for description). The results demonstrate the superiority of NCRAWL+alg2, which is able to deliver the maximum throughput in each case. Evidently, our framework in combination with the proposed scheduling algorithms is able to effectively handle the several link quality conditions.

• In the second case (Fig. 16), the overhearing probability from Bob to Chloe is set to $q = 0.7$. All channel rates are set to 24Mbps with the exception of the link Relay-Chloe which is varied. Our measurements demonstrate the inefficiency of policies oblivious to rates like the δ–FTP. In this case, the choice of a small value for δ is penalized when the Relay-Chloe link is slow enough. Instead NCRAWL+alg2 is able to handle in an effective way the several rate and link conditions and deliver important throughput gains. From figs. 15 and 16 we also observe that given that overhearing links are not perfect in terms of PDR, NCRAWL+alg2 always outperforms NCRAWL+alg1, since it is able to use feedback information.

6.4 Performance with TCP traffic

Finally, we assess the efficacy of NCRAWL in scenarios with TCP traffic. In [3], experiments with TCP have demonstrated a loss in efficiency due to packet losses and reordering. First, throughout our experiments with the Alice-Relay-Bob topology, where no losses or delays are incurred, the throughput is reduced due to the additional TCP overheads. We observe that when the 54 Mbps rate is used, the per flow throughput rate is 7 Mbps for plain 802.11 and 8.5 Mbps for NCRAWL+alg1. A slight loss in NC gain is observed; this is the result of mixing TCP ACKs with data packets. The same gain is obtained for all the other available bit rates.

Furthermore, we perform experiments with *half-cross* topologies, where flows are unidirectional (from Alice to Chloe and from Bob to Dave), with probabilities of overhearing $q_{AD} = q_{BC} = 0.7$ and several channel rates. In this case, NCRAWL+alg1 achieves a slightly lower throughput than IEEE 802.11. This is due to the fact that some packets are not correctly decoded at the destination and therefore they arrive delayed and out of order. This causes abrupt reactions from TCP and leads to throughput reduction. When adding the reordering module of COPE [3], the packets arrive always in order, however this module increases the delay for each packet. This in turn is interpreted by TCP as congestion; it ends up in TCP window increments, and thereby decreases performance. NCRAWL is not optimized to cooperate with TCP at this point and thus, it faces the common problems of TCP in wireless networks. Improving this component is the main goal of our future work.

7 Conclusions

We design and develop NCRAWL. Our framework. is an extended, generic NC framework that can be used to quickly develop networking systems in order to evaluate intersession NC and/or scheduling algorithms, entirely based on the implicit (probabilistic) acknowledgement that a packet can get decoded at the destination. The design of NCRAWL involves all the common processing steps that are always needed to implement such algorithms; these steps have been abstracted such that designers need to simply focus only on the implementation of their algorithms. Our measurements demonstrate that NCRAWL is a powerful NC development system. It offers significant throughput benefits even at high channel rates.

References

1. R. Ahlswede, N. Cai, S-Y. R.Li, and R. W. Yeung. Network information flow. In *IEEE Trans. Inform. Theory, pp. 1204-1216*, July 2000.
2. P. Chaporkar and A. Proutiere. Adaptive Network Coding and Scheduling for Maximizing Throughput in Wireless Networks. In *ACM MOBICOM*, 2007.
3. S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in The Air: Practical Wireless Network Coding. In *IEEE/ACM Trans. on Networking, Vol. 16, Iss. 3*, June 2008.
4. Kim, J., et al. A Memory Copy Reduction Scheme for Networked Multimedia Service in Linux Kernel. In *EurAsia-ICT*, 2002.

5. ANSI/IEEE 802.11-Standard. 1999 edition.
6. B. Scheuermann, W. Hu, and J. Crowcroft. Near-Optimal Co-ordinated Coding in Wireless Multihop Networks. In *ACM CONEXT*, 2007.
7. E. Rozner, A. Iyer, Y. Mehta, L. Qiu, and M. Jafry. ER: Efficient Retransmission Scheme for Wireless LANs. In *ACM CONEXT*, 2007.
8. S. Rayanchu, S. Sen, J. Wu, S. Banerjee, and S. Sengupta. Loss-Aware Network Coding for Unicast Wireless Sessions: Design, Implementation, and Performance Evaluation. In *ACM SIGMETRICS*, 2008.
9. S. Chachulski et al. Trading Structure for Randomness in Wireless Opportunistic Routing. In *ACM SIGCOMM*, 2007.
10. Click Modular Router. http://read.cs.ucla.edu/click/.
11. T.-S. Kim et al. A Framework for Joint Network Coding and Transmission Rate Control in Wireless Networks. In *IEEE INFOCOM*, 2010.
12. S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-Level Network Coding for Wireless Mesh Networks. In *ACM SIGCOMM*, 2008.
13. D. S. Lun et al. Minimum-Cost Multicast over Coded Packet Networks. In *IEEE Trans. Inform. Theory, 52(6):931-938*, 2006.
14. C. H. Liu and F. Xue. Network Coding for Two-Way Relaying: Rate Region, Sum Rate and Opportunistic Scheduling. In *IEEE ICC*, 2008.
15. H. Seferoglu and A. Markopoulou. Distributed Rate Control for Video Streaming over Wireless Networks with Intersession Network Coding. In *Packet Video*, 2009.
16. L. F. M. Vieira, A. Misra, and M. Gerla. Performance of Network Coding in Multi-Rate Wireless Environments for Multicast Applications. In *IEEE GLOBECOM*, 2007.
17. The Network Coding home Page. https://hermes.lnt.e-technik.tu-muenchen.de/DokuWiki.
18. G. S. Paschos, C. Fragkiadakis, L. Georgiadis, and L. Tassiulas. Wireless Network Coding with partial Overhearing Information. In *IEEE INFOCOM*, 2013.
19. J. Bicket et al. Architecture and Evaluation of an Unplanned 802.11b Mesh Network. In *ACM MOBICOM*, 2005.
20. MIT Roofnet. http://pdos.csail.mit.edu/roofnet.
21. SDBM hash function. http://www.partow.net/hashfunctions.
22. D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High Throughput Path Metric for MultiHop Wireless Routing. In *ACM MOBICOM*, 2003.
23. R. Draves, J. Padhye, and B.Zill. Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks. In *ACM MOBICOM*, 2004.
24. D. Raychaudhuri et al. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *IEEE WCNC*, 2005.
25. L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *Automatic Control, IEEE Transactions on*, 37(12):1936–1948, 1992.
26. N. Mckeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *Communications, IEEE Transactions on*, 47(8):1260–1267, 1999.
27. Atilla Eryilmaz. Control for inter-session network coding. In *Proc. Workshop on Network Coding, Theory and Applications*, 2007.
28. G. S. Paschos, L. Georgiadis, and L. Tassiulas. Optimal Scheduling of Pairwise XORs Under Statistical Overhearing and Feedback. In *Proc. of 7th International workshop on Resource Allocation and Cooperation in Wireless Networks–in conjunction with WiOpt 2011*, 2011.
29. G. S. Paschos, L. Georgiadis, and L. Tassiulas. Scheduling with pairwise xoring of packets under statistical overhearing information and feedback. *Queueing Syst.*, 72(3-4):361–395, 2012.
30. Iperf bandwidth measurement tool. http://iperf.sourceforge.net/.