Reliable Communications over Wireless Mesh Networks with Inter and Intra-Flow Network Coding

David Gómez[1], Eduardo Rodríguez[2], Ramón Agüero[1], Luis Muñoz[1]
University of Cantabria
Avda. De Los Castros S/N
Santander, Cantabria (Spain)
[1]{dgomez, ramon, luis}@tlmat.unican.es
[2]eduardo.rodriguezm@alumnos.unican.es

ABSTRACT

In this work we present a flexible *Network Coding* (*NC*) module integrated within the `ns-3` framework. We have exploited it to implement an *inter-flow* coding protocol, in which intermediate nodes (*routers*) combine packets belonging to different flows, as well as an *intra-flow* coding scheme, in which both the source and intermediate nodes linearly code packets of the same flow. We assess the performance of both approaches to provide reliable communication services over wireless mesh networks, considering that links are prone to cause packet drops, comparing their behavior with the one exhibited by *legacy* TCP.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Simulation, Network Coding, Mesh Networks

Keywords

Network Coding; Reliable communications; Lossy Wireless Channels

1. INTRODUCTION

Since the arrival of the *21st* century, wireless technologies have seen a continuous growth in every aspect: wholesales, users, manufacturers, standardization, etc. This success has promoted their constant evolution, and they stand at the time of writing as the most widespread access alternative. However, there is still a big challenge that has not been properly tackled: while it is clear that the present and future of communications are tightly related to wireless networking, the mainstream transport protocol nowadays, *TCP*, used for countless applications (for instance, file transfers, web browsing, etc.), is severely jeopardized when used over this type of networks.

One alternative to overcome the aforementioned limitation is the use of the *Network Coding (NC)* paradigm. It is a relatively new technique (originally proposed in 2000) that basically proposes to move away from the legacy routing procedures, which rely on the well-known *store-and-forward* paradigm, where intermediate relay nodes do not incorporate any additional processing, but they just forward the incoming packets based on the information of their routing tables. Opposed to this, *NC* follows a different approach, where routers, taking an essential role, are able to modify the content of the packets across the network, by means of different coding schemes. Using this new approach, a broad range of possibilities looms: it may reduce the number of transmissions, thus leading to energy saving; it might bring about a more reliable service; it also can be used to incorporate additional security mechanisms (the information is somehow encrypted); it finally can be used to improve the performance, among other features.

In this work we study the possibilities that are brought about by *NC* to boost the performance for reliable communications (traditionally using TCP), in particular over Wireless Mesh Networks (WMNs). For that we follow two different approaches, both of them based on a common *NC* framework that we integrated within the `ns-3` simulator [3] (namely, `ns-3.13`): first, we have implemented a solution based on the deterministic encoding of packets belonging to *different* TCP connections at the intermediate routers along the network (*Inter-flow NC*); on the other hand, we also exploit the combination of *UDP* and a *Random Linear Coding (RLC)* scheme that "mixes" datagrams belonging to the *same* UDP flow to promote a novel reliable communication service. We assess the performance of these mechanisms over two different canonical topologies and compare their behavior to the one achieved by the legacy TCP.

This document is structured as follows: Section 2 positions this work within other contributions that tackle the same topics. Sections 3 and 4 focus on the description of the *Intra* and *Inter*-flow *NC* schemes, respectively. Section 5 depicts the implementation of the *NC* layer within the `ns-3` framework, as a module placed between the network and transport layers. Section 6 depicts the simulation testbed used to assess the performance of the presented *NC* mechanisms, discussing the achieved results. Finally, Section 7 concludes the document and advocate a number of items that will be tackled in our future research.

2. RELATED WORK

The term "*Network Coding*" was originally coined by *Ahlswede et al.* in [6]. They debated the suitability of the classic *store-and-forward* paradigm in IP networks, suggesting that the integration of additional functionalities at intermediate nodes might lead to significant performance enhancements. From that moment, several works have proposed the use of these mechanisms, to get either performance improvements or more reliable communications. Furthermore, after several years of research, the scientific community accepts the division of *NC* techniques into two main groups; the first approach is based on the combination of packets belonging to different flows at intermediate routers, and some of its most popular examples are *COPE* [13] and *Coding Applied To Wireless On Mobile Ad-Hoc Networks (CATWOMAN)* [12]. The second group fosters the use of random linear combinations of packets belonging to the same flow; the reader might refer to [7] and [17] for a succinct description of this approach.

Besides, there is a new trend, aimed at merging the two aforementioned paradigms into a unique *NC* operation, so as to benefit from the advantages of each of them, as proposed by I^2NC [16] and *CORE* [14]; they show that this approach outperforms a standalone (either *inter-flow* -first group- or *intra-flow* -second group-) *NC* scheme.

However, there are not so many works addressing the use of *NC* to improve the poor performance shown by *TCP* over lossy wireless networks, since they usually do not pay attention to the transport level, focusing on the behavior of the *NC* mechanisms by themselves. The few works that have dealt with such analysis (see e.g. [13]) agree that the interplay between the *NC* schemes and the congestion control algorithms used by TCP is rather harmful.

On the other hand, the popularity that the `ns-3` simulator has recently gained, fosters the spring of different works that study the interaction between *NC* and *WMNs* using this particular platform. For instance, *Yang Chi et al.* proposed *Yet Another Network Coding Implementation (YANCI)* [5], a simplified version of the *COPE* [13] *inter-flow NC*. However, the authors did not consider the presence of packet erasures within the network. In addition, it is also worth highlighting the existence of different independent open-source frameworks that allow the possibility of linking `ns-3` as an external library to simulate the use of *NC* over different network environments, such as *NECO* [8] or Kodo [15].

Regarding our previous work, we introduced in [10] an *Inter-flow NC* protocol, which we integrated within the `ns-2` simulator, plugging the *NC* layer between the *Medium Access Control (MAC)* and *IP* levels; we thoroughly analyzed the impact of the operational parameters of intermediate coding nodes (buffer size and timeout), focusing as well on the synchronization issues between TCP flows that appeared when we introduced packet losses within the various wireless links. In [9] we presented the *NC* implementation over `ns-3`, which we discuss in this paper. Unlike in [10], we have placed the *NC* layer between the network and transport levels, addressing a direct interaction with the upper-layer protocols (TCP or UDP). This *NC* entity exploits the framework to encapsulate TCP acknowledgement traffic together with the coded data flow. However, those management segments are actually *off-coding*, since if there arose any encapsulation probability at coding nodes, the ACK would just be appended at the end of the *NC* header. Hence, these ACKs would be extracted at their corresponding destinations. Through this naive solution, we achieved an throughput improvement of 40%, compared to TCP over simple topologies, as the one which will be shown in Section 6.

3. INTER-FLOW NETWORK CODING

Based on a simplified version of *COPE* [13], our proposed *Inter-flow NC* protocol aims at improving the performance of *TCP* over wireless mesh topologies by combining the various flows that are present along the network. Besides, we can find a deeper description about the implementation and results in our previous work, i.e. [9, 10].

The cornerstone of this protocol lies on the intermediate nodes, which keep track of all TCP segments to be forwarded[1]. They use a coding buffer (named *input packet pool*) with a maximum capacity of $N \geq 1$ segments, whose size is mapped through the parameter BS. A *coding buffer timeout* (B_{TO}) is associated to each segment in the buffer, so as to limit the sojourn time of a packet in such buffer (otherwise, coding could add an unacceptable long latency for some particular arrival patterns). This buffer will group the stored packets according to their so-called *Flow ID*, which consists on a 16-bit hash of the tuple *Source IP address-Source TCP port-Destination IP address-Destination TCP port*, obtained with the popular `MD5` hashing function, using the *Open SSL Library* [4]. All the required information is serialized into a proprietary header (i.e. global information, individual info about the native packets that are encoded within the packet, etc.), whose detailed description can be found in [9].

Whenever an intermediate node receives a new segment, it checks whether another one, belonging to a different flow (i.e. having a different *Flow ID*), is already stored in its buffer. In such a case, both packets are coded together and then delivered to the lower layer. Otherwise, the segment is stored in the *input packet pool*, waiting for an eventual coding opportunity. If no new coding opportunity happens within the B_{TO} interval, the corresponding original segment will be transmitted. Given that the coding buffer has a finite size, if a new packet arrives and the buffer is full, the oldest packet will be sent (without being coded) to the MAC layer and deleted from the buffer, so as to keep the most recent one.

It is worth mentioning the relevance of appropriately selecting the nodes that will take the role of coding entities. A good choice would certainly improve the efficiency of the coding combinations, increasing the number of packets that are encoded. This would also help to reduce the failure rate at the destination node, being the event in which a coded packet is received, but the native segment cannot be retrieved, leading to a packet drop. This aspect has a key relevance for the performance of the *Inter-flow NC* scheme, but it is out of the scope of this work.

This particular approach to *Inter-flow NC* is based on the capacity of nodes to overhear packets that are being trans-

[1]For the sake of simplicity, only segments with user information would be considered to be coded under this *NC* scheme.

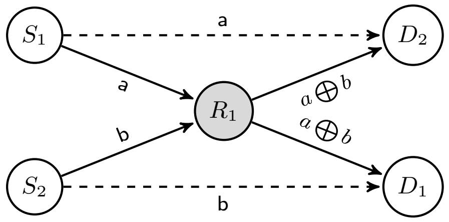


Figure 1: *Inter-flow NC* canonical scenario

mitted within their coverage area (this procedure was named in [13] as *opportunistic listening*), and therefore it does not require any further *NC* management scheme. When a node overhears a packet, it stores it in its *Decoding buffer*, even if it is not the *real* destination of such packet, considering that such segment might be eventually useful afterwards, for decoding purposes. In general, to recover the i^{th} original packet from a coded packet of m segments ($m \geq 2$), the destination needs to *have* the other $m-1$ original segments. In the work that is being presented herewith, as we are considering just two *TCP* flows, the intermediate node can encode only two original segments to generate a coded packet. As a consequence, when a destination node receives a coded packet, it can decode it only and only if its decoding buffer holds the other original segment contained in the received packet. A *XOR* operation between the coded packet and one of the two original segments allows extracting the remaining original segment.

As an illustrative example, Figure 1 represents the inherent behavior of the *Inter-flow NC* protocol over the widespread X topology. In a nutshell, two TCP connections will be established: one between S_1 and D_1 and the other one between S_2 and D_2. The distance between nodes is chosen so that D_i is out of the range of S_i, but within the range of S_j, with $i, j \in [1, 2]$ and $i \neq j$. This configuration allows D_i to overhear the packets coming from S_j to D_j for decoding purposes. In this very particular configuration, node R_1 clearly stands up as the coding router, since it is the clear articulation point of the network, and all the traffic would traverse it (four different flows in this case: two per TCP connection, one for the data segments and another one for the corresponding backwards TCP acknowledgements). Hence, R_1 will be the node in charge of combining the information belonging to the two data flows (recall that in this work we only consider the encoding of data segments).

4. INTRA-FLOW NETWORK CODING

The approach is in this case rather different from the previous one. The baseline is still the legacy *TCP*, but just because we want to have a reliable communication, since we are using *UDP* and carrying out random linear combinations over the corresponding datagrams before their delivery, taking as a reference the work carried out by *Chachulski et al.* in [7]. It is worth mentioning that other existing proposals do not properly address the interaction with the transport level layer [7, 14] or combine the coding scheme with *TCP*, thus not avoiding the problems that it shows over lossy wireless channels [17].

Unlike the *Inter-flow NC* scheme, source nodes have an essential functionality, since they are in charge of carrying out the first coding process. The *NC* entity at the source node receives the information from the upper *UDP* layer and stores the datagrams at its *transmission buffer.* As was discussed for the *Inter-flow NC* protocol, packets will be grouped in different *sub-buffers*, according to the flow they belong to (those sharing the same *Flow ID* will be stored together). When K *native* packets of the same flow have been stored, a random linear combination of them is created (from now on, the combinations carried out over the same group of native packets will be referred to as "block"), and a coded packet $p' = \sum_{i=0}^{K-1} c_i \cdot p_i$ is created, where the $c_i's$ are random coefficients generated from a finite field $GF(Q) = GF(2^q)$ and the p_is are the so-called native packets. These random coefficients can also be represented as $\overrightarrow{c} = (c_0, c_1, \cdots, c_{K-1})$, where $\overrightarrow{c}$ is the packet's *code vector.* The *NC* entity will periodically send coded packets[2], until the destination node confirms the successful decoding and reception of the current block. In order not to overflow the lower layer buffers (at the *MAC* level), the *NC* dynamically injects packets downwards according to the physical output rate, that is likely to be the actual bottleneck in a transmission; by means of a *cross-layer* technique, the *NC* entity (upper layer) is able to be aware whenever a packet of a particular flow is sent to the physical channel, and can then proceed to deliver downwards a new one, when there are no packets waiting at the *MAC* level buffer.

On the other hand, destination nodes employ two different storage entities: first, a matrix C ($K \times K$) that keeps the coded vectors (rows of such matrix), and a *reception buffer* able to keep up to K coded packets. Upon the arrival of a arbitrary coded packet p_i', its coefficient vector $\overrightarrow{c}$ will be added to the j^{th} row of the C matrix. If this last received vector ($\overrightarrow{c}$) is linearly independent from the previous ones (we can assert this using a rank check operation), the corresponding coded packet will be considered as *innovative.* In such case the coded vector will be kept at C, and the coded packet will be appended to the *reception buffer*, at the j^{th} position. On the other hand, if the coded vector is linearly dependent, it will be deleted from C and the packet will be discarded. As can be seen, the value (information-wise) of every innovative packet will be the same ($\frac{1}{K}$). However, each of them is meaningless in its own, since the receiver needs to store K innovative packets to decode the corresponding *block.* This means that when a coded packet gets lost, it is indirectly replaced by the following (and innovative) one.

Upon the reception of the K^{th} innovative packet, the coefficient matrix C is of rank K, and therefore we can calculate its inverse C^{-1} to decode the coded packets (in matrix notation, $P = C^{-1} \cdot P'$), recovering the original information. Afterwards, the K *native* packets belonging to the same *block* are simultaneously delivered to the upper layer, which receives the data in the appropriate order. It is worth mentioning that this operation will not introduce any delay (in terms of `ns-3`'s simulation time), since there do not exist any scheduled events in the corresponding forwarding process. All the datagrams will therefore reach the *UDP* entity at exactly the same time; likewise, the application at the receiver node gets the K packets altogether, after the *NC* entity has decoded the corresponding *block.* We can define the *application latency* as the time interval between the successful decoding of two consecutive *blocks.* There is a

[2]In order to avoid useless transmissions, the *null* vector is discarded and never transmitted.

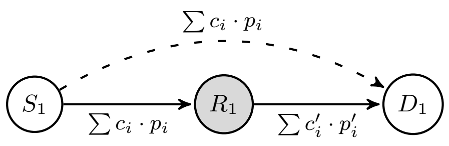


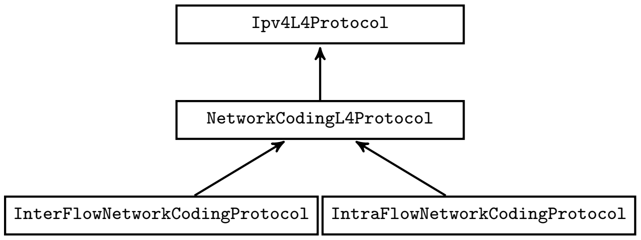
Figure 2: *Intra-flow NC* canonical scenario (RLNC)

trade-off between the size of the *block* (K) and the system performance, since higher K values lead to longer latencies (at least K transmissions are needed to decode the *block*). On the other hand, the amount of information delivered to the upper layers after each successful decoding event is proportional to K. These particular characteristics might prevent the use of this approach for highly sensitive real time traffic.

After the matrix inversion, and once the data is delivered to the upper layer, the receiver node generates an *ACK* message confirming the successful *block* reception and decoding. Upon receiving the *ACK*, the transmitter deletes the packets belonging to that *block* from the *transmission buffer* and moves forward, starting the transmission of the next one. Besides, the *NC* layer sends a signal to the MAC level to remove all the frames belonging to that particular *Flow ID*, which might be still queued at the IEEE 802.11 *MAC* transmission buffer (a *cross layer* solution to avoid the transmission of useless frames).

It is worth highlighting that all the *GF*-related tasks (i.e. vector multiplication, matrix rank and inverse calculation, etc.) are carried out by means of two external libraries: whilst the generic $GF(2^q)$ calculations use the *FFLAS*-FFPACK library [1], which works with slow *modulo* methods, the simpler finite field $GF(2)$ operations use the *IT++* library [2], whose `GF2Mat` class is much faster. Its main drawback is that is does not allow working with extended finite fields.

Along the paths, intermediate nodes can take two different roles: in the naive solution, they use the classic store-and-forward scheme, in which all the coding tasks stay within the source nodes. From now on, this solution will be referred to as *Random Linear Source Coding (RLSC)*. On the other hand, we can exploit the *NC* concept, where relay nodes are able to *re-code* the previously stored packets, thus coding the information (belonging to the same flow) as it traverses the network. In this case, this technique will be known as *Random Linear Network Coding (RLNC)*. They use a scheme akin to the one employed by the source coding process, a relay node stores all the coded and innovative packets (a rank calculation is used to discard linearly dependent coefficient vectors) at another temporary buffer (we still have one different buffer per flow), whose size, in this case, will never be greater than K. When this buffer has stored a certain number of coded packets (in this work we have used a fixed value of 2), a new coded packet $p'' = \sum_{i=0}^{T-1} c_i' \cdot p_i'$ will be generated and delivered to the lower layer, being T the number of stored packets at a particular relay node, $T \geq 2$. This new combination, as shown in [7], can be represented as a linear combination of the original *native* packets. Last, the aforementioned *cross-layer* mechanisms are also used by the intermediate nodes to dynamically re-inject the packets to be forwarded and, after receiving an ACK (which is also processed by those intermediate nodes that carry out the *RLNC* scheme), selectively flush the packets (those which belong to the acknowledged *block*) from the lower level buffers.



Ipv4L4Protocol
NetworkCodingL4Protocol
InterFlowNetworkCodingProtocol
IntraFlowNetworkCodingProtocol

Figure 3: Network Coding class inheritance diagram

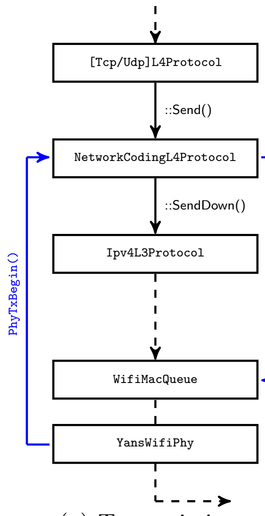
An illustrative example of both operations is depicted in Figure 2, where a source node S_1 sends application data to a receiver D_1, while a relay node R_1 forwards (or re-codes and forwards) the packets, since they cannot (from a routing protocol point of view) reach their destination by means of a single hop. However, due to the intrinsic broadcast nature of the wireless medium, D_1 might overhear a packet p_i' directly transmitted from S_1, likely with a low probability, since the distance between these nodes is long. In these cases, if R_1 just forwards p_i to D_1, this would need to discard it, since it bears the same information than p_i'. On the other hand, if R_1 combines the coded packets it had previously stored, producing a new packet p_i'', there is a higher probability for it to be innovative at the receiver entity.

In order to exchange the information required to perform the coding and decoding operations, the proposed protocol needs to include a proprietary header. It is composed by two parts: the first one, with a fixed *9-byte* length, contains all the information that must be shared between the source and destination nodes (i.e. the type of message, the number of combined packets[3], K, the *block* number and the corresponding UDP ports). On the other hand, the second part of the header is of variable length, proportional to the coefficient vector, and bounded by the K parameter and the size of the finite field $GF(Q)$, being $q = \log_2 Q$ the number of bits required for each coefficient c_i. Therefore, the whole header size will be $9 + \lceil \frac{K \cdot log_2 Q}{8} \rceil$ bytes.

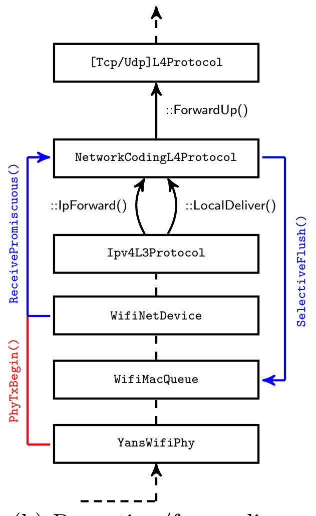
5. NETWORK CODING ON NS-3

One of the most challenging issues in this paper is the design of a common framework for the two different *NC* strategies that are addressed. To appropriately tackle that, we have designed and implemented an *NC* entity that lies between the network and transport layers. This module will be in charge of the coding and decoding operations. In terms of implementation, Figure 3 depicts the class inheritance diagram that defines the location of the protocol within the source tree. We can see a base (abstract) class, `NetworkCodingL4Protocol`, which holds all the methods that belong to the two derived classes, i.e. callbacks, essential function members, a pointer to the node that will aggregate the protocol, etc. Although the studied solutions do not have a clear placement within the stack (they might lie anywhere between 3-4 layers), they might be referred to, `ns-3` terminology, as transport level protocols, since they (indi-

[3]The field size of this parameter is 1 byte, since the implemented solution supports linear combinations of $[2, 255]$ packets.



(a) Transmission



(b) Reception/forwarding

Figure 4: Transmission and reception/forwarding flow charts of an arbitrary packet through the ns-3 protocol stack

rectly) depend on the mainstream *L4* abstract base class: `Ipv4L4Protocol`.

In the scope of this work, coding operations can be performed by different entities: whilst in the *Inter-flow NC* scheme only the defined *coding* relay nodes would actually perform the combination of packets across the network, the *Intra-flow* alternative puts the main coding functionality on source nodes, which will be the responsible of storing and coding all the segments stored at its buffer. Besides, if the *RLNC* mode is enabled, the intermediate nodes will act as "re-coding" entities, since they will combine the received packets (by means of another buffer which temporary stores them), creating new random linear combinations. As a consequence, the probability of receiving innovative packets might be increased. On the other hand, all decoding tasks are carried out by the receiver nodes (they are the actual destination, and thus they need to retrieve the original information) in both solutions.

Figure 4 provides a more thorough description of the protocols; we can see the transmission (Figure 4a) and reception (Figure 4b) flow charts of an arbitrary packet across the `ns-3`'s protocol stack (we intentionally did not include those levels that do not have any interaction with the *NC* mechanisms). Both schemes are briefly described below.

In transmission, the `NetworkCodingL4Protocol` instance intercepts the downstream *UDP*/*TCP* flow, originally headed to the `Ipv4L3Protocol::Receive()` function. The *NC* entity handles the packet and decides[4] whether it will be encoded before being forwarded downwards, or otherwise it will silently pass through the module. The rest of the flow remains unchanged, until the outgoing packet leaves the transmitter node. It is worth mentioning a particular event, which only applies to the *Intra-flow NC* scheme: we have added a *cross-layer* interaction between the *NC* entity and the physical level transmission (we use a hook to the `YansWifiPhy::PhyTxBegin` trace source), allowing us to dynamically inject (coded) data traffic from the *Intra-flow* *NC*'s transmission buffer only when the `WifiMacQueue` does not have any more packets to be sent.

[4]These decision mechanisms depend on the particular *NC* protocol currently in use.

As for the reception process, the operation becomes more complicated, since the *NC* entity captures the packets from a number of points:

1. The default reception keeps its traditional operation until the network level, where the `Ipv4L3Protocol::LocalDeliver()` function will be hooked to the *NC* stack (we modified the callback that legacy connects it to the `Ipv4L4Protocol`), which will decode (if needed) the received information, delivering it to the upper layer.

2. In a standard reception scheme, a packet would not reach the transport layer unless its IP header destination address belongs to the particular node that is parsing it. In such a case, it is the routing protocol (i.e. derived from the base class `Ipv4RoutingProtocol`) the one that takes the decision of whether forwarding or dropping the packet. When *NC* is active, intermediate nodes have a key role, since they need to process the information within those packets for coding/re-coding. This led us to tamper the legacy `Ipv4L3Protocol::IpForward()` function source code, in order to allow the *NC* layer to decide what to do with the *NC*-protocol packets, either storing/coding them or forwarding them downwards.

3. Due to the broadcast nature of the wireless medium, a node might overhear packets that are not directly addressed to it. However, these receptions might be useful for decoding procedures. Hence, intermediate nodes shall enable a *promiscuous* reception mode that allows the *NC* layer to get these packets from the `WifiNetDevice` entity.

Besides, as described in Section 4, the *Intra-flow NC* protocol needs two additional connections to the lower layer:

4. As mentioned earlier, intermediate nodes' *NC* level take care of forwarding packets (previously stored and handled at the *NC*'s buffer, see step **2** above). The *NC* entity is notified whenever a packet is about to be sent (via `YansWifiPhy::PhyTxBegin` trace source connection), and therefore it can control the injection rate at the *NC* layer.

5. In this case, once the transmitter entity (source or relay node) receives an ACK from the destination entity, it removes all the useless packets that it has previously stored from the *NC* 's buffer and shifts to the following new *block*. Besides, it would be also interesting being able to flush every "deprecated" packet that might be still stored at `WifiMacQueue`. For this reason, we have included a new member function (i.e. `WifiMacQueue::SelectiveFlush()`), which recursively removes all packets corresponding to a particular tuple *Flow ID-Old block number*.

6. SIMULATION AND RESULTS

After the description of the protocols' functionality and how they have been integrated within the `ns-3` framework, we discuss in this section a number of representative results

Table 1: Common testbed parameters

Feature	**Value**
Physical link	IEEE 802.11b (11 Mbps)
Error model	`RateErrorModel` (modified)
FER values	[0: 0.1 : 0.8]
RTS/CTS	Disabled
IEEE 802.11 RTX	3
Transport level	UDP / TCP "New Reno"
Aplication	`OnOffApplication` (20 MB)
App. data rate	CBR (11 Mbps)
Packet length	Max size allowed (MTU 1500B)
Traffic	Unicast
Simulations	50 independent runs/point

that have been obtained after a simulation campaign, using the modules described earlier.

Before showing the results, Table 1 summarizes the most relevant parameters of the simulation setup that is used for both testbeds. In a nutshell, the IEEE 802.11b recommendation (through the `ns3::YansWifiPhy` default model provided by the simulator) is used to configure the physical and *MAC* levels, using a modified `ns3::RateErrorModel` to establish the *Frame Error Rate (FER)* between links[5]. Regarding the application traffic at the source nodes, a *Constant Bit Rate (CBR)* (using the `ns3::OnOffApplication`) flow is sent downwards, ensuring that there will always be at least a packet waiting to be transmitted at the lower layer buffers (i.e. `ns3::WifiMacQueue`), leading to a saturated scenario, where the system bottleneck can be placed at the wireless channel.

Last, but not least, we show two different performance metrics to illustrate the behavior of the *NC* protocol:

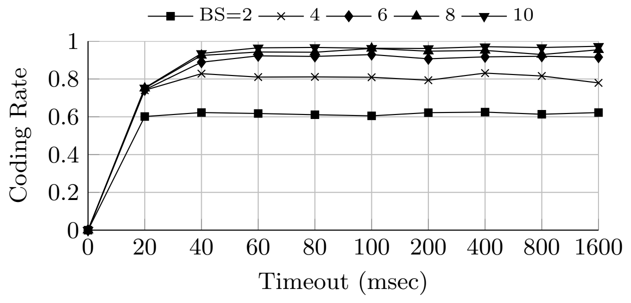
- *Throughput.* This reflects the performance from the receiver's point of view. We define it as the total number of information bytes *correctly* received at the destination node's application layer, divided by the transmission time. The value that is represented in the figures correspond to the average throughput per flow.
- *Coding Rate* (*Inter-Flow NC* exclusive). It represents the ratio between the number of coded packets transmitted over the total number of packets sent by the encoding node R_1. This parameter reflects the number of coding opportunities that were actually taken by R_1 to encode packets.

6.1 Inter-flow NC performance

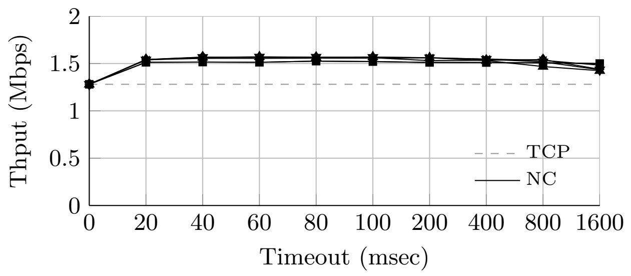
First, we assessed the capability of intermediate nodes to "mix" the information belonging to different flows. For that purpose, we deployed an *X* topology (Figure 1), which was described as the canonical scenario for an *Inter-Flow NC* communication (see Section 3).

Figure 5 shows the performance that was obtained over a lossless scenario, in which there is not any packet loss due to the propagation impairments, being the collision between transmitting stations events the only error cause. In order to study the impact of the *NC's* buffer operational parameters over the overall system, we modified both the *BS* and B_{TO} values, so as to establish the best configuration. As can be seen in Figure 5a, as long as we increase the buffer capacity, the number of coding opportunities gets greater, ranging

[5]This feature allows us to arbitrarily modify the *FER* between each pair of nodes.



(a) Coding rate vs Coding buffer timeout



(b) Throughput vs Coding buffer timeout

Figure 5: Inter flow network coding behavior as a function of the coding buffer setup

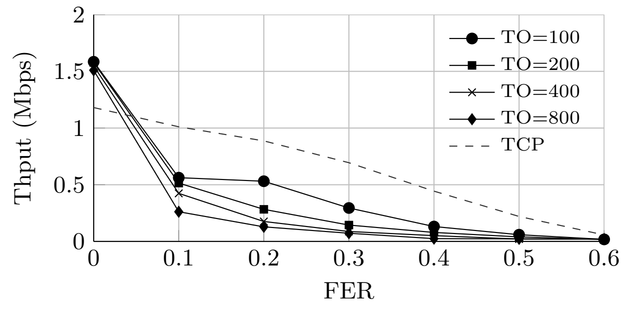


Figure 6: Throughput evolution as a function of FER on an "X" topology (Inter-flow NC)

from values of ≈ 60% (i.e. $BS = 2$ and $B_{TO} = 20$ *msec*) to ≈ 97% (with $BS = 8$ and $B_{TO} = 1600$ *msec*); in the latter configuration, most of the data packets delivered from R_1 simultaneously carry information belonging to the two *TCP* connections, thus saving a large number of transmissions. Regarding the achieved throughput, we can see in Figure 5b that the use of this *NC* scheme yields a remarkable performance enhancement (≈ 22%). However, there is a point from which both the *Coding Rate* and the *throughput* do not increase any more. We can therefore conclude that there is a particular configuration that leads to the best performance, and there is not any additional gain if we further increase the buffer size and timeout interval. We can even see that there is a small throughput reduction for the largest buffer timeout values (i.e. $B_{TO} \geq 800$).

Once we have assessed that encoding within the network can actually improve the performance of TCP over *WMNs* characterized by ideal (without errors) links, Figure 6 shows the throughput that was obtained over lossy links (in this

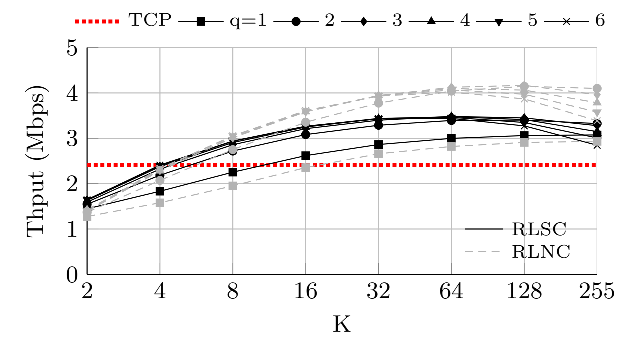


Figure 7: *Intra-flow NC* performance over a canonical three-node line topology. Throughtput (K,Q)

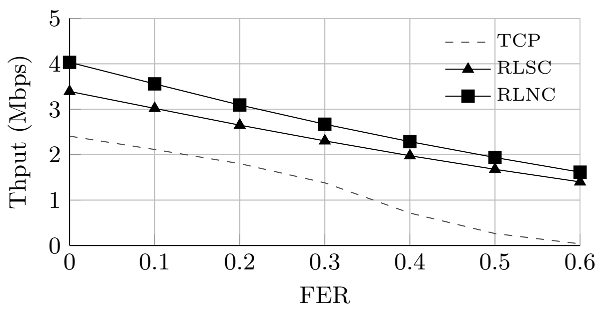


Figure 8: Throughput evolution as a function of FER on a three-node line topology (Intra-flow NC, with $K = 64$ and $Q = 3$)

particular configuration, all links are configured with the same *FER* value). The results are not very "promising", since the performance of the *NC* scheme is heavily jeopardized by the lossy channels, due to the packet losses. In fact, the throughput that was obtained for the legacy TCP was even higher. Besides, it can also be observed that a higher B_{TO} leads to a lower throughput, probably due to the fact that segments stay for too long at the coding buffer's *input packet pool*, and this might trigger *TCP* retransmissions, due to the *RTO* expiration.

We can therefore conclude that the *Inter-flow NC* scheme shows a good potential to improve TCP performance over wireless mesh networks, but only if the quality of the corresponding links is high. When the conditions of the links become worse and the loss rate starts to be relevant, the performance of the proposed scheme is heavily jeopardized, making it unsuitable for such type of scenarios.

6.2 Intra-flow NC performance

The second simulation study corresponds to a three-node topology, as depicted in Figure 2, which was selected to illustrate the performance brought about by the combination of *UDP* and an *Intra*-flow *NC* scheme. In this very particular setup, opposed to the previous case, we only use one UDP flow ($S_1 \rightarrow D_1$). The main objective of this study was to compare the performance of this combination to the one exhibited by a legacy TCP connection, since it aims to provide a reliable communication service.

The first group of results corresponds to an ideal scenario, where the *FER* over the existing wireless links can be considered as negligible (straight line in Figure 2), and the non-direct link (dashed) has a $FER = 0.6$; this link is the one over which the destination node *overhears* the transmitted packets. Figure 7 shows the throughput as a function of K and the size of the finite field $GF(Q)$, where $Q = 2^q$, for the two different relaying strategies, showing as well the performance obtained by the legacy TCP (the TCP connection uses the two-hop route, since the quality direct link is too low to properly bear it), that is constant for any value of K and $GF(Q)$. As expected, a greater Q yields a better behavior when the size of the coding vector $\overrightarrow{c}$ is small, since the number of random combinations that are non-linear gets lower as Q increases. However, there is a point from which the overhead introduced by high Q*s* over large *blocks* (i.e. $q > 4$ and $K > 64$) actually jeopardizes the transmission performance, since the improvement brought about by the transmission of a high rate of linearly independent vectors, is overcome by the amount of useful information that can be carried within each coded packet (for instance, a coded packet with $K = 255$ and $q = 6$ would require to a 201 bytes header). Besides, as hinted in Section 4, we can observe a much better performance with a *RLNC* scheme, since through *RLSC*, all the packets that have been previously overheard over the direct link $S_1 \rightarrow D_1$ would be dropped at the receiver node, since they have the same vector $\overrightarrow{c}$. This enhancement provided by the fact that the intermediate node *re-codes* the packets is $\approx 20\%$ in the best case ($K = 64$, $q = 3$).

It is also worth highlighting that TCP provides a higher throughput than the proposed *Intra-flow* solution when low-efficiency configurations are used (i.e. $K < 16$); in these cases the penalty factors associated to the packets discarded due to linear combinations and the backward *ACK* transmission have a harmful effect over the overall performance; on the other hand, for the rest of the combinations the results show a clear improvement of the TCP performance, which is $\approx 73\%$ with $K = 64$ and $q = 3$.

Finally, we can see in Figure 8 how the throughput changes as the *FER* gets higher[6], showing that the performance achieved by the two *Intra-flow NC* schemes[7] clearly outperform the one exhibited by the legacy TCP, as was also the case over the ideal channels; the results also yield that the encoding process at the intermediate routers leads to a notable throughput enhancement. The achieved gain gets even higher when the channel is worse (i.e. $FER \geq 0.4$), situations that bring about a rather poor TCP behavior.

7. CONCLUSIONS AND FUTURE WORK

In this work we have presented two different *NC* schemes, discussing their integration within a single module that we implemented within the `ns-3` framework. Their main goal is to provide a reliable transport solution for *WMNs* to complement (or even substitute) *TCP*, whose behavior is well-known to be rather poor over such networks.

By means of a thorough simulation campaign, we have assessed the performance offered by both solutions over two canonical scenarios, comparing the achieved results with the

[6]Only in links $S_1 \rightarrow R_1$ and $R_1 \rightarrow D_1$, remaining the other one ($S_1 \rightarrow D_1$) with a constant $FER = 0.6$.

[7]These measurements have been carried out using the configuration which led to the best performance over the previous scenario, i.e. $K = 64$ and $q = 3$.

behavior of a traditional *TCP* transmission. On the first hand, the *Intra-flow NC* protocol is able to yield a substantial performance gain by exploiting the combination of *UDP* and a linear coding procedure, compared with the traditional TCP performance. We have shown the enhancement ($\approx 73\%$, compared to *TCP*) that is brought about by allowing intermediate routers to *re-code* the packets. On the other hand, the *Inter-flow* scheme, based on the combination of TCP segments belonging to different flows at particular intermediate nodes, provided a certain enhancement of the overall throughput ($\approx 22\%$) over ideal wireless links (without errors). However, for worse conditions (wireless links prone to cause packet losses), the interaction between *TCP* and the *NC* scheme shows a rather poor response, leading to a notable performance decrease.

By exploiting the framework we have implemented, there are a number of aspects that we will tackle in our future research; some of the most relevant ones are briefly introduced below.

- By adding new features to the *Inter-Flow NC* scheme, we would aim to improve its performance over packet erasure channels. One particular mechanism, the encapsulation of backwards *TCP* acknowledgements within the data flow has already been added in [9]. We are currently implementing a proprietary retransmission scheme that works at the *NC* layer, and helps to reduce the long idle times caused by the legacy *TCP* congestion and control mechanisms.
- With regards to the *Intra-flow NC* scheme, a first step to be taken is to assess its performance over more complex (i.e. random) topologies and to reduce the mathematical complexity of the decoding processes.
- Another sensible step to be taken is to leverage to combined use of both schemes by means of a single hybrid solution that exploits the advantages inherent to each one. The fact that both protocols share the same `ns-3` framework would certainly help to tackle this.

Finally, it is also worth highlighting that all the *NC* protocols' source code is completely open-source and can be found in [11]. We strongly encourage other interested researchers to get the code and use it, since this would certainly help us to enhance it. We would also welcome people interested in joining this effort with contributions along some of the aforementioned open issues.

8. ACKNOWLEDGEMENTS

The authors would like to express their gratitude to the Spanish government for its funding in the project "Connectivity as a Service: Access for the Internet of the Future", COSAIF (TEC2012-38574-C02-01).

9. REFERENCES

[1] FFLAS-FFPACK. Finite Field Linear Algebra subroutines package. http://www-ljk.imag.fr/membres/Jean-Guillaume.Dumas/FFLAS/index.html.

[2] IT++ Mathematical library. http://itpp.sourceforge.net/.

[3] The ns-3 network simulator. http://www.nsnam.org/.

[4] Open SSL. The open source toolkit for SSL/TLS. http://www.openssl.org/.

[5] ns3-YANCI. Yet Another Network Coding Implementation. https://github.com/yangchi/ns3-yanci, 2012.

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

[7] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 169–180, New York, NY, USA, 2007. ACM.

[8] D. Ferreira, L. Lima, and J. Barros. NECO: NEtwork COding simulator. In O. Dalle, G. A. Wainer, L. F. Perrone, and G. Stea, editors, *SimuTools*, page 52. ICST, 2009.

[9] D. Gómez, R. Agüero, M. García-Arranz, and D. Ros. TCP Acknowledgement Encapsulation in Coded Multi-hop Wireless Networks. In *Vehicular Technology Conference (VTC Spring), 2014 IEEE 78th*, May 2014.

[10] D. Gómez, S. Hassayoun, A. Herrero, R. Agüero, and D. Ros. Impact of network coding on TCP performance in wireless mesh networks. In *23th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE Proceedings*, September 2012.

[11] D. Gómez, E. Rodríguez, M. Puente, and R. Agüero. Network coding architecture source code and documentation (ns-3). https://github.com/dgomezunican/network-coding-ns3.

[12] M. Hundebøll, J. Ledet-Pedersen, J. Heide, M. Pedersen, S. Rein, and F. Fitzek. Catwoman: Implementation and performance evaluation of ieee 802.11 based multi-hop networks using network coding. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5, 2012.

[13] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in the Air: Practical Wireless Network Coding. *Networking, IEEE/ACM Transactions on*, 16(3):497 –510, 2008.

[14] J. Krigslund, J. Hansen, M. Hundeboll, F. Fitzek, and T. Larsen. CORE: COPE with MORE in Wireless Meshed Networks. In *IEEE VTC2013-Spring: Cooperative Communication, Distributed MIMO and Relaying*, Dresden, Germany, June 2013.

[15] M. V. Pedersen, J. Heide, and F. H. P. Fitzek. Kodo: An open and research oriented network coding library. In *Proceedings of the IFIP TC 6th International Conference on Networking*, NETWORKING'11, pages 145–152, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] H. Seferoglu, A. Markopoulou, and K. Ramakrishnan. I2nc: Intra- and inter-session network coding for unicast flows in wireless networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1035–1043, 2011.

[17] J. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros. Network Coding Meets TCP: Theory and Implementation. *Proceedings of the IEEE*, 99(3):490 –512, March 2011.