Application of network coding in wireless networks : coding conditions and adaptive redundancy control

Thuong Van Vu

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

Pour l'obtention du titre de

Docteur en Informatique

École Doctorale Informatique, Télécommunications et Électronique de Paris

Application du codage réseau dans l'environnement san fil: conditions de codage et contrôle de redondance adaptatif

présentée par Thuong Van VU

devant le jury composé de:

M. André-Luc BEYLOT *Professeur à IRIT-ENSEEIHT*	Rapporteur
M. Steven MARTIN *Maître de Conférences HDR à l'Université Paris-Sud*	Rapporteur
M. Naceur MALOUCH *Maître de Conférence HDR à l'Université Pierre et Marie Curie*	Examinateur
M. Michel BOURDELLÈS *Ingénieur à THALES Communications*	Examinateur
M. Sidi Mohammed SENOUCI *Professeur à l'Université de Bourgogne*	Examinateur
M. Guy PUJOLLE *Professeur à l'Université Pierre et Marie Curie*	Encadrant
Mme. Thi Mai Trang NGUYEN *Maître de Conférence HDR à l'Université Pierre et Marie Curie*	Co-directrice de Thèse
Mme. Nadia BOUKHATEM *Professeur à Télécom ParisTech*	Co-directrice de Thèse

Acknowledgements

First of all, I would like to thank Prof. Andr-Luc BEYLOT, IRIT-ENSEEIHT Toulouse and Prof. Steven MARTIN, University of Paris-Sud for their time and expertise in the review of the manuscript. I also appreciate Prof. Naceur MALOUCH, University of Pierre et Marie Curie, Prof. Sidi-Mohammed SENOUCI, University of Bourgogne and Mr. Michel BOURDELLÈS, THALES Communications for participating my Phd defense committee and for their interest over my Phd subject.

To the end of this 3-year-Phd journey, I would like to show my great gratitude to my supervisors, Prof. Thi-Mai-Trang NGUYEN, Prof. Nadia BOUKHATEM and Prof. Guy PUJOLLE, for their support, guidance in work (conducting research and writing improvement) and even encouragement in life.

I also take this chance to thank Ba Lam TO, Dinh Phong VO and everyone else in "Paris Tech Club" who shared many great memories with me in this City of Light.

I acknowledge the help of the administrative and technical staff of LIP6, particularly, Sabrina VACHERESSE, Irphane KHAN, Julia FRANKE, Véronique VARENNE, Aziza LOUNIS, Pierre-Emmanuel LE ROUX and Konstantin KABASSANOV, all members of PHARE team and LIP6, who smiles at and gives me friendly and warm talks in work everyday.

I give my words to Phuong for her love and trust in me.

Finally, I would like to say thanks to my family, for their love and never-ending support.

Contents

Chapter 1

Introduction

Communication networks have been significantly developed since the early of 1960s, the decade when Advanced Research Projects Agency Network (ARPANET), one of the world's first operational packet switching networks, was born. ARPANET is the first network implementing Transmission Control Protocol/Internet Protocol (TCP/IP), keeping evolved to what becomes the global Internet. Surprisingly, transmitting a packet is still sharing the same fundamental principle of travelling by a car. That is, information or human beings will be transmitted or transported but they have to be separate and their wholeness is maintained. Throughout many decades, this great assumption has been a *de facto* for all network components (routing, data storage, ...) to work on.

Since its first introduction in [1], network coding has gained a significant attention from the research communities in the need of improving the way of communication in computer networks. In short, network coding is a technique which allows the nodes to combine several native packets into one coded packet for transmission (i.e, coding packets) instead of simply forwarding packets one by one. With network coding, a network can save the number of transmissions to reduce data transfer time and increase throughput. This breaks the great assumption about keeping information separate and whole. Information must not be **tampered** but it can be *mixed, and transformed*. To this very point, there are three main questions about network coding that lead the community's research:

- How does network coding evolve the current network systems? Based on what?
- What are the benefits of network coding?
- Where can a network coding layer be added? Which application is network coding suitable for?

Certainly, our work only covers a small part of the answers to these three questions. In the context of this thesis, our contributions belong to two main research branches of network coding: inter-flow network coding (i.e, opportunistic network coding) for throughput enhancement and intra-flow network coding (i.e, random linear network coding) for transmission reliability.

Inter-flow network coding is "encoding packets from multiple traffic flows at a common node". One of practical inter-flow network coding considerations, called opportunistic network coding, has been firstly proposed in Coding Opportunistically (COPE) [2]. COPE XORs multiple packets from different traffic flows and broadcast a single packet based on a particular designed coding condition, called "two-hop pattern". Consequently, COPE can reduce the number of transmissions and enhance the bandwidth utilization. With the inspiration of COPE, more extended works, notably, MAC-layer proactive mixing for Network Coding (BEND) [3] and Distributed Coding-Aware Routing (DCAR) [4] have been proposed in order to achieve better network performances over COPE. The coding condition is a key factor which allows COPE and its successors to know if an encoded packet can be decoded successfully later, so that only the traffic flows fulfilling the coding condition are encoded and decoded in a network. In the case of COPE, the encoders and decoders (i.e., the nodes encoding and decoding packets, respectively) must be neighbors to each other and all of them must be on the routing path of the traffic flows. By giving enhanced coding conditions, DCAR and BEND increase the chances of having encodable flows and outperform COPE. DCAR allows decoders to be more than one hop away from the encoders. Nevertheless, DCAR requires the encoders to be on the routing path of all flows. BEND does not require the later condition because an overhearing node (i.e., a node receiving packets not intended for it) can also be the encoder. However, BEND still insists the encoder and decoders to be neighbors like COPE.

On the other hand, intra-flow network coding is "encoding packets from the same traffic flow". Random linear network coding is mainly applied into intra-flow network coding and widely supported in research community. Random linear network coding has been recently proposed to provide transmission reliability. Instead of being transmitted separately, native packets of a flow are coded in the form of random linear combinations. At recipients (the destination or the intermediate nodes), innovative coded packets are stored until they are sufficiently decoded for original data. In Batch Coding [5, 6, 7, 8, 9], a popular and fundamental random linear network coding scheme, native packets are grouped in to many consecutive batches (called *generation*), encoded and transmitted by linear network coding combination to reduce the complexity of network coding layer at a node. As a trade-off, generations

cause two problems of *high decoding delay* and *generation discard*. In lossy environments like wireless networks, a redundancy control is also needed to mitigate random losses. Some implementations (e.g., [10]) use the fixed redundancy control (i.e, locating the number of redundant packets at fixed intervals to resist losses) which can waste bandwidth due to over-redundancy or degrade network performances with packet generation discards due to packet insufficiency. To overcome this, other propositions (e.g. [11] and [12]) suggest figuring out the required number of redundant packets flexibly by predicting the link quality to compensate errors. The proposed solutions are not quite effective because they are traffic-related (TCP only, like [11], [12]) or do not obtain the "link quality" exactly due to the measurement packet-probing [5], which still wastes bandwidth for over-redundancy. In summary, generations in Batch Coding introduces two problems of *high decoding delay* and *generation discard*. Moreover, the fixed redundancy control or predicting the loss rate by packet-probing do not tackle the problem of *random losses* effectively.

In this thesis, we put our efforts on opportunistic network coding (inter-flow network coding) and random linear network (intra-flow network coding). Our first approach is *to improve network throughput* by an enhanced opportunistic coding system which surpasses COPE, BEND and DCAR. Summarily, if BEND can enrich the coding chances "vertically" (between coders), DCAR can extend the coding chances "horizontally" (along the traffic flow), Distributed Opportunistic and Diffused Coding (DODE) [13], our proposition, is deployed with a new coding condition which combines coding conditions from COPE, BEND and DCAR; and grasps the coding chances "vertically and horizontally", hence, improving the network performance considerably. Afterwards, we have extended DODE into Distributed Opportunistic and Diffused Coding with Multiple Decoders (DODEX) [14], which is a significant next step forward with the new concept of multiple decoders. Basically, a trivial limitation in previous works, even in DODE, is that for every coded packet (the combination of natives packets from different traffic flows), we always have only one encoder (which creates the coded packet) and one decoder (which retrieves the desired packet for the destination). This limitation dramatically restricts the coding chances which can be found in a network topology. DODEX proposed a new coding condition which allows that for one encoder, we can have multiple decoders on the path (up to the number of other traffic flows involved in the encoding process). We complete our inter-flow network coding system by improving the generalized coding condition with Distributed Opportunistic and Diffused Coding with Multiple Encoders and Multiple Decoders (DODEX+). In previous works, the coded packets, after generated, are only forwarded from the encoder to the decoder. Even though some coding chances

can be found with this traffic, the intermediate nodes do not perform the re-encoding process. DODEX+ allows the re-encoding of coded packets with other codable flows to maximize the chances to apply network coding in a network. Thus, more coding chances can be found, improving the network performance. We also introduce a simple routing metric called Short Paths of Enriched Neighbors Metric (SPENM), which bolsters more coding chances found.

Our second approach is to provide *transmission reliability* by new practical random linear network coding systems. Transmission reliability can be achieved by a link-by-link adaptive redundancy control (i.e, losses informed by intermediate nodes or destination) or an end-to-end adaptive redundancy control (i.e, losses informed only by the destination). We believe that an adaptive redundancy control can adjust the redundant traffic effectively if it is integrated and supported by a well-designed coding scheme. We have investigated the current research and found that Pipeline Coding [10] is a generation-based coding scheme which tackles the problem of *high decoding delay*. In Pipeline Coding, upon receiving a new data packet, the source will create the coded packet including all packets up to the new packet in the batch. If all coded packets are delivered successfully, they can be decoded immediately at their arrival time without waiting for generation completion. As a result, Pipeline Coding helps to reduce the decoding delay and improve the throughput. We would like to propose Multi-batch Pipeline Coding with Adaptive Redundancy Control (ARC) [15], a tailored coding scheme over Pipeline Coding integrated with a **link-by-link** adaptive redundancy control. *Multi-batch pipeline coding* is the coding scheme where packets are transmitted consecutively by generations, which are buffered at the destination until well-decoded. In case of losses, the adaptive redundancy control at the source (or forwarders) is triggered to send the traffic accordingly. To achieve a link-by-link adaptive redundancy control to mitigate *random losses*, we interpret the acknowledgement from the current transmission-acknowledgement mechanism at MAC layer IEEE 802.11 as the indicator of "link quality". We decide to find *the suitable time* to transmit the redundant packets to cover losses. An algorithm is proposed to calculate which time is appropriate for transmitting the redundant packet. We argue that the state of link quality should be rechecked before every transmission to deduce if we should adjust the redundant packets or we continue streaming new data to the network. Consequently, we ensure not only the packet sufficiency for decoding but also the new information being transmitted smoothly.

We propose an **end-to-end** adaptive redundancy control to deal a problem of *random losses* in TCP/IP. TCP/IP does not react to random losses in wireless networks smoothly. If there are some losses not related to any congestion in the network,

TCP/IP still treats them as signals of congestion and cuts down the sending rate, leading to the performance degradation [16]. Transmission Control Protocol with Network Coding (TCP/NC) [17] can resolve the problem. Whenever TCP wants to transmit a packet, TCP/NC will send one or more coded packets, depending on the redundancy level. The coded packet in TCP/NC is the combination of all non-ACKed packets remained in the TCP congestion window. TCP ACKs will be sent back to the source to inform that the destination acknowledges every degree of freedom (i.e., new data stored in one linear combination reaches the destination successfully) even if it does not retrieve the data yet. Indeed, the coding scheme of TCP/NC resolves the problem of *high decoding delay* and avoids the problem of *generation discard*. Consequently, packet losses are essentially masked by TCP/NC from the congestion control algorithm. However, the redundancy control provided at fixed intervals in TCP/NC does not recover *random losses* in time. We present Dynamic Coding (DynCod) [18], a tailored coding scheme from TCP/NC integrated with an **end-to-end** adaptive redundancy control, to address this lingering problem upon TCP. Our main idea is *how the destination can inform to the source whether the latest data sent from the source is codable or not and how many packet losses occur via TCP ACKs*. Particularly, we change the principle of the information delivered by TCP ACKs: the destination does not only acknowledges *every degree of freedom*, but also announces how many *unseen packets* (≥ 0) there are in the coding window at the destination. *Unseen packets* are the native packets which are not yet decoded and remained in the coding window at the destination. We argue that the number of *unseen packets* (introduced in [17]) somehow reflects the number of packet losses. If this reflection is shown correctly (i.e, one unseen packet indicates one loss), we can use the number of *unseen packets* as a *loss indicator*. First, we re-design the coding scheme: when TCP wants to transmit a packet, DynCod will send only **one** innovative coded packet. The latest data in DynCod is always transformed and presented by only **one** coded packet *at a time*. Second, thanks to this coding scheme, losses on the transmission will **create** unseen packets in the coding window at the destination. Therefore, one unseen packet in the coding window at the destination can be interpreted as a loss. The source can be informed about the number of lost native packets via TCP ACKs and adjusts the number of redundant packets accordingly.

In DynCod, another minor problem is arise that there may be a potentially high end-to-end delay due to the *random losses* of TCP ACKs. We extend DynCod to support multipath transmission, Multipath Dynamic Coding (MP-DynCod) to tackle this problem. MP-DynCod allows multipath transmission by classifying the neighbors of a node into one primary forwarder and backup forwarders. The primary forwarder is

the neighbor with the highest probability of packet reception and the other neighbors are backup. In other words, the primary forwarder handles the innovative packets while backup forwarders transmit the redundant packets to compensate losses. The probability of packet reception is calculated based on the link-quality routing metric (e.g., Expected Transmission Count (ETX) [19]). Generally, in intra-flow network coding with multipath transmission, we need to re-encode the traffic at an intermediate node to avoid sending the non-innovative coded packets. As the primary forwarder in MP-DynCod always transmits the innovative packets from the source, only backup forwarders need to re-encode the redundant traffic. This will help to reduce the complexity at forwarders. The redundant control is provided hop-by-hop, not from the source only like DynCod, so that it can help to reduce the potentially high end-to-end delay.

The remainder of this thesis is organized as follows. In Chapter 2, we present the related work and how a practical network coding scheme can be deployed to improve the network performance via two main trends inter-flow network coding and intra-flow network coding. We explain the benefits of network coding and introduce current applications of network coding. We also states what concrete problems are solved in the scope of this thesis. In Chapter 3, we introduce three contributions in inter-flow network coding: DODE, DODEX and DODEX+. In Chapter 4, we present three contributions in intra-flow network coding: ARC, DynCod and MP-DynCod. Finally, we draw some conclusions and future research directions in Chapter 5.

Chapter 2

Related work

This chapter presents the background knowledge and provides the state of the art of network coding (Sections 2.1, 2.2, 2.3). Two main trends of network coding: inter-flow network coding and intra-flow network coding are introduced in the details (Sections 2.5 and 2.6). We also categorize applications of network coding in current network systems (Sections 2.7). Problem statement declares which issues we tackle in this thesis (Section 2.8).

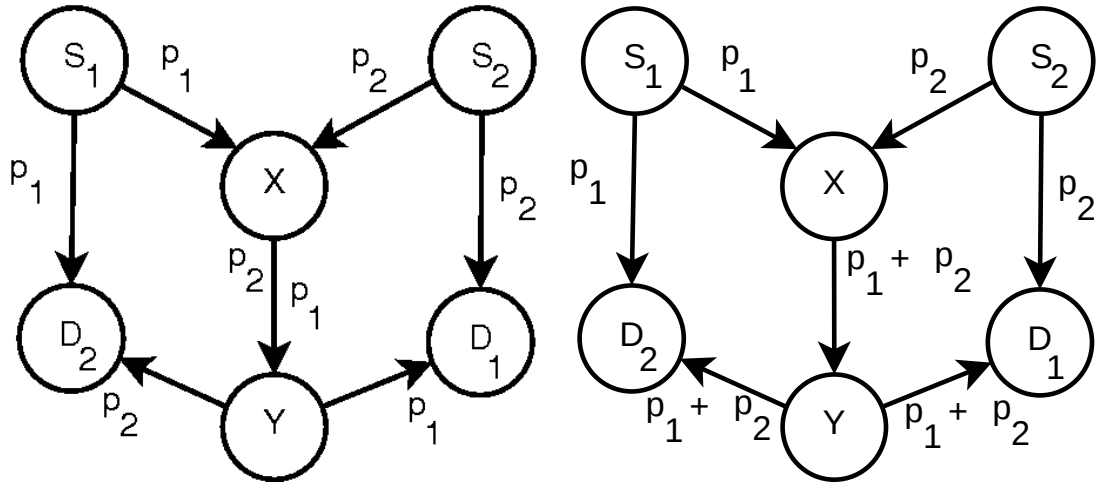
2.1 Network Coding and its benefits

The definition of network coding is not straightforward, even though in [1], the authors refer "coding at a node in a network as network coding". However, we need an explicit and simple definition that explains the purpose and functionality of network coding as below:

Definition 1 *Network coding is a technique which allows the nodes to combine several native packets into one coded packet for transmission (i.e, coding packets) instead of simply forwarding packets one by one in order to maximize network capacity.*

Network capacity is the amount of traffic that a network can carry. How is the combination of native packets for transmission can maximize network capacity over conventional forwarding? We will move to a basic example. The very first idea for network coding was introduced in [1], given by the famous butterfly topology (Fig 2-1). We suppose there are two multicast flows from sources S_1 and S_2 to two receivers D_1 and D_2 via a common link $X \rightarrow Y$ and assume links are loss-free with a capacity of one packet per time unit. With the conventional transmission mechanism, X only forwards either p_1 or p_2 at a time unit. The throughput of each receiver (D_1 or D_2) is

approximately 1.5 packets per time unit. Network capacity in this case is 3.0 packets per time unit.



(a) Butterfly topology without network coding (b) Butterfly topology with network coding

Figure 2-1: Butterfly networks

However, if X intelligently XORs p_1 and p_2 together and sends the coded packet $p = p_1 + p_2$, receivers D_1, (or D_2) can extract the desired packet p_2 (or p_1) successfully by XOR-ing the coded packet p with the obtained packet p_1 (or p_2). In the comparison with the conventional transmission mechanism, network coding reduces the number of transmission and maximizes network capacity (4 packets per time unit instead of 3). Network capacity in this case is maximized. That is the first benefit of network coding: **throughput enhancement**. Generally, below is **Theorem** 1 taken from [20]:

Theorem 1 *Assume that the source rates are such that, without network coding, the network can support each receiver in isolation (i.e. each receiver can decode all sources when it is the only receiver in the network). With an appropriate choice of linear coding coefficients, the network can support all receivers* ***simultaneously*** *[1], [21].*

In network coding, not only the intermediate nodes encode the packets, but also the source also participates in coding its own data. Let's consider the wireless butterfly topology (Fig. 2.2), a multicast flow with only one source S and two receivers D_1 and D_2. The source S wants to send two packets p_1 and p_2, so that it generates and transmits two coded packets: $p_{c_1} = 1 \times p_1$ and $p_{c_2} = 1 \times p_2$. The intermediate nodes A, B, C and E re-encode the received packets and transmit the new linear

combinations: $p_{c_3} = 1 \times p_1 + 1 \times p_2$, $p_{c_4} = 1 \times p_1 + 2 \times p_2$, $p_{c_5} = 2 \times p_1 + 1 \times p_2$ and $p_{c_6} = p_{c_5}$, respectively.

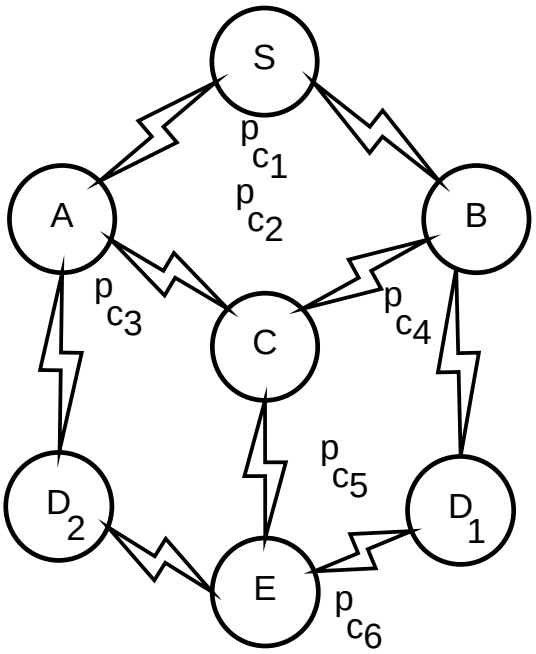


Figure 2-2: Wireless butterfly topology

In Fig. 2.2, the information (packets p_1 and p_2) from single source S is distributed to both sinks D_1 and D_2 but intermediate nodes constituting the transmission path don't need to exchange any routing decision on which packets (p_1 or p_2) are prioritized for transmissions. Currently, the conventional transmission mechanism needs to send the data in order and recognizes whether each packet is lost or not to perform the retransmission (e.g., Automatic Repeat Query (ARQ) [22]). Instead, with network coding, coded packets are considered equally important to each other and carry only one "piece of information" from the source. If the destination sufficiently collects the "pieces", the destination can retrieve the data successfully. If a loss occurs, the forwarders (or even the source) only need to generate another coded packet for compensation. The forwarders (or the source) have to know *how many* redundant packets are required to cover errors. This is a simplified task in comparison with the conventional transmission mechanism that requires to know *which specific* packet is lost. Consequently, network coding reduces the complexity of the transmission mechanism and recovers the data from losses more easily. This is the second benefit of network coding: **transmission reliability**.

Moreover, from two examples above, we can see that there are two approaches to perform the encoding: the first is inter-flow network coding, i.e., encoding packets

from multiple flows (Fig 2-1(b)) and the second is intra-flow network coding, i.e., encoding packets from the same flow (Fig 2.2). Each approach has its own advantages and challenges that define how network coding is applied into current network systems.

2.2 Linear Network Coding versus Non-linear Network Coding

Many works so far still assume on using linear algebra as the fundamental theory to develop network coding. Interestingly, there exist network topologies where the maximum capacity could not be achieved if only linear network coding solutions are used. This phenomenon was firstly described in [23] and many debates, along with open questions, are still left about which approach is better. The complexity of non-linear network coding is not much higher than linear network coding [24, 25]. To our best knowledge, there are some applications of using non-linear network coding to counter many issues involved in security ([26]) or special networks that linear coding could not satisfy ([27, 28, 29]).

2.3 Linear Network Coding - Deterministic versus Random

Many researches [21], [30] and [1] show that network coding can be practically applied if we develop it as "linear network coding". Consider the network system of which intermediate nodes can act as information forwarders. Conventionally, forwarders only need to *forward* (i.e, receive and re-transmit) packets to the next hop. The information (packets) is kept intact during the transmission. With linear network coding, nodes constituting the traffic flows are allowed to linearly combine a number of incoming (i.e., received or created) native packets into various *coded* packets for transmission.

To ensure the independence among coded packets, some deterministic algorithms to find coding coefficients [21], [31] are provided. In [31], the algorithm checks each node in network and decides the encoding vector each node will choose. The algorithm is polynomial in time and not scaling even this is a decentralized solution. On the other hand, we can use *Random Linear Network Coding* [32]. Each node independently selects the coefficients for coded packets at random. This is a lemma taken from [32]:

Lemma 1 *For a feasible multicast connection problem with independent or linearly correlated sources and a network code in which some or all code coefficients are chosen independently and uniformly over all elements of a finite field* $\mathbb{F}_q$, $(q = 2^s)$ *(some coefficients can take fixed values as long as these values preserve feasibility), the probability that all the receivers can decode the source processes is at least* $(1-\frac{d}{q})^v$ *for* $q > d$, *where* d *is the number of receivers and* v *is the maximum number of links receiving signals with independent randomized coefficients in any set of links constituting a flow solution from all sources to any receiver.*

For example, we assume there is a 3-hop single-path unicast traffic flow in a wireless network. If we consider the finite field $\mathbb{F}_{2^s}$ with s equal to 8 bits (1 byte), we have the probability that the receiver can decode the packets of $(1 - \frac{1}{256})^1 = 0.996$ which is very high and possible. Moreover, the decoding probability greatly increases if we consider a larger s in $\mathbb{F}_{2^s}$. As *Random Linear Network Coding* is a simple and effective approach, it is widely used and also the basic platform for us to propose the nouvel coding scheme.

2.4 Network coding approaches in the thesis

In the scope of this thesis, we follow linear network coding because it is simple and widely used. Let's recall, from the examples in Figs 2-1(b) and 2.2, that we see two clear trends how to apply network coding into the current network systems: packets from different traffic flows are coded and transmitted to improve the network performance, which is called *inter-flow network coding* and packets in the same traffic flow are coded and transmitted for the enhanced transmission reliability, which is called *intra-flow network coding*. Indeed, these trends are the main interesting topics for the current research community. In the term of practical application of network coding system, *Opportunistic Network Coding* [2] is the pioneer of inter-flow network coding and *Random Linear Network Coding* [32] is the simple and effective proposition of intra-flow network coding. We will go through these two in the following sections by detailing many proposed implementations which inspired our works. Fig 2.4 shows the visual illustration about our work and how it fits into the current research of network coding. The works on inter-flow network coding are sorted as the coding condition is extended to support more network topologies and helps to improve network performance. On the other hand, the works on intra-flow network coding are sorted as the new coding schemes are proposed. The blue boxes represent our contributions.

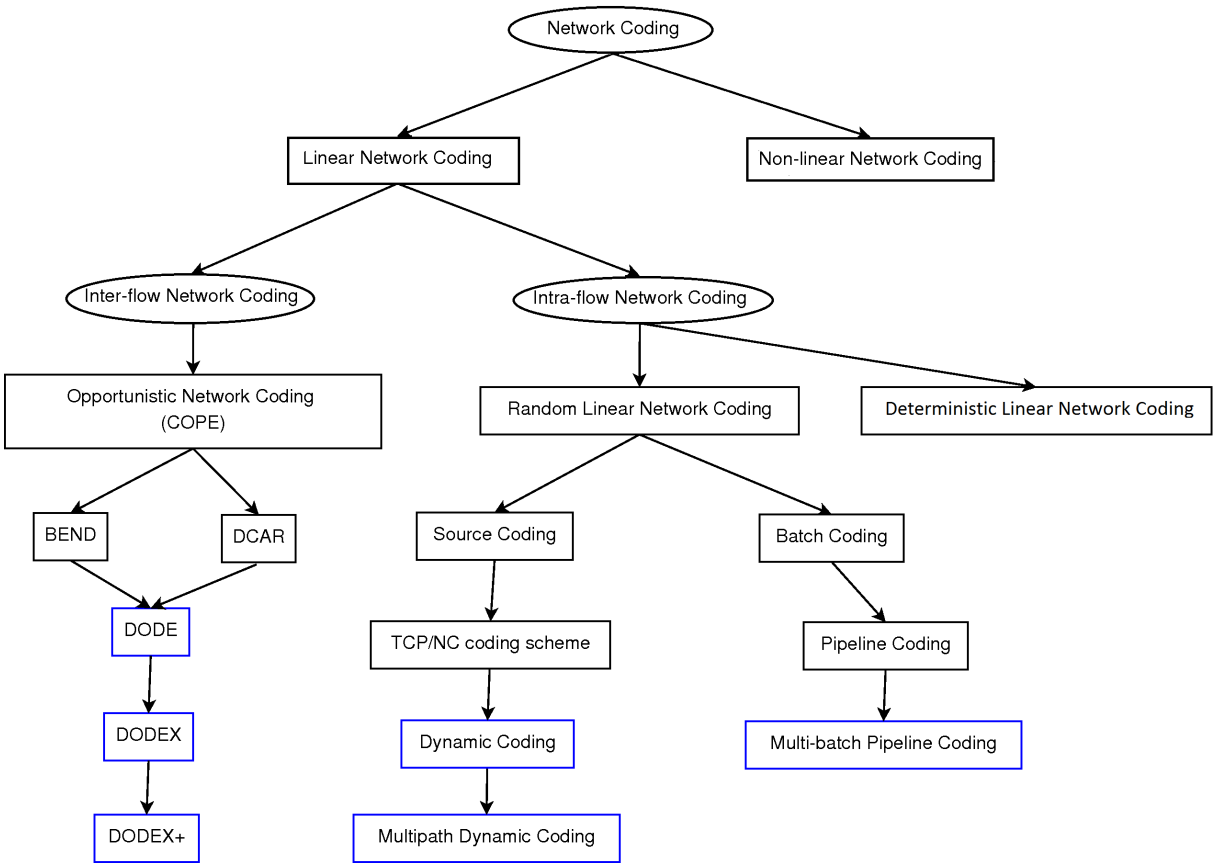


Figure 2-3: Thesis contributions

2.5 Inter-flow network coding (Opportunistic Network Coding)

In this section, we introduce one popular and practical inter-flow network coding scheme called *Opportunistic Network Coding* (ONC). ONC is first introduced in COPE [2] which is applied to unicast traffic in wireless network. ONC is a technique which combines packets (i.e., XORs them together) from traffic flows traversing in opposite ways at an intermediate node called *encoder*. Encoded packets are then acquired to retrieve the original packets at *decoders*. ONC is a practical inter-flow network coding system of which design is based on two key principles:

- *No point-to-point assumption and exploitation of the broadcast nature of wireless networks*: In conventional wireless networks, nodes always discard the non-intended packets even though they can **overhear** them due to adapting the principle of point-to-point transmission from wired networks. However, ONC

can exploit the broadcast nature of wireless environment to use the overheard packets in the term of network enhancement.

- *Inter-flow network coding for wireless unicast traffic*: ONC practically mixes and transformes data from multiple flows and gain an extra of network throughput. Consequently, an enhanced network system is proposed.

ONC systems are heavily based on a *coding condition* to decide which flows are codable together because there's no point to code packets without ensuring their successful decoding. In ONC, the coding chances are gained more based on how the coding condition is defined (i.e., covering more network flowing cases). We will detail these in the next subsections.

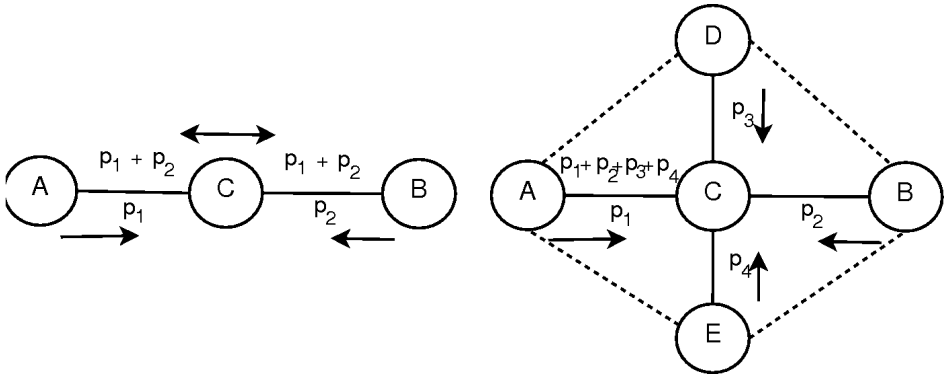
2.5.1 Coding Opportunistically (COPE)

Figure 2-4(a) and 2-4(b) show how COPE works in the basic scenario. The dashed lines imply that the related nodes are neighbors (i.e., within the transmission range) to each other and the arrows indicate the direction of traffic flows. Both end-to-end connections go through the intermediate node C which serves as a forwarding node. Suppose that node A and B want to send packet p_1 and p_2 to node B and A, respectively. Without network coding, node C simply forwards packets p_1 and p_2 to the destinations using two separate transmissions. Using network coding, node C can smartly combine these two native packets by XOR-ing them together and broadcast the *coded* packet $(p_1 + p_2)$ to nodes A and B in a single transmissions. Nodes A and B will decode the coded packet to get the desired packet by XOR-ing the coded packet with the packet they originate. As a result, node C can save one transmission and improve the network utilization. Node C is called the *encoder* and two nodes A and B serve as *decoders*.

More coding scenarios are possible if we consider one important feature introduced by COPE: the opportunistic listening in which each wireless node can overhear other transmissions and use the overheard packets for later use (e.g, for the decoding process). Consider the scenario presented in Figure 2-4(b) in which nodes A, B, D and E wish to send packets p_1, p_2, p_3 and p_4 to its opposite nodes B, A, E and D, respectively. Each extremity node (A, B, D and E) will perform the opportunistic listening because it is the neighbor of the other nodes except its opposite node. C intelligently encodes four native packets to broadcast $p_c = p_1 + p_2 + p_3 + p_4$. All receivers are able to extract the desired packets because they have collected sufficiently other packets for decoding. For example, node A can decode the coded packet p_c to get p_2 successfully because p_3 and p_4 are obtained from opportunistic listening (p_3

and p_4 are overheard by transmissions $(D \rightarrow C)$ and $(E \rightarrow C)$ respectively) and p_1 is its own packet. In summary, COPE exploits the broadcast nature of the wireless channel to perform the *opportunistic listening* and *opportunistic encoding*, reducing the number of forwarded packets and improving the network utilization.

Indeed, the ONC system needs to check if the flows are codable or not based on a *coding condition*. The coding condition in ONC will define which flow is codable to another and which node is the encoder or the decoder. There is no point in creating the coded packets if they are un-decodable. To check the coding condition, COPE needs to obtain the "neighbor state", or which packets the neighbors could receive. In the absence of deterministic information, COPE utilizes a link state routing protocol with the link-quality routing metric ETX [19] to guess the "neighbor state" intelligently. Certainly, the node can make a false decision, reducing the coding chances by not coding the codable native packets or producing un-decodable coded packets. How to guess the "neighbor state" intelligently is quite challenged. In the scope of this thesis, we leave this problem open as we focus on a more certain existing problem: extending the coding condition of COPE to gain more coding chances.



(a) Simple case without the opportunistic listening

(b) Complex case with the opportunistic listening

Figure 2-4: Two scenarios applied for COPE

To determine whether flows are codable, COPE defines the coding condition called *the two-hop coding pattern*. The coding condition of COPE is provided to detect if multiple flows are codable together or not. Every pairs of flows must be checked and met the coding condition. This is the principle required in ONC. Before we move further into the details of the coding condition of COPE, we need some mathematical notations to describe the ideas more clearly:

- A network is formally presented by a graph G. A graph $G = (V, E)$ is a set of nodes V connected by a set of links E.

- A node $v \in V$ is a terminal node (e.g., the user's station) or an intermediate node (e.g., the switch or the router).
- A link $e = (v, t) \in E$ is a connectivity between two nodes v and t. The link $e = (v, t)$ consists of one node v called the start node and one node t called the end node.
- A traffic flow $F = (V_F, E_F)$ of k links is a subgraph of the network G, representing a way to get from a source to a destination by traversing links in the network. A flow $F = (V_F, E_F)$ ($V_F \subset V$ and $E_F \subset E$) is uni-directional and can be written as an ordered list of nodes: $V_F = \{v_{1_F}, v_{2_F}, ..., v_{k_F}, v_{k+1_F}\}$ connected by an ordered list of directed links: $E_F = \{(v_{1_F}, v_{2_F}), (v_{2_F}, v_{3_F}), ..., (v_{k_F}, v_{k+1_F})\}$.
- v_{1_F} is the source of flow F.
- v_{k+1_F} is the destination of flow F.
- v_{i+1_F} is the next hop of node v_{i_F}.
- v_{i-1_F} is the previous hop of node v_{i_F}.
- The set of neigbors of node v_i, denoted as $N(v_i)$, contains all nodes that have a link with node v_i:
  $N(v_i) = \{\forall v_j, j \neq i | (v_i, v_j) \in E\}$.
- The set of upstream nodes of node v_{i_F} on flow F, denoted as $Up(v_{i_F}, F)$, is the set of nodes on transmission path from the source v_{1_F} to the previous hop v_{i-1_F}:
  $Up(v_{i_F}, F) = \{\forall v_{j_F} | j_F < i_F\}$
- The set of downstream nodes of node v_{i_F} on flow F, denoted as $Dn(v_{i_F}, F)$, is the set of nodes on transmission path from the next hop v_{i+1_F} to the destination v_{k+1_F}:
  $Dn(v_{i_F}, F) = \{\forall v_{j_F} | j_F > i_F\}$
- The set of neighbors of all nodes on flow $F = (V_F, E_F)$, denoted as $N(F)$, is the union of sets of neighbors of all nodes on flow F:
  $N(F) = \bigcup_{j=1, j \neq i}^{k+1} N(v_{j_F})$
- The set of neighbors of a group of nodes T, denoted as $N(T)$, is the union of sets of neighbors of nodes in T:
  $\forall v_j \in T, N(T) = \bigcup_{j=1, j \neq i}^{|T|} N(v_j)$

The two-hop coding pattern of COPE is then described in Condition 1 and Condition 2. The first condition (Condition 1) explains how two traffic flows are codable to each other and the second condition (Condition 2) explains how multiple flows are codable together.

Condition 1 *Two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other, at node $c = v_{f_{F_i}} = v_{g_{F_j}} \in (V_{F_i} \bigcap V_{F_j})$ if and only if:*

- $|D_{ij}| = |\{v_{f+1_{F_i}}\} \bigcap (\{v_{g-1_{F_j}}\} \bigcup N(v_{g-1_{F_j}}))| = 1.$
- $|D_{ji}| = |\{v_{g+1_{F_j}}\} \bigcap (\{v_{f-1_{F_i}}\} \bigcup N(v_{f-1_{F_j}}))| = 1.$

Literally, Condition 1 states: two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other at node c which is intersected by both flows if the next hop of c on flow F_i (or flow F_j) is the previous hop of c on flow F_j (or flow F_i) or is the neighbor of that previous hop. D_{ij} (or D_{ji}) is the set of decoders of flow F_i (or flow F_j) to remove the native packet of flow F_j (or flow F_i) from the coded packet generated by encoder c.

Condition 2 *Native flows belonging to a set of flows $\mathbb{F} = \{F_k | k \in [1, n]\}$ (n is the number of flows) are considered as codable together at node $c \in \bigcap_{j=1, j\neq i}^{n} V_{F_j}$ if and only if:*

- $\forall F_i, F_j \in \mathbb{F}(i \neq j) \mid F_i$ *and* F_j *are considered as codable to each other at node* c.
- $\forall F_i \in \mathbb{F}, |D_i| = |\bigcup_{j=1, j\neq i}^{|\mathbb{F}|} D_{ij}| = 1.$

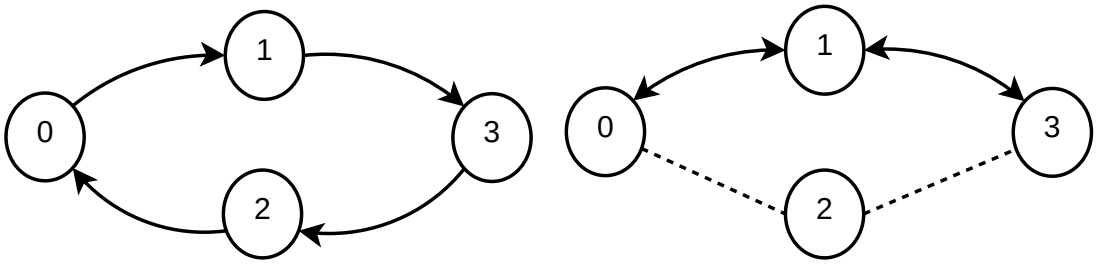
Condition 2 specifies the constraint for multiple flows to be codable. First, each pair of flows must be codable and intersected at a common node c, which is the only encoder for all flows. Second, for each flow, there is only one decoder ($|D_i| = 1$), which is $d_i \in D_i$, D_i is the set of decoders of flow F_i to retrieve the native packet of flow F_i from the coded packet.

With the two-hop coding pattern of COPE, we can realize that node c is the encoder and node $d_i \in D_i$ is the decoder. In other words, the coded packet generated by the coder c is decoded by only one decoder d_i of flow F_i. That is a trivial limitation that DODEX (Section 3.2) breaks to find more coding chances. However, as a pioneer, COPE filled a significant gap between the theory and practical consideration

in network coding. COPE gets the great attention from the research community and many works focus on how to improve COPE from many aspects: extending the coding condition of COPE (like BEND, DCAR, or our contributions DODE, DODEX and DODEX+ (Chapter 3)), investigating the constraints affecting the coding condition of COPE [33, 34, 35, 36, 37, 38], or finding the routing path that gain the most coding chance from COPE ([39, 40, 41, 4, 42, 43, 44, 45, 46, 47, 48, 49]).

2.5.2 Distributed Coding-Aware Routing (DCAR)

COPE has two separate limitations: the *routing path dependency* and the *strict two-hop coding pattern*. The first limitation is that *the coding chances inevitably depend on the established route*. The best routing path (e.g., the shortest path) may not be the best coding-aware one. Consider the example in Figs 2-5(a) and 2-5(b) with two traffic flows. Without being aware of the potential coding chances, two separate routing paths shown in Fig 2-5(a) may be chosen. On the other hand, if a coding-aware routing decision is made, a common routing path via node 1 is used. Node 1 becomes the encoder and has the coding chance to generate the coded packets (in Fig 2-5(b)). In this example, coding-aware routing will provide the improved performance.



(a) Non-coding-aware routing (b) Coding-aware routing

Figure 2-5: Routing decision affects the coding chance

The second limitation is that traffic is *strictly coded based on the two-hop coding pattern*. Let's take Fig 2-6 as an example and consider the scenario where two flows $F_1(1 \rightarrow 2 \rightarrow 3 \rightarrow 4)$ and $F_2(5 \rightarrow 3 \rightarrow 6 \rightarrow 7)$ intersect at node 3. In this scenario, COPE cannot detect the coding chance because two hop coding pattern is not met. Indeed, there is a coding chance where node 3 is an encoder and decoders are 4 and 7. Node 3 can generate the coded packet from both flows. Even though node 6 cannot perform the decoding, it can forward the coded packet to node 7. Node 7 obtains enough packets to perform the decoding to retrieve the desired data. In general, the decoder can be **more than one hop-away** from the encoder. If we can re-design

the coding condition in 1, these coding chances can be detected and the bandwidth efficiency and throughput can be enhanced.

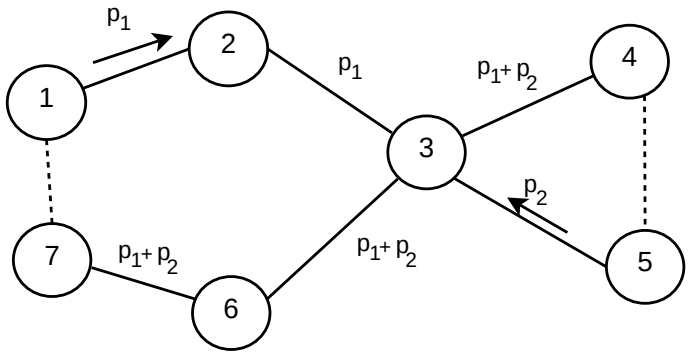


Figure 2-6: Generalized coding condition of DCAR breaks two hop coding pattern

To cope with the first limitation of COPE, DCAR [4] tries to make routing decisions and concentrate traffic through some specific nodes for more coding chances. Even though this solution enhances COPE in some cases, concentrating traffic solely via one encoder can lead to packet collisions and packet drops at that intersecting node. This problem is discussed in the next section 2.5.3. DCAR has solved the second limitation by extending the two-hop coding pattern. The decoders are not necessarily one-hop neighbors of the encoder.

The generalized coding condition of DCAR is described in Condition 3 and 2. DCAR only changes the coding condition between two traffic flows (Condition 3) and re-uses the coding condition of multiple flows of COPE (Condition 2). Condition 3 states that two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other at a node c if and only if there is one downstream node of c on flow F_i (or on flow F_j) being the upstream node of c on flow F_j (or on flow F_i) or at least being the neighbor of that upstream node. D_{ij} (or D_{ji}) contains decoders which do not need to be neighbors of encoder c.

Condition 3 *Two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other, at node $c \in (V_{F_i} \bigcap V_{F_j})$ if and only if:*

- $|D_{ij}| = |Dn(c, F_i) \bigcap (Up(c, F_j) \bigcup N(Up(c, F_j)))| = 1.$
- $|D_{ji}| = |Dn(c, F_j) \bigcap (Up(c, F_i) \bigcup N(Up(c, F_i)))| = 1.$

For illustration on how to apply the generalized coding condition of DCAR, let's reconsider Figure 2-6 with two flows $F_1(1 \rightarrow 2 \rightarrow 3 \rightarrow 4)$ and $F_2(5 \rightarrow 3 \rightarrow 6 \rightarrow 7)$, one

realizes that COPE cannot detect the coding chances of two traffic flows but DCAR can. As the coding condition of DCAR says, we can easily see $c = 3$, $d_1 = 4$ and $d_2 = 7$ as $c \in V_{F_1} \bigcap V_{F_2}$, $d_1 \in D_1 = D_{12} = Dn(3, F_1) = \{4\} \subset N(5)$, $5 \in Up(3, F_2)$ and $d_2 \in D_2 = D_{21} = Dn(3, F_2) = \{7\} \subset N(1)$, $1 \in Up(3, F_1)$.

2.5.3 MAC-layer proactive mixing for Network Coding (BEND)

Concentrating traffic through only one encoder which is also the intersecting node of two flows like COPE or DCAR can cause the performance problem. Considering a wireless mesh network with dense traffic, concentrating traffic via the encoders result in packet collisions and packet drops at the intersecting node. The problem can be worse if the dropped packets are the coded ones. BEND is a MAC layer solution to practical network coding in multi-hop wireless networks, which solves the problem. In contrast to COPE and DCAR which use a single node to encode packets, BEND deploys a group of neighboring nodes which can share the encoding process, called *the encoder group*. In fact, the transmission of a node is heard by its neighbors, which can be used to encode the packets. In BEND, any node can code and forward a packet even when this node is not on the routing path of the packet, as long as it ensures that the decoding process can be successfully performed at the decoders to obtain the original packets.

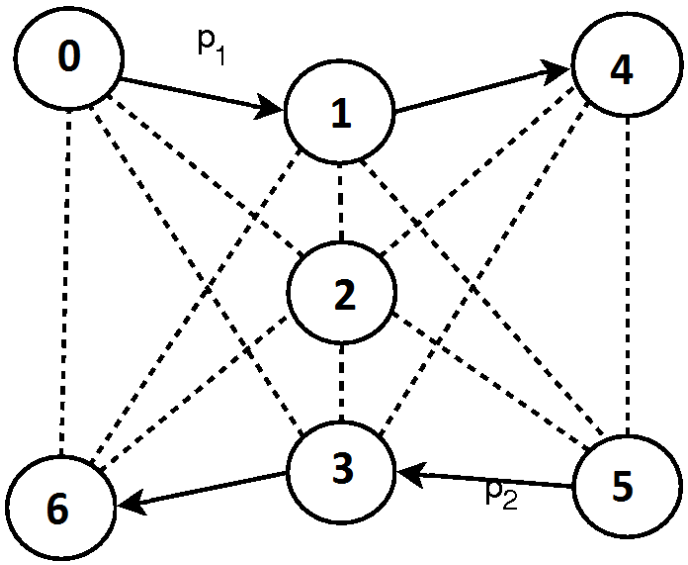


Figure 2-7: An encoder group in BEND

Specifically, BEND takes advantage of the encoder group to share the encoding process instead of concentrating the traffic through only one encoder. The encoder

group can grasp the overheard packets and combine them with the packets that they have in their queues to provide the coded packet. In comparison with COPE and DCAR, the overheard packets are not only used in the decoding process but also in *the encoding process*. As illustrated in Figure 2-7, there are two flows $F_1(0 \rightarrow 1 \rightarrow 4)$ and $F_2(5 \rightarrow 3 \rightarrow 6)$. The packets will be forwarded by an encoder group of three intermediate nodes 1, 2 and 3, i.e., nodes 1, 2 and 3 share the encoding process. For example, if node 3 is more idle than nodes 1 and 2, it can combine both native packets into one and broadcast $(p_1 + p_2)$. The coded packets are broadcasted by the most idle intermediate node. All other nodes in the encoder group overhear the coded packet and stop the intended transmission for packet duplication avoidance. This feature called the *diffused gain by neighborhoood* indeed solves the first problem *routing dependency* mentioned in Section 2.5.2 and increases the network performance.

The two-hop coding pattern of COPE has also been extended by the diffused gain of BEND: the encoder is not necessarily the intersecting node of the two flows. The two-hop coding pattern of BEND is described in Condition 4 and Condition 5. BEND changes the definition of the encoder c which affects the coding condition between two flows of COPE (Condition 1) and the coding condition of multiple flows of COPE (Condition 2). Condition 4 states: two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other at an encoder group C_{ij} if and only if there exist two nodes $v_{f_{F_i}}, v_{g_{F_j}} \in L_{ij}$ such that the next hop of $v_{f_{F_i}}$ (or $v_{g_{F_j}}$) on flow F_i (or on flow F_j) is the previous hop of $v_{g_{F_j}}$ (or $v_{f_{F_i}}$) on flow F_j (or on flow F_i) or at least is the neighbor of that previous hop. L_{ij} is the set of nodes u on flows F_i and nodes t on flow F_j such that there are links between u and t or $u = t$. C_{ij}, the encoder group that generates coded packets between two flows F_i and F_j, is the set of neighbors of four nodes $v_{f-1_{F_i}}, v_{g-1_{F_j}}, v_{f+1_{F_i}}$ and $v_{g+1_{F_j}}$. D_{ij} (or D_{ji}) is the set of decoders of flow F_i (or flow F_j) to remove the native packet of flow F_j (or flow F_i) from the coded packet generated by the encoder group C_{ij}.

Condition 4 *Two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other at a group of nodes $C_{ij} \subset L_{ij} = \{\forall u \in V_{F_i}, \forall t \in V_{F_j} | u = t \bigvee (u,t) \in E\}$ if and only if:*

- $\exists v_{f_{F_i}}, \exists v_{g_{F_j}} \in L_{ij}$.
- $|D_{ij}| = |\{v_{f+1_{F_i}}\} \bigcap (\{v_{g-1_{F_j}}\} \bigcup N(v_{g-1_{F_j}}))| = 1$.
- $|D_{ji}| = |\{v_{g+1_{F_j}}\} \bigcap (\{v_{f-1_{F_i}}\} \bigcup N(v_{f-1_{F_i}}))| = 1$.
- $|C_{ij}| = |N(v_{f-1_{F_i}}) \bigcap N(v_{g-1_{F_j}}) \bigcap N(v_{f+1_{F_i}}) \bigcap N(v_{g+1_{F_j}})| \geq 1$

Condition 5 does not change much from Condition 2, only an encoder group C_i is used instead of only one encoder c like COPE. Condition 5 states that each pair of flows must be codable and intersected at an encoder group C_{ij}, which contains the encoders of coding packets between two flow F_i and F_j. Besides, for each flow, there is only one decoder ($|D_i| = 1$), which is $d_i \in D_i$, and C_i is the encoder group containing all encoders to create coded packets of multiple flows $\forall F_i \in \mathbb{F}$, which must be the intersection of all C_{ij}: $C_i = \bigcap_{j=1, j\neq i}^{n} C_{ij}$ ($|C_i| \geq 1$).

Condition 5 *Native flows belonging to a set of flows $\mathbb{F} = \{F_k | k \in [1, n]\}$ (n is the number of flows) are considered as codable together at a group of nodes $\forall F_i \in \mathbb{F}, C_i = \bigcap_{j=1, j\neq i}^{n} C_{ij}$ if and only if:*

- $\forall F_i, F_j \in \mathbb{F}(i \neq j) \mid F_i$ *and* F_j *are considered as codable to each other at the group of node* C_{ij}.
- $\forall F_i \in \mathbb{F}, |D_i| = |\bigcup_{j=1, j\neq i}^{|\mathbb{F}|} D_{ij}| = 1$.
- $|C_i| \geq 1$

For example, let's reconsider network topology in Fig 2-7. As we have $L_{12} = \{0, 6, 1, 3, 4, 5\}$, there exist two nodes $1, 3 \in L_{12}$ such that 4 (or 6), the next hop of 1 on flow F_1 (or 3 on flow F_2), is the neighbor of 5 (or 0) which is the previous hop of 3 on flow F_2 (or 1 on flow F_1). Consequently, decoder on flow F_1 (or F_2) is 4 (or 6) and the encoder group is $\{1, 2, 3\} = N(0) \bigcap N(4) \bigcap N(5) \bigcap N(6)$.

2.6 Intra-flow network coding (Random linear network coding)

2.6.1 Source coding versus batch coding

Random linear network coding (RLNC) has been introduced almost at the same time since the idea of network coding emerged. As shown in Figs 2-1(b) and 2.2, RLNC is focusing on "coding the packets belonging to the same traffic flows". The nature of wireless network tends to be error-prone and exposed to interference and congestion. To provide a simple and effective transmission reliability, instead of transmitting

native packets in order, native packets are coded to broadcast the random linear combinations. At recipients, innovative coded packets are stored until they are sufficiently decoded for original data. In lossy environments like wireless networks, a redundancy control is also needed to mitigate losses. Automatic Repeat Query (ARQ) [22] and Forward Error Correction (FEC) [50] are two techniques which can be used:

- ARQ is an error-control method for data transmission relying on acknowledgements and timeouts to ensure the transmission reliability.
- FEC is an error-control method for data transmission which allows sending the information along with some redundant data to control errors over the lossy communication environment.

In this thesis, we focus on FEC-related coding schemes, such as source coding scheme [51] and batch coding scheme [1, 9]:

- Source coding scheme: the source generated innovative coded packets, along with some redundant coded packets, and transmits them to the destination. Forwarders only forward the coded packets to the destination.
- Batch coding scheme: the source generated innovative coded packets, along with some redundant coded packets, and transmits them to the destination. Forwarders perform the re-encoding process over the input coded packets, generate and transmit new linear network coding combinations to the destination.

In this section, we would like to show the popular coding schemes of RLNC. We present Batch Coding (Section 2.6.3), Pipeline Coding (Section 2.6.4) and TCP/NC (Section 2.6.5), along with some frequently used terms in Table 2.1. Throughout the examples in Figs 2-8, 2-9, 2-10, the fixed redundancy control used in Batch Coding, Pipeline Coding and TCP/NC coding scheme has the redundancy level $R = 1.25$ per one transmission, which means 1 redundant transmission every 4 transmissions.

Table 2.1: Defitions of terms used in intra-flow network coding

Term	Definition
Source coding scheme	Source-side only coding.
Batch coding scheme	Source-side and relay coding.
Batch Coding	Every coded packet will encode all data packets within the same generation. Coding and decoding begins only when the generation rank is full
Pipeline Coding	Coded packets will be generated upon every new data packet arriving. Destinations decode the data packets progressively if possible.
Generation	A set of packets that are encoded or decoded together as a unit.
Coding vector (Encoding vector)	A vector of coefficients that reflect the linear combination of data packets.
Rank (Degree of freedom)	Number of linearly independent combinations of data packets.
Innovative packet	A packet that increases the rank.
Coding redundancy	Number of coded packets sent per generation divided by generation size.

2.6.2 Generation

Basically, due to the constraints of practical implementation, a RLNC system requires that packets are grouped into many consecutive batches with the same fixed size, called *generations*. The size of generation and how generation is generated will give a huge impact on the performance of network coding [52]. Generations help to reduce the complexity of decoding process as the calculation over a decoding matrix with the fixed size is quite simple [20]. Original data is only decoded after the whole generation is well-received, thus, causing a high decoding delay. Moreover, coded packets in a generation need to be decoded successfully before packets in a newer generation can be transmitted [32]. If a generation is discarded (e.g., not enough innovative packets), the node has to retransmit that generation, leading to an increasing decoding delay. In summary, the main limitation that generations introduce is the *high decoding delay* and potential information loss due to whole *generation discard*, especially with fixed redundancy control (i.e, redundant traffic is adjusted into the network at pre-set time intervals) [10], [6], [7], [8], [9]. For illustration, in Fig 2-8, generation 1 is well-decoded but the destination has to wait for sufficient packets to perform the decoding process, leading to the high decoding delay. Because there are not sufficient packets to perform the decoding, whole generation 2 is discarded.

Many propositions [10], [11] introduce a minimum redundancy level to ensure that the generation discard does not happen. However, this may introduce the over-redundancy.

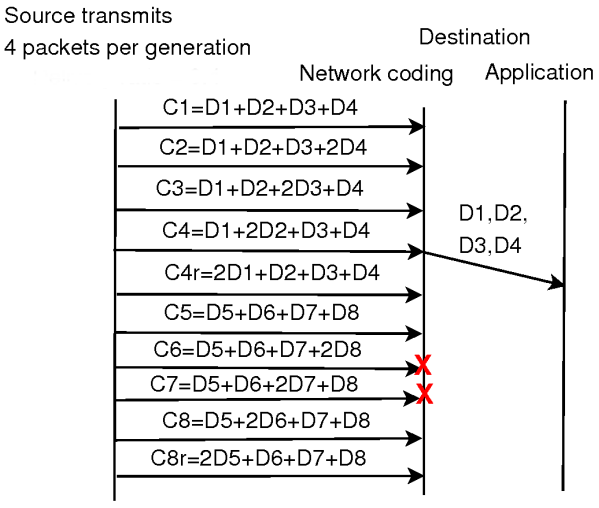


Figure 2-8: Batch coding

2.6.3 Batch Coding

This scheme is the fundamental mechanism of many works [5], [6], [7], [8], [9]. With n packets in the g^{th} generation ($g > 0$), the i^{th} coded packet generated for transmission is:

$$c_i = \sum_{j=1}^{n} e_j \mathbf{p}_{(g-1)\times n+j} \tag{2.1}$$

where e_j is the coding coefficient ($e_j > 0$), an element deployed from a particular Galois field $\mathbb{F}$, $\mathbf{p}_j$ is the j^{th} native packet in the g^{th} generation and $(g-1)\times n$ denotes the number of transmitted packets. This coding scheme exposes two problems that are described in Section 2.6.2.

2.6.4 Pipeline Coding

Pipeline Coding is a generation-based coding scheme which encodes and decodes packets progressively instead of waiting for the entire batch (i.e., generation) like many previous works [5], [6], [7], [8], [9]. For n packets in the g^{th} generation, the i^{th} coded packet generated for transmission is:

$$c_i = \sum_{j=1}^{i} e_j \mathbf{p}_{(g-1)\times n+j} \tag{2.2}$$

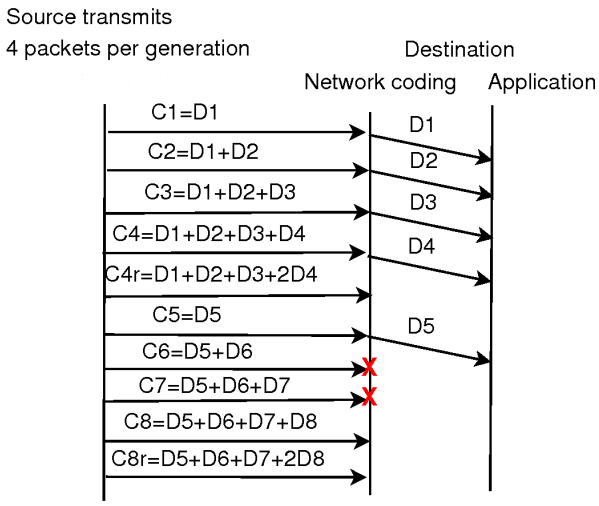


Figure 2-9: Pipeline Coding

Fig 2-9 shows the mechanism of Pipeline Coding: upon receiving the i^{th} new data

packet, the source will instantly trigger the encoding process to create the coded packet which is the combination of all currently received data packets (up to the i^{th} packet in the g^{th} generation). If coded packets arrive at the destination successfully, original data can be decoded immediately without waiting for transmitting the whole generation. We can say the coding scheme of Pipeline Coding is *progressive*, i.e, coded packet is decoded one by one at the arrival time without generation completion. Thanks to the progressive coding scheme, the problem of *high decoding delay* is resolved. However, Pipeline Coding still suffers the problem of *generation discard*.

2.6.5 Transmission Control Protocol with Network Coding (TCP/NC)

In a lossy environment like wireless network, random losses are treated by TCP as signals of congestion. As a result, TCP will dramatically reduce the sending rate, leading to network performance degradation [16]. TCP/NC has been proposed to resolved this problem. TCP/NC is intended to incorporate network coding into TCP/IP layer as a network coding layer between TCP layer and routing layer without any major changes to TCP/IP stacks. Using this idea as the core philosophy to develop, authors of TCP/NC only change mechanisms of packet generation and packet acknowledgement. First, at the source, a coding window, of which the size is the same with the size of TCP congestion window, is used in TCP/NC layer to receive packets from TCP layer. Whenever TCP layer at the source wants to send a packet, TCP/NC will generate and transmit at least one coded packet (depending on the redundancy level R) which combines that packet with other non-ACKed packets in the coding window. TCP/NC uses the fixed redundancy control to tackle the problem of random losses. In TCP/NC, only the source generates and transmits the coded packets to the destination. If it's a multi-hop traffic flow, forwarders constituting the transmission path will forward the traffic without re-encoding them. Second, at destination, the destination acknowledges *every degree of freedom*, which indicates an arrival of one linear network coding combination which contributes a new piece of information, even though it does not retrieve the desired data yet. With the new interpretation of TCP ACKs, packet losses are essentially masked from TCP congestion control. The source then reacts to packet losses smoothly, without cutting down the sending rate. Summarily, to be compatible with TCP mechanisms, TCP/NC is a intra-flow source coding layer. Moreover, as described in the coding scheme, TCP/NC coding scheme is *non-generation-based* but it is also *progressive*, which allows TCP/NC to avoid the problems of *high decoding delay* and *generation discard* as explained in Section 2.6.2.

When i^{th} packet p_i is allowed for transmission, $\lfloor NUM \rfloor$ coded packets c_{i_r} are generated as follows:
for $r = 1 \rightarrow \lfloor NUM \rfloor$:

$$c_{i_r} = \sum_{j=i-n}^{i} e_j \mathbf{p}_j \tag{2.3}$$

and

$$NUM = \begin{cases} NUM + R \;\; before\, packet\, generation \\ NUM - \lfloor NUM \rfloor \;\; afterward \end{cases} \tag{2.4}$$

where R denotes the redundancy level, n denotes the number of non-ACKed packets remained in coding window.

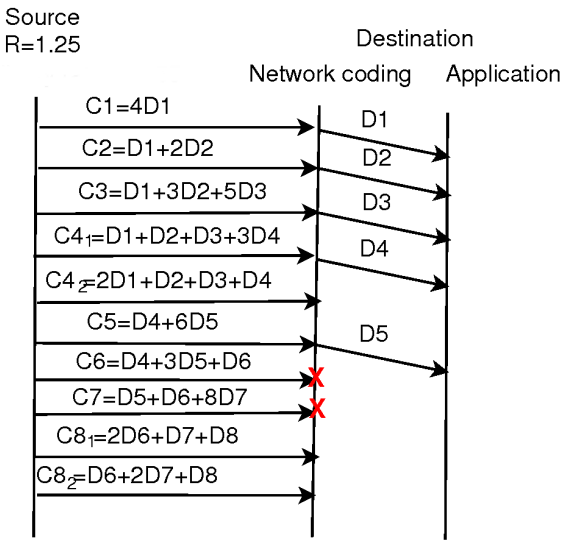


Figure 2-10: TCP/NC

We do not illustrate the ACK transmissions in Fig 2-10 but TCP ACKs help to remove the packets already reached the destination from the coding window. TCP/NC provides the *progressive* encoding and decoding processes like Pipeline Coding as coded packets can be decoded at the time they arrive. Due to the losses of C_6, C_7, the arrival of the redundant packet C_{8_2} is not sufficient for decoding. The decoding will be performed until the next redundant packet is sent (i.e., the transmission of packet C_{12_2}).

2.7 Applications of network coding into current network systems

2.7.1 Wireless networks

2.7.1.1 Multi-hop traffic flows in wireless networks

COPE [2], BEND [3] and DCAR [4] provide practical network coding with multi-hop traffic flows in wireless networks. The coding condition becomes generalized from each extension (from COPE to DCAR). As shown in Figs 2-1(b) and 2-4(a), network coding can improve throughput of wireless network when more than 2 traffic flows intersect at a common node. This is also extended to support any network of which physical layer using broadcast. Being dependent on the type of traffic (e.g. multicast or unicast), network coding will be applied into differently situational network topologies. In 2-1(b), two multi-cast traffic flows are transmitted in the same direction while in 2-4(a), two unicast traffic flows are transmitted in opposite directions. Moreover, there exist problems like transmission interference, channel fading, energy consumption in multi-hop wireless networks. [53, 54] try to find solutions to send multicast traffic in interference-prone wireless networks effectively. In order to reduce the effort and complexity to perform network coding, [55] suggests selecting nodes which need encoding instead of all nodes. CodeCast [8], I^2NC [56], [57] and [58] were developed further the idea "XORs in the air" from COPE and assessed on real scalable testbeds. [57] proposes the combination of network coding and physical broadcast in wireless networks in the mutual exchange of information between two nodes as a distributed scheme which is robust to random losses and delay.

2.7.1.2 Broadcast in wireless networks

Broadcasting is one of basic operations in communication networks. Flooding is the typical method of broadcast in multi-hop wireless networks. Flooding allows one node to forward a packet to its neighbors whenever the packet arrives at the node at first. In wireless networks, flooding may cause packet collisions, resulting in wasteful energy consumption due to great amount of packets repeatedly forwarded (i.e, broadcast storm problem [59] and its solutions [60, 61]). Network coding can help to solve the broadcast storm problem more effectively because the node can encode number of packet needed for broadcast in the form of one linear combination and reduce the number of broadcast transmissions. The broadcast scheme that one source (or multiple sources) can transmit packet to other nodes is called single-source broadcasting (or multiple-source broadcasting). [62] proposes a broadcasting scheme with RLNC

while [63] proposes a deterministic approach to multiple-source broadcasting. Flooding is very robust to packet losses because each node has high chances of receiving broadcast packets. As a result, network coding can help to reduce the number of broadcast transmissions but data is delivered with the low probability of packet loss ([64, 65, 66]).

2.7.1.3 Coding-aware routing metric

How the traffic flows are steered through the network to reduce the transmission delay and maximize network capacity is an interesting application of network coding. Network coding is used as a routing metric to decide routing decisions. One proposition in this is Routing with Opportunistically Coded Exchanges (ROCX) [39], which gain much attention. Following works are extended the idea of ROCX to solve more complicated problems: Interference-aware ROCX (IROCX) [40], Topology control ROCX (TC-IROCX) [41], DCAR [4], [42], Adaptive Routing in Dynamic Ad Hoc Networks (AROD) [43], Coding-Aware Multi-path Routing (CAMP) [44], Coding-aware real-time routing (CARTR) [45], Self-recommendation Coding-Aware Routing (SCAR) [46], Coding-aware opportunistic routing (CORE) [47], [48], [49], etc. Many routing protocols (e.g., Ad hoc On-Demand Distance Vector (AODV) [67], Dynamic Source Routing (DSR) [68], Dynamic Destination-Sequenced Distance-Vector (DSDV) [69]) are extended to support the coding-aware routing metric. Essentially, coding-aware routing metric is find the transmission path that provides the maximum number of encoder to encode the number of native packets as many as possible. However, the number of native packets encoded together is small, as written in [70]:

"In a generic coding structure, there is one coding node and n coding flows whose packets can be encoded by the coding node. Clearly, the encoding number in this coding structure is at most n. In [2], the authors assume that n is unbounded because there can be infinite nodes surrounding the coding node. However, we show that due to physical limitations of the wireless channel and the geometrical constraints for proper decoding, n is indeed upper bounded by a small constant determined by the physical layer parameters. In particular, the upper bound is $\Theta((\frac{r}{\delta})^2)$ *for 2D space and* $\Theta((\frac{r}{\delta})^3)$ *for 3D space, where r is the "reliable transmission range" between the a transmitter and a receiver and* δ *is the channel parameter such that the receiver with a distance larger than* $r + \delta$ *can only hear the transmission with a very low probability. For more details, please refer to [71]"*

2.7.1.4 Opportunistic routing

Opportunistic routing has recently emerged as a mechanism to resist losses in the error-prone environment, like wireless networks. In current network systems, conventional routing chooses the next hop per packet before any transmissions. However, when the links are lossy, the probability of packet transmission is be very low, leading to the performance degradation. In contrast, opportunistic routing suggests any node obtaining the packets can participate into forwarding packets to destination. Let's consider a two-hop transmission from A to B (Fig 2-11). We have 5 forwarders $C_i, i \in [1, 5]$. The probability of successful packet transmission on the link between A and C_i is around 20% while the probability of successful packet transmission on the link between C_i and B is 100%. Rapidly, we realize the traditional routing can only offer the sending rate 1 successful packet transmission every 5 packets despite which nexthop should be picked. However, if opportunistic routing is used, the successful rate should be 100% because packet is handled by any of 5 forwarders, which reduces the probability of packet loss from 80% to 0%. Certainly, this should be an ideal case because the practical performance may be affected by many other factors. Nevertheless, the benefit of opportunistic routing is not deniable.

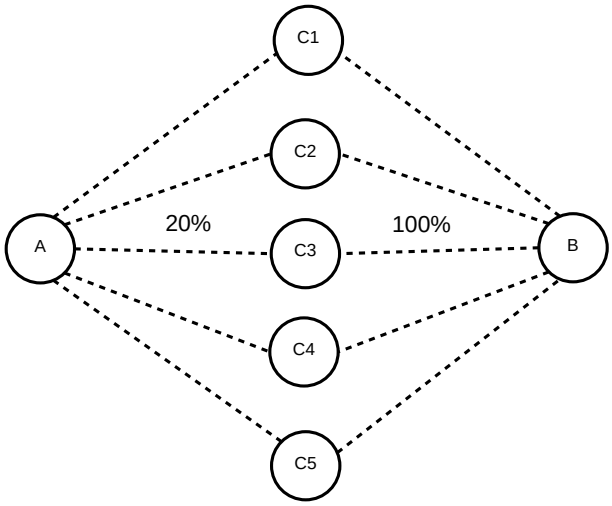


Figure 2-11: An example of opportunistic routing

The very first proposition of opportunistic routing is Extremely Opportunistic Routing (ExOR) [72, 73] even though ExOR and some successors (e.g., Simple Opportunistic Adaptive Routing (SOAR) [74]) do not absorb network coding into its mechanism. ExOR is a cross-layer modification between datalink layer and routing

layer. However, the idea "any nodes as the next hop" can be prone to many challenges. Multiple nodes overhear the packet and try to forward it. It leads into a broadcast loop if nodes farther away from destination are so eager to act. Therefore, how to find the suitable nodes and how they cooperate to forward the traffic effectively (no broadcast loop, no duplication) are the main questions with opportunistic routing. ExOR provides a highly structured scheduler on the routing access queried by MAC layer. ExOR builds up a forwarder list and sort the forwarders by their probability of packet transmission, from the highest to the lowest. Forwarders need to transmit in that order and one at a time. All nodes try to overhear the transmission to avoid the duplication. If the transmission is not successful, the next forwarder in line will be allowed transmitting the packet. The process is repeated until the packet reaches the destination. Even though this kind of routing is quite interesting, the highly structured scheduler from ExOR is indeed complicated, quite hard to extend to support more kind of traffic (e.g., multicast) or exploit the medium spatial-reuse, i.e., multiple non-interference packets transmitted at the same time.

To suppress the limitations of ExOR, MAC-independent Opportunistic Routing and Encoding (MORE) [5] introduces the effect of network coding into opportunistic routing, which significantly improves the network performance. MORE, which uses RLNC to transmit packets of a flow in the form of multiple linear combinations, gains two important aspects: i) removing the complexed design of the highly structured scheduler of ExOR and ii) no routing information exchanged among forwarders to decide which packet is prioritized for forwarding. Many researches are done to extended the idea of MORE, such as Coding in Opportunistic Routing (CodeOR) [75], Online Opportunistic Network Coding (SlideOR) [76], Multipath TCP with Network Coding (CoMP) [77], Network Coded Multipath to Support TCP (CodeMP) [78], A Game Theoretic Framework for Wireless Multipath Network Coding (DICE) [79], and Cumulative Coded Acknowledgments (CCACK) [80].

2.7.2 Ad-hoc sensor networks

A sensor network deploys sensor nodes which are simple and cheap devices to collect a specific kind of information (e.g., the weather-like data) at a located area. During the exchange of information from many tiny sources to a sink node (probably the data center), using the RLNC to steer the traffic and resist losses is a good solution [81], [82]. In dense sensor networks with many of tiny sensor terminals, the multi-hop path from many sources (the sensors) to one destination (the sink) always exists. With RLNC, we don't need to find the path but data can be combined and broadcasted. Even though a specific protocol is not given, the authors from [82] conclude that

RLNC can provide the benefit more than the conventional way. Other works on applying RLNC into ad-hoc sensor network, the authors from [83] propose Partial Network Coding (PNC). This implementation of RLNC which removes all out of date coded packets with a specific time constraint, defined by the wireless sensor network, shows good performances in a real environment. However, the traffic in PNC needs to be application-specific. AdapCode [64] provides an extra gain by network coding by coding traffic from upstream (e.g., updates from the control center) to all leaves. Indeed, this implementation helps to increase the performance but the data from upstream is not quite mainstream in wireless sensor network. Similarly, [84] provide a link-by-link feedback mechanism in wireless sensor networks to save up the energy consumption.

To this very end, Zigbee Index-coding (ZInC) [85] suggests that applying the RLNC as "the whole package" from others like wireless networks will not be a well-considered solution. ZInC is an index coding scheme that is tailored by the constraints of wireless sensor networks. Generally, ZInC is an enhancement of ZigBee, the most widespread technology for sensor networking and wireless personal area networks. ZInC is designed to focus on coding the data from downstream to upstream which dominates the bandwidth of wireless sensor network. The authors notices even though the packet data in sensor networks are quite small, the overhead (e.g., packet header) is excessive. They proposes in the many-to-one wireless sensor network, each source data can dominate a part of data packet (i.e, only few bits or up to few bytes) and this part can help to identify the source, hence, it's called the index coding scheme. Through each hierarchical node, the overhead is significantly reduced while data are still aggregated to the upstream. Consequently, ZInC improves the network performance.

2.7.3 Peer to peer (P2P) file distribution

In a P2P content distribution network (e.g., BitTorrent [86]), a *tracker*, the one obtaining the file, becomes the first *seeder*, the node which splits the sharing file into many small blocks and distributes them to a swarm of nodes which needs the files (called *leechers*). Because the swarm are located randomly around the Internet, the structure of BitTorrent network is also random. Some leechers can get the blocks directly from seeder while others can't. Instead, leechers in the swarm start exchanging the blocks after they get sufficient new blocks from their neighbours. The distribution is spread around until the leechers receive the whole file, becoming the seeders themselves. Each node in the BitTorent network can join and leave freely. Avalanche [87] is an enhancement of BitTorrent that adapts the RLNC into the BitTorrent system to

resist the loss of blocks more effectively. Network coding helps Avalanche to reduce the knowledge of block distribution throughout the network by simply broadcasting the linear coding combinations instead. Each leecher starts transmitting the linear coding combinations generated by their own blocks whenever they know there is new information (i.e., new blocks) needed exchanging among them. Because each block is around hundreds of kilobytes, the packet overhead (e.g., the coefficient list attached to each coded packet...) can be ignored for the overhead.

More similar works [88, 89, 90, 91, 92, 93] follow the similar discipline to solve many different challenges arise with network coding in distributed systems.

2.7.4 Network security

There are many applications of network coding for the security:

- Resistance against eavesdropping
- Detection of malicious nodes in networks
- Resistance against jamming attacks

By the nature of network coding, data is mixed, transformed and widespread, making it harder to eavesdrop [94, 95, 96]. One important note is in network security, data can be combined with random information in order to reduce the chance of unauthorized data recovery. Moreover, the authors of [97] suggests that network coding itself can support the security if in the network, we ensure all parts of information (i.e., the sufficient number of linear network coding combinations) are only obtained by the dedicated recipient. Consequently, eavesdroppers cannot restore data in this case. The modification of coefficient list is also used to improve the mechanism [98]. The intermediate nodes are able to modify the coded packets (i.e., re-encode the packets) with some random information, thus, the attackers do not apprehend correct data. For the jamming attacks, the authors from [99] suggest the similar solutions.

2.8 Problem statement

After looking through the current research progress of network coding, the challenges which we are facing in the wireless environment are solidified to

- In inter-flow network coding, how do we propose a generalized coding condition to maximize network capacity?

- In intra-flow network coding, how do we propose an effective coding scheme which is robust to random losses to provide the transmission reliability?

We introduce an inter-flow network coding system to improve the throughput and another intra-flow network coding to provide the transmission reliability. Our approach is straight-forward: for inter-flow network coding, we will re-design the coding condition while for intra-flow network coding, we suggest the new coding schemes. Via simulations, we show the promising results. Details of our contributions will be lined out in next chapters.

2.9 Chapter conclusion

In this chapter, we present what network coding is and how it can be used to maximize network capacity. Two important benefits: throughput enhancement and robustness to random losses to provide transmission reliability are also described. These two benefits are derived directly from two branches of network coding: inter-flow network coding (ONC) and intra-flow network coding (RLNC). The state of the art of network coding in current wireless networks is shown, along with descriptions of typical works (i.e, inter-flow network coding systems: COPE, BEND and DCAR; intra-flow network coding systems: Batch Coding, Pipeline Coding and TCP/NC). Main applications of network coding are discussed with references to notable works in each field. We conclude the chapter with our problem statement.

Chapter 3

Inter-flow network coding

This chapter provides our contributions on inter-flow network coding or more precisely, Opportunistic Network Coding (ONC). In ONC, the coding condition is a key factor which allows an ONC system to know if the encoded packet can be decoded successfully later, so that only the traffic flows satisfying the coding condition are encoded and decoded to improve network performance. We briefly introduce what ONC is in previous Chapters 1 and 2. We will use the definitions of traffic flow, coding chance, encoder and decoder as below:

Definition 2 ***Traffic flow*** *is a subgraph of the network, representing a way to get from a source to a destination by traversing links in the network.*

Definition 3 ***Coding chance*** *is the ability of intermediate nodes to detect if traffic flows are feasible to be coded under a specific coding condition declared by each opportunistic coding system. The higher coding chance is found, the more network performance can improve.*

Definition 4 ***Encoder*** *is the node which receives packets from multiple codable traffic flows and performs the coding process to create coded packets for transmissions.*

Definition 5 ***Decoder*** *is the node which performs the decoding process on coded packets, generated by the encoder, to retrieve the desired packets successfully.*

We propose 3 contributions: DODE, DODEX and DODEX+ which are briefly introduced in Chapter 1. Next sections 3.1, 3.2 and 3.3 will detail their coding conditions and illustrate how they can gain more throughput than COPE, BEND and DCAR. Section 3.4 shows the design of our propositions. Section 3.5 shows the simulation configurations and results. Fig 3-1 illustrates our work on inter-flow network coding.

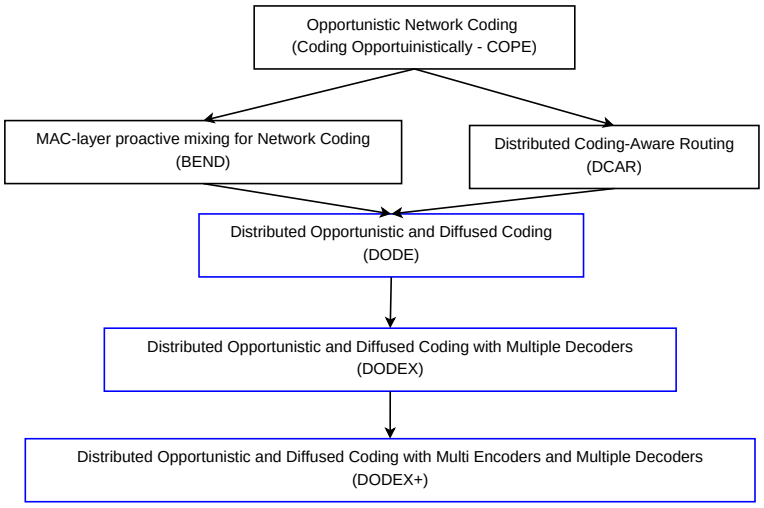


Figure 3-1: Opportunistic Network Coding schemes

3.1 Distributed and Diffused Encoding (DODE)

COPE, one of the pioneers of inter-flow network coding, has two limitations: (i) the coding chances inevitably depend on the established route, (ii) traffic is strictly encoded based on the two-hop coding pattern. We propose our novel and enhanced network coding architecture, Distributed Opportunistic Diffused Encoding (DODE). DODE combines (i) the diffused gain from BEND and (ii) the generalized coding condition from DCAR to completely solve the limitations present in the three previous propositions.

COPE suggests codable traffic flows being the flows which traverse through some similar intermediate nodes on the routing path. These nodes will become the encoders generating the encoded packets. Encoders and decoders must be neighbors to each others and all of them are on the routing path of the traffic flows. This will ensure the encoded packets are handled by correct nodes. Otherwise, traffic is forwarded normally. Because of the nature of wireless transmission, overheard packets can also be exploited for the decoding process. In other words, not only the next hops of codable flows are the candidates to be decoders but also their one-hop neighbors. By giving the enhanced coding condition, DCAR and BEND increase the chances of coding flows and outperform COPE. The proposition of BEND is an adjust to which nodes can be encoders. Encoders in BEND are nodes which receive or overhear packets from codable traffic flows. Because the number of nodes possible to be coders increases, it also increases the coding chances, thus, giving the improvement from BEND over COPE. DCAR finds another approach to redefine which are the decoders.

Decoders in DCAR are not limited to one-hop neighbor to coders but any nodes which stay on the routing path and obtain sufficient extra information to perform the decoding process successfully.

We propose DODE as a combination of all features from COPE (opportunistic listening and coding), DCAR (the general coding condition) and BEND (the diffused gain). We can see the generalization of coding condition from COPE to its successors (Fig 3-2). At first, COPE requires encoders and decoders to be one-hop neighbors because the coding condition is easily checked to be met via the information existed in exchanged messages of current routing protocols (e.g., DSDV). Next, BEND allows overhearing nodes to be encoders also. The overheard packets in BEND is also re-used for coding process, not only decoding process like in COPE. Finally, DCAR proposes that coders and decoders might not be one-hop neighbors by showing a specific case that coding chances could be found.

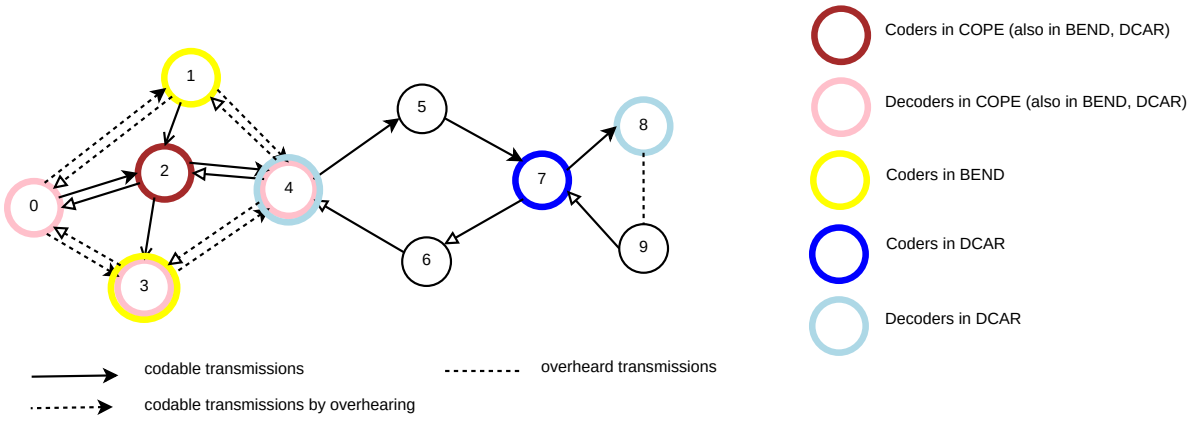


Figure 3-2: Generalizing the coding condition from COPE to DCAR

Fig 3-2 illustrates "the evolution" of coding condition from all previous works to DODE. Consider three codable flows $F_1(0 \to 2 \to 4 \to 5 \to 7 \to 8)$, $F_2(9 \to 7 \to 6 \to 4 \to 2 \to 0)$ and $F_3(1 \to 2 \to 3)$. At first, with COPE, only one coding chance is found at node 2, encoder is node 2 and nodes $0, 3$ and 4 are decoders. In BEND, more coding chances are found at coders 1 and 3. Finally, DCAR suggests another coding chance found at node 7, encoder is 7 and decoders are 4 and 8.

We can conclude that two extensions given by BEND and DCAR from COPE are possible to be *combined* together to create an enhanced network coding system. Consequently, DODE is designed to increase the number of possible coders and decoders, hence, improving the network performance.

3.1.1 Coding chance improvement

Let's consider the topology depicted in the Figure 3-3 with two flows $F_1(1 \rightarrow 2 \rightarrow 3 \rightarrow 5)$ and $F_2(5 \rightarrow 6 \rightarrow 4)$. COPE, BEND and DCAR cannot find the coding chance in this example. COPE and DCAR cannot be applied in this case because the two flows are not intersected. BEND cannot be applied because nodes 2 and 4 are not neighbors. However, it is easy to detect that it is possible to code the two flows with node 3 as encoder, and nodes 4 and 5 as decoders.

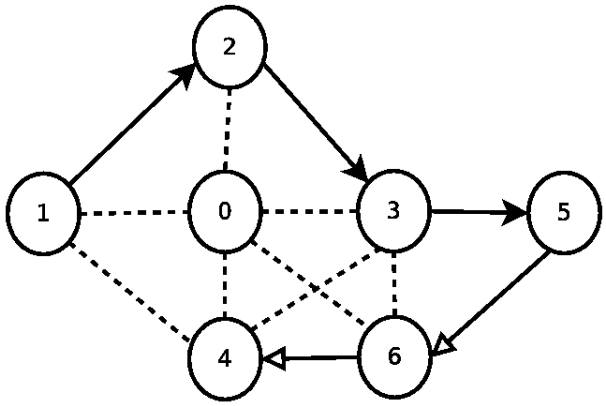


Figure 3-3: The missing coding chance scenario with COPE, BEND and DCAR

3.1.2 Generalized coding condition of DODE

From this observation, the coding condition of DODE combined from DCAR and BEND is presented by Condition 6 and Condition 5 (i.e, the coding condition of multiple flows of BEND in Section 2.5.3). DODE only replaces the definition of the decoder (D_{ij} and D_{ji}) of BEND by the definition of the decoder of DCAR and introduces the Condition 6. Condition 6 states that two native flows F_i and F_j ($i \neq j$) are considered as codable to each other at an encoder group C_{ij} if and only if there exist two nodes $v_{f_{F_i}}, v_{g_{F_j}} \in L_{ij}$ such that one downstream node of $v_{f_{F_i}}$ (or $v_{g_{F_j}}$) on flow F_i (or on flow F_j) is the upstream node of $v_{g_{F_j}}$ (or $v_{f_{F_i}}$) on flow F_j (or on flow F_i) or at least is the neighbor of that upstream node. L_{ij} is the set of nodes u on flows F_i and nodes t on flow F_j such that there are links between u and t or $u = t$. C_{ij}, the encoder group that generates coded packets between two flows F_i and F_j, is the set of neighbors of four nodes $v_{f-1_{F_i}}, v_{g-1_{F_j}}, v_{f+1_{F_i}}$ and $v_{g+1_{F_j}}$. D_{ij} (or D_{ji}) is the set of decoders of flow F_i (or flow F_j) to remove the native packet of flow F_j (or flow F_i) from the coded packet generated by the encoder group C_{ij}. The coding condition of multiple flows is inherited from BEND (Condition 5).

Condition 6 *Two native flows F_i and F_j $(i \neq j)$ are considered as codable to each other at a group of nodes $C_{ij} \subset L_{ij} = \{\forall u \in V_{F_i}, \forall t \in V_{F_j} | u = t \bigvee (u,t) \in E\}$ if and only if:*

- $\exists v_{f_{F_i}}, \exists v_{g_{F_j}} \in L_{ij}$.
- $|D_{ij}| = |Dn(v_{f_{F_i}}, F_i) \bigcap (Up(v_{g_{F_j}}, F_j) \bigcup N(Up(v_{g_{F_j}}, F_j)))| = 1$.
- $|D_{ji}| = |Dn(v_{g_{F_j}}, F_j) \bigcap (Up(v_{f_{F_i}}, F_i) \bigcup N(Up(v_{f_{F_i}}, F_i)))| = 1$.
- $|C_{ij}| = |N(v_{f-1_{F_i}}) \bigcap N(v_{g-1_{F_j}}) \bigcap N(v_{f+1_{F_i}}) \bigcap N(v_{g+1_{F_j}})| \geq 1$

For example, let's reconsider the network situation in Fig 3-3. As we check $L_{12} = \{1, 4, 3, 6, 5\}$, there exist two nodes $3, 6 \in L_{12}$ such that $Dn(3, F_1) \ni 5 \in Up(6, F_2)$ and $Dn(6, F_2) \ni 4 \in N(1), 1 \in Up(3, F_1)$. However, only 3 is the encoder because $3 \in C_{12} = N(2) \bigcap N(5) \bigcap N(4) = \{3\}$. As there are two flows coded to each other, $C_1 = C_2 = C_{12} = C_{21} = \{3\}$, $D_1 = \{5\}$ and $D_2 = \{4\}$.

With the new designed coding condition, DODE can capture more coding chances to enhance the network utilization.

3.2 Distributed and Diffused Encoding with Multiple Decoders (DODEX)

For better performance, the aim of network coding systems COPE, BEND, DCAR and DODE is trying to find more and more coding chances in network topologies. However, they restricted the finding within a simple rule "a pair of encoder and decoder". For every coded packet sent on the traffic flow, we always have one encoder and one decoder. We redesign the coding condition in DODEX for each coded packet at the encoder, which allows multiple decoders in order to find more coding chance and improve network utilization. After one encoder creates the coded packets, multiple decoders can share the decoding process to retrieve the desired packet for the destination. More coding chances can be found, leading to an improved network performance.

We would like to discuss a bit the core idea of decoupling the pair of encoder and decoder. Actually, the idea goes straight from a limitation that how an intermediate nodes obtains sufficient extra information to become a decoder, especially, the multiple packet coding (> 2) case. Practically, the number of codable flows coded together are greatly constrained and is a small number [71]. Multiple packet coding

(> 2) barely exists except the ideal wheel topology (Fig 3-4). In the wheel topology, one node is a neighbor of other nodes except its opposite node which is the node to receive its sending packet. For example, in Fig 3-4, node 5 is a neighbor of nodes $0, 2, 3, 4, 6, 7$ and 8 except node 1. Wheel topology in Fig 3-4 allows maximum 8-packet coding (there are 8 nodes on the wheel and each node transmits packets to its opposite node).

With the concept of "one coder- multiple decoders", nodes obtaining insufficient (or partly) extra information can co-operate together to perform the decoding process. Consequently, more coding chances will be found. We will look into the details in next section.

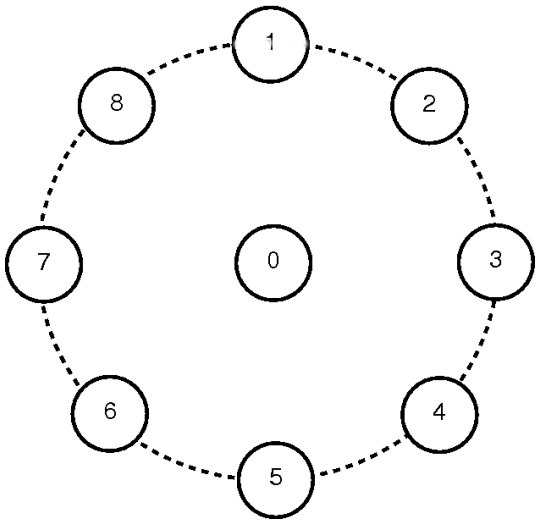


Figure 3-4: Wheel topology

3.2.1 Coding chance improvement

In DODEX, encoders can combine many native packets for transmission and multiple decoders will recover the desired packet. Actually, because multiple nodes insufficiently obtain native packets of other flows to perform the decoding process by itself. These nodes *partially decode* the coded packet and transmit that "partially"-decoded packet to next decoders. The decoding process is repeated on the next decoder until the coded packet reach the last decoder which decodes the coded packet to get the original native packet successfully. In other words, the decoding process is sharing among the nodes. The coded packets are not maintained *"untouched"* but eventually *"peeled out"* the unnecessary packets until the original native packet reaches the destination. Consider Fig 3-5, we can see three flows $F_1(0 \rightarrow 3 \rightarrow 4 \rightarrow 5)$, $F_2(2 \rightarrow 3 \rightarrow 6)$

and $F_3(8 \rightarrow 7 \rightarrow 3 \rightarrow 1)$ can be coded together but on the flow F_1 there are two decoders 4 (for leaving out packet p_2) and 5 (removing p_3 to get the desired p_1). With previous implementation, only 2-packet coding can be found instead of 3-packet coding like our proposition DODEX. Because DODEX inherits the advantage of general coding condition from DCAR, the encoder will know which next hops can perform the decoding processes to remove the packets from different paths and obtain the desired one successfully.

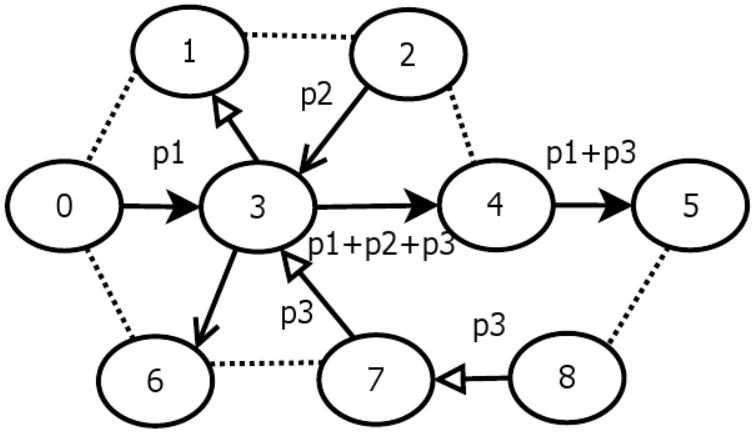


Figure 3-5: Coding chance improvement with DODEX

3.2.2 Generalized coding condition of DODEX

The coding condition of DODEX can be written as in Condition 6 (i.e, the coding condition of two flows of DODE in Section 3.1.2) and Condition 7. As DODEX introduces a new concept "one coder - multiple decoders", DODEX keeps intact the coding condition between two flows of DODE (Condition 6) and changes the coding condition of multiple flows (Condition 7). Condition 7 states: each pair of flows must be codable and intersected at an encoder group C_{ij}, which contains the encoders of coding packets between two flow F_i and F_j. Besides, for each flow, the number of decoders of a flow F_i can be more than 1 ($|D_i| \geq 1$), and C_i is the encoder group containing all encoders to create coded packets of multiple flows $\forall F_i \in \mathbb{F}$, which must be the intersection of all C_{ij}: $C_i = \bigcap_{j=1, j \neq i}^{n} C_{ij}$ ($|C_i| \geq 1$).

Condition 7 *Native flows belonging to a set of flows $\mathbb{F} = \{F_k | k \in [1, n]\}$ (n is the number of flows) are considered as codable together at a group of nodes $\forall F_i \in \mathbb{F}, C_i =$*

$\bigcap_{j=1,j\neq i}^{n} C_{ij}$ *if and only if:*

- $\forall F_i, F_j \in \mathbb{F}(i \neq j) \mid F_i$ *and* F_j *are considered as codable to each other at the group of node* C_{ij}.
- $\forall F_i \in \mathbb{F}, |D_i| = |\bigcup_{j=1,j\neq i}^{|\mathbb{F}|} D_{ij}| \geq 1$.
- $|C_i| \geq 1$

For example, let's consider three flows $F_1(0 \to 3 \to 4 \to 5)$, $F_2(2 \to 3 \to 6)$ and $F_3(8 \to 7 \to 3 \to 1)$ in Fig 3-5. Let's check Condition 6 and Condition 7:

- $L_{12} = \{0, 3, 4, 2, 6\}, \exists 3 \in L_{12} | Dn(3, F_1) \ni 4 \in N(2), 2 \in Up(3, F_2) \bigwedge Dn(3, F_2) \ni 6 \in N(0), 0 \in Up(3, F_1) \Longrightarrow C_{12} = C_{21} = \{3\}$, $D_{12} = \{4\}$ and $D_{21} = \{6\}$
- $L_{13} = \{0, 1, 3, 7, 5, 8\}, \exists 3 \in L_{13} | Dn(3, F_1) \ni 5 \in N(8), 8 \in Up(3, F_3) \bigwedge Dn(3, F_3) \ni 1 \in N(0), 0 \in Up(3, F_1) \Longrightarrow C_{13} = C_{31} = \{3\}$, $D_{13} = \{5\}$ and $D_{31} = \{1\}$
- $L_{23} = \{1, 2, 3, 6, 7\}, \exists 3 \in L_{23} | Dn(3, F_2) \ni 6 \in N(7), 7 \in Up(3, F_3) \bigwedge Dn(3, F_3) \ni 1 \in N(2), 2 \in Up(3, F_2) \Longrightarrow C_{23} = C_{32} = \{3\}$, $D_{23} = \{6\}$ and $D_{32} = \{1\}$
- $\Longrightarrow C_1 = C_2 = C_3 = \{3\}$, $D_1 = D_{12} \bigcup D_{13} = \{4, 5\}$, $D_2 = D_{21} \bigcup D_{23} = \{6\}$ and $D_3 = D_{31} \bigcup D_{32} = \{1\}$

3.3 Distributed and Diffused Encoding with Multiple Encoders and Multiple Decoders (DODEX+)

We improve the network performance of ONC by extending the coding condition. In this final phase, we complete the coding condition by presenting the idea of re-encoding in ONC for the throughput enhancement. Normally, in DCAR, DODE or DODEX, the coded packets are only forwarded until they reach the decoders. The coded traffic is untouched even though there may be any coding chances found. In the complete form of general coding condition of ONC, we propose "the re-encoding" or multiple encoders on the same traffic: for the coded packets, the re-encoding can be performed if the coding condition is met again with this traffic. We extend DODEX to Distributed and Diffused Encoding with Multiple Coders and Multiple Decoders

(DODEX+). With this, more coding chances are found and the performance will be improved.

The idea is totally different from the re-encoding in RLNC [7]. The re-encoding in RLNC is to avoid the non-innovative transmission when the intermediate nodes, especially in the opportunistic routing (e.g., MORE [5]), need to forward the packets from the source to the destination. If they simply forward the coded data, it could lead to some non-innovative and wasted transmissions. On the other hand, the re-encoding will ensure that packets are coded and innovative when they arrive at the destination.

In contrast, our idea about the re-encoding is proposed to complete the generalized coding condition in ONC. The heart of the idea is that all traffic can be coded together whenever there are any coding chances. Because checking coding condition of ONC is quite sensitive over transmission time and delay, coding packets which they are native or coded will help to maximize network capacity.

3.3.1 Coding chance improvement

Consider the situation in Fig 3-6, the coded packet p_1+p_2 at node 5 is only forwarded. Even though we notice that flow F_3 can be coded with flow F_2 at node 5, the coding is only performed between the native packets of F_2 and F_3. The limitation is not quite necessary as we easily notice the coded packet $p_1 + p_2$ and native packet p_3 can be coded together to produce $p_1 + p_2 + p_3$. Decoders $6, 2$ can obtain the desired packet successfully (e.g., node 6 gets p_2 because it has p_1 by overhearing and p_3 is its own).

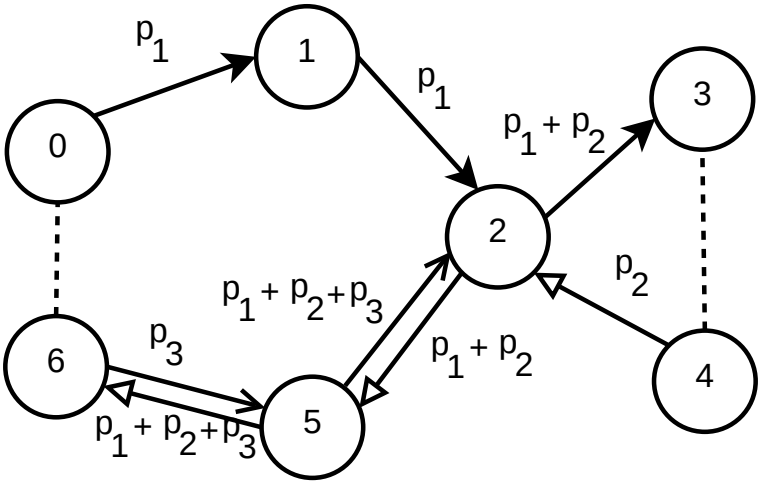


Figure 3-6: Coding chance improvement with DODEX+

3.3.2 Generalized coding condition of DODEX+

The generalized coding coding of DODEX+ is presented in Condition 8 and Condition 9. The coding condition between two flows and the coding condition of multiple flows are kept intact from Condition 6 and Condition 7 but extended to support coded flows. Condition 8 states: two native or coded flows F_i and F_j ($i \neq j$) are considered as codable to each other at an encoder group C_{ij} if and only if there exist two nodes $v_{f_{F_i}}, v_{g_{F_j}} \in L_{ij}$ such that one downstream node of $v_{f_{F_i}}$ (or $v_{g_{F_j}}$) on flow F_i (or on flow F_j) is the upstream node of $v_{g_{F_j}}$ (or $v_{f_{F_i}}$) on flow F_j (or on flow F_i) or at least is the neighbor of that upstream node. L_{ij} is the set of nodes u on flows F_i and nodes t on flow F_j such that there are links between u and t or $u = t$. C_{ij}, the encoder group that generates coded packets between two flows F_i and F_j, is the set of neighbors of four nodes $v_{f-1_{F_i}}, v_{g-1_{F_j}}, v_{f+1_{F_i}}$ and $v_{g+1_{F_j}}$. D_{ij} (or D_{ji}) is the set of decoders of flow F_i (or flow F_j) to remove the packet of flow F_j (or flow F_i) from the coded packet generated by the encoder group C_{ij}.

Condition 8 *Two native or coded flows F_i and F_j ($i \neq j$) are considered as codable to each other at a group of nodes $C_{ij} \subset L_{ij} = \{\forall u \in V_{F_i}, \forall t \in V_{F_j} | u = t \bigvee (u,t) \in E\}$ if and only if:*

- $\exists v_{f_{F_i}}, \exists v_{g_{F_j}} \in L_{ij}$.
- $|D_{ij}| = |Dn(v_{f_{F_i}}, F_i) \bigcap (Up(v_{g_{F_j}}, F_j) \bigcup N(Up(v_{g_{F_j}}, F_j)))| = 1$.
- $|D_{ji}| = |Dn(v_{g_{F_j}}, F_j) \bigcap (Up(v_{f_{F_i}}, F_i) \bigcup N(Up(v_{f_{F_i}}, F_i)))| = 1$.
- $|C_{ij}| = |N(v_{f-1_{F_i}}) \bigcap N(v_{g-1_{F_j}}) \bigcap N(v_{f+1_{F_i}}) \bigcap N(v_{g+1_{F_j}})| \geq 1$

Condition 9 states: each pair of native or coded flows must be codable and intersected at an encoder group C_{ij}, which contains the encoders of coding packets between two flow F_i and F_j. Besides, for each flow, the number of decoders of a flow F_i is can be more than 1 ($|D_i| \geq 1$), and C_i is the encoder group containing all encoders to create coded packets of multiple flows $\forall F_i \in \mathbb{F}$, which must be the intersection of all C_{ij}: $C_i = \bigcap_{j=1, j\neq i}^{n} C_{ij}$ ($|C_i| \geq 1$).

Condition 9 *Native or coded flows belonging to a set of flows $\mathbb{F} = \{F_k | k \in [1, n]\}$ (n is the number of flows) are considered as codable together at a group of nodes $\forall F_i \in \mathbb{F}, C_i = \bigcap_{j=1, j\neq i}^{n} C_{ij}$ if and only if:*

- $\forall F_i, F_j \in \mathbb{F}(i \neq j) \mid F_i$ *and* F_j *are considered as codable to each other at the group of node* C_{ij}.
- $\forall F_i \in \mathbb{F}, |D_i| = |\bigcup_{j=1,j\neq i}^{|\mathbb{F}|} D_{ij}| \geq 1$.
- $|C_i| \geq 1$

One important note is that the re-encoding cannot happen in COPE or BEND due to their two hop coding pattern, where decoders are the one-hop neighbors of coders and rip off the unnecessary packets immediately. Consider the chain topology in Fig 3-7 with two flow $F_1(0 \rightarrow 3)$ and $F_2(3 \rightarrow 0)$, at encoders 1 and 2, the re-encoding of coded packets does not happen because for every encoded packet received, the decoding process will occur first to extract the native packets. Afterwards, another separate encoding process may be performed. For example, assume at node 1 there are two packets p_1 and p_2 coded together to produce the coded packet $p_1 + p_2$. At node 2, the coded packet will be decoded to get p_1. This packet will be encoded with $p_{2'}$, another packet from flow F_2 of node 3. As we can see, there no re-encoded packet $p_1 + p_2 + p_{2'}$ because next hop (node 2) will act as a decoder to remove the non-intended packet.

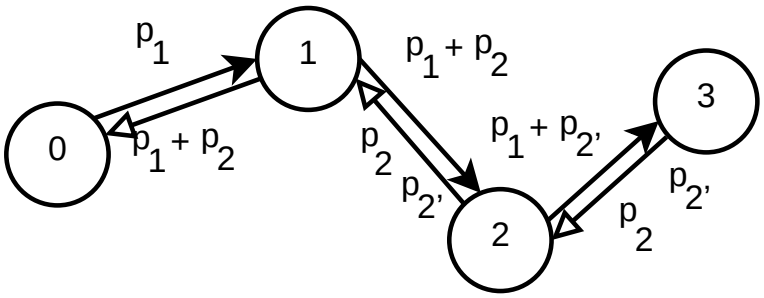


Figure 3-7: No re-encoding in COPE or BEND

3.4 Design

3.4.1 Node architecture

To perform the coding condition above, DODE, DODEX and DODEX+ have to collect the information *"who sends what"* on the transmission path of each flow and detect the coding chances based on that. They store and forward native packets,

generate and forward coded packets when coding chances exist. Particularly, each node changes its behaviors to handle the native and coded traffic. To do this, they need to collects the network coding information. In DODE, DODEX and DODEX+, the network coding information includes the neighbor list, the source routing list and the decoder list (details in Sections 3.4.1.1 and 3.4.1.2). The network coding information is required by nodes to check the coding condition over the packets in the queuing system. We also modify Destination-Sequenced Distance-Vector Routing (DSDV) [69], a well-known table-driven routing scheme for wireless networks, so that the network coding information is broadcast over the network via the routing messages. As ONC has to know "the neighbor state" (or the network topology) to check the coding condition and make coding decisions, we use the table-driven routing protocol like DSDV for its simpleness in the mechanism and easy modification (e.g., full routing table updates to broadcast network coding information). AODV or OLSR can be also used to broadcast network coding information but more complicated modification may need to be done.

3.4.1.1 Neighbor list and source routing list

Each node maintains its neighbor list and also its neighbors' neighbor lists. Besides the original information of DSDV, each route entry in the update message will also carry the list of nodes constituting the route and their neighbors. The neighbor list is obtained by collecting the senders' addresses of DSDV messages. Before broadcasting out the DSDV messages, the sender adds its address to the current list of routing path in the entry. After receiving the routing updates from neighbors, each node will process the routing update messages as the original DSDV routing protocol does and cache the network coding information. The receiver, in turn, will broadcast the update after adding its network coding information (its neighbor list). The process will be repeated per update. Thus, all nodes in the network gradually acquire enough information to perform the correct coding. The source routing list is built through a the similar procedure.

3.4.1.2 Decoder list

Because multiple decoders need to co-operate to perform the decoding, coded packets broadcasted by coders have to maintain a list storing the addresses of decoders. The list is to be computed when encoder perform the encoding process. The result is the addresses of decoders. The encoder will add the decoder list into coded packets for transmissions. Each node will check if it is the decoder by comparing its address with

the addresses in the list. The node then performs the decoding process to remove as much as possible the extra information and forward the remained if possible.

3.4.1.3 Queuing system

DODE, DODEX and DODEX+ use four different queues: Q_{native} for storing the native packets, Q_{ovrhrd} for storing the overheard packets, $Q_{codable}$ for the linked lists of native packets that can be combined together to create coded packets, and Q_{coded} for storing coded packets that need forwarding or encoding (only DODEX and DODEX+ need this queue). Figure 3-8 shows a representation of queues. The algorithm to find the coding chances is straight-forward: packet received from data-link layer is checked with linked list of packets stored in $Q_{codable}$ first, and then existed packets in Q_{native}, Q_{ovrhrd} and Q_{coded}. If there is a match, the packet will be added into the link list of $Q_{codable}$ or a new link list of both matched packets is created and moved into $Q_{codable}$. Each queue is allowed to forward the packets down to physical layer for transmission in a round robin fashion. The algoritm to check the coding condition is polynomial in time and equal to $\Theta(|Q_{codable}| + |Q_{native}| + |Q_{ovrhrd}| + |Q_{coded}|)$.

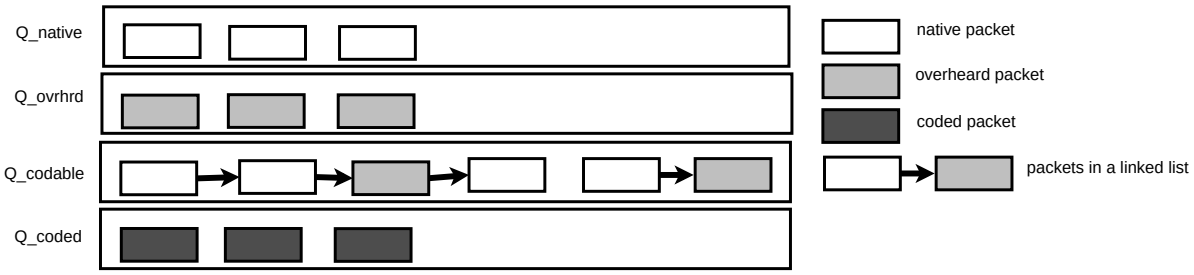


Figure 3-8: Queuing system

3.4.2 Routing metric with coding chance discovery for DSDV protocol - SPENM

Routing decision can affect the efficiency of network coding. Let's recall the example shown in Figure 3-3. As the routing protocol has decided the path $1 \to 2 \to 3 \to 5$ for flow F_1 and the path $5 \to 3 \to 4$ for flow F_2, BEND cannot be applied because node 2 is not a neighbor of node 4. If the path of flow F_1 were $1 \to 0 \to 3 \to 5$, BEND can be used. It means that if the routing path is correctly chosen so that the next hop is the node with the highest number of neighbors, we can greatly increase the chance to fulfil the coding condition. From this observation, we introduce "the Shortest Path

with Enriched Neighborhood routing Metric" (SPENM) with a simple discipline: for equal shortest paths to a destination, we will use the one with the highest number of neighbors for intermediate nodes.

With this modified DSDV and SPENM, we gain two benefits for the *coding+routing discovery process*. First, we only do a little modification to DSDV. We reuse the original mechanism to exchange more routing information needed for the network coding functionality among nodes. Second, DODE provides SPENM which does not need a long convergence to find coding chances. We can view the SPENM as a "guide" to keep the traffic flows going through the intermediate nodes with the most potential coding chances in DODE network system. For example, in Fig 3-9, without SPENM, traffic flows F_1 and F_2 can choose the transmission paths which do not gain the highest number of neighbors for intermediate nodes. With SPENM, the routing paths can be selected so that both flows are codable with maximum coding chances.

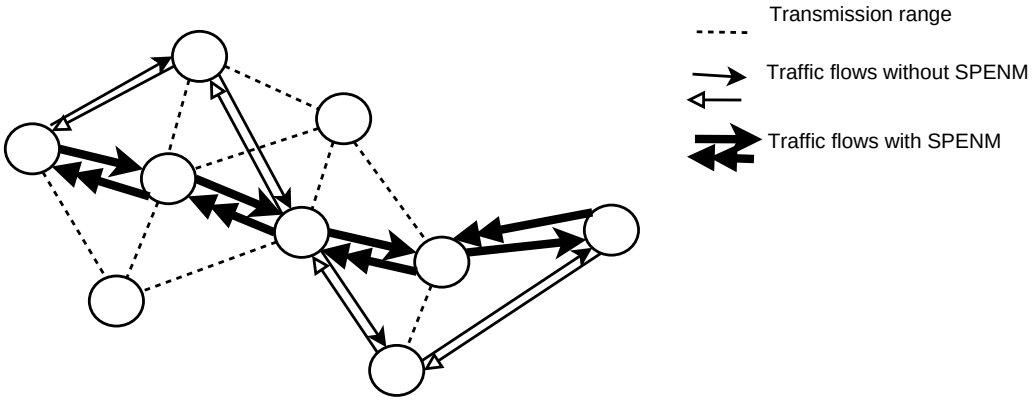


Figure 3-9: An illustration for benefits gained by SPENM

In order for the routing process to select the shortest path with highest coding chance, among the shortest paths with equal number of hop counts p_i, the path with the highest number of neighbors will be chosen based on the SPENM metric which is calculated as follows:

$$SPENM(p_i) = \sum_{j}^{n} neighbors_j$$

where:

- j is the intermediate node on the routing path p_i.
- n is the number of intermediate nodes on p_i.
- $neighbors_j$ is the number of neighbors of the node j.

3.4.3 Modified DSDV packet format

In this section, we will describe the modified packet format used in our DSDV to support the network coding at MAC layer. The format of DSDV update packet is depicted in Figure 3-10.

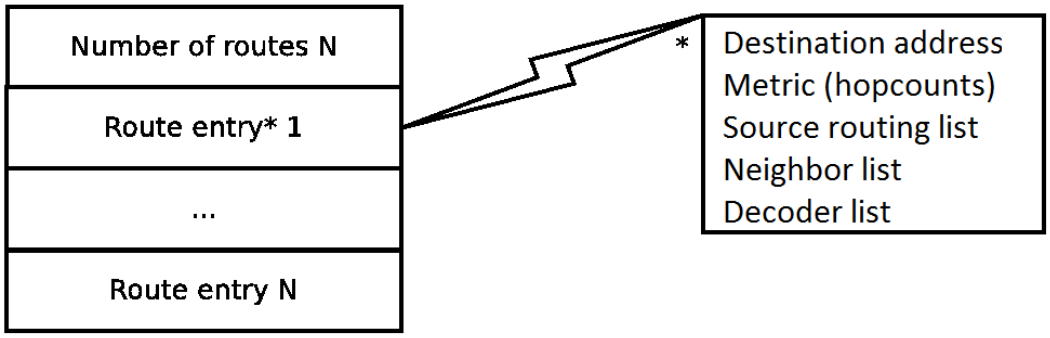


Figure 3-10: The modified DSDV update packet format

As we discussed above, we keep the original information of the original DSDV (i.e., destination, metric and sequence number) intact because we want to have the same functionalities but adding the mechanism to support the network coding. Here, we add three more lists: the neighbor list, the source routing list and the decoder list. The decoder list is only used by DODEX and DODEX+.

3.4.4 Node behavior

Each node in DODE, DODEX and DODEX+ has to perform packet reception and packet transmission. We will illustrate the node behavior via the flow chart.

3.4.4.1 COPE and BEND

Node behavior is shown in Fig 3-11. We do not illustrate the packet transmission from COPE to DODEX because it is simple: the network coding layer only forwards the packet based on the upper layer's request. For packet reception, COPE and BEND will check if the packet is coded or not to perform the decoding. They do not support decoders that are two-hop away from coders and the decoding process must produce a native packet. Otherwise, received packet gets dropped. If the packet is native, packet is checked with others from every queues successively: the codable queue $Q_{codable}$, the native queue Q_{native} and the overheard queue $Q_{overheard}$ to meet the coding condition. Whenever it is a match, the packet is added to the linked list,

along with related codable packets. The linked list is then included into the codable queue $Q_{codable}$. If the packet is not codable after the check, the packet is stored into the associated queues.

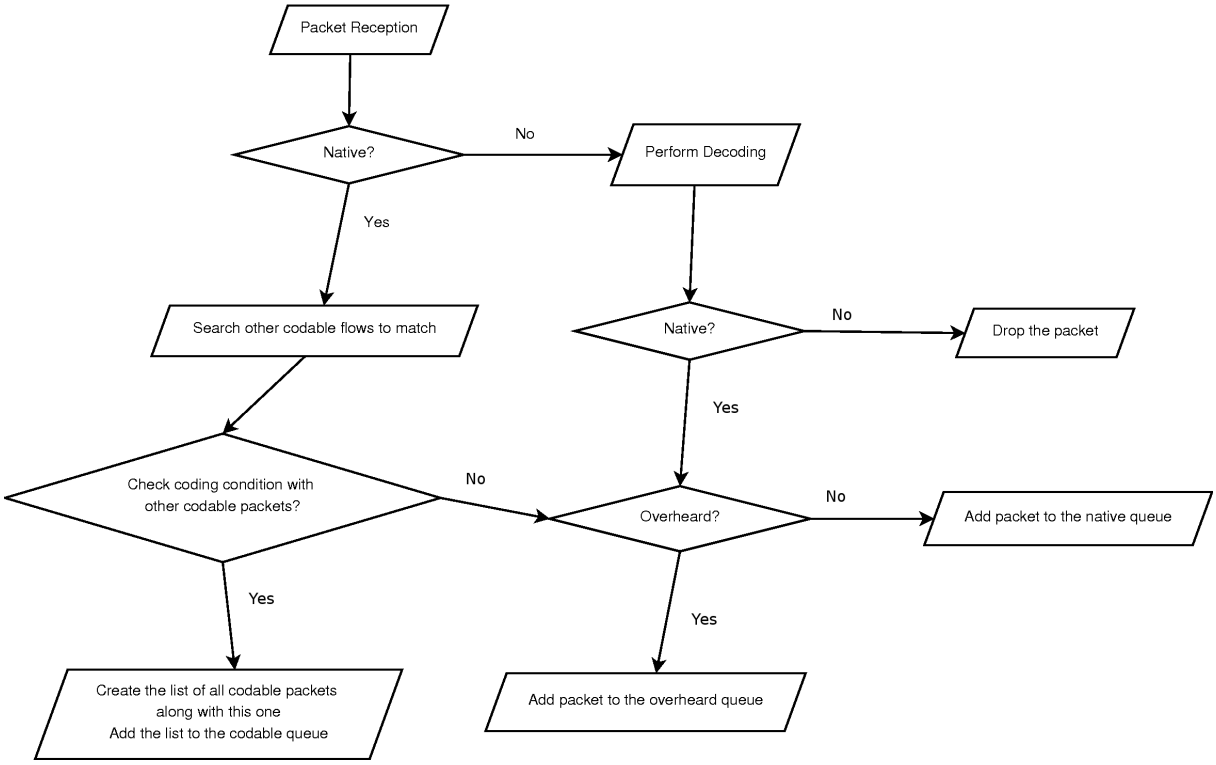


Figure 3-11: Packet reception of COPE and BEND

3.4.4.2 DCAR and DODE

In DCAR and DODEX, the packet reception (Fig 3-12) is quite similar except the coding condition, which leads to the new mechanism called *forwarding coded packets*. Because they support the decoders over two-hop away from coders, the coded packets will go through some validations. If the coded packet is forward-able and intended to the node, it will be kept. The node will ensure the coded packets intact for transmission by adding them into a separate coded queue Q_{coded}. Otherwise, packets will get dropped.

3.4.4.3 DODEX and DODEX+

Because DODEX and DODEX+ support "multiple decoders" feature, each encoded packet maintain the decoder list to ensure the correct forwarding. The decoder list

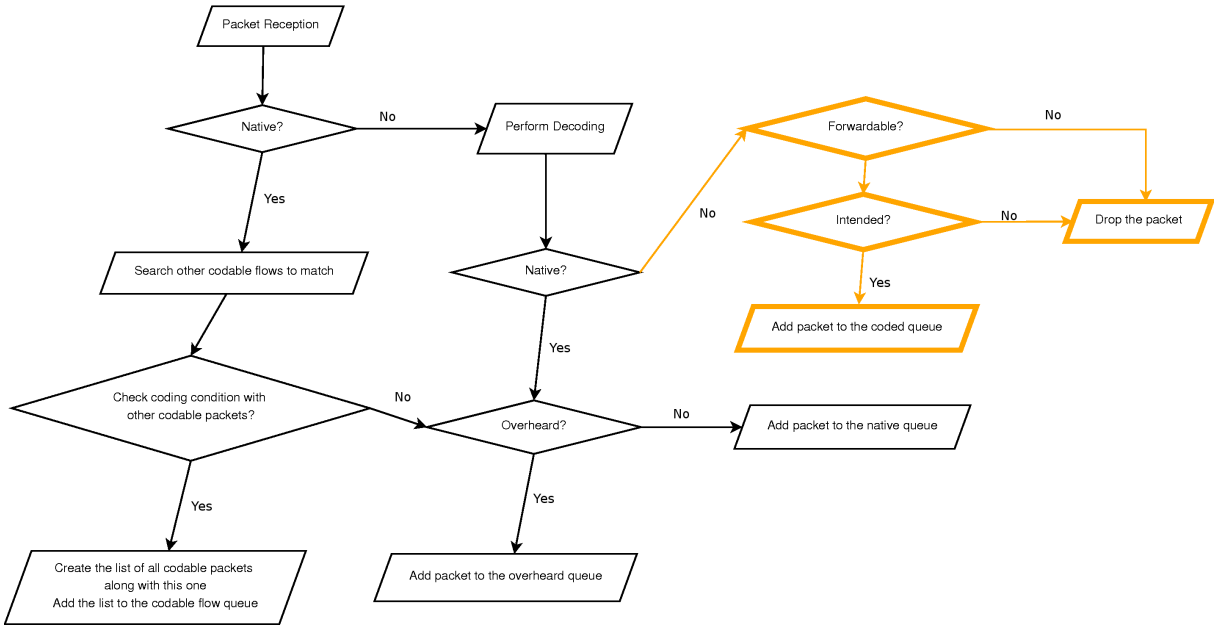


Figure 3-12: Packet reception of DCAR and DODE

is created by the encoder and included into the packet header for the next hops to check. Based on other network coding information (e.g., the neighbor lists...) the node will conclude if it is appropriate to forward the packet (i.e., the decoder is still on the routing path). If the forwarding is wasteful, the packet will get dropped. We add a validation before the decoding, so the traffic can be properly forwarded.

DODEX+ support "the re-encoding" feature and the encoded packets will be checked with the coding condition again to collect as much as possible the coding chance (Fig 3-13 and 3-14) before the transmission. We decide the packets are checked to meet the coding condition again before the transmission because it does not affect to the current mechanism of DODEX much.

3.5 Simulation and results

3.5.1 DODE

We use NS-2 as the simulator to compare the performances of DODE with the previous architectures: the original IEEE 802.11, COPE, BEND and DCAR. We use two topologies as illustrated in Figures 3-15(a) and 3-15(b). The first topology (Figure 3-15(a)) is provided for test scenario of 2-packet coding (x-packet coding, i.e. the combination of x packets in a single transmission). The second topology (Figure

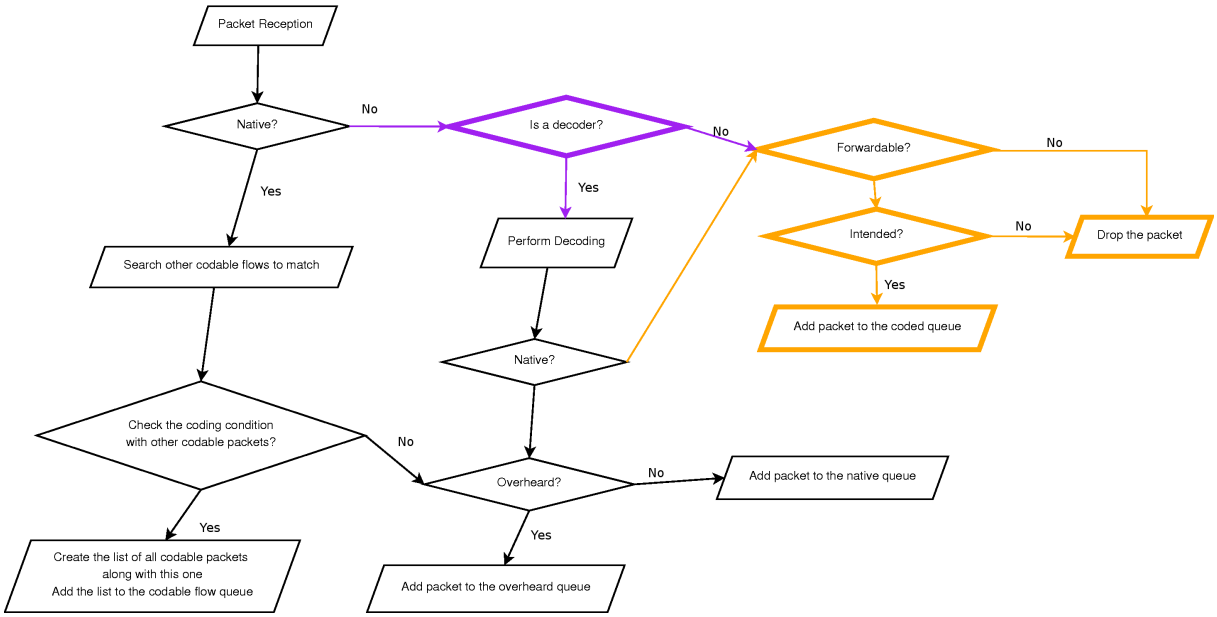


Figure 3-13: Packet reception of DODEX and DODEX+

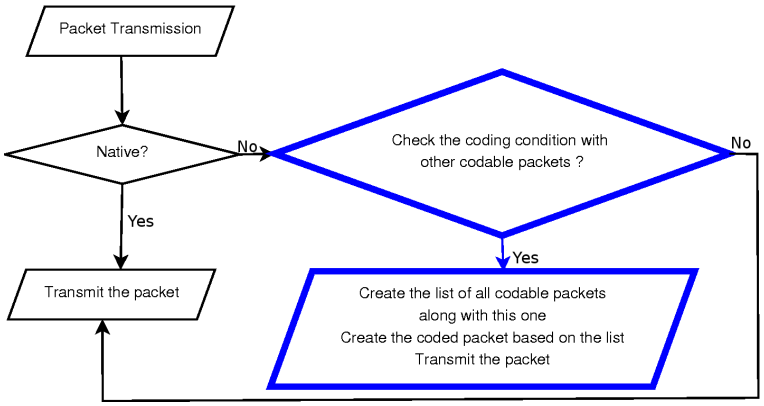
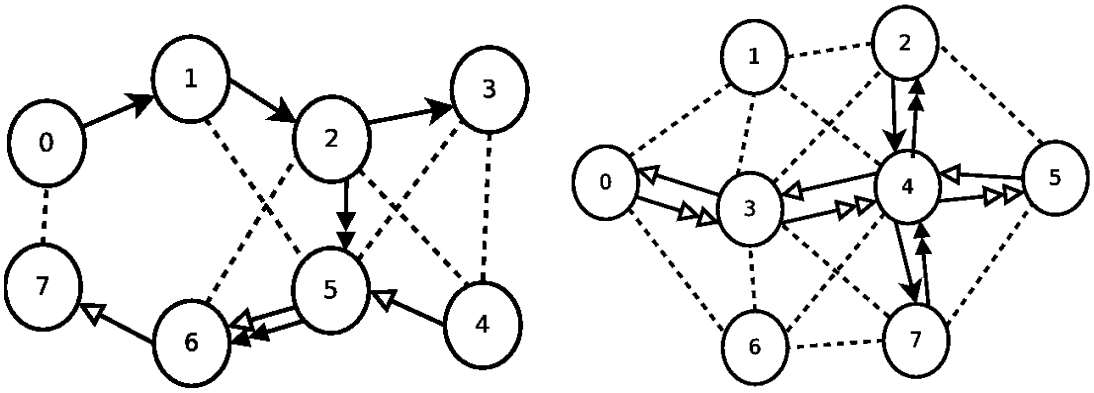


Figure 3-14: Packet transmission of DODEX+

3-15(b)) is used for test scenario of 3 or 4-packet coding.

Each topology is created in a flat area of $1000m \times 1000m$. The data traffic in the network are all CBR (Constant Bit Rate) flows sent over UDP (User Datagram Protocol) using 1000-byte datagrams with an arrival interval of 0.01s and traffic generation duration at source of 150s. The performances is then evaluated by two performance metrics, the throughput and the number of coded packets. We vary the traffic flows in test scenarios as shown in Table 3.1 . Afterwards, the test scenarios will be executed with each traffic-flow variety for all implementations and the result is collected with a 95% confidence interval.



(a) Test case 1 topology for coding 2 native packets (b) Test case 2 topology for coding 3-4 packets

Figure 3-15: Test cases for exposing 2, 3-packet coding in DODE

Table 3.1: Flows in Test scenarios

3 flows varied in test scenario 1	$F_1(0 \to 1 \to 2 \to 3)$, $F_2(4 \to 5 \to 6 \to 7)$, $F_3(2 \to 5 \to 6)$
5 flows varied in test scenario 1	$F_1(0 \to 1 \to 2 \to 3)$, $F_2(4 \to 5 \to 6 \to 7)$, $F_3(2 \to 5 \to 6)$, $F_4(3 \to 2 \to 1 \to 0)$, $F_5(7 \to 6 \to 5 \to 4)$
4 flows varied in test scenario 2	$F_1(0 \to 3 \to 4 \to 5)$, $F_2(5 \to 4 \to 3 \to 0)$, $F_3(2 \to 4 \to 7)$, $F_4(7 \to 4 \to 2)$
6 flows varied in test scenario 2	$F_1(0 \to 3 \to 4 \to 5)$, $F_2(5 \to 4 \to 3 \to 0)$, $F_3(2 \to 4 \to 7)$, $F_4(7 \to 4 \to 2)$, $F_5(1 \to 3 \to 6)$, $F_6(6 \to 3 \to 1)$

In the first topology, we run the test with three flows $F_1(0 \to 1 \to 2 \to 3)$, $F_2(4 \to 5 \to 6 \to 7)$ and $F_3(2 \to 5 \to 6)$. In the second test case, there are five flows: the above-mentioned flows F_1, F_2 and F_3 and two additional ones $F_4(3 \to 2 \to 1 \to 0)$ and $F_5(7 \to 6 \to 5 \to 4)$. The test case is then repeated after adding two more codable flows $F_4(3 \to 2 \to 1 \to 0)$ and $F_2(7 \to 6 \to 5 \to 4)$. The purpose of this test case is exposing the advantage of DODE over the formers by two keys: the generalized coding condition helps DODE to detect the code chances (while COPE, BEND does not) and the diffused gain which will reduce the collision of traffic flow through encoders, thus, giving DODE an upper hand over the previous architectures.

In each test case, a number of intermediate nodes are placed between source and destination. Based on the purporse of test case, the number of flows UDP/CBR (Constant Bit Rate 800kbps on UDP) can be varied, but maintained data transfer during the test case runtime. The data transfer is 150s.

All parameters and values that are used in running the simulation are referenced in the Table 3.2.

As shown in Figures 3-16(a) and 3-16(b), the throughput obtained by DODE gains

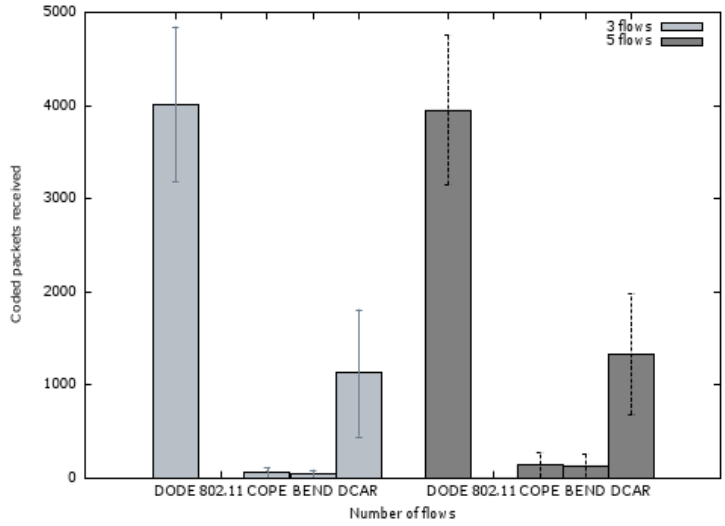
Table 3.2: Simulation parameters

Parameter	Value	Comments
Flat Grid	1000x1000	Simulation space, a flat field 1000mx1000m
Bandwidth	1Mbps	The bandwidth of wireless network
Mac layer	IEEE 802.11, COPE, BEND, DCAR, and DODE	IEEE 802.11 with support the network coding
Routing layer	DSDV	DSDV with support the network coding
Propagation model	Two-Ray Ground	A propagation model supported by ns-2 to simulate the signal propagation
Interference model	Thermal Threshold	If a packet power is lower than a minimum threshold (set fixed), packet will get dropped
Error model	ErrorModel80211	An error model supported by ns-2 to simulate the loss on wireless links
UDP/CBR rate	800kbps	Rate for application Constant Bit Rate on UDP
Test case duration	320s	The testcase runtime
Data transfer duration	150s	The data transfer runtime
CBR packet size	1000 bytes	Data in each CBR packet
UDP traffic	UDP/CBR	UDP constant bit rate at 800kbps

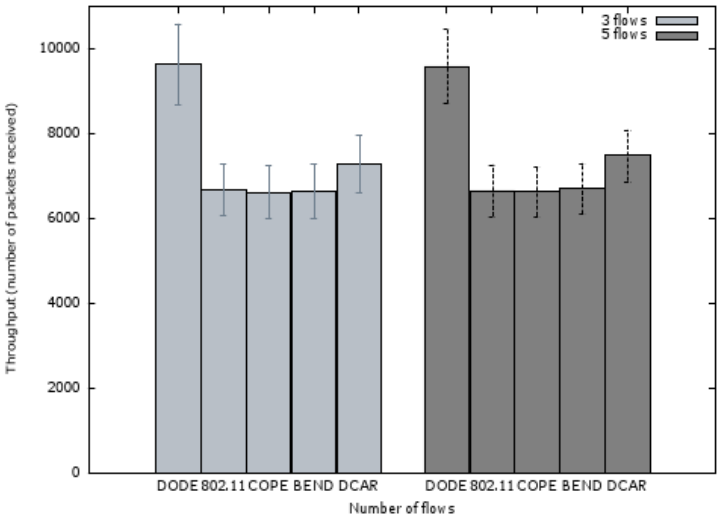
the extra 21% - 30% over DCAR, BEND, COPE and IEEE 802.11. First, based on the two-hop coding pattern, COPE or BEND cannot detect the coding chances, forcing them to transmit the native packets during data transfer. That's the reason why there is almost no coding packets for COPE or BEND, and their throughputs achieved are nearly the same as the throughput of IEEE 802.11. Second, in the case of DCAR, it requires the encoder to be on the routing paths of traffic flows, so that only either 2 or 5 is the encoder, depending on the routing result in each simulation running. Due to the presence of the flow $F3$ whose traffic is $(2 \rightarrow 5 \rightarrow 6)$, the traffic going through node 2 or 5 is dense and collisions are increased, resulting in packet drops at node 2 or 5. With the diffused gain, DODE can overcome this problem by sharing the load of coding process between 2 and 5 while still keeping the traffic of F_3 sending.

COPE or BEND does not detect the coding chance fully like DODE or DCAR, leaving them to perform 2 packet encoding while DODE or DCAR (sometimes) perform 3 or 4 packet encoding. There are plenty of equivalent encoders on the path from source to destination, causing the packet collision at encoder on routing path in case of DCAR, reducing its throughput gain compared with DODE.

We also have a note on the coded packets gained by COPE and BEND in test case 2 are greatly reduced after adding 3 more reverse flows. This happens because COPE, BEND intend to do the 2-packet encoding at node 4 or 8. Only the packet pairs of



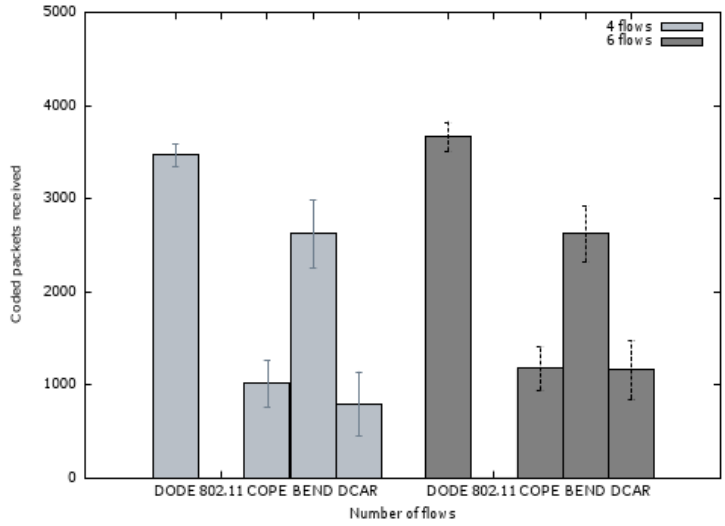
(a) Coded packets result for Test case 1



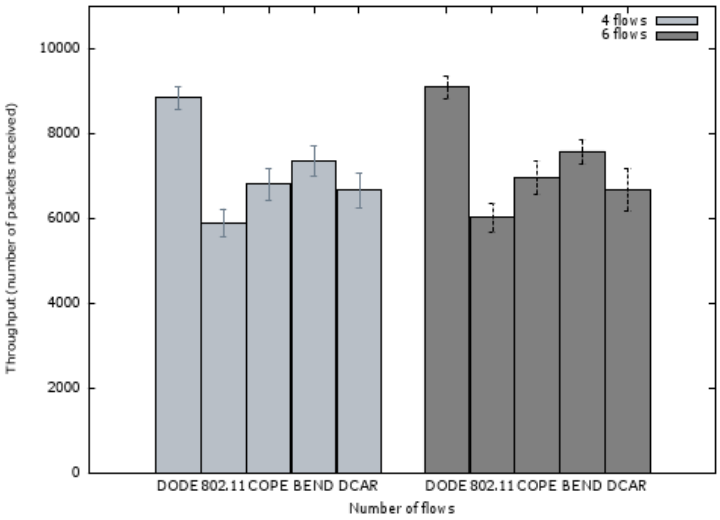
(b) Throughput result for Test case 1

Figure 3-16: Test results for coding 2 packets

(F_2, F_3), (F_2, F_5), (F_3, F_6) can be coded together. However, the packets from flow F_1 are now considered as non codable and contributed to the collision with the codable ones. Moreover, the source or destination of the other flows can be the intermediate nodes of the routing path for flow F_1. All these constraints can create a great collision around the node 4 or 8 which reduces the number of successful coded packets, thus, decreasing the throughput gained by COPE or BEND. On the other hand, DODE and DCAR manage to overcome the problem. They can detect more coding chances thanks to the generalized coding condition, resulting in releasing the packets in the queue faster so that reducing the collision.



(a) Coded packets result for Test case 2



(b) Throughput result for Test case 2

Figure 3-17: Test case result for coding 3-4 packets

In the second topology, the performances of DODE, IEEE 802.11, COPE, BEND and DCAR are presented in Figures 3-17(a) and 3-17(b). There are two test cases: the first one with four flows ($F_1(0 \rightarrow 3 \rightarrow 4 \rightarrow 5)$, $F_2(5 \rightarrow 4 \rightarrow 3 \rightarrow 0)$, $F_3(2 \rightarrow 4 \rightarrow 7)$, $F_4(7 \rightarrow 4 \rightarrow 2)$) and the second with four flows mentioned above and two more ($F_5(1 \rightarrow 3 \rightarrow 6)$, $F_6(6 \rightarrow 3 \rightarrow 1)$. Particularly, DODE uses the metric SPENM, which helps maintaining traffic going through the path with the most coding chances as illustrated in Figure 3-15(b) while the previous suffer from their own limitation of choosing the different paths: COPE, BEND uses the shortest routing metric supported by DSDV, which can select the non-codable paths; DCAR requires the encoder

to stay on the routing paths of flows, so traffic is concentrating on encoding nodes. For example, with COPE, BEND, traffic for F_1, F_2 may flow by $F_1'(0 \to 1 \to 2 \to 5)$ and $F_2(5 \to 4 \to 3 \to 0)$. With DCAR, it keeps selecting 3 and 4 as encoders for test simulation. These problems force the previous coding systems not to encode packets fully (COPE, BEND) or to suffer packet drop (COPE, DCAR), giving DODE a better performance 16%-33% over the formers as shown in Figures 3-20(a) and 3-20(b). COPE, BEND and DCAR may select transmission paths not feasible to perform the encoding caused by their routing metrics. For example, all the network architectures will choose the path $(2 \to 4 \to 7)$ for flow F_2 and the reversed one for F_3. Nevertheless, the previous may choose the path $(0 \to 1 \to 4 \to 5)$ for flow F_1 and DODE chooses $(0 \to 3 \to 4 \to 5)$ for flow F_1 by using SPENM.

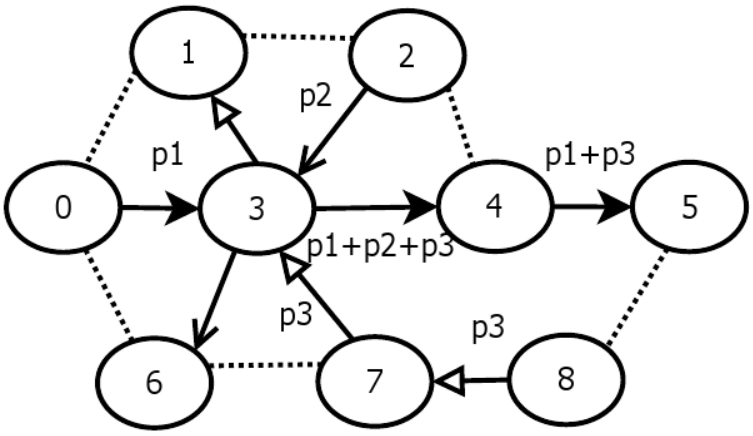
3.5.2 DODEX

We use NS-2 as the simulator to compare the performances of DODEX with the previous architectures: IEEE 802.11, COPE, BEND, DCAR and the non-extended DODE. We use two topologies as illustrated in Fig 3-18(a) and 3-18(b). The first topology (Fig 3-18(a)) is provided for test scenario of maximum 3-packet coding with light traffic. The second topology (Fig 3-18(b)) is used for test scenario of maximum 4-packet coding with stress traffic causing high packet collisions and drops.

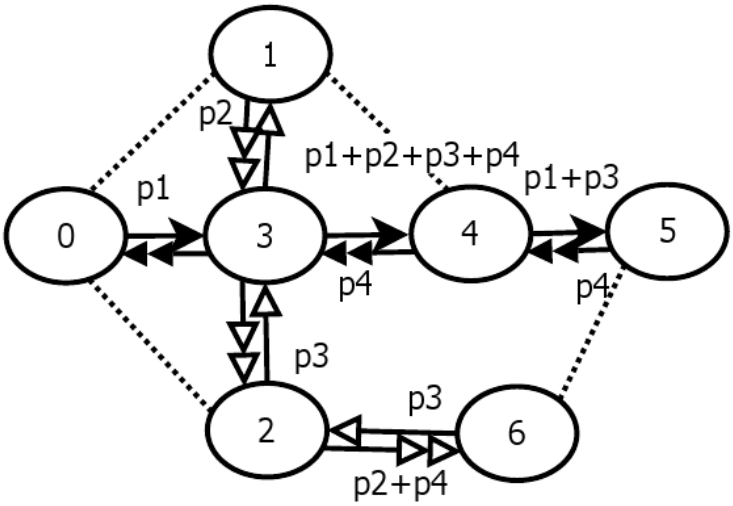
Each topology is created in a flat area of $1000m \times 1000m$. The data traffic in the network are all CBR (Constant Bit Rate) flows sent over UDP (User Datagram Protocol) using 1000-byte datagrams with an arrival interval of 0.01s and traffic generation duration at source of 150s. The performances is then evaluated by two performance metrics, the throughput and the number of coded packets. We vary the traffic flows in test scenarios as shown in Table 3.3. Afterwards, the test scenarios will be executed with each traffic-flow variety for all implementations and the result is collected with a 95% confidence interval.

Table 3.3: Flows in Test scenarios

3 flows varied in test scenario 1	$F_1(0 \to 3 \to 4 \to 5)$, $F_2(2 \to 3 \to 6)$, $F_3(8 \to 7 \to 3 \to 1)$
6 flows varied in test scenario 1	$F_1(0 \to 3 \to 4 \to 5)$, $F_2(2 \to 3 \to 6)$, $F_3(8 \to 7 \to 3 \to 1)$, $F_4(5 \to 4 \to 3 \to 0)$, $F_5(6 \to 3 \to 2)$, $F_6(1 \to 3 \to 7 \to 8)$
7 flows varied in test scenario 2	$F_1(6 \to 2 \to 3 \to 1)$, $F_2(1 \to 3 \to 2 \to 6)$, $F_3(0 \to 3 \to 4 \to 5)$, $F_4(5 \to 4 \to 3 \to 0)$, $F_5(5 \to 6)$, $F_6(1 \to 4)$, $F_7(0 \to 2)$
10 flows varied in test scenario 2	$F_1(6 \to 2 \to 3 \to 1)$, $F_2(1 \to 3 \to 2 \to 6)$, $F_3(0 \to 3 \to 4 \to 5)$, $F_4(5 \to 4 \to 3 \to 0)$, $F_5(5 \to 6)$, $F_6(1 \to 4)$, $F_7(0 \to 2)$, $F_8(5 \to 6 \to 2)$, $F_9(2 \to 6 \to 5)$, $F_{10}(0 \to 2 \to 6)$



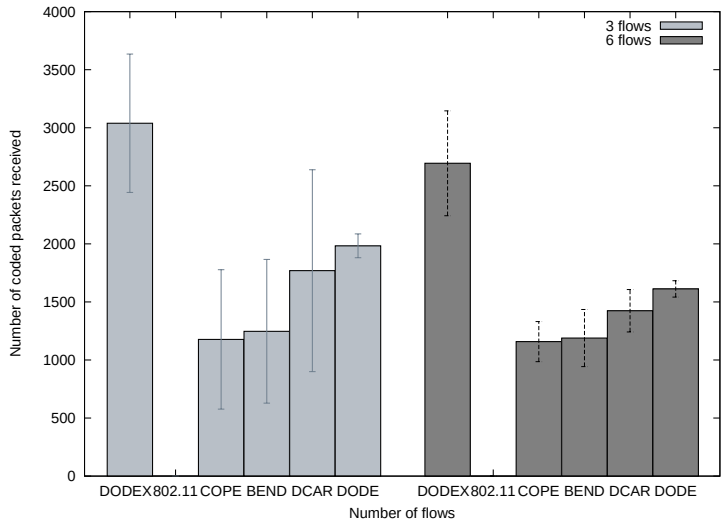
(a) Test topology 1 for 3-packet coding



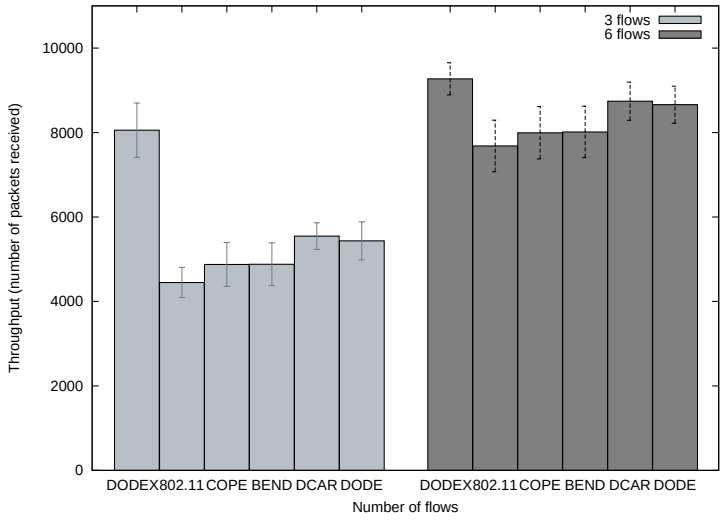
(b) Test topology 2 for 4-packet coding

Figure 3-18: Test scenarios for exposing 3 and 4-packet coding in DODEX

As shown in Fig 3-19(a) and 3-19(b), with the light traffic, DODEX outperforms previous implementations significantly. In the case of 3 flows, only DODEX discovered the 3-packet coding, and the throughtput gain over the previous is quite promising (32% over our old DODE and 30%-45% over the others). First, COPE and BEND only detects the 2-packet coding between F_2 and F_1. However, coder 3 also serves as a forwarder for F_3, increasing packet collisions and drops at node 3 due to concentrating traffic via the coder. Second, DODE and DCAR are also able to detect the 2 packet-coding between F_1 and F_3 thanks to the general coding condition but the same problem happened too because the diffused gain does not help BEND, DODE or even DODEX much (only 3 as the coder). With 3-packet coding, DODEX allows more packets delivered to destination, result in higher throughput. For the next case (6



(a) Coded packets result for Test scenario 1

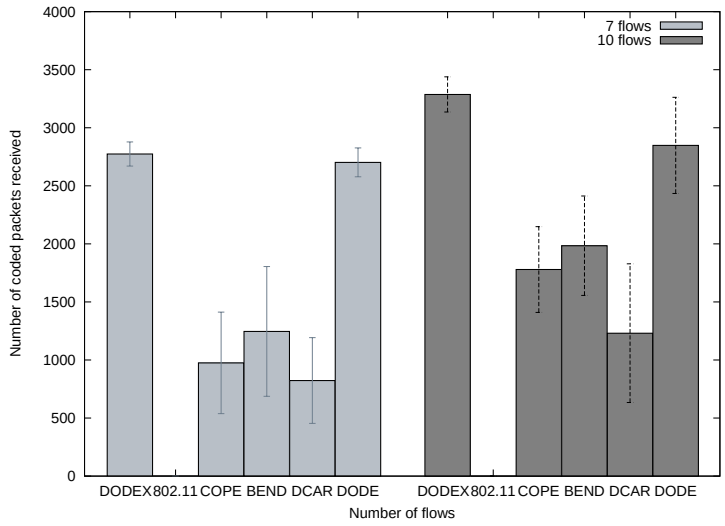


(b) Throughput result for Test scenario 1

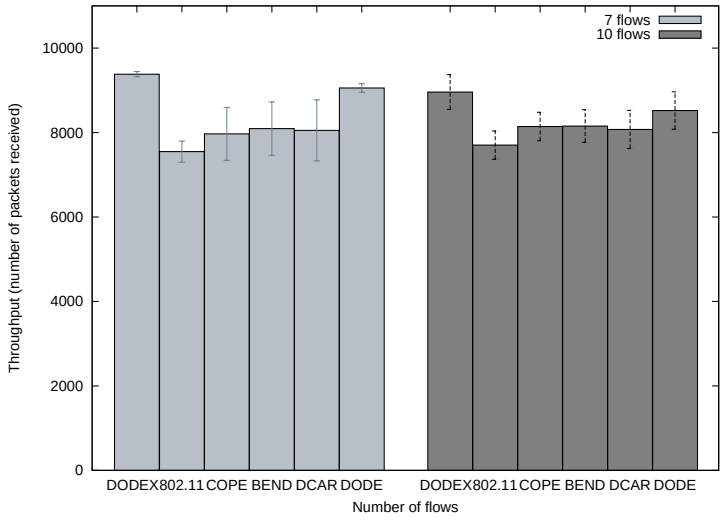
Figure 3-19: Test scenarios 1 results

flows), because there are more coding chances (F_4, F_5 and F_6 are in reverse directions with F_1, F_2 and F_3, respectively), the previous works can compensate the throughput gain and the number of coded packets by creating the coded packets from these flows with opposite ways. Nevertheless, DODEX still maintains higher throughput because DODEX can detect not only all coding chances like the previous but also the 3-packet coding with multiple decoders.

In the second topology exposing the coding 3 or 4 packets in a single transmission, the performances of all previous and DODEX are presented in Figs 3-20(a) and 3-20(b). We would like to check if under heavy traffic, DODEX still keeps the high



(a) Coded packets result for Test scenario 2



(b) Throughput result for Test scenario 2

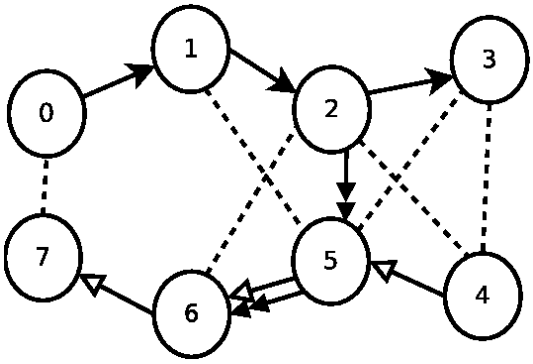
Figure 3-20: Test scenario 2 results

throughput and coded packets. By applying 4-packet coding with multiple decoders, DODEX transfers more data even in the interference of the non-codable flows. Compared to DODEX, in case of 10 flows (4-6 of them are non-codable), all previous implementations are losing throughput because it takes more transmissions for 2-packet coding. Due to high interference from the non-codable flows, the throughputs gained from all architectures are reduced, both from competition of accessing MAC layer to send coded packets or the packet collisions and drops. DODEX can alleviate the problem by discovering more coding chances via coding with multiple decoders, draining the packets from the queue at the forwarder faster than the others (4-packet

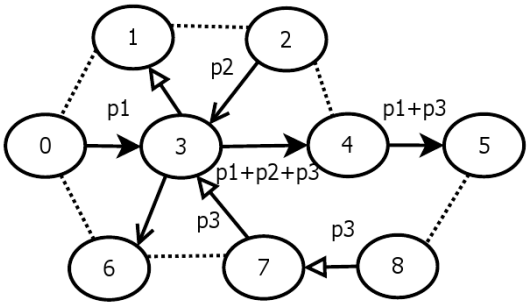
coding compared to 2-packet coding), thus, giving DODEX a better performance over the formers (6% over DODE, and 10%-20% over the others as Figs 3-20(a) and 3-20(b) shows)

3.5.3 DODEX+

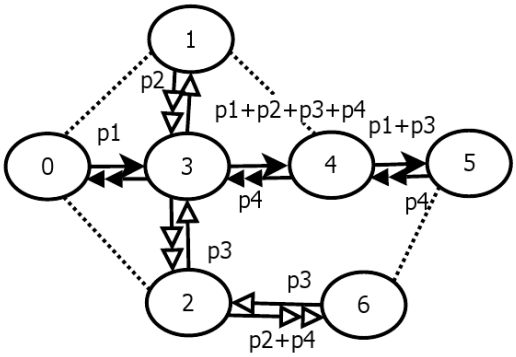
Similar to DODE, DODEX, we use NS-2 to compare the performance against the previous propositions: IEEE 802.11, COPE, BEND, DCAR and DODEX. There are 3 testing topologies shown in Figs 3-21(a), 3-21(b) and 3-21(c). Testing parameters will keep intact as they are in the running tests of DODE, DODEX. Two testing metrics the throughput (the number of received packets during the running time 150s) and the number of coded packets reached the destination. Each topology is created in a flat area of $1000m \times 1000m$. The data traffic in the network are all CBR (Constant Bit Rate) flows sent over UDP (User Datagram Protocol) using 1000-byte datagrams with an arrival interval of 0.01s and traffic generation duration at source of 150s.



(a) Test topology 1



(b) Test topology 2



(c) Test topology 3

Figure 3-21: Test topologies in DODEX+

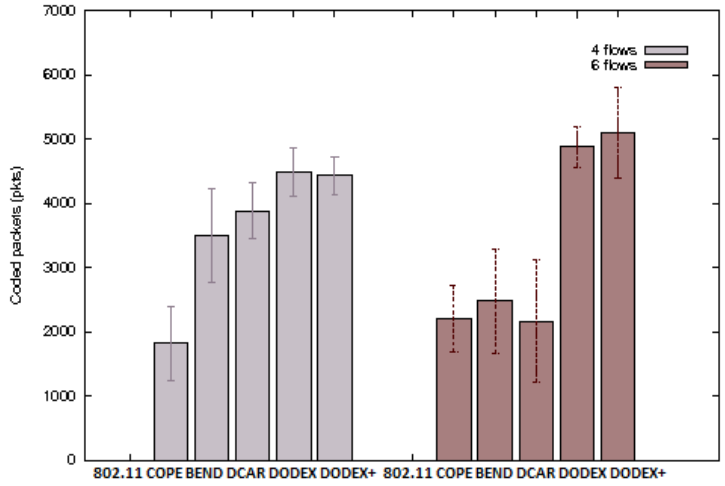
We vary the traffic flows in test scenarios as shown in table 3.4. Afterwards, the test scenarios will be executed with each traffic-flow variety for all implementations and the result is collected with a 95% confidence interval.

Table 3.4: Flows in Test scenarios

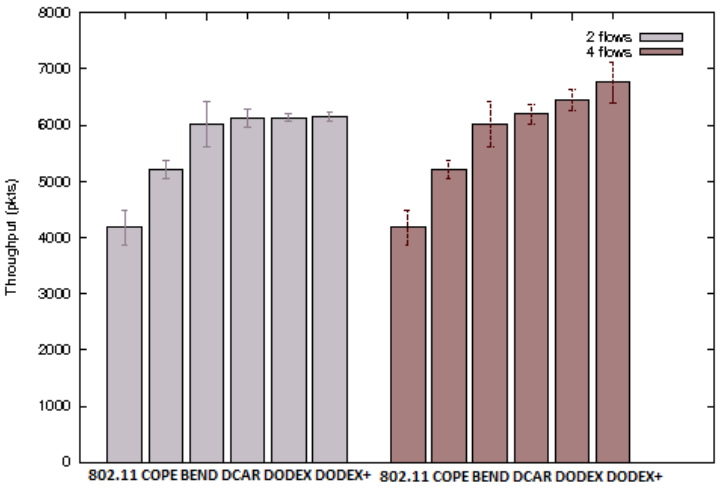
2 flows varied in test scenario 1	$F_1(0 \rightarrow 3)$, $F_2(4 \rightarrow 7)$
4 flows varied in test scenario 1	$F_1(0 \rightarrow 3)$, $F_2(4 \rightarrow 7)$, $F_3(3 \rightarrow 0)$, $F_4(7 \rightarrow 4)$
3 flows varied in test scenario 2	$F_1(0 \rightarrow 5)$, $F_2(2 \rightarrow 6)$, $F_3(8 \rightarrow 1)$
6 flows varied in test scenario 2	$F_1(0 \rightarrow 5)$, $F_2(2 \rightarrow 6)$, $F_3(8 \rightarrow 1)$, $F_4(5 \rightarrow 0)$, $F_5(6 \rightarrow 2)$, $F_6(1 \rightarrow 8)$
5 flows varied in test scenario 3	$F_1(6 \rightarrow 1)$, $F_2(1 \rightarrow 6)$, $F_3(0 \rightarrow 5)$, $F_4(5 \rightarrow 0)$, $F_5(5 \rightarrow 6)$
10 flows varied in test scenario 3	$F_1(6 \rightarrow 1)$, $F_2(1 \rightarrow 6)$, $F_3(0 \rightarrow 5)$, $F_4(5 \rightarrow 0)$, $F_5(5 \rightarrow 6)$, $F_6(1 \rightarrow 4)$, $F_7(0 \rightarrow 2)$, $F_8(5 \rightarrow 2)$, $F_9(2 \rightarrow 5)$, $F_{10}(0 \rightarrow 6)$

In Fig 3-22(b), DODEX+ outperforms DODEX and others around 1-5% and 31-42%, respectively. We see that the extra gain of DODEX+ over DODEX is not much because in the first test case, the "re-encoding" feature is only used in the transmissions between node 6 and 7 or 0 and 1. Coded packets are re-encoded again with other native packets and transmitted instead of simply forwarded. That's the reason why the number of coded packets are similar to the one of DODEX (only 1-5%, in Fig 3-22(a)).

The second and third test cases (Figs 3-21(b) and 3-21(c)) are the same from ones of DODEX (Figs 3-18(a) and 3-18(b)). Thanks to the new feature "re-encoding", DODEX+ can gain the extra throughput 1-3% and 1-5% over DODEX in these two test cases, respectively (Figs 3-23(b) and 3-24(b)). This is very interesting if we know in the second test case, DODEX and DODEX+ perform the 3-packet coding. Combining two features from DODEX and DODEX+ pose a problem: in the lossy wireless environment, more data are combined, more risky they are to be lost. In DODE and its extensions, we leave the random loss and interference for farther research (see Chapter 3) and focus on finding more coding chance based on extending the coding condition. Even though DODEX+ saves more transfer time, it risks losing more data than DODEX. Consequently, the extra gain is not much. The number of coded packets in both cases is similar to the one in DODEX as DODEX+ takes advantages of the current coded traffic to perform the coding, "compressing" the data again to save more transfer time. It means that DODEX+ re-encodes the current coded packet, so that the number of coded packets could not increase. We can see that the number of coded packets slight decreases by 1-2% due to the lossy wireless environment. We



(a) Coded packets result for Test scenario 1

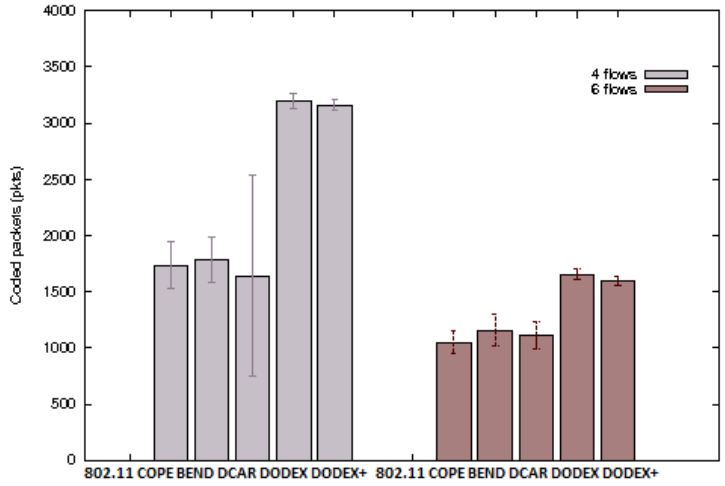


(b) Throughput result for Test scenario 1

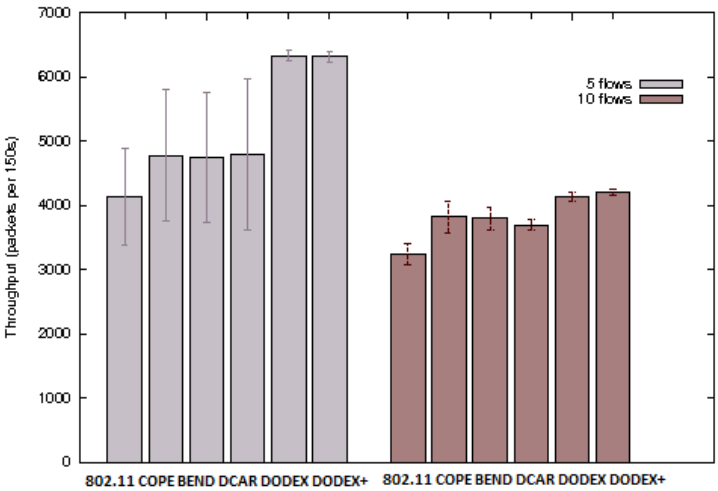
Figure 3-22: Test scenarios 1 results

can conclude that the "re-encoding" feature indeed gives some extra throughput in case with the light traffic (the first test case). However, in the case with heavy traffic, with the lossy and interfered wireless network, the gain is not much.

Another important note is that the benefit given by the "re-encoding" feature is not quite a boost. There are two reasons: the re-encoding of coded packets is not an only option, the coding condition is still met among the native packets and the lossy environment can reduce the coding chances for re-encoding. Let's recall the situation in Fig 3-6. Because of the property of ONC, there are always the native transmissions from coders to next hops. The native packets will compete with the



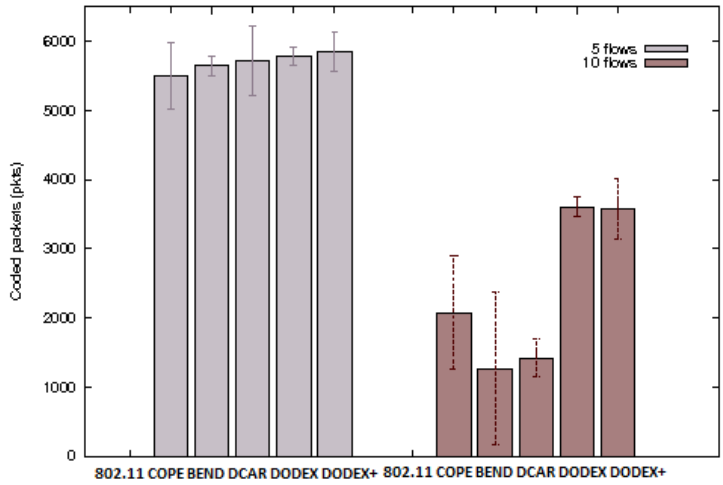
(a) Coded packets result for Test scenario 2



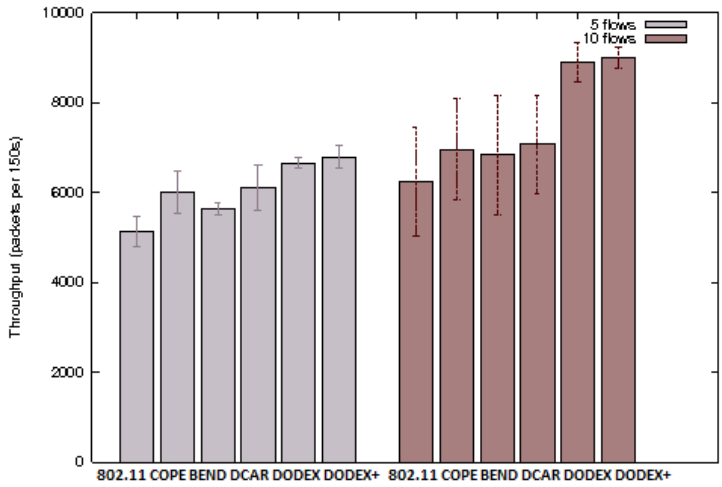
(b) Throughput result for Test scenario 2

Figure 3-23: Test scenarios 2 results

encoded packets in the encoding process, which reducing the effect of "re-encoding" feature. In our farther work (out of the scope of this thesis), we need some mechanism of transmission reliability to reduce the effect of loss and interference. With that, the performance is well-increased.



(a) Coded packets result for Test scenario 3



(b) Throughput result for Test scenario 3

Figure 3-24: Test scenarios 3 results

3.6 Chapter conclusion

ONC is a practical inter-flow network coding which is under development and research. Extending the coding condition is one of the challenges to bolster ONC. COPE, BEND and DCAR are works which defines and extends the coding condition by suggesting many features to maximize network capacity.

- COPE: the feature of *opportunistic listening and encoding*

- BEND: the feature of *diffused gain*
- DCAR, the feature of *generalizing the coding condition*

Our work is a contribution to extend the coding condition by the followings:

- DODE combines of features of COPE, BEND and DCAR.
- DODEX adds the *encoding process with multiple decoders*.
- DODEX+ adds the *re-encoding process with multiple encoders*.

Certainly, the more the coding condition is generalized, the more complex its mechanism is to learn the neighbor's state. Learning the neighbor's state is crucial to ONC because it leads to the decision to create the encoded packet or not. Network topology change can lead to wrong match in the coding condition, which results in reducing the performance by unnecessary forwarding native transmissions or undecodable packet broadcasts. It's an interesting problem open to future research, especially, in other greatly lossy and mobile environments, e.g. vehicular networks.

Chapter 4

Intra-flow network coding

In this chapter, we describe our contributions which provide transmission reliability with intra-flow network coding in wireless networks. Currently, in lossy environments like wireless networks, random losses cause the performance degradation. A redundancy control, which is triggered to transmit redundant traffic, can indeed alleviate the effect of random losses and provide transmission reliability. Let's recall how RLNC works in intra-flow network coding. Instead of transmitting packets of a flow separately, packets are grouped and transmitted to the destination as RLNC combinations. Each linear combination is considered equally important and carries a "piece" of coded data of the packets in a group. When the destination collects a sufficient number of "pieces", it can perform the decoding process to retrieve the original data. If a loss occurs, the source only needs to generate and transmit another linear combination to cover the loss. In fact, RLNC is efficient to simplify the mechanism of providing *transmission reliability* which our research aims for. However, this appealing benefit can be given by RLNC under specific conditions: coded packets need to be properly generated, transmitted and decoded in an acceptable decoding delay. To control random losses, some researches have suggested fixed redundancy control (e.g., [10]) or adaptive redundancy control by calculating the loss rate based on sending probes to estimate the network quality (e.g., [5]). These two approaches do not provide the redundancy in time and pose the risk of wasting bandwidth by over-redundancy or unsuccessful decoding due to an insufficient number of received packets, dramatically degrading the performance. We would like to follow a different approach to provide transmission reliability. We believe that an adaptive redundancy control can efficiently adjust the redundant traffic if it is integrated and supported by a well-designed coding scheme. We propose two solutions: the link-by-link adaptive redundancy control based on the feedback of link quality at MAC layer, called Adaptive Redundancy Control (ARC) and the end-to-end adaptive redundancy control

based on the feedback of transmission quality at TCP layer, called Dynamic Coding (DynCod). Multipath DynCod (MP-DynCod) is an enhancement of DynCod to support multipath transmission and alleviate a minor limitation of DynCod. Details on each proposition and current related state of the art are given in next sections. Section 4.1 details ARC. DynCod and MP-DynCod are introduced in Sections 4.2 and 4.2.6, respectively. The simulation and obtained results are presented and analyzed in Sections 4.1.5, 4.2.7.1 and 4.2.7.2.

4.1 Multi-batch Pipeline Coding with Adaptive Redundancy Control (ARC)

We need to recall some *terms* used in RLNC before going into detail. As formulated in Section 2.6.3, Batch Coding [5], [6], [7], [8], [9] is a *generation-based* batch coding scheme. Within a batch of n packets of the same flow, native packets are combined and transmitted by $n+r$ random linear combinations. $R = \frac{n+r}{n}$ is the value of *redundancy level*, which Batch Coding uses to resist random losses. The traffic flow is encoded in many consecutive *generations*. At the recipient side, *innovative coded packets* (i.e, linear combinations which reveal a new "piece" of information) are stored until they are sufficient for the decoding. As explained in Section 2.6.2, *generation-based coding* is a technique that may introduce the problems of *high decoding delay* and *generation discard*. Pipeline Coding [10] (mentioned in Section 2.6.4) is an enhancement of Batch Coding to avoid the problem of *high decoding delay*. Instead of generating n linear combinations such that each of them is a linear combination of n native packets of the batch, n linear combinations in Pipeline Coding are generated based on the incremental amount of native packets j ($1 \leq j \leq n$). More specifically, the first coded packet is generated from the first native packet, the second coded packet is generated by combining the first and second native packets and so on. Thanks to this coding scheme, the encoding and decoding processes are performed *progressively*. If all linear combinations up to the current coded packet successfully arrive at the destination, the decoding process does not need to wait for n arrivals of coded packets like Batch Coding to start off. Pipeline Coding indeed *alleviates the effect of high decoding delay*. Nevertheless, Pipeline Coding still *suffers from the problem of generation discard* even though it does give a fixed redundancy control.

To this point, we would like to propose a **link-by-link** adaptive redundancy control based on a new tailored coding scheme over Pipeline Coding in order to provide transmission reliability. Our proposition is called *Multi-batch Pipeline Coding with*

Adaptive Redundancy Control (ARC). It resolves the problems of *generation discard* and *random losses*. First, to *suppress* the potential *generation discard*, we propose the concept of *multi-batch pipeline coding* in which all sent batches are buffered at the sender and the receiver until they are well-decoded. In case of losses, the adaptive redundancy control is triggered at the sender to send redundancy packets accordingly. Second, to provide an adaptive redundancy control to mitigate *random losses*, we interpret the MAC layer IEEE 802.11 acknowledgement as an indicator of "link quality" and also as the acknowledgment of a coded packet. We define a new metric, the MAC layer *Data-frame Round Trip Time* (DRTT) which is the duration from the time when a MAC data-frame is sent to the time when its acknowledgement is received. The DRTT is used to determine the interval between two successive packets to send. Based on DRTT, the sender can adapt the sending rate of coded packets to the current state of link quality. Finding the **exact** number of redundant packets is merely impossible. So that we decide to find *the suitable time* to transmit the redundant packets to cover losses. Thanks to multi-batch coding scheme, new data is transmitted to the network while interpreting the received MAC acknowledgements as the feedback on the state of link quality. From that, an algorithm is proposed to calculate which time is appropriate for transmitting the redundant packet. We argue that the state of link quality should be rechecked before every transmission to deduce if we should adjust the redundant packets or we continue to send new data to the network. Consequently, we ensure not only the packet sufficiency for decoding but also the new information being transmitted smoothly. Summarily, our contribution ARC is two-fold:

- We suggest multi-batch pipeline coding scheme to tackle the problem of *generation discard*.

- We present an adaptive redundancy control based on the MAC acknowledgements, which decides the suitable time to adjust the redundant packets to tackle the problem of *random losses*.

4.1.1 Design

We suggest a cross-layer solution. ARC is implemented as a layer "2.5" between network and data link layer to provide an intra-flow unicast RLNC system as shown in Fig 4-1. One coded packet generated by ARC will be packed inside a MAC data-frame, transmitted and acknowledged by a MAC ACK. ARC interprets the MAC IEEE 802.11 acknowledgement as the feedback on the state of link quality. Traffic

is buffered into multiple batches. Native packets are then pipeline-coded and transmitted generation by generation. A generation is only removed from the buffer if it is well-received by the destination, which avoids the problem of generation discard. An adaptive redundancy control at the sender is triggered to send redundant packets if losses occur. ARC is designed to continuously send data from sender to the recipient while interpreting the MAC acknowledgements as the feedback on the link quality. Each generation transmission in ARC starts by sending the whole generation while collecting the number of acknowledgements to compute the value of **DRTT** (details in Section 4.1.2). Afterwards, ARC will run the adaptive redundancy control to check whether it is the suitable time to send the redundant packet. Otherwise, ARC will start over with the new batch in case the batch is well received. ARC continues transmitting this way until all data are sent from source to destination. As the requirement in the design, ARC intends to mask losses exposed to the application layer. It means that if a loss occurs, ARC keeps finding the suitable time to adjust the redundancy while streaming new data. The more losses occur, the more redundancy ARC adjusts to the network. With this, ARC ensures the smooth traffic flow from source to destination.

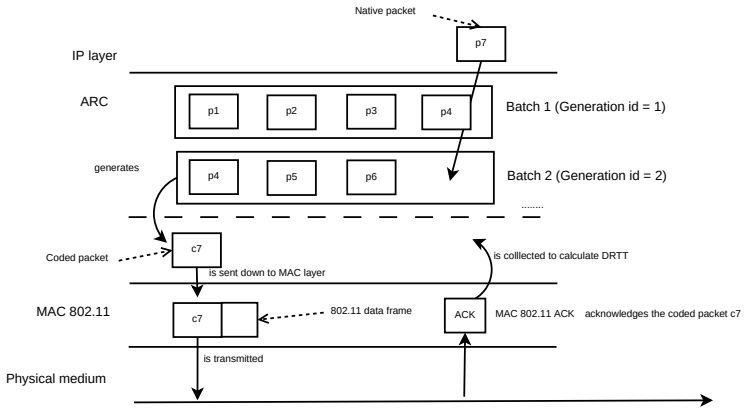


Figure 4-1: ARC layer in TCP/IP stack

4.1.2 DRTT estimation

The DRTT reflects the time needed for a successful transmission of a data frame at the MAC layer. The DRTT value depends on the distance between two nodes and the quality of the radio link. It also depends on the number of collisions at the MAC layer. If the distance between two nodes is long, the link quality is bad and there are many collisions at the MAC layer, a node will experience a long DRTT. ARC uses

the DRTT value as an indicator of link quality and schedules the next transmission of a coded packet based on the DRTT. ARC also uses the DRTT value to calculate the adaptive redundancy as presented in Section 4.1.3.

In order for the sender not to react too quickly to spontaneous and temporary changes of the link quality, the DRTT estimation (given in Algorithm 1) is calculated with a smooth factor α. Every time the sender receives a MAC ACK, it gets a DRTT measurement and updates the smoothed DRTT value as follows:

$$\mathbf{DRTT} = \alpha \times DRTT_{old} + (1 - \alpha) \times DRTT_{mes}$$

where:

- **DRTT** is the smoothed data frame round trip time.
- $DRTT_{mes}$ is the newly measured DRTT upon receiving a MAC ACK.
- $DRTT_{old}$ is the value of **DRTT** before updated by $DRTT_{mes}$.
- α is the smooth factor.

4.1.3 Adaptive redundancy scheme

As the main idea is introduced above, we will find *the suitable time* to add the redundancy instead of finding the number of redundant packets to send. We argue our approach based on an observation: even if an adaptive redundancy control measures the loss rate exactly to decide the number of redundant packets (> 1), at the time when the second redundant packet is transmitted, the loss rate may be not the same. The redundancy control has to re-measure the loss rate to ensure the exact number of redundant packets. Instead, we propose at time interval possible for MAC layer to transmit a packet (i.e, the **DRTT** calculated above), ARC will decide if a redundant packet or a new innovative coded packet is needed for transmission. Below is Algorithm 2 showing our scheme:

Algorithm 1 - Acknowledgement Reception

For each acknowledgement ack intended to the coded data packet p
if $gen\_id_{ack} \geq 0$ **then**
  Find batch b_i such that $i = gen\_id_{ack}$
  if b_i exists **then**
    Increase $recvcnt_{b_i}$ by 1
    Calculate $DRTT_{mes} = timestamp - NOW$
    Calculate **DRTT** $= \alpha \times DRTT_{old} + (1 - \alpha) \times DRTT_{mes}$
    Update scheduled-time-for-next-packet = NOW - **DRTT**
  end if
end if

Algorithm 2 - Adaptive Redundancy Scheme

$id_{batch} = -1$
For each batch $b_i (i \in [0, \mathbb{N}))$ stored in buffers
if $recvcnt_{b_i} < rank_{b_i} \& recvcnt_{b_i} < GEN\_SIZE \& min\_gen > gen_{b_i}$ **then**
  //only batches not well-received, find the one with minimum generation id to meet condition:
  if $sendcnt_{b_i} > GEN\_SIZE$ **then**
    //the redundant packet has to wait for a while before transmission
    if b_i waited for $\frac{recvcnt_{b_i}}{sendcnt_{b_i} - GEN\_SIZE + 1} \times$ **DRTT then**
      $id_{batch} = i$
      $min\_gen = gen_{b_i}$
    end if
  else
    //keep sending the packets if there's still new information
    if $sendcnt_{b_i} \leq rank_{b_i}$ **then**
      $id_{batch} = i$
      $min\_gen = gen_{b_i}$
    end if
  end if
end if
return id_{batch}

where:

- min_gen is the minimum value of generation ids of batches in transmission.

- $sendcnt_{b_i}$ is the number of coded packets generated and sent by batch b_i.
- $recvcnt_{b_i}$ is the number of acknowledgements received by batch b_i.
- $rank_{b_i}$ is the rank of batch b_i.
- $\mathbb{N}$ is the number of batches.
- GEN_SIZE denotes the generation size.
- *timestamp* is the time when the packet is transmitted.
- NOW is the current time when the packet is proceeded.

As the algorithm 2 describes, each batch b_i in ARC contains: the rank $rank_{b_i}$, *recvcnt* is the number of acknowledgement from the recipient, *sendcnt* is the number of packets sent. Because ARC handles multiple batches instead of a single batch, for every timing interval, ARC will decide either to keep transmitting new data or to add the redundancy. Each redundant packet has to wait for the duration as below:

$$\frac{\mathbf{recvcnt}_{b_i}}{\mathbf{sendcnt}_{b_i}\mathbf{- GEN\_SIZE + 1}} \times \mathbf{DRTT} \tag{4.1}$$

before ARC starts transmitting a packet. The ratio $\frac{recvcnt_{b_i}}{sendcnt_{b_i} - GEN\_SIZE + 1}$ indicates the number of successful sent data during the time transmitting the whole batch. If the number is high (receiving many acknowledgements) and nearly reaches to the generation size, ARC senders can be patient for receiving more data and the waiting duration is longer. If not, the waiting duration is shorter and rapidly reduced after each redundant packet is sent. Consequently, less or more redundant packets are requested to send to recover losses based on the interpretation of MAC acknowledgements. One more important note is that ARC deploys multi-batch pipeline coding scheme, which avoids the problem of generation discard and high decoding delay completely. Additionally, generation size is also set to a small value (e.g., in our simulation, generation size is set to 10) so thats our adaptive redundancy control reacts to losses more effectively. In case of losses in a generation, ARC takes priority of sending the redundant packets for the batch with the lowest generation id. After a while, the generation is recovered and ARC can move to the next generation.

4.1.4 Node behavior

ARC does not transmit every packet received from application layer (at the source) or MAC layer (at the forwarder). Instead, ARC only generated coded packets every

time-inverval **DRTT** because during **DRTT**, MAC has a high chance to be busy. Consequently, ARC only asks MAC to transmit packets at the time which is more possibly successful.

Algorithm 3 shows the coded packet reception at the destination or the forwarders. At first, the packet is added into the correct batch and is performed Gaussian elimination to check if the batch or the packet is decoded or not. The packet is then forwarded to the upper layer (at destination) or the lower layer for transmission (at forwarders). Otherwise, packet gets dropped. Algorithm 5 shows how the source handles a native packet which arrives from the upper layer. A batch will be created if the packet belongs to the new batch. The packet is then added to the batch. Algorithm 4 is triggered for packet transmission.

Algorithm 3 - Coded Packet Reception

1: **if** Intended coded packet p_{rcv} from lower layer **then**
2: For each coded packet p_{rcv} received
3: **if** $gen\_id_{p_{rcv}} \geq 0$ AND $gen\_id_{p_{rcv}} \in batchb_i$ **then**
4: Add packet p_{rcv} to batch b_i
5: Perform the Gaussian reduction on batch b_i
6: **if** batch b_i is fully decoded **then**
7: Forward the remained packets in b_i to upper layer
8: **else if** latest packet p_{rcv} decoded **then**
9: Forward p_{rcv} to upper layer
10: **end if**
11: **end if**
12: **else if** Coded packet p_{fwd} requested to forward **then**
13: For each coded packet p_{fwd} received
14: **if** $gen\_id_{p_{fwd}} \geq 0$ AND $gen\_id_{p_{fwd}} \in batchb_i$ **then**
15: Add packet p_{rcv} to batch b_i
16: Perform the Gaussian reduction on batch b_i
17: **if** $rank_{b_i}$ increased **then**
18: Call Algorithm 4 for packet transmission
19: **end if**
20: **end if**
21: **end if**

Algorithm 4 - Coded Packet Transmission

For each time-interval **DRTT**, run Algorithm 2 to find the appropriate batch b_i to send
if batch b_i exists **then**
  Generate the random linear combination **c** from batch b_i
  if **c** is generated **then**
    Increase $sendcnt_{b_i}$ by 1
    Set current time as the start of waiting time for the acknowledgement
    Request the lower layer to proceed sending packet
  end if
end if

Algorithm 4 shows the coded packet transmission at a node. For each interval **DRTT**, Algorithm 2 is triggered to locate an appropriate batch to send. A coded packet is generated from that batch for transmission. The current time is recorded as the start of waiting time for the MAC ACK. The counter $sendcnt_{b_i}$ also increases by 1.

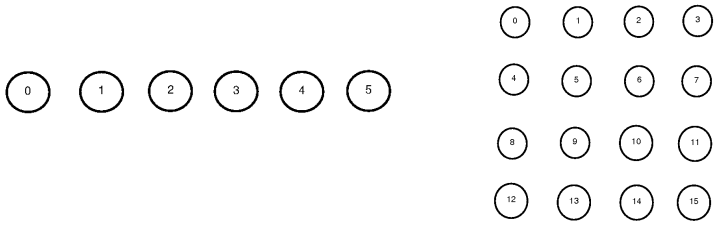
Algorithm 5 - Native Packet Reception

if Native packet p_{nativ} from upper layer **then**
  Create a batch b_i
  Add p_{nativ} to b_i
  Call Algorithm 4 for packet transmission
end if

4.1.5 Simulation and results

We use NS-2 as the simulator to compare performances of ARC with previous architectures: IEEE 802.11 and FRC - Fix Redundancy Control, a similar network coding system using fixed redundancy scheme to cover losses. Two topologies are illustrated in Figs 4-19(a) and 4-19(b). Nodes in test topologies are indexed from 0 to $\mathbb{N} - 1$ ($\mathbb{N}$ is the number of nodes in topology). The first topology (Fig 4-19(a)) is a chain of nodes for to test the packet loss and performance with light traffic via multi-hop. The second topology (Fig 4-19(b)) is a grid topology with multiple traffic flows to test the high load traffic (maximum 10 flows). Each topology is created in an area of $1000m \times 1000m$. Distance between two successive nodes is 150m. Data flows in network are all CBR (Constant Bit Rate) flows sent over UDP (User Datagram Protocol) using **1000**-byte datagrams with an arrival interval of **0.01**s. Traffic generates

12500 packets during **250**s. Channel bit-rate is set to 1Mbps and default link loss value is set to **20%**. Handshake RTS/CTS is turned off during the simulation. Protocol OLSR is used in network layer to determine the transmission path. Performances are then evaluated by two metrics, the throughput and the number of lost packets. We vary the traffic flows in test scenarios as shown in Tables 4.1. The test scenarios will be executed with each traffic flow variety for all implementations and results are collected with 95% confidence interval.



(a) Chain (b) Grid

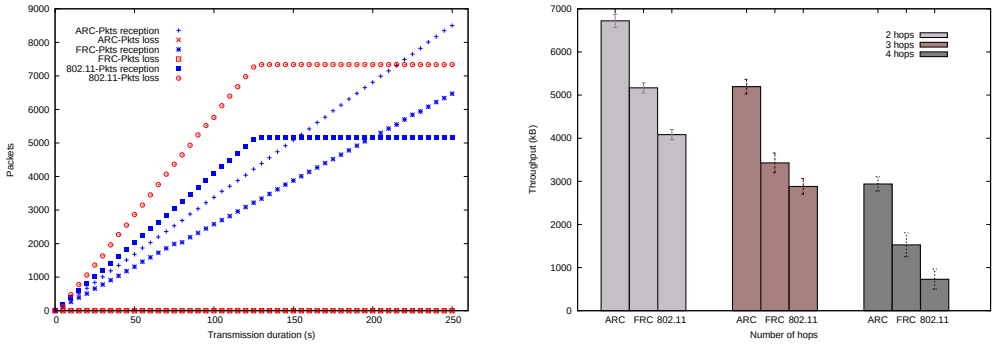
Figure 4-2: Test topologies

Table 4.1: Flows in Test scenarios

1 flow in Test scenario chain	$F_1(0 \rightarrow 1 \rightarrow 2)$
2 flows in Test scenario chain	$F_1(0 \rightarrow 1 \rightarrow 2$ and $F_2(2 \rightarrow 1 \rightarrow 0)$
2 flows in Test scenario grid	$F_1(0 \rightarrow 5 \rightarrow 10 \rightarrow 15)$ and $F_2(3 \rightarrow 6 \rightarrow 9 \rightarrow 12)$
5 flows in Test scenario grid	$F_1(0 \rightarrow 1 \rightarrow 2 \rightarrow 3)$, $F_2(15 \rightarrow 14 \rightarrow 13 \rightarrow 12)$, $F_3(1 \rightarrow 5 \rightarrow 9 \rightarrow 13)$, $F_4(14 \rightarrow 10 \rightarrow 6 \rightarrow 2)$ and $F_5(7 \rightarrow 11)$
10 flows in Test scenario grid	$F_1(0 \rightarrow 1 \rightarrow 2 \rightarrow 3)$, $F_2(15 \rightarrow 14 \rightarrow 13 \rightarrow 12)$, $F_3(0 \rightarrow 4 \rightarrow 8 \rightarrow 12)$, $F_4(15 \rightarrow 11 \rightarrow 7 \rightarrow 3)$, $F_5(4 \rightarrow 5 \rightarrow 6 \rightarrow 7)$, $F_6(11 \rightarrow 10 \rightarrow 9 \rightarrow 8)$, $F_7(0 \rightarrow 5 \rightarrow 10 \rightarrow 15)$, $F_8(3 \rightarrow 6 \rightarrow 9 \rightarrow 12)$, $F_9(1 \rightarrow 5 \rightarrow 9 \rightarrow 13)$ and $F_{10}(14 \rightarrow 10 \rightarrow 6 \rightarrow 2)$

As shown in Fig 4-3(a), even with light traffic (1 flow over two-hop connection) ARC and FRC provides a continuous and smooth streaming data (no loss) against IEEE 802.11 thanks to the mechanisms of DRTT estimation and adaptive redundancy for loss recovery. As the desired application rate is 800kbps, 802.11 doesn't have enough time to control the flow but pushing packets to the collision on the media, which results in a lot of packet drops. Moreover, the desired rate is too high for total

packets (12500 packets) to spend on the whole duration (250s). 802.11 keeps sending packets until nothing left after about 130s and the number of lost packets keeps constant. On the other hand, ARC and FRC use DRTT estimation as the interval between two successive packets to send. The desired rate of ARC and FRC now is $\frac{8000}{\mathbf{DRTT}}$. Because of **DRTT** obtained from the feedback on the state of link quality, the desired rate now is adapted flexibly to reduce packet collisions and drops on the media. Besides, using the adaptive redundancy control mentioned in Algorithm 2, ARC is not only compensated for losses by appropriate redundant packets but also provides an extra throughput (23%-35%) over FRC and IEEE 802.11, respectively. Particularly, the adaptive redundancy control gives ARC a better performance over the remained (in Fig 4-21(b)) around 24%-40% (2-hop) or 51%-70% (4-hop) over FRC and 802.11, respectively. It means, ARC is more resistant to the increasing interference and forwarding delay in the network.This can be explained if we compare the performance of ARC with FRC. If there're more hops in transmissions, FRC performance is rapidly reduced due to FRC consideration on the link quality only at the end of sending the whole batch to calculate the fixed redundancy. Based on the calculated number of redundant packets, FRC will eventually send these packets, which results in wasted traffic of over-redundancy sent to destination, decreasing the performance of FRC.

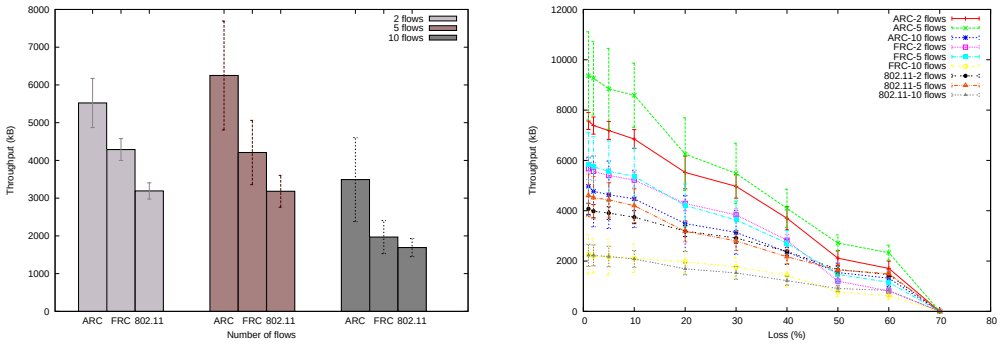


(a) Accumulative received and lost packets for one two-hop traffic flow

(b) Throughput result for one flow and its reversed on test scenario chain

Figure 4-3: Test scenario chain results

As Fig 4-4(a) shows, under high loaded traffic leading to great packet collisions and drops, ARC maintains the better throughput over 802.11 and FRC about 21%-40%, respectively. For a better illustration, we also show the result of throughput over increasing loss percent (%) in Fig 4-20(b). Let's also recall ARC (and its similar FRC) provides a lossless streaming data during the whole runtime but IEEE 802.11 does not. In the test scenario with throughput over increasing loss percent (%), our proposed adaptive redundancy scheme stably gains ARC an extra throughput over



(a) Throughput result for varied number of flows

(b) Throughput over increasing loss percent

Figure 4-4: Test scenario grid results

the others.

4.2 Dynamic Coding (DynCod)

In lossy environment like wireless networks, random losses are treated by TCP as signals of congestion and TCP cuts down the sending rate, leading to the performance degradation [16]. The practical network coding system TCP/NC [17] can address this problem. As explained in Section 2.6.5, TCP/NC masks the random losses by allowing the destination to acknowledge every *degree of freedom* even though original data is not decoded yet. Consequent, TCP/NC smoothly reacts to random losses without reducing the performance. TCP/NC is **progressive non-generation-based** coding scheme which avoids problems of *high decoding delay* and *generation discard*. The sole problem from TCP/NC is that TCP/NC uses the fixed redundancy control which does not recover *random losses* in time. We suggest an **end-to-end** adaptive redundancy control based on re-designing the coding scheme from TCP/NC. Our main idea is *how the destination can inform to the source whether the latest data sent from the source is decodable or not and how many packet losses occur via acknowledgement packets*. Particularly, we change the principle of the information delivered by TCP ACKs: the destination does not only acknowledge *every degree of freedom*, but also announces how many *unseen packets* (≥ 0) there are in the coding window at the destination. The proposition to show what is a *unseen packet* is taken from [17]:

Definition 6 *Seeing a packet: A node is said to have seen a packet p_k if it has enough information to compute a linear combination of the form $(p_k + q_k)$, where $q_k = \sum_{l>k} \alpha_l p_l$*

with $\alpha_l \in \mathbb{F}_{2^s}$ for all $l > k$. Thus, q is a linear combination involving packets with indices larger than k.

Proposition 1 *If a node has seen packet p_k, then it knows exactly one linear combination of the form $p_k + q_k$ such that q_k is itself a linear combination involving only* ***unseen*** *packets.*

For example, let's assume that one destination receives two coded packets $c_1 = p_1 + 2 \times p_2 + 3 \times p_3 + 4 \times p_4$ and $c_2 = p_1 + p_2 + 5 \times p_3 + 3 \times p_4$. The seen packets are p_1 and p_2 and the unseen packets are p_3 and p_4 because the destination can compute $2 \times c_2 - c_1 = p_1 + 7 \times p_3 + 2 \times p_4$ and $c_1 - c_2 = p_2 - 2 \times p_3 + p_4$ which are linear combinations of the form $p_k + q_k$ mentioned in **Definition** 6 and **Proposition** 1.

We argue that the number of *unseen packets* somehow reflects the number of packet losses. If by some way, we can make the reflection accurate (i.e, these two numbers have the same value), we can interpret *one unseen packet* in the destination's coding window *as a loss on the transmission path.* We present Dynamic Coding (DynCod), our re-designed coding scheme integrated with an adaptive redundancy control, to realize our idea. First, when TCP wants to transmit a packet, DynCod will send only **one** innovative coded packet. Consequently, the latest data can be decoded immediately if all coded packets up to the latest are transmitted successfully (it's very similar with the coding scheme of Pipeline Coding but it is non-generation-based). The latest data in DynCod is always transformed and presented by only **one** coded packet *at a time*. Second, thanks to this *progressive* coding scheme, losses on the transmission will **create** unseen packets in the coding window at the destination. For example, in Fig 4-5, under the *progressive* coding scheme of DynCod, a sequence of 5 coded packets c_1, c_2, c_6, c_7 and c_8 is transmitted to the destination and added to the coding window. Throughout examples in Figs 4-5, 4-7, 4-8(a) and 4-8(b), we assume that TCP ACKs will be sent immediately after its acknowledged coded packet arrives at the destination. In Fig 4-5, 3 losses create 3 unseen packets. In other words, the number of *unseen packets* can be interpreted as the number of losses. This information can be informed back to the source via TCP ACKs and the source adjusts the redundant traffic accordingly. We will describe our solution more clearly after we state the packet definition in the next section.

4.2.1 Packet definition

The i^{th} combination c_i of the latest data p_i with $n_{(c_i)}$ non-ACKed packets remained in the coding window at the time c_i generated, is as below:

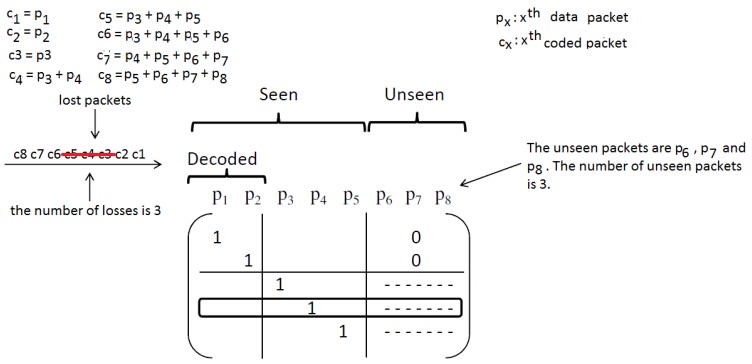


Figure 4-5: The number of unseen packets in DynCod

$$c_i = v_i^T \otimes \mathfrak{p}_i = \sum_{j=i-n_{(c_i)}}^{i} e_j \mathbf{p}_j \tag{4.2}$$

where: v_i is the encoding vector, $v_i^T = (e_1, ..., e_{n+1})$, $\mathfrak{p}_i$ is the information vector, n_{c_i} is the number of non-ACKed packets remained in the coding window at the transmission time of packet p_i, $\mathfrak{p}_i^T = (p_{i-n_{c_i}}, ..., p_i)$ and $\otimes$ is the multiplication between two vectors.

The reflection between the number of unseen packets and the number of losses is only correct if a well-designed coding scheme is used. In DynCod, the latest data is presented by only one coded packet at a time. Consequently, the latest data p_k can only be presented by the linear combination of the form $p_k + q_k$ with $q_k = 0$ if all packets up to coded packet c_k are received. Otherwise, $q_k = \sum_{j>k}^{l} p_j$ with $\alpha_l \in \mathbb{F}_{2^s}$ for all $l > k$ if there are some consecutive losses up to the arrival of packet c_l. The number of losses is the number of unseen packet in q_k. We can simply prove it. Let's assume that the destination of a traffic flow receives coded packets from c_1 to c_{k-1} and packet p_l but losses occur from packet p_k to packet p_{l-1}. We also assume $n_{c_i} = n_{c_{i-1}} + 1, \forall p_i$ without loss of generality. By the packet definition 4.2, if the destination computes the linear combination of the form $p_m + q_m (m \in [1, k-1])$ with coded packet c_m, that form is actually $c_m - c_{m-1} = p_m + 0$ with $q_m = 0$. In case there are some losses, the destination has to compute the linear combination of the form $p_k + q_k$ with coded packet p_l instead, the form turns out to be $c_l - c_{k-1} = p_k + p_{k+1} ... + p_l$ with $q_k = p_{k+1} ... + p_l$. We can easily see that the number of unseen packets in q_k is the same value as the number of losses from packet p_k to packet p_{l-1}. For instance, Fig 4-6 shows how the linear combination of the form $p_k + q_k$ is computed in DynCod.

In other words, the progressive coding scheme of DynCod will allow the destination

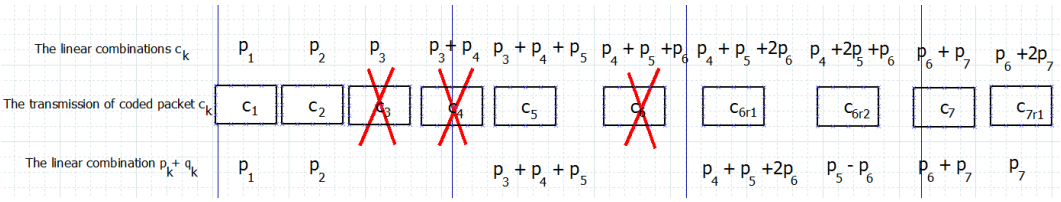


Figure 4-6: The linear combination of the form $p_k + q_k$ computed in DynCod

to acknowledges not only *every degree of freedom* (like TCP/NC) but also *how many unseen packets remains in the coding window*. The number of unseen packets exposes the need on *how many linear combinations of these unseen packets* are required to **fully decode** the coding matrix (i.e, the coding window) at the destination. In fact, it turns out to be the number of losses at the time when the source transmits the latest data. Fig 4-7 shows a traffic flow under the coding scheme of DynCod. $CX(X > 0)$ is the X^{th} coded packet while DX is the X^{th} native packet. The destination quickly notices **2** losses after the arrival of coded packet $C8$. $C8$ is not decoded because there are **2** unseen packets $D7$ and $D8$. The destination will inform the number of unseen packets to the source. Afterwards, the adaptive redundancy control at the source will be triggered to send **2** redundant packets to compensate losses. The idea is straightforward and we have an adaptive redundancy control to tackle random losses based on a tailored coding scheme.

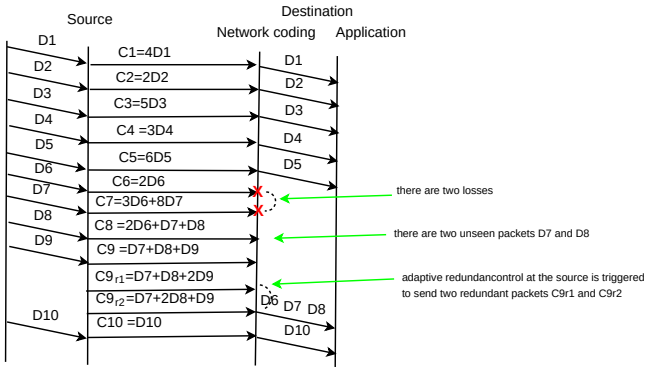


Figure 4-7: The number of losses and the number of unseen packets in DynCod

As a result, DynCod classifies the coded traffic into innovative coded packets and redundant coded packets:

- The innovative packet is the coded packet containing at least one new native packet (i.e., the latest packet that TCP requires to send).

- The redundant packet is the coded packet containing no new information but old non-ACKed packets, only generated and transmitted by the adaptive redundancy control if losses are detected.

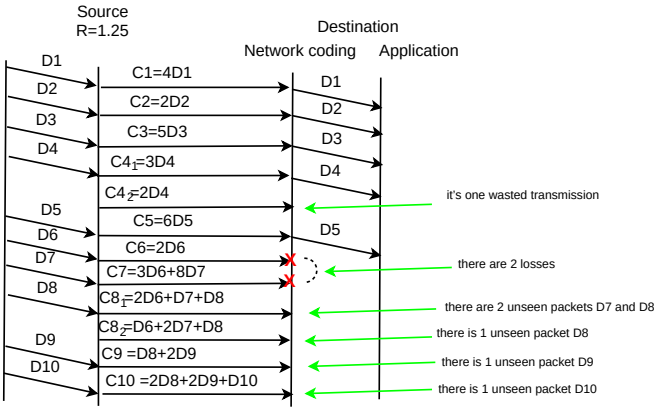
This classification will help to reduce the packet overhead, which is introduced in Section 4.2.5. The redundancy level of TCP/NC is replaced by an adaptive redundancy control, which is only triggered to send redundant coded packets if losses are detected.

We would like to discuss the reason why in the original coding scheme of TCP/NC, the number of **unseen packets** does not reflect the number of losses. TCP/NC completely ignores the constraint of "one latest data packet transmitted in a form of one innovative coded packet at a time" like DynCod. If we assume that packets from TCP layer arrive at TCP/NC layer more than one packet at a time, TCP/NC will transmit linear combinations that cannot be decoded immediately, making the number of *unseen packets* totally meaningless. For example, we consider a traffic flow from the source to the destination in Figs 4-8(a) and 4-8(b). The difference between two cases is that in Fig 4-8(b), two packets from TCP layer arrive at TCP/NC layer at a time instead of one packet in Fig 4-8(a). In both cases, there are two losses but the number of unseen packets is changed variably and cannot be used as a loss indicator.

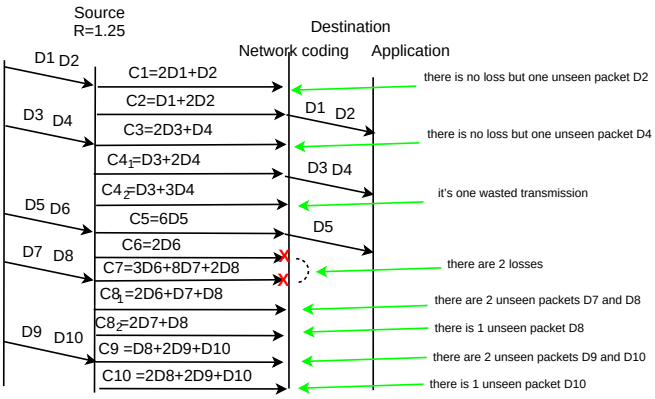
4.2.2 Design

DynCod is a network coding layer between the transport layer and the network layer of the protocol stack. The source receives packets from TCP layer and buffers them into the coding window, until they are ACKed by the sink. Each packet arrival from TCP layer will trigger DynCod to generate and send one random linear network coding combination of latest data with others non-ACKed in the coding window. If required, the redundancy control is also activated to transmit the redundant packets. At the destination side, the sink will perform the Gaussian elimination after any innovative packet arrivals. The linear equation system (4.3) can be solved progressively to reconstruct the original data.

$$\begin{bmatrix} c_{i-n} \\ \vdots \\ c_i \end{bmatrix} = \begin{bmatrix} e_1^{(c_{i-n})} & 0 & \dots & 0 \\ e_1^{(c_{i-n+1})} & e_2^{(c_{i-n+1})} & & \vdots \\ \vdots & & \ddots & 0 \\ e_1^{(c_i)} & e_2^{(c_i)} & \dots & e_{n+1}^{(c_i)} \end{bmatrix} \begin{bmatrix} p_{i-n} \\ \vdots \\ p_i \end{bmatrix} \tag{4.3}$$



(a) One packet from TCP arrives at TCP/NC at a time



(b) Two packets from TCP arrive at TCP/NC at a time

Figure 4-8: The number of losses and the number of unseen packets in TCP/NC

4.2.3 Dynamic information vector

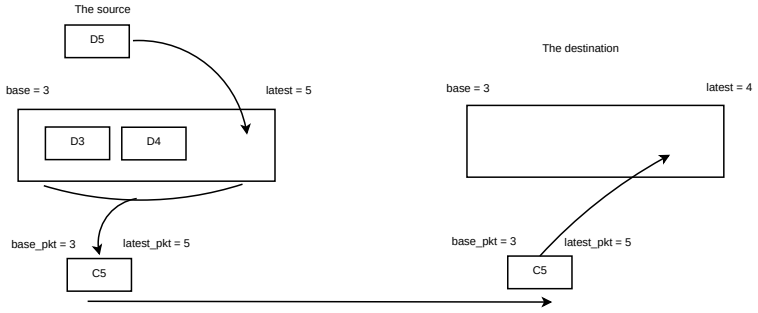


Figure 4-9: Range of native packets transmitted in combination

To keep track of information stored in the coding window, the index of the earliest coming data which still remains non-ACKed in the coding windows is stored into the variable *base* while the index of the latest data is stored into the variable *latest*. In

other words, from Equation (4.2), $base = i - n$ and $latest = i$. The coded packet generated by *DynCod* then similarly contains two variables: $base_{pkt}$ and $latest_{pkt}$ which is set based on *base* and *latest* of the source at transmission time. The index i ($i > 0$) of the packet is based on the order of packets that TCP layer wants to transmit. For example, let's examine the scenario in Fig 4-9. Currently, the source has $base = 3$ and $latest = 4$ (i.e, it contains 2 packets D3 and D4). After receiving packet $D5$ from TCP, the source add this packet to the coding window and changes $base = 3$ and $latest = 5$ (i.e, it contains 3 packets D3, D4 and D5). The source generates and transmits the coded packet $C5 = D3 + D4 + D5$ with $base_{pkt} = 3$ and $latest_{pkt} = 5$. At the destination, packet $C5$ is added into the coding window. The destination performs the Gaussian elimination and changes $base = 3$ and $latest = 5$.

In summary, let's imagine the data stored in the coding window as the information matrix $\mathfrak{M}_{i-1}$ (i.e, the latest data stored in $\mathfrak{M}_{i-1}$ is the data packet p_{i-1}) and the information vector $\mathfrak{p}_i$ in the coded packet c_i. For simplicity, we don't illustrate the coefficients along with each packets p_i. The procedure to add the information vector $\mathfrak{p}_i$ to the information matrix at a node can be illustrated as below:

$$\underbrace{\begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & & \vdots & \\ \vdots & & \ddots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & p_i \end{bmatrix}}_{\mathfrak{p}_i \text{ as an information matrix}} + \underbrace{\begin{bmatrix} p_{i-n} & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & & \vdots & \\ \vdots & & \ddots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & 0 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}}_{\mathfrak{M}_{i-1}} = \underbrace{\begin{bmatrix} p_{i-n} & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & & \vdots & \\ \vdots & & \ddots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & p_i \end{bmatrix}}_{\mathfrak{M}_i} \tag{4.4}$$

For simplicity, we assume that the packet information vector $\mathfrak{p}_i$ and the information matrix $\mathfrak{M}_i$ have the same $n_{\mathfrak{p}} = n_{\mathfrak{M}} = n$ non-ACKed packets. Otherwise, we can include more 0s into or remove some already-ACK-ed data from the information vector $\mathfrak{p}_i$. That is, if $n_{\mathfrak{p}} < n_{\mathfrak{M}} = n$ ($n_{\mathfrak{p}} + m = n$):

$$\begin{bmatrix} 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \ddots & \vdots & & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & p_{i-n+m} & p_{i-n+m+1} & \dots & p_{i-1} & p_i \end{bmatrix} + \begin{bmatrix} p_{i-n} & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \ddots & \vdots & & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-n+m} & p_{i-n+m+1} & \dots & p_{i-1} & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \end{bmatrix} \tag{4.5}$$

$$= \begin{bmatrix} p_{i-n} & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \ddots & \vdots & & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-n+m} & p_{i-n+m+1} & \dots & p_{i-1} & 0 \\ 0 & 0 & \dots & p_{i-n+m} & p_{i-n+m+1} & \dots & p_{i-1} & p_i \end{bmatrix}$$

If $n_{\mathfrak{p}} > n_{\mathfrak{M}} = n$ ($n_{\mathfrak{p}} = n + m$):

$$\begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & & \vdots & \\ \vdots & & \ddots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & p_i \end{bmatrix} + \begin{bmatrix} p_{i-n} & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & & \vdots & \\ \vdots & & \ddots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & 0 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} = \begin{bmatrix} p_{i-n} & 0 & \dots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & & \vdots & \\ \vdots & & \ddots & 0 & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & 0 \\ p_{i-n} & p_{i-n+1} & \dots & p_{i-1} & p_i \end{bmatrix} \tag{4.6}$$

It means, the useful and innovative data from the packet will be added exactly into the coding window with the dynamic size. The node then performs the Gaussian elimination over the information matrix $\mathfrak{M}$ so the row i^th will contain the data i^th (if decoded) or the data i^th along with other data with higher indices. For example, the matrix $\mathfrak{M}$ below would be a typical matrix after a Gaussian elimination:

$$\begin{bmatrix} p_{i-n} & 0 & \dots & 0 \\ 0 & p_{i-n+1} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & 0 & \dots & p_i \end{bmatrix} \tag{4.7}$$

The matrix $\mathfrak{M}$ in Equation 4.7 contains all decoded data which are ready to be forwarded to the upper layer or to the next hop.

4.2.4 Adaptive redundancy control

The main point of our proposed coding scheme DynCod is the interpretation of the number of unseen packets at the destination. The unseen packet p_i is still the linear combination of other native packets, and p_i is not decoded yet. Thanks to the progressive decoding of DynCod, one unseen packetat the destination can be interpreted as a loss. The number of unseen packets or the number of losses, stored in variable *unseen*, is calculated as:

$$unseen = latest_{pkt} - latest \tag{4.8}$$

where $latest_{pkt}$ is the index of the latest data arrived at the destination side and *latest* is the index of the data that the sink expects to receive. Besides $base, latest$, $base_{pkt}, latest_{pkt}$ mentioned in Section 4.2.2, let's denote $latest_{ack}$ as the index of ACKed data packet, *r_unseen* as the number of redundant packets needed for transmission and *r_pkt* as the index of the latest redundant packets. Fig 4-10 and 4-11 show the packet reception at both sides. For each packet requested to send by TCP, the source will generate one coded packet along with a number of redundant traffic based on *r_unseen*. The number of redundant packets *r_unseen* is calculated according to the number of losses *unseen* shown in Fig 4-10. Whenever there is a

loss ($unseen > r\_unseen > 0$), the value of r_unseen is set. The difference between $latest\_ack$ and r_pkt indicates losses even with the redundant packets. For the sink, it just needs to calculate the number of losses $unseen$ based on the difference between $latest_{pkt}$ and $latest$ as in Equation (4.8), and informs this value back to the source via ACKs.

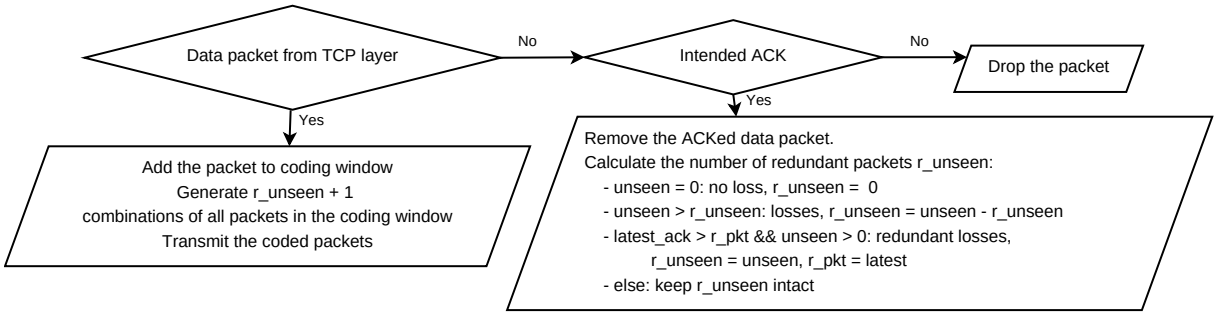


Figure 4-10: Packet reception at the source

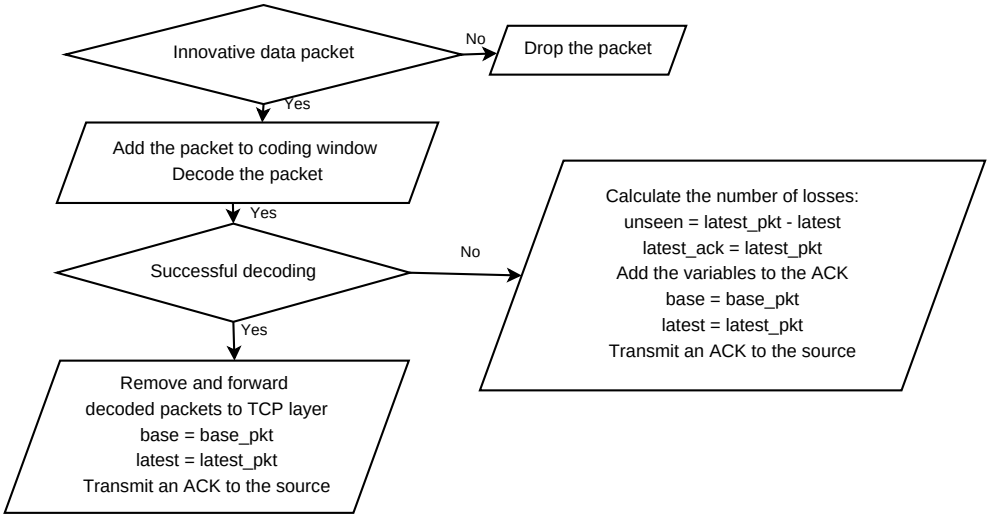


Figure 4-11: Packet reception at the sink

4.2.5 Simplified encoding vectors to reduce overhead

In DynCod, the latest data is presented by one coded packet at a time. DynCod separates coded packets into innovative (coded) packets and redundant (coded) packets. As the result, the next innovative packet always contains at least one native packet "fresher" than the previous packet. Innovative packets are ensured to be independent in spite of how their linear combinations are. We take advantage of this

property to reduce the packet overhead. Any pairs of coded packets c_i, c_j in general are independent to each other as in (4.9):

$$c_i = e_i \mathbf{p}_i + \sum_{j=i-n_{(c_i)}}^{i-1} e_j \mathbf{p}_j \leq e_i \mathbf{p}_i + \sum_{j=i-n_{(c_{i-1})}}^{i-1} e_j \mathbf{p}_j = e_i \mathbf{p}_i + c_{i-1} \tag{4.9}$$

or c_i is the combination of the new data (i.e., p_i) with at most the previous coded packet c_{i-1}. In other words, *DynCod* can select $v_i^T = (1, ..., 1)$ and remove v_i from the coding header to reduce the overhead. However, encoding vectors of redundant packets cannot be simplified because they are considered as re-broadcasts of lost packets, not transmissions of the latest data. Nevertheless, the number of redundant packets is quite small, compared with the number of normal packets transmitted and we still have the benefit from this.

4.2.6 Multipath DynCod (MP-DynCod)

In DynCod, there may be a potentially high end-to-end delay due to the losses of TCP ACKs. MP-DynCod is an extension of DynCod to support multi-path transmission and tackle the problem. We keep the same mechanism in MP-DynCod, but replace the redundancy control with a heuristic hop-by-hop adaptive redundancy control. MP-DynCod calculates the loss rate based on a link quality routing metric (e.g., ETX [19]). This information will be transferred from one node to its *forwarders* to adjust the traffic accordingly. MP-DynCod is implemented as a layer "3.5" between TCP and network layer to provide an intra-flow unicast random linear source coding system.

4.2.6.1 Design

Because we keep the same principle to handle TCP traffic from DynCod, the coded packet definition in MP-DynCod is similar as in Section 4.2. However, intermediate nodes in MP-DynCod also need to maintain the coding windows like source and sink. The *re-encoding* is performed at forwarders to avoid the non-innovative transmissions since many nodes can transmit based on the same input data. The challenge remained is the multi-path routing or how intermediate nodes co-operate to forward the data to the destination.

4.2.6.2 Forwarding belt assignment

As a part of multipath routing, we need to identify which nodes can share the task of forwarding. This is a prerequisite required for multipath routing protocol like MORE, CodeMP, or CCACK to avoid unnecessary or duplicated transmissions. In MP-DynCod, we create the forwarding belt by concluding all nodes with the same distance from the source, illustrated in Fig 4-12. For ease of implementation, we use the number of hop counts as distance metric. For transmissions, one node decides that in its next forwarding belt (containing its neighbors possible to forward traffic to the destination), which neighbor is the *primary forwarder*, which neighbor are the *backup forwarders* and add the forwarding belt with the **fractions from ETX** of each forwarder to coded packet headers based on the ETX obtained by packet-probing. The calculation of **fraction from ETX** at one node is detailed in next Section 4.2.6.3.

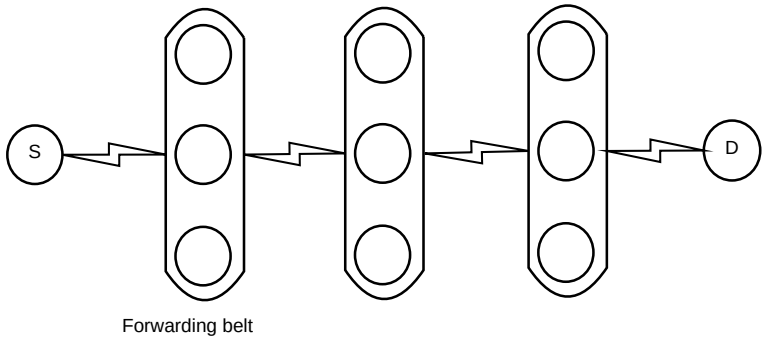


Figure 4-12: Forwarding belts

4.2.6.3 Feedback of link quality

MP-DynCod does not uses the feedbacks of transmission quality via TCP ACKs like DynCod. The TCP ACKs are only used in MP-DynCod for removing the packets *seen* (received but not decoded yet) by the destination (see Fig 4-15). In fact, the end to end feedback from destination to source like DynCod can increase packet forwarding delay. To address the remained problem in DynCod, we suggest a hop-by-hop upstream control based on a link quality routing metric (e.g. ETX).

Unlike MORE, MP-DynCod allows multipath transmission by classifying the next hops of a node into one primary forwarder and backup forwarders. Considering a traffic flow from the source to the destination, at one node in MP-DynCod, the primary forwarder is the neighbor with the highest probability of packet reception and the

other neighbors are backup forwarders. In other words, the primary forwarder handles the innovative packets while backup forwarders transmit the redundant packets to compensate losses. The probability of packet reception is calculated based on the link-quality routing metric (e.g., ETX [19]). Generally, in intra-flow network coding with multipath transmission, we need to re-encode the traffic at an intermediate node to avoid sending the non-innovative coded packets. As the primary forwarder in MP-DynCod always transmits the innovative packets from the source, only backup forwarders need to re-encode the redundant traffic.

The idea for redundancy control is straightforward: the value of ETX can be assumed as *the number of transmissions to forward the coded packets successfully from upstream* at downstream nodes (e.g., ETX at an upstream node is 1.5, which means 15 transmissions from upstream will successfully deliver 10 packets at downstream). Consequently, for every packet forwarding, one primary forwarder has 1.0 from ETX and other backup forwarders share the remainder of ETX $\delta_{ETX_{pr}}$ for transmission. The source will generate both innovative traffic and redundant traffic based on the value of ETX. In case of only 1 next hop, that node handles both traffic. Summarily, for each downstream i, **the fraction from ETX for transmission** Δ_{dwn_i} will be calculated by a upstream node as below:

$$\Delta_{dwn_i} = \begin{cases} 1.0 \;\; for\, one primary forwarder \\ \delta_{ETX_{pr}} \times \dfrac{ETX_{bk_i}}{\sum_{j=1}^{n} ETX_{bk_j}} \; else \end{cases} \tag{4.10}$$

where:

- $\delta_{ETX_{pr}} = \Delta_{ETX_{pr}} - 1.0$, $\Delta_{ETX_{pr}}$ is the ETX value from the current node to one downstream primary forwarder.
- ETX_{bk_i} is the ETX value from the current node to backup i. This is the lowest among all values of ETX from one node to its forwarders.
- $\sum_{j=1}^{n} ETX_{bk_j}$ is the sum of ETXs from current node to backup forwarders.

4.2.6.4 Adaptive redundancy control

As the idea is introduced in previous Section 4.2.6.3, *MP-DynCod* maintains a *credit system* that provides the number of transmission to forward a packet *num*. For example, in case of no loss, the number of transmission that allows primary to forward one packet is *1.0* (1 transmission per forwarding) and the number of transmissions

for backups are always *0.0* (no redundant traffic). Nodes in the forwarding belt will forward packets based on *num*, without or with re-encoding them (primary forwader or backup forwarders, respectively). The value of *num* can increase by *the fraction from ETX* informed by upstream nodes and decrease by transmissions. Whenever ETX is changed ≥ 1.0, the fraction from ETX > 1.0 will indicate the loss of the link. This will triggers downstream nodes to adjust the redundancy. According to the credit system, one node will calculate the fractions from ETX for transmissions of its forwarders. Forwarders will accumulate this value into *num* which only triggers the transmissions every integral count (e.g., $num = 1.2$ triggers 1 transmission and is reduced to 0.2). Certainly, before forwarding the packet, one node calculates the fractions from ETX for its forwarders. Eventually, the number of lost packets will be considered and compensated adaptively (Fig 4-14). Fig 4-13 illustrates a basic case of how multipath dynamic coding works. Assume there is a 2-hop traffic flow and two forwarders (one primary forwarder and one backup forwarder). The traffic is coded and transmitted from source to destination. At the time, there's a loss on the link (packet C_3). This would affect to the ETX metric calculated by the source, let's say, it gives the *fraction from ETX* to backup as "0.5". After receiving two *fractions*, the backup forwarder will transmit 1 redundant packet and reduce the fraction of *num* to 0.0. As a result, destination will decode packets successfully.

This adaptive redundancy control has two advantages. First, the diversity over the required traffic and the redundant traffic will avoid the complexed calculation for the number of required transmissions (MORE [5]). Instead, we only deal how many redundant packets needed for each backup per forwarding. That problem is much reduced to be solved by a simple formula suggested in Section 4.2.6.3. Second, we also avoid the complexity of designing and providing the good feedbacks as the adaptive redundancy control like CCACK [80] because actually, there is **no** explicit feedback from downstream nodes in MP-DynCod.

4.2.6.5 Node behavior

Each node in *MP-DynCod* has to perform coded packet and acknowledgement reception and packet transmission which are all described in Figs 4-14 and 4-15.

4.2.7 Simulation and results

4.2.7.1 DynCod

We use NS-2 as the simulator to compare performances of *DynCod* with TCP Reno and TCP/NC. Two topologies are illustrated in Figs 4-19(a) and 4-19(b). Nodes in

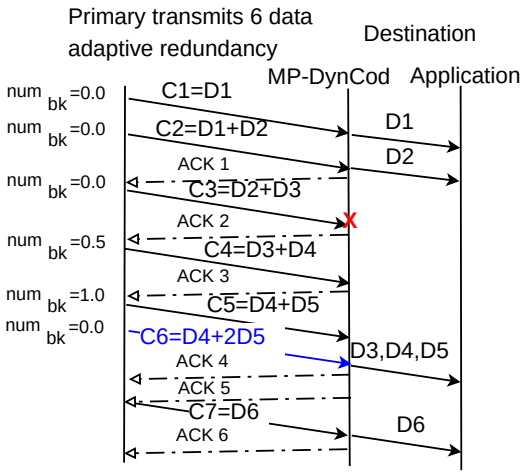


Figure 4-13: MP-DynCod

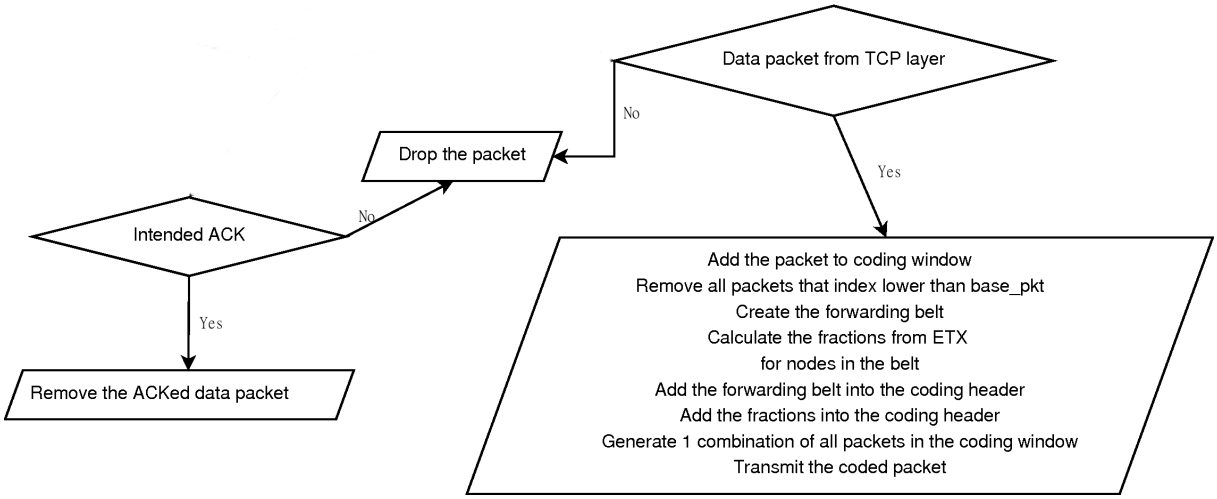


Figure 4-14: Packet reception at the sender

test topologies are indexed from 0 to $\mathbb{N}-1$ ($\mathbb{N}$ is the number of nodes in topology). The first topology (Fig 4-19(a)) is a chain of nodes to test the performance with light traffic via multi-hop. The second topology (Fig 4-19(b)) is a grid topology with multiple traffic flows to test the load traffic with high interference and loss. Each topology is created in an area of $1000m \times 1000m$. Distance between two successive nodes are 180m. Data flows in network are all FTP traffic sent over TCP Reno using 1000-byte datagrams with an arrival interval of 0.1s. Traffic is continuously generated during the test runtime of 250s. Channel bit rate is set to 1Mbps and default link loss value is set to 20% (i.e., at maximum 20% of sent packets are lost during the test). Propagation delay between two nodes is around 100ms. Performances are

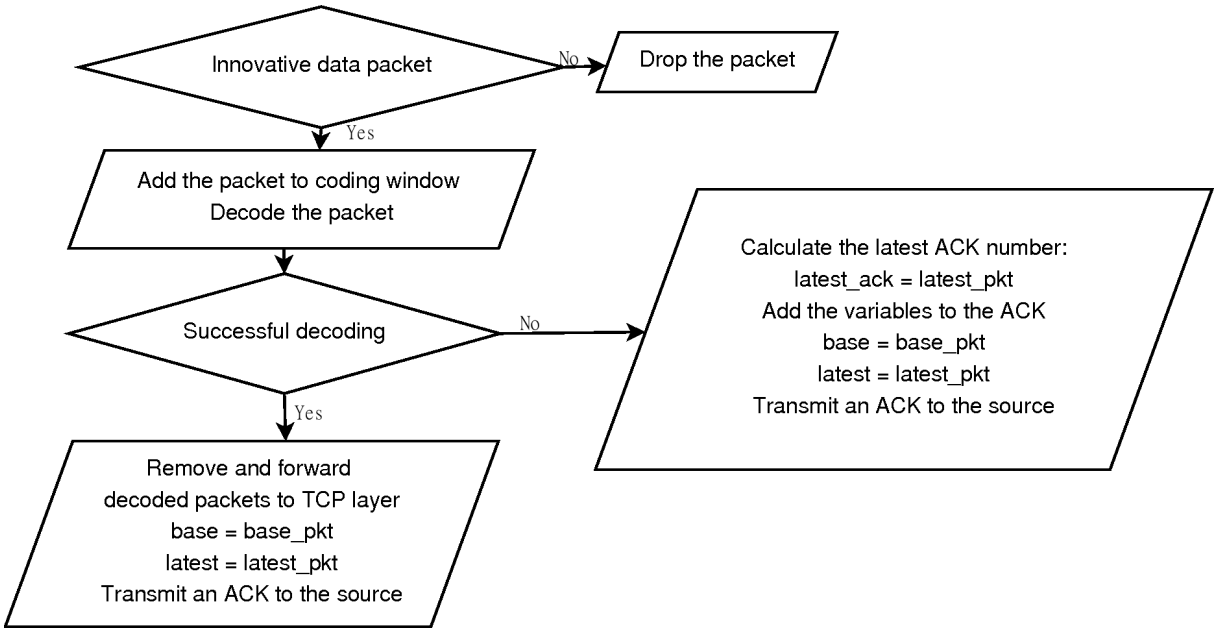
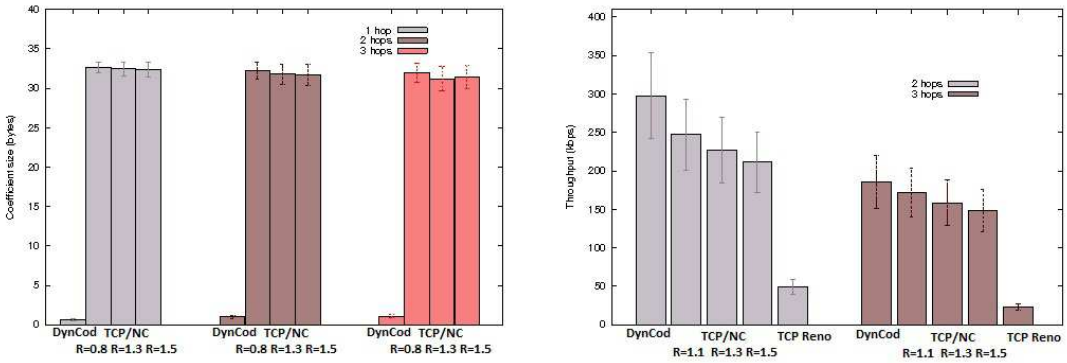


Figure 4-15: Packet reception at the receiver

then evaluated by throughput, coefficient list size per packet received at destination, and *end to end* packet delivery time (**pdt**). For each test scenario, we randomly choose sources and destinations for the traffic flows which are 2 or 3-hop length (chain topology) or 3-hop length only (grid topology). Results are collected with 95% confidence interval.



(a) Coefficient list size per packet

(b) Throughput

Figure 4-16: Test results of chain topology

As shown in Figs 4-21(b) and 4-16(a), *DynCod* achieved a thoughput which is 10-26% better than TCP/NC and 83-87% better than TCP Reno, respectively. The extra gain for *DynCod* is derived from the reduced packet overhead (around 97% against

TCP/NC) and the adaptive redundancy control. Moreover, setting the redundancy factor R like TCP/NC is quite complex and degrades the performance if TCP itself can recover from the random losses. For example, in Fig 4-20(b), in case of loss ratio 1%, even TCP Reno surpasses TCP/NCs.

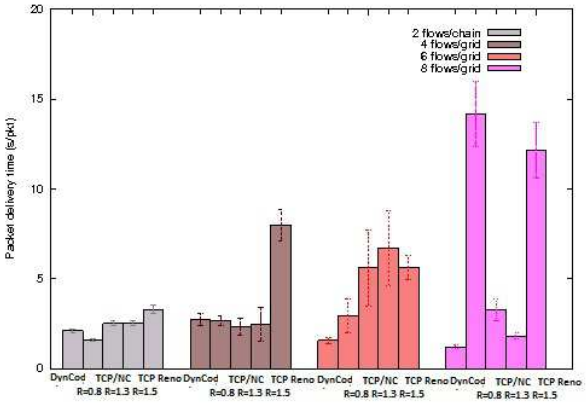


Figure 4-17: Packet delivery time for 3-hop flows in test scenarios

In a lossy environment with high interference, not only the data but also the routing information is suffered from losses. Fig 4-17 shows the pdt of 3-hop traffic flows in both test scenarios. The adaptive redundancy control allows *DynCod* to deliver packets to TCP layer at destination within an acceptable pdt (around 1.5-2.5s per packet). On the other hand, TCP Reno and TCP/NC with different R can increase the pdt significantly. For instances, with 8 traffic flows in grid topology, pdt of TCP Reno and pdt of TCP/NC increase around 50-600% and 50-700% over *DynCod*, respectively. The reason is that due to high loss and interference, packet insufficiency will result in the long TCP timeouts (TCP Reno) or decoding delays (TCP/NC), especially, in case of under-redundancy (TCP/NC R=0.8). For a better illustration, we also show the result of throughput over increasing loss ratio (%) in Fig 4-20(b). In test topology Grid with increasing loss (Fig 4-20(b)), DynCod can resist the losses (from 1% to 40%) and improve the network performance over TCP/NC.

4.2.7.2 MP-DynCod

We use NS-2 as the simulator to compare performances of *MP-DynCod* with previous architectures: *DynCod* and TCP/NC with fixed redundancy control. Two topologies are illustrated in Figs 4-19(a) and 4-19(b). Nodes in test topologies are indexed from 0 to $\mathbb{N}-1$ ($\mathbb{N}$ is the number of nodes in topology). The first topology (Fig 4-19(a)) is a chain of nodes for to test the performance with light traffic via multi-hop. The

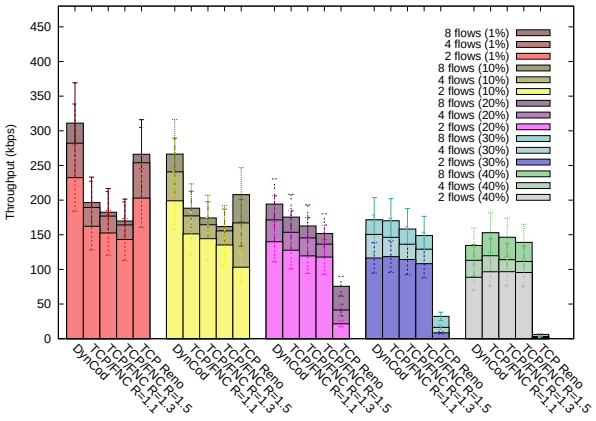
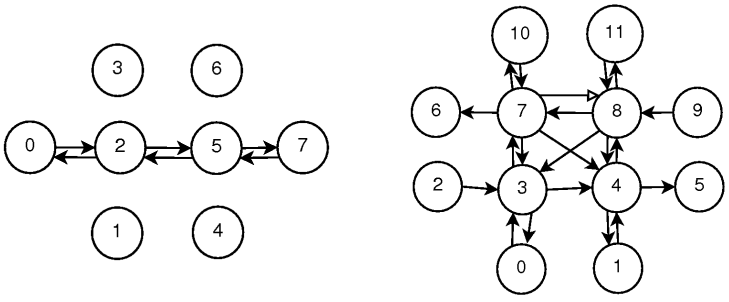


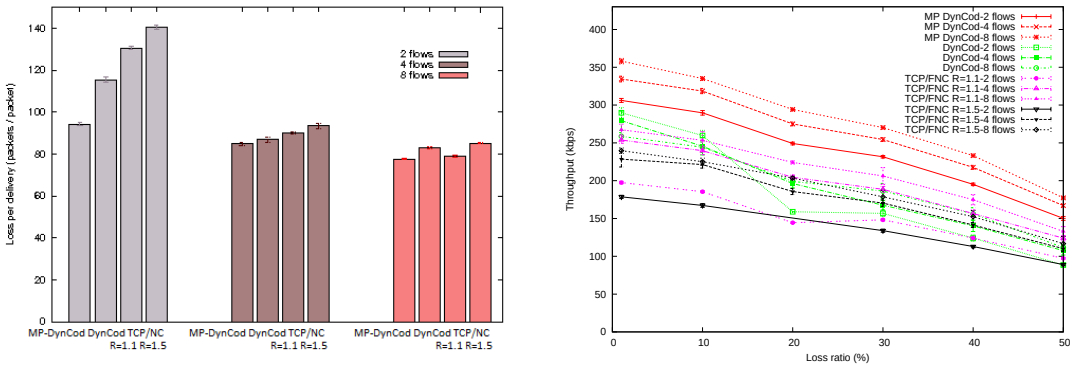
Figure 4-18: Throughput over increasing loss (%)

second topology (Fig 4-19(b)) is a grid topology with multiple traffic flows to test the load traffic with high interference and loss. Each topology is created in an area of $1000m \times 1000m$. Distance between two successive nodes are 150m. Data flows in network are all FTP traffic sent over TCP Reno using **1000**-byte datagrams with an arrival interval of **0.1**s. Traffic is continuously generated during the test runtime **250**s. Channel bit-rate is set to 1Mbps and default link loss value is set to **20%**. Propagation delay between two nodes is around 100ms. Handshake RTS/CTS is turned off during the simulation. Protocol OLSR is used in network layer to determine the transmission path. *DynCod* does not apply "coefficient list compression" to reduce the packet delivery time. Performances are then evaluated by throughput, packet overhead, and loss per delivery. For each test scenarios, we randomly pick sources and destinations for the traffic flows which are 2 or 3-hop away (chain topology) or 3-hop away only (grid topology). The test scenarios will be executed with each traffic flow variety for all implementations and results are collected with 95% confidence interval.



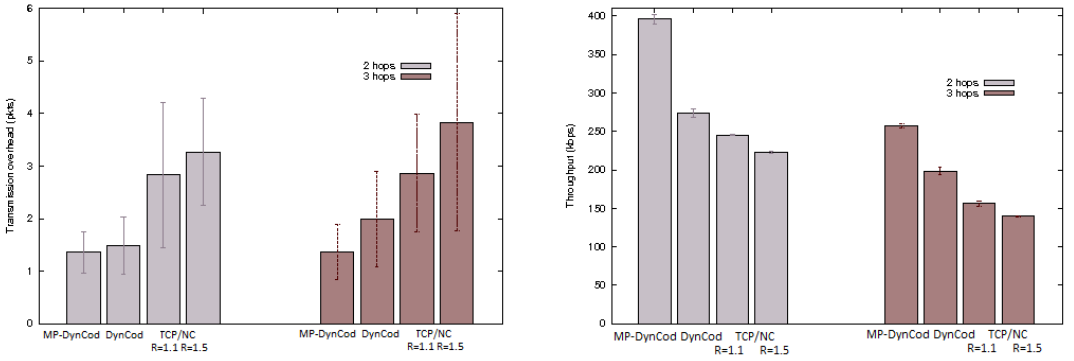
(a) Chain (b) Grid

Figure 4-19: Test topologies



(a) Loss over delivery (b) Throughput

Figure 4-20: Test results of grid topology



(a) Transmission overhead of traffic flow F_1 (b) Throughput

Figure 4-21: Test results of chain topology

As shown in Figs 4-21(b) and 4-21(a), *MP-DynCod* resisted losses and maintained the higher throughput than *DynCod* and TCP/NC approximately 20-31% and 42-45%, respectively. The extra gain for *MP-DynCod* is derived from the reduced transmission overhead (only 1.4 against 2.8 of *DynCod*, 3.2 of TCP/NC) and the adaptive redundancy control. Transmission overhead is the ratio between the number of transmission at source and the number of received packets at destination. Because *MP-DynCod* takes advantage of multipath transmission, the redundancy will not be delayed until the feedbacks (TCP ACKs) are sent back to source like *DynCod*. Instead, based on the routing metric ETX, one node will calculate the loss rate and inform its forwarders to transmit more traffic for compensation.

Fig 4-20(a) shows the loss per delivery (lpd), which is the ratio between the total number of lost packets and the number of received packets at destination. Even though under high interference and losses, *MP-DynCod* still maintains a reasonable

lpd over the others (80-96 lost packets per delivery against 85-140 of TCP/NC and 90-135 of *DynCod*). The support of multipath transmission help reducing the interference as it diverts the traffic into different routes but not simultaneously transmitted but based on the credit system. That's the reason why *MP-DynCod* gains an extra throughput around 26-28% and 28-36% compared to *DynCod* and TCP/NC, respectively (Fig 4-20(b)).

To further illustration, Fig 4-20(b) shows the throughput of all implementations with the loss increased by percent %. *MP-DynCod* provides a higher resistance against losses over the others. One note is that *DynCod* has the performance similarly equal to the ones of TCP/NC R=1.1 and TCP/NC R=1.5 because the end to end adaptive redundancy control still exposes the feedbacks (TCP ACKs) to losses, leading to the reduced throughput if loss reaches over 20%. On the other hand, *MP-DynCod* utilizes the hop by hop adaptive redundancy control. The information for redundancy control is provided based on metric ETX by packet probing. The throughput gained from *MP-DynCod* is much higher. MP-DynCod is more adaptive to losses in network, and outperforms the previous propositions.

4.3 Chapter conclusion

RLNC is well-researched and widely supported in the network coding community. In this chapter, we suggest the nouvel coding schemes to improve transmission reliability:

- ARC is the multi-batch pipeline coding with an adaptive redundant control based on the feedback of link quality provided by MAC ACKs. ARC avoids figuring out how many packets are required for the redundancy but suggests the suitable time when the physical link is more feasible for the redundant transmissions. Consequently, ARC reduces the random losses by packet congestion and interference, thus, provides the improved performance for UDP traffic.

- Dynamic Coding is the progressive non-generation based source coding scheme like TCP/NC. The difference is that our DynCod suggests the latest data being presented by only one coded packet at a time. One unseen packet in the coding window at the destination can be interpreted as a loss on the transmission. Consequently, an adaptive redundancy control is designed to support the redundant packets accordingly.

- Multipath Dynamic Coding is the multipath progressive non-generation based batch coding scheme. A hop by hop heuristic credit generating mechanism is included to help reducing the high end-to-end delay introduced in DynCod.

To support transmission reliability, we agree that a transparent network coding system is quite effective and robust to random losses. We provide three new coding schemes which help to improve the network performance.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Network coding is an interesting and promising solution that helps to enhance network performance. There are many benefits of deploying network coding, from the conventional throughput improvement to transmission reliability. The core idea of network coding is that data flowing through the network can be mixed and transformed (i.e., coded) but still lossless and recoverable (i.e., decoded) at the recipient. The main trade-off of using network coding is that the computational capability of current network stack at nodes need to be powerful. This is feasible because with modern terminals and routers, processing is becoming faster and faster, completing vast processing tasks in a blink. Network coding utilizes less expensive computational power to trade network efficiency [20].

As an emerging field to the networking research community, how to design an effective network coding system is still challenging and open to many propositions. Linear network coding is used thanks to its popularity and wide support from the research community. Moreover, linear network coding is well suited to our challenges from the thesis: the throughput improvement and the transmission reliability. In the context of applying network coding into wireless networks with unicast traffic, we propose two main contributions: one for inter-flow network coding and one for intra-flow network coding.

The first contribution, we address the benefit of network coding as throughput improvement that traffic flows from opposite directions but intercepted at an intermediate nodes can be encoded to save the transmissions, thus, more data to be transferred. We have proposed a nouvel and practical network coding architecture for unicast traffic in the wireless mesh network. We studied the previously proposed network coding systems, including COPE, BEND and DCAR, their advantages and

limitations. Based on this, the enhanced network coding system, called DODE, is redesigned and proposed to take advantages of the formers and overcome their remained issues. In the next step, we suggest DODEX (an extension of DODE) with a nouvel idea: for more than 2 packet coding, DODEX will allow *more than one* decoder on the path instead of only one. Decoders perform the decoding process cooperatively to provide the packets to destination. DODEX breaks through a basic assumption that the decoder for a traffic flow has to be only one node decoding coded packets. With that, more coding chances will be explored and more network performance can be improved. Moreover, we introduce DODEX+ to complete the coding condition. Normally, the coded packets are kept intact and forwarded to decoders even though some coding chance is found with that kind of traffic. DODEX+ allows re-encoding the coded traffic to provide the better performance. In spite of very first results that show DODEX+ is a minor improvement, DODEX+ is promising to give the enhanced network coding system. Via simulations, we show our propositions DODE, DODEX and DODEX+ surpass their previous works.

The second contribution follows the random linear network coding, a simple and effective intra-flow network coding. Random linear network coding is the distributed approach to avoid the node co-operation from locating the non-innovative encoding vectors. Randomized network coding combination are generated and proved to contain independent encoding vectors with the high probability [7]. Via well-investigating the pros and cons of current propositions (e.g., Batch Coding, Pipeline Coding and TCP/NC), we suggest a new coding scheme and an adaptive redundancy control by using MAC ACKs for UDP traffic, named ARC. ARC uses the multi-batch pipeline coding scheme that is continuously streaming new packets from generation to generation. Generations are not discarded if losses occur. They are kept and awaits the redundant packets to be decoded, which is ensured by the adaptive redundancy control provided by ARC. ARC is based on the use of MAC acknowledgements as the feedback on the state of link quality. Thus, senders can adapt the sending rate of coded packets to the current state of link quality. While interpreting the received MAC acknowledgements as the feedback on the state of link quality, ARC can find the suitable time to adjust the redundant traffic to the network to cover losses if necessary. We show that our proposition can indeed help the transmission resist against the loss, which results in smooth and lossless data streaming and increasing the bandwidth utilization. For TCP traffic, we introduce the dynamic coding scheme DynCod. In DynCod, when TCP wants to transmit a packet, DynCod will code this packet with others non-ACKed in TCP congestion window. Therefore, the latest data are always transformed and presented by only one coded packet at a time. This is the

heart of our proposed coding scheme which allows one unseen packet in the coding window at the destination to be seen as a loss on the transmission. The source can be informed via ACKs and adjust the redundant packets accordingly. TCP ACKs are interpreted as the feedback of transmission quality and DynCod can accordingly add the redundant packets. In DynCod, the end to end delay can be high due to the packet congestion and interference causing the losses of TCP ACKs. We reuse our dynamic coding scheme and adapt it to support the multipath transmission. Based on routing metric ETX, a hop-by-hop credit distributed system is proposed. MP-DynCod can classify the forwarding traffic into one primary forwarder and backup forwarders so that MP-DynCod can accordingly add the redundant packets. MP-DynCod helps to reduce the end-to-end delay and the redundant packets are provided by all related nodes, not just the source as in DynCod. Via simulation, we show ARC, DynCod and MP-DynCod will provide the better performance over previous approaches.

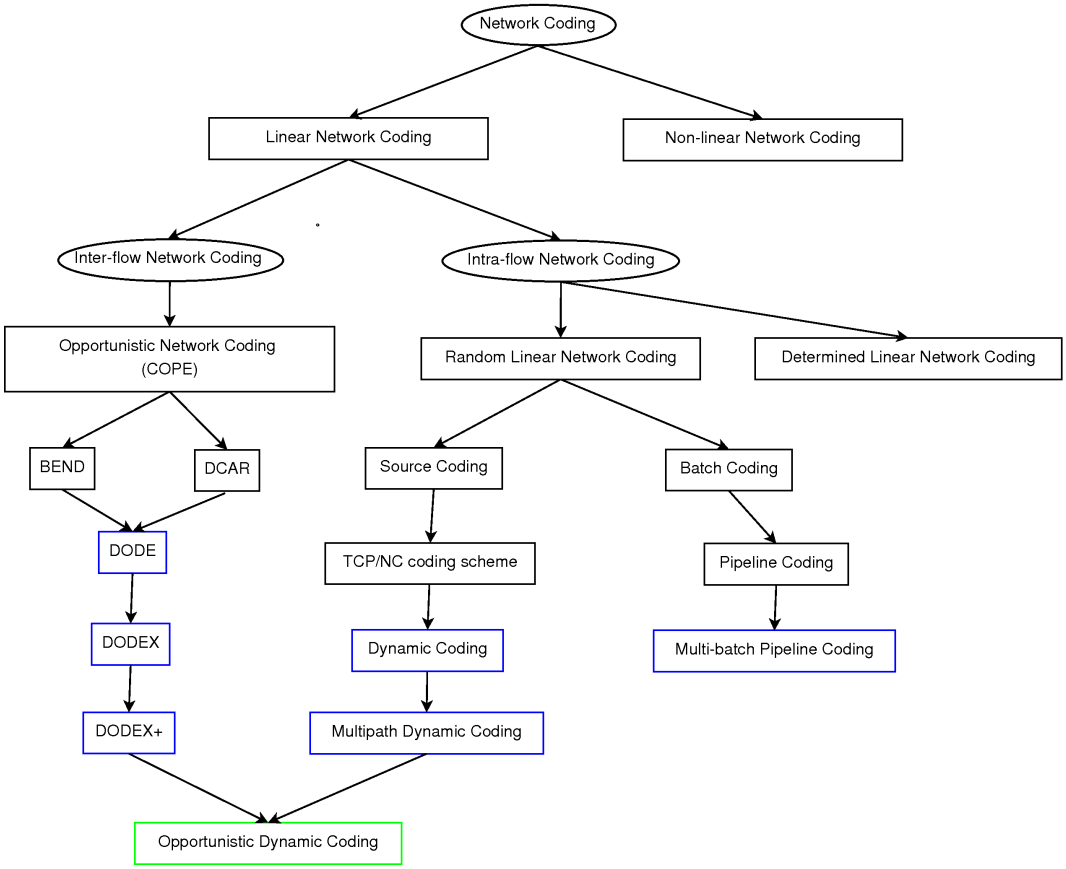


Figure 5-1: Future work

5.2 Future Work

We already explained how a generation coding condition is evolved from COPE through BEND, DCAR to DODE, DODEX and DODEX+. We still leave an practical issue unanswered: the interference between multi-flows. This problem is partially solved by PiggyCode [100] but we think the core problem can be solved if a more effective coding scheme is developed from the current dynamic coding scheme. In other words, the interference problem should be resolved at lower layer, for example, channel network coding would be a possible way to go. Combined with our current 3.5 network coding layer, the result should be very promising. We can see the two first future works pointing to the same direction: combining the two systems between DODEX+ and DynCod can provide an enhanced network coding system. We show in Fig 5-1, the future approaches open from this thesis.

Publications

CONFERENCE PAPERS

[C.1] T. V. Vu, T. M. T. Nguyen and G. Pujolle, *Distributed Opportunistic and Diffused Coding in Multi-hop Wireless Networks*, ICC, 5583-5587, 2012.

[C.2] T. V. Vu, T. M. T. Nguyen and G. Pujolle, *Distributed Opportunistic and Diffused Coding with Multiple Decoders in Wireless Mesh Networks*, MSWiM, 409-412, 2012.

[C.3] T. V. Vu, N. Boukhatem, T. M. T. Nguyen and G. Pujolle, *Adaptive Redundancy Control with Network Coding in Multi-hop Wireless Networks*, WCNC, 1510-1515, 2013.

[C.4] V. C. Phung, T. V. Vu, T. M. T. Nguyen, *DCAR Coding Gain Modeling and Analysis*, NoF, 2013.

[C.5] T. V. Vu, N. Boukhatem, T. M. T. Nguyen and G. Pujolle, *Dynamic Coding for TCP Transmission Reliability in Multi-hop Wireless Networks*, WoWMoM, 2014.

Appendices

Appendix A

Mathematical Background

In the appendix, we would like to discuss the mathematical aspects of network coding [101] and how we fit them into our proposition. We also explain some practical considerations that allows an network coding system to be practically realized.

A.1 Linear Framework $\mathbb{F}_{2^s}$

The reason why we consider a linear framework (hence, linear network coding) because the linear algebra is well-understood and the framework is quite simple and effective. Let's assume that each packet has data of L bits (e.g., in our simulation $L = 1000\text{B} = 8000\text{b}$). For the simplicity, we also consider all native packets in our system has the same size $= L$ bits. In general case of different sizes among packets, we can paddle successive "0" into the end of shorter ones to meet the same. Mathematically, we can interpret s consecutive bits of packets as a symbol over the field $\mathbb{F}_{2^s}$. Consequently, a packets will has $\frac{L}{s}$ symbols (e.g., in our simulation $s = 8$). Our proposed practical linear network coding system will provide intra-flow unicast coded traffic $Y(I)$ mentioned in the **Definition** below:

Definition 7 *Let $\mathcal{G}$ be a delay-free communication network. $\mathcal{G}$ is a $\mathbb{F}_{2^s}$ - linear network if for all links, the coded packet $Y(I)$ at node I satisfies:*

$$Y(I) = \begin{cases} \sum_{l=1}^{\mu(I)} \alpha_l X(I,l) \; if \; source \\ \sum_{K:prev(I)} \beta_K Y(K) \; else \end{cases} \tag{A.1}$$

where:

- *$Y(I)$ denotes the egress coded packet at node I.*

- *$X(I, l)(l \in (0, \mu(I)])$ denotes the packets at node I for transmission from the application indexed by l.*
- *$\mu(I)$ denotes the number of applications at node I.*
- *prev(I) denotes the previous forwarders which are nodes constituting before node I on the transmission path.*
- *α_l, β_K is the coefficients chosen from a finite field $GF = 2^m$.*

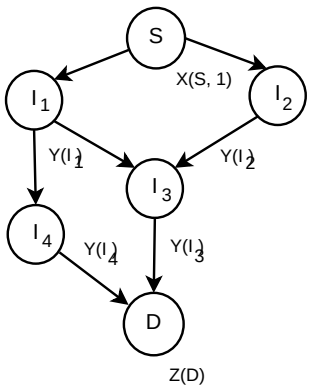


Figure A-1: An example of linear network

For illustration, Fig A-1 shows the packet $Z(D)$ at node D as
$Z_1(I_6) = Y(I_3) = \beta_{1I_3}Y(I_1) + \beta_{2I_3}Y(I_2) = \beta_{1I_3}\beta_{1I_1}\alpha_1 X(S,1) + \beta_{2I_3}\beta_{1I_2}\alpha_1 X(S,1)$
$= (\beta_{1I_3}\beta_{1I_1}\alpha_1 + \beta_{2I_3}\beta_{1I_2}\alpha_1)X(S,1)$
$Z_2(I_6) = \beta_{I_4,1}Y(I_1) = \beta_{1I_4}\beta_{1I_1}\alpha_1$
Definition 7 describes how actually a coded packet is created. Indeed, it is the combination between the received coded packets from previous forwarders. Moreover, Fig A-1 only illustrates a part of transmission path. If we look at the whole picture, the coded packets received at the destination $Z(D)$ is actually the combination of all native packets from source. Consequently, via a linear network coding system, data are *maintained the integrity but transformed*:

$$Z(D) = \sum_{l=1}^{\mu(S)} \prod_{K:prev(D)} (\beta_K)\alpha_l X(S,l) \tag{A.2}$$

From the definition and illustration, packets (elements) in a linear network coding system could be performed to satisfy the superposition and homogeneity properties:

- Combination of packets (even coded ones) will be a new packets. For example, in Fig. A-1, $Z_1(I_6) = Y(I_3) = \beta_{1I_3}Y(I_1) + \beta_{2I_3}Y(I_2) = \beta_{1I_3}\beta_{1I_1}\alpha_1 X(S,1) + \beta_{2I_3}\beta_{1I_2}\alpha_1 X(S,1)$
- Each packet could be multiplied by a *scalar*, or a *coefficient* from the Galois field $\mathbb{F}_{2^s}$, e.g., $Z_2(I_6) = \beta_{I_4,1}Y(I_1) = \beta_{1I_4}\beta_{1I_1}\alpha_1$

In other words, we can deduct how the nodes in network coding system perform the encoding process and the decoding process.

A.1.1 Encoding

- **At source**
  Assume we have n native packets $X_1(S,\mu(S)),...,X_j(S,\mu(S)),...,X_n(S,\mu(S))$ are generated by source S. To generate a coded packet $Y(S)$, source S needs to associate a list of coefficients $(\alpha_{\mu(S)1},...,\alpha_{\mu(S)j},...,\alpha_{\mu(S)n})$ in $\mathbb{F}_{2^s}$, so $Y_j(S) = \sum_{l=1}^{\mu(S)} \alpha_l X_j(S,l)$. $(\alpha_{\mu(S)1},...,\alpha_{\mu(S)j},...,\alpha_{\mu(S)n})$ is called *encoding vector* or *coefficient vector*, $Y_j(S)$ is called *information vector*.
- **At forwarders**
  Encoding process can be performed repeatedly with the coded pacekts at forwarders. It's called "re-encoding". For example, consider node I_3 in Fig A-1, it has two coded packets from I_1 and I_2: $Y(I_1)$ and $Y(I_2)$, respectively. I_3 then generates a coded packet $Y(I_3) = \beta_{I_31}Y(I_1) + \beta_{I_32}Y(I_2)$. Generally, after a re-encoding process, at an intermediate node I, a new coded packet is produced $Y(I) = \sum_{K:prev(I)} \beta_K Y(K)$ which is mentioned in **Definition** 7.

A.1.2 Decoding

Consider destination D has received m coded packets $Z(D_j)$. Node D needs to solve the linear problem $\left\{ Z(D_j) = \sum_{j=1}^{n} \prod_{K:prev(D)} (\beta_K)\alpha_{\mu(S)} X_j(S,\mu(S)) \right\}$. A crucial conditions is that $m \geq n$, meaning the number of received coded packets at least equal to the number of original packets. Otherwise, some of linear combinations are not *linearly independent* or lost on transmission. At this point, the problem is that how we ensure the linear combination (i.e., coded packets) are innovative (i.e., independent to each others) and loss-resistant. These two goals are in the aim of this thesis, which we discuss in details in next sections.

A.2 Matrix and Gaussian elimination

In a linear network, Gaussian elimination is an algorithm for solving systems of linear equations, which is a sequence of *row operations* performed on the *decoding matrix* of coefficients and reducing it into a upper triangular matrix. There are 3 types of elementary row operations:

- Swapping two rows.
- Multiplying a row by a non-zero scalar.
- Adding to one row a scalar multiple of another.

An illustration is shown in the example below:

$$
\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{matrix} \\ \xrightarrow{-3} \\ \xleftarrow{+} \end{matrix}
$$

$$
\Rightarrow \begin{bmatrix} 1 & 2 \\ 0 & -6 \end{bmatrix} \begin{matrix} \\ \\ \cdot -\frac{1}{6} \end{matrix}
$$

$$
\Rightarrow \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{matrix} \xleftarrow{+} \\ \xrightarrow{-2} \end{matrix}
$$

$$
\Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
$$

A.3 Finite field operations

Network coding requires the manipulation over packets, or performs the operations in $\mathbb{F}_{2^s}$. The operations are defined as below:

- Addition $\oplus$ is the standard bitwise XOR between two packets.
- Subtraction $\ominus$ is also the stand bitwise XOR between the sum and the subtrahend.
- Multiplication $\otimes$ is a multiplication following by a modulo an irreducible reducing polynomial (e.g., Rijndael's finite field $\mathbb{F}_{2^s}$ is $1 + x + x^3 + x^4 + x^8$). The result is the product of two packets. Summarily, in $\mathbb{F}_{2^s}$, deduce the sequence of s bits as the polynominal $b_0 + b_1 x + ... + b_{s-1} x^{s-1}$, compute the product of two deduced polynominals modulo the selected irreducible polynominal (e.g., like one in Rijndael's finite field $\mathbb{F}_{2^s}$).

- Division $\oslash$ is computed by using *Extended Euclidian algorithm* to find the multiplicative reverse b^{-1} of divisor b, that is, $a \oslash b = a \otimes b^{-1}$. The *Extended Euclidian algorithm* states that we can find b^{-1} thanks to $a \otimes b^{-1} + b \otimes a^{-1} = gcd(a, b)$ (in our case, with the Galois field $GF(2^s) = GF(256)$, $gcd(a, b) = 256$).

Appendix B

Practical Considerations

B.1 Decoding matrix

To decode packets effectively, each node maintain a *decoding matrix*, that is, containing the pair *encoding vector* and *information vector* row by row (e.g., like in Eq B.1). At first, the coded packet is added to the last row of the *decoding matrix*. Using Gaussian elimination, the matrix is transformed to triangular matrix. If the rank of matrix increased, the coded packet is innovative (i.e., new information). Otherwise, the row of this packet is reduced to all 0s. After n innovative packets are well received and Gaussian-eliminated, the triangular matrix will be transformed into diagonal matrix The remain in *information vector* will be the original packets. Decoding process is done.

$$\underbrace{\left[\begin{array}{ccc} X_1 & 0 & 0 \\ 0 & X_2 & 0 \\ 0 & 2 & X_3 \end{array}\right.}_{encoding\ vector} \underbrace{\left.\begin{array}{c} Y_1 \\ Y_2 \\ Y_3 \end{array}\right]}_{information\ vector} \tag{B.1}$$

B.2 Decoding delay

Network coding has an special impact on the decoding delay. The very proposition [6] suggest the *batch coding* scheme: k packets are grouped into a batch and at least k linear combinations will be generated and transmitted from this batch. This coding scheme continues being adapted to many propositions [5], [80], [75], [76], [77], [79] like a fundamental mechanism. Let's consider the batch coding with fixed redundancy control in [6]. Generally, assume we use a batch coding with size = k, redundancy = r. Furthermore, assume t_i is the time packet i^{th} added into the coding window,

t_{c_i} is the time coded packet intended for packet i^{th} transmission is sent. Eventually, let's denote d_{trx} is the average transfer delay per transmission including processing, transmission, propagation and queuing delay and d_c is the average decoding delay per packet. In case of no loss, the packet delivery time pdt of packet i^{th} calculated by [6] is:

$$pdt_i = t_{c_{i+k-(i\%k+1)+l}} - t_i + d_{trx} + d_c \tag{B.2}$$

Equation B.2 shows that the recipient has awaited at least k successful transmission before it can decode the packets. Moreover, each packet contains all native ones mixed together, leading to a careful selection of coefficients to ensure the linear independence. Even with Random Linear Network Coding [32], there is a probability to get non-innovative if we consider the high traffic and multipath streaming. More coding and decoding schemes are proposed [10], [11] to significantly reduce this impact.

Appendix C

Ad-hoc routing protocols

C.1 Destination-Sequenced Distance Vector (DSDV)

Destination-Sequenced Distance-Vector Routing (DSDV) [69] is a table-driven routing scheme for ad hoc mobile networks based on the Bellman Ford algorithm. Each entry in the routing table maintains a sequence number. If a link is present, the sequence number is even. Otherwise, the sequence number is odd. The destination generates the sequence number. If one node has a routing path to the destination, the node has to send the routing update with this sequence number. Routing information is exchanged among nodes by sending full routing table updates and incremental entry updates. Incremental entry updates are frequently triggered to send while full routing table updates are infrequently transmitted. The route with the latest sequence number is selected. If there are two routes with the same sequence number, the route with the better metric is selected. If a routing entry is not updated for a while, it becomes *stale*. Stale entries along with the associated routes will be removed from the routing table.

C.2 Ad-hoc On-demand Distance Vector (AODV)

Ad-hoc On-demand Distance Vector (AODV) [67] is a reactive routing protocol for mobile ad hoc networks (MANETs) and other wireless ad hoc networks. In AODV, if there is a routing demand from a transmitter to a receiver, the transmitter will broadcast a routing request to the network. Other nodes forward the request, and record which node they hear the request from. Consequently, a number of routing requests are transmitted to the receiver. If there is one node on the routing path which already obtains the route to the destination, that node will send a routing

reply travelling back to the transmitter via the forwarders which forwarded the routing request. Otherwise, the receiver receives the routing request and uses the route with the least number of hops. Unused routing entries will be purged after a duration. When a link fails, a routing error message is passed backward to the transmitter, and the routing process above repeats.

References

[1] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. XORs in the air: Practical wireless network coding. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 243–254, New York, NY, USA, 2006. ACM.

[3] Jian Zhang, Yuanzhu Peter Chen, and Ivan Marsic. MAC-layer proactive mixing for network coding in multi-hop wireless networks. *Computer Networks*, 54(2):196–207, 2010.

[4] Jilin Le, John Chi Shing Lui, and Dah Ming Chiu. DCAR: Distributed coding-aware routing in wireless networks. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ICDCS '08, pages 462–469, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. Trading structure for randomness in wireless opportunistic routing. In *SIGCOMM*, pages 169–180, 2007.

[6] Philip A. Chou, Yunnan Wu, and Kamal Jain. Practical network coding. In *Allerton Conference on Communication, Control, and Computing*, 2003.

[7] Tracey Ho, Muriel Medard, Jun Shi, Michelle Effros, and David R. Karger. On randomized network coding. In *In Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing*, 2003.

[8] Joon-Sang Park, Mario Gerla, Desmond S. Lun, Yunjung Yi, and Muriel Médard. CodeCast: a network-coding-based ad hoc multicast protocol. *IEEE Wireless Commun.*, 13(5):76–81, 2006.

[9] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Netw.*, 11(5):782–795, 2003.

[10] Chien-Chia Chen, Soon Young Oh, Phillip Tao, Mario Gerla, and M.Y. Sanadidi. Pipeline network coding for multicast streams. In *Proceedings of the 5th International Conference on Mobile Computing and Ubiquitous Networking (ICMU), Seattle, U.S.A*, 2010.

[11] Chien-Chia Chen, Clifford Chen, Soon Y. Oh, Joon-Sang Park, Mario Gerla, and M. Y. Sanadidi. ComboCoding: Combined intra-/interflow network coding for TCP over disruptive MANETs. *Journal of Advanced Research*, 2:241–252, 2011.

[12] Hamlet Medina Ruiz, Michel Kieffer, and Batrice Pesquet-Popescu. An adaptive redundancy scheme for TCP with network coding. In *"IEEE International Symposium on Network Coding (NETCOD), United States*. IEEE, 2012.

[13] Thuong Van Vu, Thi Mai Trang Nguyen, and Guy Pujolle. Distributed opportunistic and diffused coding in multi-hop wireless networks. In *ICC*, pages 5583–5587. IEEE, 2012.

[14] Thuong Van Vu, Thi Mai Trang Nguyen, and Guy Pujolle. Distributed opportunistic and diffused coding with multiple decoders in wireless mesh networks. In *Proceedings of the 15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '12, pages 409–412. ACM, 2012.

[15] Thuong Van Vu, N. Boukhatem, Thi Mai Trang Nguyen, and G. Pujolle. Adaptive redundancy control with network coding in multi-hop wireless networks. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 1510–1515, April 2013.

[16] Sumit Rangwala, Apoorva Jindal, Ki-Young Jang, Konstantinos Psounis, and Ramesh Govindan. Understanding congestion control in multi-hop wireless mesh networks. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, MobiCom '08, pages 291–302, New York, NY, USA, 2008. ACM.

[17] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Michael Mitzenmacher, and João Barros. Network coding meets TCP. In *INFOCOM*, pages 280–288, 2009.

[18] Thuong Van Vu, N. Boukhatem, Thi Mai Trang Nguyen, and G. Pujolle. Dynamic coding for tcp transmission reliability in multi-hop wireless networks. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) 2014*, June 2014.

[19] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, MobiCom '03, pages 134–146, New York, NY, USA, 2003. ACM.

[20] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, January 2006.

[21] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[22] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach, 3rd Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[23] Søren Riis. Linear versus non-linear boolean functions in network flow. In *CISS*, 2004.

[24] A. Blasiak, R. Kleinberg, and E. Lubetzky. Lexicographic products and the power of non-linear network coding. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 609–618, 2011.

[25] Hongyi Yao and E. Verbin. Network coding is highly non-approximable. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 209–213, 2009.

[26] Oliver Kosut, Lang Tong, and David Tse. Nonlinear network coding is necessary to combat general byzantine attacks. In *in 47th Annual Allerton Conference on Communication, Control, and Computing*, 2009.

[27] R. Dougherty, C. Freiling, and K. Zeger. Insufficiency of linear coding in network information flow. *Information Theory, IEEE Transactions on*, 51(8):2745–2759, Aug 2005.

[28] T. Koike-Akino, P. Larsson, P. Popovski, and Vahid Tarokh. Non-linear network coding in two-way relaying discrete channels. In *Wireless Communications*

Signal Processing, 2009. WCSP 2009. International Conference on, pages 1–5, 2009.

[29] F. Davoli, M. Marchese, and M. Mongelli. Non-linear coding and decoding strategies exploiting spatial correlation in wireless sensor networks. *Communications, IET*, 6(14):2198–2207, 2012.

[30] S. Y. R. Li and R. W. Yeung. Network multicast flow via linear coding. 1998.

[31] Peter Sanders, Sebastian Egner, and Ludo Tolhuizen. Polynomial time algorithms for network information flow. In *in 15th ACM Symposium on Parallel Algorithms and Architectures*, pages 286–294, 2003.

[32] Tracey Ho, Ralf Koetter, Muriel Mdard, David R. Karger, and Michelle Effros. The benefits of coding over routing in a randomized setting. In *In Proceedings of 2003 IEEE International Symposium on Information Theory*, 2003.

[33] Yu Wang, Hancheng Lu, Peilin Hong, and Kaiping Xue. Practical wireless network coding with constrained decoding buffers. In *Network Coding (NetCod), 2010 IEEE International Symposium on*, pages 1–6, 2010.

[34] Jinyi Zhou, Shutao Xia, Yong Jiang, and Haitao Zheng. Decoding buffer management in practical wireless network coding. In *Network Coding (NetCod), 2011 International Symposium on*, pages 1–6, 2011.

[35] Chao Dong, Guihai Chen, Weibo Yu, Panlong Yang, and Hai Wang. How to discover the coding opportunity for practical wireless network coding? In *Wireless Communications Signal Processing, 2009. WCSP 2009. International Conference on*, pages 1–4, 2009.

[36] T. Aktas, A.O. Yilmaz, and E. Aktas. Practical wireless network coding and decoding methods for multiple unicast transmissions. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pages 6–11, 2012.

[37] Huan Wang, Kaiping Xue, Peilin Hong, and Hancheng Lu. Impact of traffic pattern on benefits of practical multi-hop network coding in wireless networks. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 1197–1201, 2011.

[38] S. Omiwade, Rong Zheng, and Cunqing Hua. Practical localized network coding in wireless mesh networks. In *Sensor, Mesh and Ad Hoc Communications*

and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on, pages 332–340, 2008.

[39] Bin Ni, Naveen Santhapuri, Zifei Zhong, and Srihari Nelakuditi. Routing with opportunistically coded exchanges in wireless mesh networks. In *WiMesh*, 2006.

[40] Youghourta Benfattoum, Steven Martin, and Khaldoun Al Agha. IROCX: Interference-aware routing with opportunistically coded exchanges in wireless mesh networks. In *WCNC*, pages 1113–1118, 2011.

[41] Youghourta Benfattoum, Steven Martin, and Khaldoun Al Agha. TC-IROCX: Network coding with topology control and interference awareness. In *WCNC*, pages 1970–1975, 2012.

[42] Shinji Kano, Takahiro Wada, and Iwao Sasasei. Route switching protocol for network coding in ad hoc networks. In *Iwano Communication and Electronics (ICCE)*, 2010.

[43] Cong Liu and Jie Wu. Adaptive routing in dynamic ad hoc networks. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 2603–2608, March 2008.

[44] Song Han, Zifei Zhong, Hongxing Li, Guihai Chen, E. Chan, and A.K. Mok. Coding-aware multi-path routing in multi-hop wireless networks. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 93–100, 2008.

[45] Seong-Hee Lee, Junwhan Kim, and Ho gil Cho. Coding-aware real-time routing in multi-hop wireless sensor networks. In *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pages 443–444, 2011.

[46] Jin Wang, Cenzhe Zhu, Qinfeng Guo, Chai Teck Yoong, and Wai-Choong Wong. SCAR: A dynamic coding-aware routing protocol. In *Signal Processing and Communication Systems (ICSPCS), 2012 6th International Conference on*, pages 1–5, 2012.

[47] Yan Yan, Baoxian Zhang, Jun Zheng, and Jian Ma. Core: a coding-aware opportunistic routing mechanism for wireless mesh networks [accepted from open call]. *Wireless Communications, IEEE*, 17(3):96–103, June 2010.

[48] Xin Wei, Li Zhao, Ji Xi, and Qingyun Wang. Network coding aware routing protocol for lossy wireless networks. In *Wireless Communications, Networking*

and Mobile Computing, 2009. WiCom '09. 5th International Conference on, pages 1–4, 2009.

[49] Bin Guo, Hongkun Li, Chi Zhou, and Yu Cheng. Analysis of general network coding conditions and design of a free-ride-oriented routing metric. *Vehicular Technology, IEEE Transactions on*, 60(4):1714–1727, 2011.

[50] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[51] Luigi Rizzoi. Effective erasure codes for reliable computer communication protocols. *SIGCOMM Comput. Commun. Rev.*, 27(2):24–36, April 1997.

[52] Christina Fragouli, Jrg Widmer, and Jean-Yves Le Boudec. A network coding approach to energy efficient broadcasting: from theory to practice. In *IN PROC. OF IEEE INFOCOM*, 2006.

[53] Yufang Xi and E.M. Yeh. Distributed algorithms for minimum cost multicast with network coding. *Networking, IEEE/ACM Transactions on*, 18(2):379–392, April 2010.

[54] Tracey Ho, Jia-Qi Jin, and Harish Viswanathan. On network coding and routing in dynamic wireless multicast networks. 2006.

[55] Jingyao Zhang and Pingyi Fan. On network coding in wireless ad-hoc networks. *IJAHUC*, 2(3):140–148, 2007.

[56] Hulya Seferoglu, Athina Markopoulou, and K. K. Ramakrishnan. I^2NC: Intra- and inter-session network coding for unicast flows in wireless networks. In *INFOCOM*, pages 1035–1043, 2011.

[57] Y. Wu, P. A. Chou, and S. Y. Kung. Information exchange in wireless networks with network coding and physical-layer broadcast. In *Proc. 39th Annual Conf. Inform. Sci. and Systems (CISS)*, 2005.

[58] Sachin Katti, Dina Katabi, Wenjun Hu, Hariharan Rahul, and Muriel Medard. The Importance of Being Opportunistic: Practical Network Coding for Wireless Environments. In *Allerton Annual Conference on Communication, Control and Computing*, 2005.

[59] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 151–162, New York, NY, USA, 1999. ACM.

[60] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3866–3875, Jan 2002.

[61] Wei Peng and Xi-Cheng Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proceedings of the 1st ACM International Symposium on Mobile Ad Hoc Networking & Computing*, MobiHoc '00, pages 129–130, Piscataway, NJ, USA, 2000. IEEE Press.

[62] T. Matsuda, T. Noguchi, and T. Takine. Broadcasting with Randomized Network Coding in Dense Wireless Ad Hoc Networks. *IEICE Transactions on Communications*, 91:3216–3225, 2010.

[63] Li Li, R. Ramjee, M. Buddhikot, and S. Miller. Network coding-based broadcast in mobile ad-hoc networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1739–1747, May 2007.

[64] Ihong Hou, Yuen Tsai, Tarek F. Abdelzaher, and Indranil Gupta. AdapCode: Adaptive network coding for code updates. In *in Wireless Sensor Networks, in Proceedings of IEEE INFOCOM*, 2008.

[65] Shuhui Yang and Jie Wu. Efficient broadcasting using network coding and directional antennas in MANETs. *Parallel and Distributed Systems, IEEE Transactions on*, 21(2):148–161, Feb 2010.

[66] Zhenyu Yang, Ming Li, and Wenjing Lou. R-Code: Network coding based reliable broadcast in wireless mesh networks with unreliable links. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–6, Nov 2009.

[67] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. IETF RFC 3561, 2003.

[68] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007.

[69] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, pages 234–244, New York, NY, USA, 1994. ACM.

[70] Jilin Le, John C. S. Lui, and Dah-Ming Chiu. On the performance bounds of practical wireless network coding. pages 1134–1146, 2010.

[71] Jilin Le, J.C.S. Lui, and Dah Ming Chiu. How many packets can we encode? - An analysis of practical wireless network coding. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[72] Sanjit Biswas and Robert Morris. Opportunistic routing in multi-hop wireless networks, 2005.

[73] A. Tiwari, A. Ganguli, Ashwin Sampath, D.S. Anderson, Bao-Hong Shen, N. Krishnamurthi, J. Yadegar, M. Gerla, and D. Krzysiak. Mobility aware routing for the airborne network backbone. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7, 2008.

[74] Eric Rozner, Jayesh Seshadri, Yogita Ashok Mehta, and Lili Qiu. SOAR: Simple opportunistic adaptive routing protocol for wireless mesh networks. *IEEE Trans. Mob. Comput.*, 8(12):1622–1635, 2009.

[75] Yunfeng Lin, Baochun Li, and Ben Liang. CodeOR: Opportunistic routing in wireless mesh networks with segmented network coding. In *ICNP*, pages 13–22, 2008.

[76] Yunfeng Lin, Ben Liang, and Baochun Li. SlideOR: Online opportunistic network coding in wireless mesh networks. In *INFOCOM*, pages 171–175, 2010.

[77] Steluta Gheorghiu, Alberto López Toledo, and P. Rodriguez. Multipath TCP with network coding for wireless mesh networks. In *ICC*, pages 1–5, 2010.

[78] Chien-Chia Chen, Guruprasad Tahasildar, Yu-Ting Yu, Joon-Sang Park, Mario Gerla, and M. Y. Sanadidi. CodeMP: Network coded multipath to support tcp in disruptive MANETS. In *MASS*, pages 209–217, 2012.

[79] Xinyu Zhang and Baochun Li. DICE: a game theoretic framework for wireless multipath network coding. In *MobiHoc*, pages 293–302, 2008.

[80] Dimitrios Koutsonikolas, Chih-Chun Wang, and Y. Charlie Hu. CCACK: Efficient network coding based opportunistic routing through cumulative coded acknowledgments. In *INFOCOM*, pages 2919–2927, 2010.

[81] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran. Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 111–117, April 2005.

[82] Dragan Petrovi, Kannan Ramchandran, and Jan Rabaey. Overcoming untuned radios in wireless networks with network coding. *IEEE TRANSACTIONS ON INFORMATION THEORY*, pages 2649–2657, 2006.

[83] P. Ingtoft T. Madsen R. Jacobsen, K. Jakobsen and F.H.P. Fitzek. Practical Evaluation of Partial Network Coding in Wireless Sensor Networks. In *4th International Mobile Multimedia Communications Conference (MobiMedia 2008)*, Oulu, Finland, July 2008. ICTS/ACM.

[84] Vahid Shah-Mansouri and Vincent W. S. Wong. Link-by-link feedback mechanism for intra-session random linear network coding in wireless sensor networks. In *ICC*, pages 681–686, 2012.

[85] Ismail Salhi, Erwan Livolant, Yacine Ghamri-Doudane, and Stéphane Lohier. ZInC: Index-coding for many-to-one communications in zigbee sensor networks. In *ICC*, pages 783–788, 2012.

[86] www.bittorrent.com. 2001.

[87] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2235–2245 vol. 4. IEEE, March 2005.

[88] M. Martalo, M. Picone, R. Bussandri, and M. Amoretti. A practical network coding approach for peer-to-peer distributed storage. In *Network Coding (NetCod), 2010 IEEE International Symposium on*, pages 1–6, 2010.

[89] Y. Wu. Existence and construction of capacity-achieving network codes for distributed storage. *Selected Areas in Communications, IEEE Journal on*, 28(2):277–288, 2010.

[90] M. Martalo, M. Picone, M. Amoretti, G. Ferrari, and R. Raheli. Randomized network coding in distributed storage systems with layered overlay. In *Information Theory and Applications Workshop (ITA), 2011*, pages 1–7, 2011.

[91] Ning Wang and Jiaru Lin. Network coding for distributed data storage and continuous collection in wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–4, 2008.

[92] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *Information Theory, IEEE Transactions on*, 52(6):2809–2816, 2006.

[93] A.G. Dimakis, P.B. Godfrey, Y. Wu, M.J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*, 56(9):4539–4551, 2010.

[94] Ning Cai and R.W. Yeung. Secure network coding on a wiretap network. *Information Theory, IEEE Transactions on*, 57(1):424–435, 2011.

[95] Young-Sik Kim. Refined secure network coding scheme with no restriction on coding vectors. *Communications Letters, IEEE*, 16(11):1907–1910, 2012.

[96] Ning Cai and T. Chan. Theory of secure network coding. *Proceedings of the IEEE*, 99(3):421–437, 2011.

[97] Muxi Yan and A. Sprintson. Weakly secure network coding for wireless cooperative data exchange. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5, 2011.

[98] Yawen Wei, Zhen Yu, and Yong Guan. Efficient weakly-secure network coding schemes against wiretapping attacks. In *Network Coding (NetCod), 2010 IEEE International Symposium on*, pages 1–6, 2010.

[99] C. Gkantsidis and P.R. Rodriguez. Cooperative security for network coding file distribution. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–13, 2006.

[100] Luca Scalia, Fabio Soldo, and Mario Gerla. Piggycode: A MAC layer network coding scheme to improve TCP performance over wireless networks. In *Proceedings of the Global Communications Conference, 2007. GLOBECOM 07, Washington, DC, USA, 26-30 November 2007*, pages 3672–3677. IEEE, 2007.

[101] N. R. Wagner. The laws of cryptography with java code, 2000.