University of Crete
School of Sciences and Engineering
Computer Science Department

NETWORK CODING

by

NIKOLAOS PAPPAS

Master's Thesis

Heraklion, June 2007

UNIVERSITY OF CRETE
SCHOOL OF SCIENCES & ENGINEERING
COMPUTER SCIENCE DEPARTMENT

NETWORK CODING

by

NIKOLAOS PAPPAS

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

Author:

_____

Nikolaos Pappas, Computer Science Department

Supervisory Committee:

_____

Apostolos Traganitis, Professor, Supervisor

_____

Vassilis Siris, Assistant Professor, Member

_____

Panagiotis Tsakalides, Associate Professor, Member

Approved by:

_____

Panos Trahanias, Professor

Chairman of the Graduate Studies Committee

Κωδικοποίηση Δικτύου

Νικόλαος Παππάς

Μεταπτυχιακός Φοιτητής

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Επόπτης Καθηγητής Μετ. Εργασίας: Απόστολος Τραγανίτης

Περίληψη

Η κωδικοποίηση δικτύου είναι ένας νέος ερευνητικός τομέας που παρουσιάζει ενδιαφέρουσες εφαρμογές όχι μόνο στη θεωρία πληροφοριών και κωδικοποίησης, αλλά και σε πρακτικά συστήματα δικτύων. Η κωδικοποίηση δικτύου επεκτείνει τη λειτουργικότητα των δικτύων πέρα από την παραδοσιακή δρομολόγηση ή τις τεχνικές αποθήκευσης και προώθησης, εισάγωντας την μαθηματική επεξεργασία των δεδομένων (κωδικοποίηση) μέσα στα δίκτυα. Ο Ahlswede κ.ά. μελέτησαν την πολυεκπομπή σε ένα δίκτυο χωρίς απώλειες συνδέσμων και έδειξαν ότι η επίτευξη της χωρητικότητας της πολυεκπομπής απαιτεί γενικά τη χρήση ενός κώδικα δικτύου. Περαιτέρω εργασίες έδειξαν τρόπους σχεδίασης κωδίκων δικτύου και επίσης μελέτησαν αρκετές ιδιότητές τους όπως η κατανεμημένη σχεδίασή τους.

Στα πλαίσια αυτής της εργασίας εξετάζουμε διάφορα θεωρητικά και πρακτικά ζητήματα της κωδικοποίησης δικτύου. Ξεκινάμε τη μελέτη μας με την εξέταση μερικών δικτύων από την άποψη του κέρδους που έχουμε με τη χρήση της κωδικοποίησης δικτύου. Σε μερικές περιπτώσεις, η κωδικοποίηση δικτύου μπορεί να διπλασιάσει την χωρητικότητα και εν γένει εξοικονομεί ενέργεια σε ένα ασύρματο δίκτυο λόγω των λιγότερων μεταδόσεων που απαιτούνται. Παρουσιάζουμε έναν απλό αλλά επαρκή συγκεντρωτικό αλγόριθμο για την κατασκευη κώδικα δικτύου, ο οποίος εφαρμόζεται σε ένα ακυκλικό δίκτυο πολυεκπομπής μιας πηγής. Στη συνέχεια μελετάμε το πρόβλημα του μεγέθους του αλφάβητου ενός κώδικα δικτύου. Η μνήμη καθώς και η υπολογιστική πολυπλοκότητα που απαιτούνται εξαρτώνται σε μεγάλο βαθμό από το μέγεθος του

αλφαβήτου του κώδικα που χρησιμοποιούμε. Επίσης, παρουσιάζουμε μια μέθοδο για την εύρεση του μεγέθους του αλφάβητου χρησιμοποιώντας μόνο τα εξερχόμενα κανάλια από την πηγή. Τέλος, παρουσιάζουμε μια μέθοδο για να μειώσουμε το μέγεθος του αλφάβητου για τα δίκτυα συνδυασμού.

Network Coding

Abstract:

Network coding is a new research area that is showing promise for interesting applications not only in information and coding theory, but in practical networking systems too. Network Coding generalizes network operation beyond traditional routing or store and forward approaches allowing for mathematical operations (coding) within networks. Ahlswede et al. studied the multicast in a network of lossless links and showed that achieving the multicast capacity requires in general the use of network coding. Further work by various authors showed how to design network codes and also demonstrated new properties of these codes such as distributed design etc.

This thesis considers a number of theoretical and practical issues from the network coding perspective. We begin our study with the examination of some networks and the gain we may have by using network coding techniques. In some cases network coding can double our capacity and saves energy in wireless network due to reduced transmissions. We present a simple but efficient algorithm for network coding, which is centralized and can be easily applied to a single source acyclic multicast network. Next we study the problem of finding the alphabet size for a network code. The size of the alphabet plays an important role because the memory requirements and the computational complexity depend on it. We present a method for determining the alphabet size by using only the outgoing channels from the source. Finally, we present a method to reduce the alphabet size for combination networks.

Thesis Supervisor: Apostolos Traganitis

Title: Professor

ΕΥΧΑΡΙΣΤΙΕΣ

Καταρχήν θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες προς τον κ. Απόστολο Τραγανίτη, σαν ακαδημαϊκό σύμβουλο στην αρχή και σαν επόπτη καθηγητή της μεταπτυχιακής μου εργασίας στη συνέχεια. Η βοήθεια και η υποστήριξη που μου παρείχε, όπως επίσης και ο χρόνος που αφιέρωσε για όλες τις συζητήσεις καθώς και κατά την διόρθωση του κειμένου ήταν σημαντικά για την ολοκλήρωση αυτής της εργασίας.

Στη συνέχεια θα ήθελα να ευχαριστήσω όλα τα παιδιά του εργαστηρίου Τηλ/νιών & Δικτύων του Ινστιτούτου Πληροφορικής και ειδικά αυτά που είμασταν μαζί στο «*Κλουβί*», τον Βαγγέλη Αγγελάκη, Στέφανο Παπαδάκη, Χαρίτων Μελισσάρη, Γιώργο Σταματάκη και Ρόη Φλουρή.

Στο σημείο αυτό θα ήθελα να πω ένα μεγάλο ευχαριστώ στους γονείς μου Δημήτρη και Αμαλία για την πίστη που μου έχουν δείξει σε κάθε προσπάθειά μου. Χωρίς αυτούς δεν θα βρισκόμουν ποτέ στο σημείο αυτό.

Ολοκληρώνοντας θα ήθελα να ευχαριστήσω την Μαρία, που είναι δίπλα μου όλα αυτά τα χρόνια, ανέχεται τις παραξενιές μου και με στηρίζει στα δύσκολα. Σε ευχαριστώ Μαρία.

Στους γονείς μου

Αμαλία, Δημήτρη

Contents

List of figures

Chapter 1 Introduction

1.1 Historical perspective

Like many fundamental concepts, Network coding is based on a simple but powerful idea, which was first stated in the seminal paper by R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow", (IEEE Transactions on Information Theory, IT-46, pp. 1204-1216, 2000). The core notion of network coding is to allow and encourage mixing of data at intermediate network nodes. Network coding is a generalization of traditional store and forward technique. In existing computer networks, each node functions as a switch in the sense that it either relays information from an input link to an output link, or it replicates information received from an input link and sends it to a certain set of output links. From the information-theoretic point of view, there is no reason to restrict the function of a node to that of a switch. Rather, a node can function as an encoder in the sense that it receives information from all the input links, encodes and sends information to all the output links. The main advantage of using network coding can be seen in multicast scenarios where, given a network with specific capacities on links, we have to compute the maximum multicast throughput possible for communication between a source node and a set of receivers.

The main idea behind network coding was to design networks capable of achieving the maximum flow bound on the information transmission rate in a multicast scenario. The major finding in [1] was that it is in general not optimal to consider information to be multicast in a network as a “fluid” which can simply be routed or

replicated at the intermediate nodes but as a mathematical entity that can be operated upon. It allows interior network nodes to perform arbitrary operations on information from different incoming links.

In the following we list some of applications of network coding.

P2P file distribution

Probably the most widely known application using network coding is Avalanche [2]. Generally, in a peer-to peer content distribution network, a server splits a large file into a number of blocks. Peer nodes try to retrieve the original file by downloading blocks from the server but also distributing downloaded blocks among them. Peers maintain connections to a limited number of neighboring peers (randomly selected among the set of peers) with which they exchange blocks. In Avalanche, the blocks sent out by the server are random linear combinations of all original blocks. Similarly, peers send out random linear combinations of all the blocks available to them (idea of Random Network Coding presented by Ho [3]). A node can either determine how many innovative blocks it can transmit to a neighbor by comparing its own and the neighbor's matrix of decoding coefficients, or it can simply transmit coded blocks until the neighbor receives the first non-innovative block. The node then stops transmitting to this neighbor until it receives further innovative blocks from other nodes. Coding coefficients are transmitted together with the blocks, but since blocks usually have a size of hundreds of kilobytes, this overhead is negligible.

Wireless Networks

Network coding can improve throughput when two wireless nodes communicate via a common base-station. In a wireless environment, network coding can be used to offer benefits in terms of battery lifetime because it saves energy in wireless network because of less transmissions needed. Network coding offer benefits to wireless bandwidth and delay too.

Security

In [4], the authors investigate the problem of designing secure network codes for wiretap networks, where certain links can be accessed by attackers. They assume

that it is known which links are tapped. The source combines the original data with random information and designs a network code in a way that only the authorize receivers are able to decode the original packets. Furthermore, the mutual information between the packets obtained by the eavesdroppers and the original packets is zero (security in the information theoretic sense). The fact that with network coding, nodes can only decode packets if they have received a sufficient number of linearly independent information vectors allows for a weaker form of security [5]. Such codes are more efficient, but an attacker who has $n-1$ out of n linear combinations, only has to guess the content of a single packet to be able to decode all n packets (hence the name "weak security"). Finally, network coding simplifies the protection against modified packets in a network [6]. At a normal network (and no additional protection), an intermediate attacker may make arbitrary modifications to a packet to achieve a certain reaction at the attacked destination. However, in the case of network coding, an attacker cannot control the outcome of the decoding process at the destination, without knowing all other coded packets the destination will receive. Given that packets are routed along many different paths, this makes controlled man-in-the-middle-attacks more difficult.

Integration with existing infrastructure

As communication networks evolving, a challenging task is to incorporate the emerging technologies such as network coding, into the existing network architecture. Ideally, we would like to be able to profit from the functionalities that network coding can offer, without incurring important changes in the existing equipment and software. A related open question is, how could we integrate network coding in current networking protocols. Making this possible is also an area of current research.

1.2 Some examples

Example 1.1 [1]

The first example highlighting the utility of network coding was given by Ahlswede et al. [1]. Figure below shows their famous example of a network called butterfly network for which coding in the interior of the network is necessary in order to achieve the maximum possible multicast transmission rate.

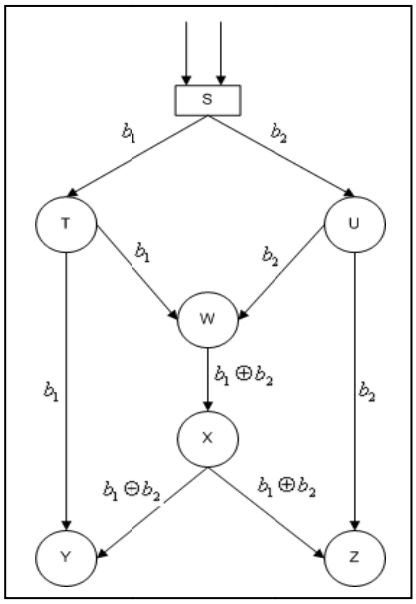


Fig. 1.1 Butterfly network

In this network we multicast two bits from the source node S to Y and Z. Node W derives from the received b_1 and b_2 the exclusive-OR bit $b_1 \oplus b_2$. The channel from W to X transmits $b_1 \oplus b_2$, which is then replicated at X for passing on to Y and Z. Then the node Y receives b_1 and $b_1 \oplus b_2$, from which the bit b_2 can be decoded. Similarly, the node Z decodes the bit b_1 from the received bits b_2 and $b_1 \oplus b_2$. In this way, all the nine channels in the network are used exactly once. The derivation of the exclusive-OR bit is a simple form of coding. If the same communication objective is to be achieved simply by bit replication at the intermediate nodes without coding, at least one channel in the network must be used twice, so that the total number of channel usage will be at least ten. Thus, coding offers the potential advantage of

minimizing both latency and energy consumption, and at the same time maximizing the bit rate.

Example 1.2 [7]

Figure 1.2 depicts two neighboring base stations labeled ST and S'T', of a communication network at a distance twice the wireless transmission range.

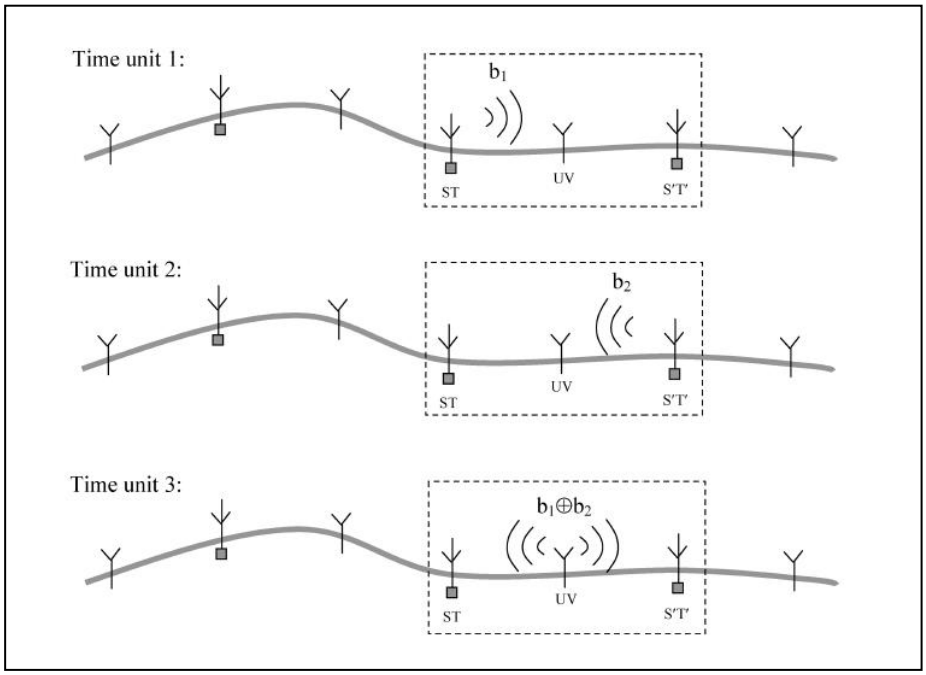


Fig. 1.2 Operation of the relay transceiver between two wireless base stations

Installed at the middle is a relay transceiver labeled by UV, which in unit time either receives or transmits one bit. Through UV and using network coding, the two base stations transmit one bit of data to each other in three unit times: In the first two unit times, the relay transceiver receives one bit from each side. In the third unit time it broadcasts the exclusive-OR bit to both base stations, which then can decode the bit from each other.

This model can also be applied to satellite communications, with the nodes ST and S'T' representing two ground stations communicating with each other through a satellite represented by the node UV as you can see in figure 1.3. By employing very simple coding at the satellite as prescribed, the downlink bandwidth can be reduced by 50%.

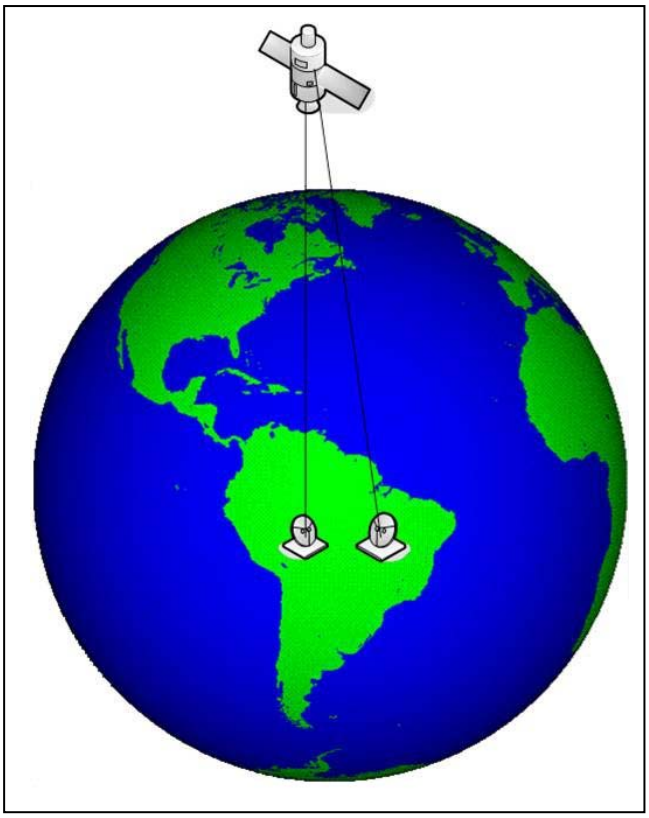


Fig. 1.3 A satellite with two earth stations.

1.3 Thesis outline

The remainder of this thesis is divided into four chapters (ch2-6).

In chapter two, we present background theory that is necessary for the better understanding of network coding theory. In the first part of chapter two we give some definitions from graph theory and after that we focus on some relevant theorems. In the second part of the chapter two we continue with the mathematical formulation of Network Coding Theory. In that chapter we limit our study to one source directed acyclic multicast networks and linear network codes only.

In chapter three, we examine the benefits from network coding in specific networks and some generalizations of them. We study some networks using transmissions in time slots and, after that we calculate the gain, applying network coding techniques.

In chapter four we present our algorithm for the construction of a network code for a multicast network. We restrict our study to acyclic directed graphs as models of

our network. The algorithm depends on the knowledge of the topology of the network.

Chapter five presents a new way to find the size of the alphabet of a network code using only outgoing channels from source. At the end of the chapter we show a method to reduce the size to two for some networks.

Finally Chapter 6 outlines our conclusions and presents ideas for future work.

Chapter 2 Background

In this chapter we present some background theory that is necessary for an introduction to the network coding theory.

In the first part of this chapter we give some definitions from graph theory and after that we focus on some interesting theorems.

In the second part of this chapter we continue with mathematical formulations for Network Coding Theory. For the material of this part we used “Network Coding Theory”, a tutorial from Yeung, Li, Cai and Zhang [7], as well as the monograph by Christina Fragouli “Network Coding Fundamentals” [8] which is under publication.

For a more detailed study, a good reference is the Tutorial of Yeung etc[7] which focuses on the information theoretic aspects of network coding. Another reference is the monograph by Christina Fragouli [8] which is an excellent book to start with. A rich source of references is the homepage of network coding [www.networkcoding.info], that has an exhaustive list of literature in the bibliography section.

In the present chapter we will limit our study to one source directed acyclic multicast networks and linear network codes only.

2.1 Graph Theory

In mathematics and computer science, graph theory is the study of graphs, which are mathematical structures used to model pair wise relations between objects from a certain collection. A "graph" in this context refers to a collection of vertices and a collection of edges that connect pairs of vertices. A **graph** may be **undirected**, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be **directed** from one vertex to another. We denote a graph as $G = (V, E)$ where V is the vertex set and E is the edge set. An edge $e \in E$ where $e = (x, y)$ iff $x, y \in V$. In an undirected graph, edges $e_1 = (x, y)$ and $e_2 = (y, x)$ are identical, but they are not in a directed graph. A directed graph or digraph G is an ordered pair $G = (V, E)$ where V is a set, whose elements are called vertices or nodes, and E is a set of ordered pairs of vertices, called directed edges. An edge $e = (x, y)$ is considered to be directed from x to y; y is called the head and x is called the tail of the arc; y is said to be a direct successor of x, and x is said to be a direct predecessor of y.

Two edges of a graph are called adjacent, if they share a common vertex. Similarly, two vertices are called adjacent if they share a common edge, in which case the common edge is said to join the two vertices. An edge and a vertex on that edge are called incident.

In graph theory, a **flow network** is an assignment of flow to the edges of a directed graph, where each edge has a **capacity**, such that the amount of flow along an edge does not exceed its capacity. A directed graph with edge capacities is called a **network**. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, except when it is a source, which has more outgoing flow, or sink, which has more incoming flow. A network can be used to model traffic in a road system, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

Suppose $G = (V, E)$ is a finite directed graph in which every edge $(u, v) \in E$ has a non-negative, real-valued capacity $c(u, v)$. If $(u, v) \notin E$ then we assume that $c(u, v) = 0$. We distinguish two vertices: **a source s and a sink t.** A flow network is a real function $f : V \times V \to \mathbb{R}$ with the following three properties for all nodes u and v:

1) **Capacity constraints**: $f(u, v) \le c(u, v) \quad \forall (u, v) \in E$ .The flow along an edge cannot exceed its capacity.
2) **Skew symmetry**: $f(u, v) = -f(v, u)$ .The net flow from u to v must be the opposite of the net flow from v to u.
3) **Flow conservation**: $\sum_{u \in V} f(u, w) = 0$ unless u = s or u = t. The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow.

If a value of a flow is as large as possible then the flow is called **a maximum flow**.

A **cut** of a graph $G = (V, E)$ is a partition of the vertices V into two sets *S* and *T*. Any edge $(u, v) \in E$ with $u \in S$ and $v \in T$ is said to be crossing the cut and is a **cut edge**. The **size** of a cut is the total number of edges crossing the cut. In **weighted graphs**, the size of the cut is defined to be sum of weights of the edges crossing the cut. In network flow, the size of a cut is defined to be the sum of weights of the edges crossing the cut from the source side to the sink side (but not the ones that go the other way). A cut is **minimal** if the size of the cut is not larger than the size of any other cut.

For unit capacity edges, the **value** of a cut equals the number of edges in the cut, and it is sometimes referred to as the size of the cut. There exist a **unique minimum cut** value and possibly several minimum cuts. One can think minimum cut as a **bottleneck** for information transmission between source S and receiver T.

The maximum flow minimum cut theorem states that the maximum information rate we can send from S to T is equal to the minimum cut value.

Maximum flow minimum cut theorem:

Consider a graph $G = (V, E)$ with unit capacity edges, a source vertex s and a receiver vertex t. If the minimum cut between s and t equals h, then the information can be send from s to t at a maximum rate equal to h. Equivalently there exist exactly h edge-disjoint paths between S and R (Menger's Theorem).

For a detailed study there are several books, a good one to start is "Graph theory with Applications" by Bondy and Murty [9]. For an advanced reader Diestel [10] is recommended, which is available online. You can also find useful the books by Bollobas like[11].

2.2 Network Coding: A mathematical formulation

A **Communication network** is a finite directed graph, where multiple edges from one node to another are allowed. The **source node** is a node without any incoming edges depicted by a **square**, any other node is called **non-source** node and is depicted by a **circle**. An edge is also called **channel** and represents **a noiseless communication link** for the transmission of a data unit per unit time. A communication network is said to be **acyclic** if it contains no directed cycles. On the other hand if a communication network contains at least one directed cycle is said to be **cyclic**. There exists at least one source node on every acyclic network. In our study we suppose that there is a **unique source node** denoted by **S**. Fig 2.1 shows an acyclic and a cyclic communication network respectively.

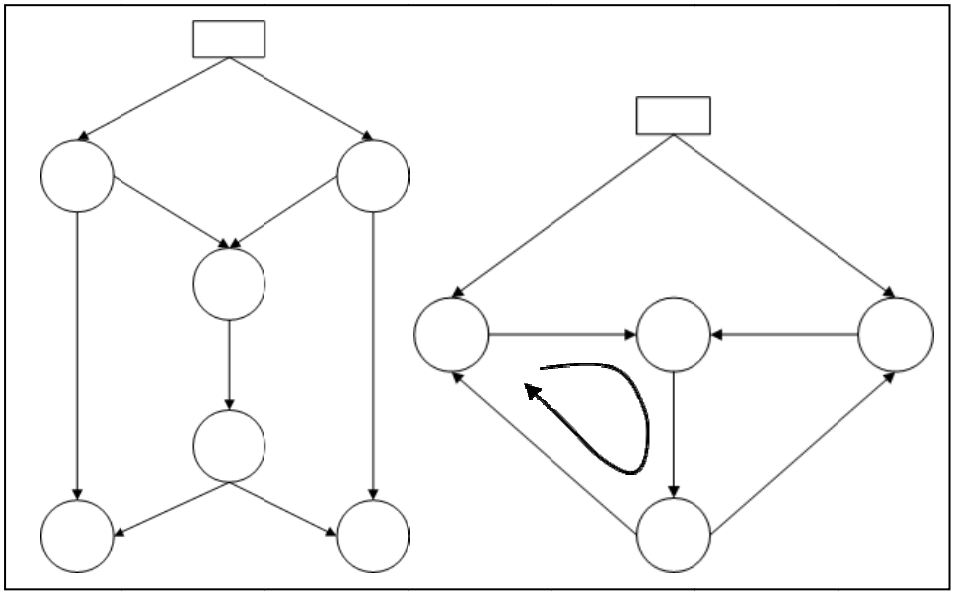


Fig. 2.1 Acyclic and cyclic communication networks

The **capacity** of a direct transmission from a node to another is determined by the **multiplicity of the channels** between them.

Network **multicast** refers to the simultaneous transmission of the same information from a source node to multiple receivers (a subset of the nodes in the graph) in the network. In a multicast network there is only one source node and multiple target nodes, and all messages are available at the source, while they are demanded by all the target nodes. Multicast information flow problems have been studied extensively.

A **network code** can be formulated in various ways. In general, a source node generates a set of messages to be multicast to certain destinations. When the communication network is **acyclic**, operation at all nodes can be synchronized in such a way that the message is individually encoded and propagated from the upstream nodes to the downstream nodes. That is the processing of each message is independent of the subsequent messages. In this way, the network coding problem is independent of the propagation delay over the channels as well as processing delay at nodes. When the network is **cyclic,** the propagation and encoding of sequential messages could convolve together. In this case the amount of delay

becomes part of the considerations in network coding; it's a different problem than the one for an acyclic network. As we mentioned in the start of the chapter we will study network coding for the case of a single message over an acyclic network.

For every node T, let $\partial^{-}(T)$ denote the set of **incoming channels** to T and $\partial^{+}(T)$ the set of **outgoing channels** from T. Similarly, let $\partial^{-}(S)$ denote a set of **imaginary channels**, which terminate at the source node S but are without originating nodes. The number of these imaginary channels is context dependent and always denoted by **ω**. Figure 2.2 below illustrates an acyclic network (the well known butterfly network) with ω = 2 imaginary channels terminating at the source node S.

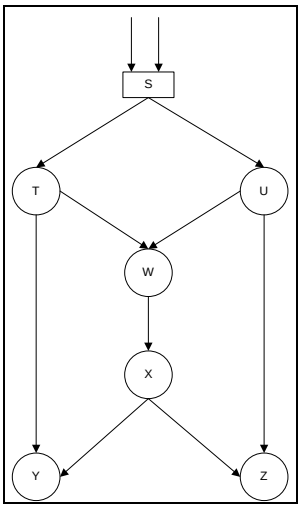


Fig. 2.2 The butterfly network with ω=2

An **information symbol** is represented by an element of a certain base of a Galois Field ***F***. For example if *F* is a GF(2) then the information symbol is a bit. A **message** consists of **ω data units** and is therefore represented by an ω-dimensional row vector $x \in F^{\omega}$. The source node S generates a message x and sends it out by transmitting a symbol over the outgoing channels. Message propagation through the network is achieved by the transmission of a symbol over every channel e in the network. A symbol that is transmitted over a channel e is denoted by $\tilde{f}_e(x) \in F$. Obviously the symbol is a function of the message x. A non-source node may not receive enough information to identify the value of the whole message x. Its **encoding function** simply maps the incoming symbols from all the incoming channels

to a symbol for each outgoing channel. A **network code** is specified by such an encoding mechanism for every channel.

In the single-source, acyclic case, there are two equivalent ways to define the network code, **local encoding mapping** and **global encoding mapping**. We give the definitions below:

Local encoding mapping:

Let F be a finite field and ω a positive integer. An ω -dimensional F-valued **network code** on an acyclic communication network consists of a **local encoding mapping** $\tilde{k}_e : F^{|\partial^-(T)|} \to F$ for each node T in the network and each channel $e \in \partial^+(T)$.

The acyclic topology of the network provides an upstream-to downstream procedure for the local encoding mappings to build up the values $\tilde{f}_e(x)$ transmitted over all channels e. The above definition of a network code does not explicitly give the values of $\tilde{f}_e(x)$. Therefore, we give an **equivalent definition** below, which describes a network code by both the local encoding mappings as well as the recursively derived values $\tilde{f}_e(x)$.

Global encoding mapping:

Let F be a finite field and ω a positive integer. An ω -dimensional F-valued network code on an acyclic communication network consists of a **local encoding mapping** $\tilde{k}_e : F^{|\partial^-(T)|} \to F$ and a **global encoding mapping** $\tilde{f}_e : F^{\omega} \to F$ for each channel e in the network such that:

1)) For every node T and every channel $e \in \partial^+(T)$, $\tilde{f}_e(x)$ is uniquely determined by $(\tilde{f}_d(x), d \in \partial^-(T))$, and $\tilde{k}_e$ is the mapping via $(\tilde{f}_d(x), d \in \partial^-(T)) \mapsto \tilde{f}_e(x)$

2) For the ω imaginary channels e, the mappings $\tilde{f}_e$ are the projections from the space F^{ω} to the ω different coordinates, respectively.

A global encoding mapping $\tilde{f}_e$ is **linear**, iff $\tilde{f}_e(x) = x \cdot f_e$.

The following examples illustrate the definitions above:

<u>Example 2.1:</u>

Consider the butterfly network of figure 2.3, where $x = (b_1, b_2) \in F^2 \equiv GF(2)^2$.

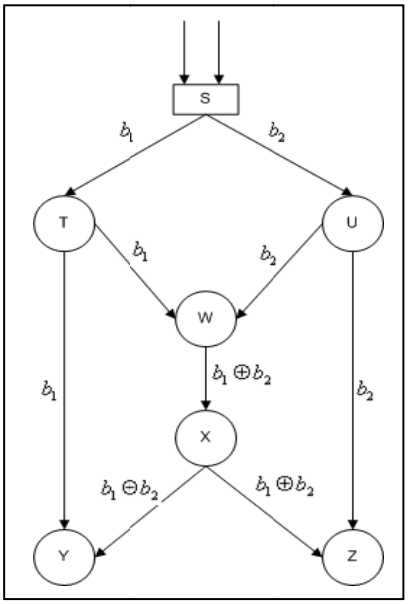


Fig. 2.3 Butterfly network

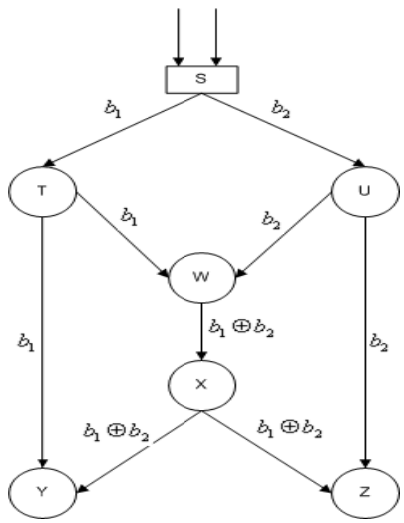
A two dimensional binary network code is shown, with the following global encoding mappings:

$$\tilde{f}_e(x) = \begin{cases} b_1, \text{ for channels } e = OS, ST, TW, TY \\ b_2, \text{ for channels } e = OS', SU, UW, UZ \\ b_1 \oplus b_2, \text{for channels } e = WX, XY, XZ \end{cases}$$

Where OS,OS' denote the two imaginary channels.

The corresponding local encoding mappings are: { $\tilde{k}_e : F^{|\partial^-(T)|} \to F$ }

$$\tilde{k}_{ST}(b_1, b_2) = b_1$$
$$\tilde{k}_{SU}(b_1, b_2) = b_2$$
$$\tilde{k}_{TY}(b_1) = b_1$$
$$\tilde{k}_{TW}(b_1) = b_1$$
$$\tilde{k}_{UW}(b_2) = b_2$$
$$\tilde{k}_{UZ}(b_2) = b_2$$
$$\tilde{k}_{WX}(b_1, b_2) = b_1 \oplus b_2$$
$$\tilde{k}_{XY}(b_1 \oplus b_2) = b_1 \oplus b_2$$
$$\tilde{k}_{XZ}(b_1 \oplus b_2) = b_1 \oplus b_2$$



Below we formulate a linear network code as a network code where all the local and global encoding mappings are **linear** (that is $\tilde{k}_e(y) = y \cdot k_e$ and $\tilde{f}_e(x) = x \cdot f_e$ respectively). A linear network code was originally called a **linear code multicast** (LCM) [12]. Physical implementation of message propagation with network coding results in **transmission delay** over the channels as well as **processing delay** at the nodes. Nowadays node processing is the dominant factor of the total delay in message delivery through the network. It is therefore desirable that the coding mechanism inside a network code be implemented by **simple and fast circuitry**. For this reason, network codes that involve only linear mappings are of particular interest.

<u>Local encoding mapping for *linear* network code</u>

Let F be a finite field and ω a positive integer. An ω-dimensional F-valued **linear network code** on an acyclic communication network consists of a scalar $k_{d,e}$, called the **local encoding kernel**, for every adjacent pair (d,e). Meanwhile, the **local encoding kernel** at the node T means the $|\partial^-(T)| \times |\partial^+(T)|$ matrix $K_T = [k_{d,e}]_{e \in \partial^+(T)}^{d \in \partial^-(T)}$

<u>Global encoding mapping for *linear* network code</u>

Let F be a finite field and ω a positive integer. An ω -dimensional F-valued **linear network code** on an acyclic communication network consists of a scalar $k_{d,e}$ for every adjacent pair (d,e) in the network as well as an ω -dimensional column vector f_e for every channel e such that:

1) $f_e = \sum_{d \in \partial^-(T)} k_{d,e} f_d$, where $e \in \partial^+(T)$.

2) The vectors f_e for the ω imaginary channels $e \in \partial^-(S)$ form the natural basis of the vector space F^{ω} .

The vector f_e is called the **global encoding kernel** for the channel e.

Assume that the source generates a message x in the form of an ω-dimensional row vector. A node T receives the symbols $\tilde{f}_d(x) = x \cdot f_d$, $d \in \partial^-(T)$, from which it calculates the symbol $\tilde{f}_e(x) = x \cdot f_e$ to send onto each channel $e \in \partial^+(T)$ via the linear formula

$$\tilde{f}_e(x) = x \cdot f_e = x \cdot \sum_{d \in \partial^-(T)} k_{d,e} \cdot f_d = \sum_{d \in \partial^-(T)} k_{d,e}(x \cdot f_d) = \sum_{d \in \partial^-(T)} k_{d,e} \cdot \tilde{f}_d(x)$$

Given the local encoding kernels for all the channels in an acyclic network, the global encoding kernels can be **calculated recursively** in any upstream-to-downstream order.

The above expression of $\tilde{f}_e(x)$ can be considered in a more generic form:

We have a node T with $|\partial^-(T)| = n$ and $|\partial^+(T)| = k$. That is, we have:

$\partial^-(T) = \{e_{i,m}, 1 \le m \le n\}$ and $\partial^+(T) = \{e_{o,l}, 1 \le l \le k\}$

Then we have:

$$f_{e_{o,l}} = \begin{pmatrix} f_{e_{i,1}}^T \\ f_{e_{i,2}}^T \\ \vdots \\ f_{e_{i,n}}^T \end{pmatrix}^T (K_T \cdot u_l) \quad \forall \quad 1 \le l \le k$$ where u_l is the l-th unitary vector.

That is $(K_T.u_l)$ is the l-th row of matrix K_T.

The previous equation may help us to express a network code in one equation.

Note: A partial analogy can be drawn between the global encoding vectors f_e for the channels in a linear network code and the columns of a generator matrix of a linear error-correcting code [7].

<u>Example 2.2:</u>

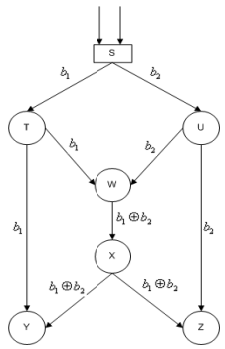
This example illustrates the construction of a linear network code for the butterfly network of the previous example. Assume the alphabetical order among the channels OS,OS',ST, . . . ,XZ. Then, the local encoding kernels at the nodes are the following matrices: $K_S = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $K_T = K_U = K_X = \begin{pmatrix} 1 & 1 \end{pmatrix}$ and $K_W = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

The corresponding global encoding vectors are:

$$f_e = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{ for channels } e = OS, ST, TW, TY$$

$$f_e = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \text{ for channels } e = OS', SU, UW, UZ$$

$$f_e = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \text{ for channels } e = WX, XY, XZ$$



The global encoding vectors and the local encoding kernels are summarized in the figure below. These vectors and kernels remain the same for any choice of the base field.

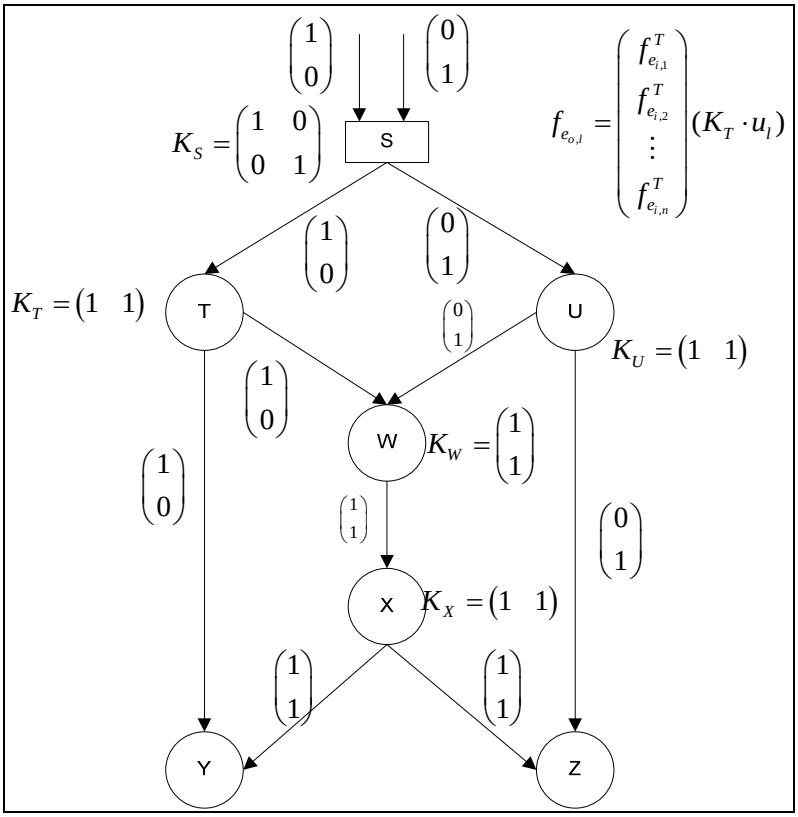


Fig. 2.4 Global & local encoding mappings for butterfly network

The figure below highlights the operation of the local encoding kernels:

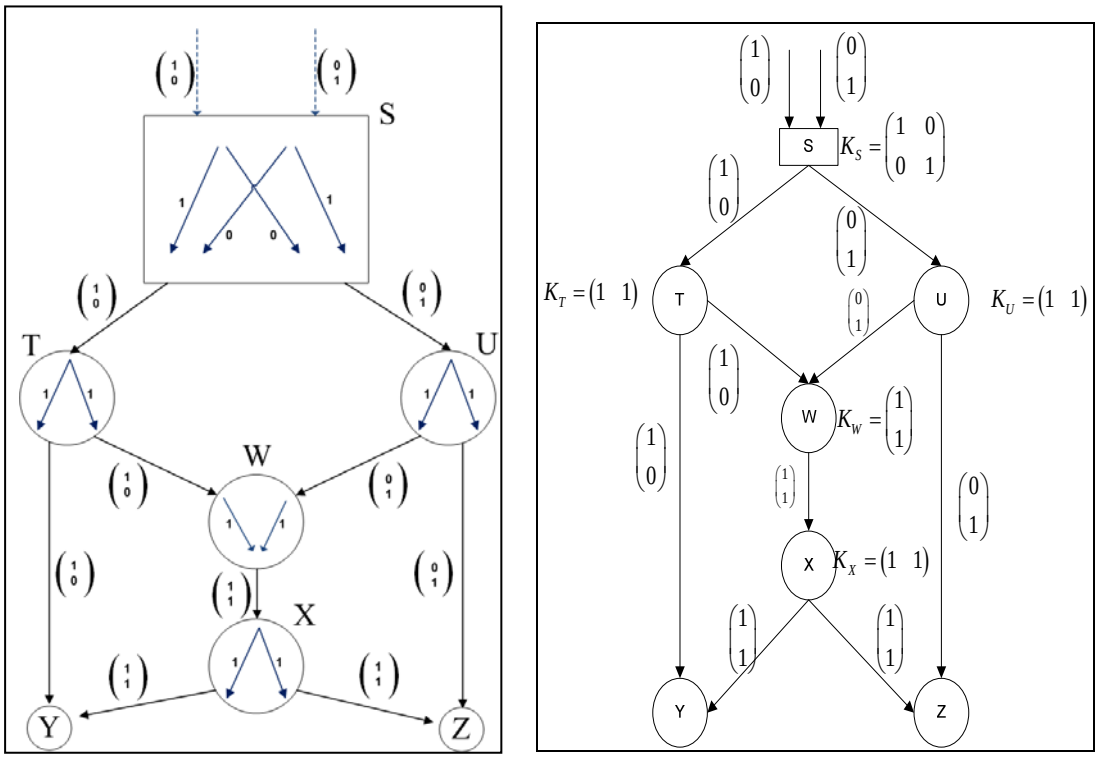


Fig. 2.5 An inside view of local encoding mappings

Example 2.3:

In this example we will see a **general** two dimensional network code for the butterfly network. The local encoding kernels can be expressed as:

$K_S = \begin{pmatrix} n & q \\ p & r \end{pmatrix}, K_T = \begin{pmatrix} s & t \end{pmatrix}, K_U = \begin{pmatrix} u & v \end{pmatrix}, K_W = \begin{pmatrix} w \\ x \end{pmatrix}$ and $K_X = \begin{pmatrix} y & z \end{pmatrix}$ where p,q,r,…,z are variables. Starting with $f_{OS} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $f_{OS'} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ we can compute the global encoding vectors recursively as follows:

$$f_{ST} = \begin{pmatrix} f_{OS}^T \\ f_{OS'}^T \end{pmatrix} (K_S \cdot u_1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} n & q \\ p & r \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} n \\ p \end{pmatrix}$$

$$f_{SU} = \begin{pmatrix} f_{OS}^T \\ f_{OS'}^T \end{pmatrix} (K_S \cdot u_2) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} n & q \\ p & r \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} q \\ r \end{pmatrix}$$

$$f_{TW} = (f_{ST}^T)^T (K_T \cdot u_1) = \begin{pmatrix} n & p \end{pmatrix}^T \begin{pmatrix} s & t \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = s \begin{pmatrix} n \\ p \end{pmatrix} = \begin{pmatrix} ns \\ ps \end{pmatrix}$$

$$f_{TY} = (f_{ST}^T)^T (K_T \cdot u_1) = \begin{pmatrix} n & p \end{pmatrix}^T \begin{pmatrix} s & t \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = t \begin{pmatrix} n \\ p \end{pmatrix} = \begin{pmatrix} nt \\ pt \end{pmatrix}$$

$$f_{UW} = (f_{SU}^T)^T (K_U \cdot u_1) = \begin{pmatrix} q \\ r \end{pmatrix} \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} qu \\ ru \end{pmatrix}$$

$$f_{UZ} = (f_{SU}^T)^T (K_U \cdot u_2) = \begin{pmatrix} q \\ r \end{pmatrix} \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} qv \\ rv \end{pmatrix}$$

$$f_{WX} = \begin{pmatrix} f_{TW}^T \\ f_{UW}^T \end{pmatrix}^T K_W 1 = \begin{pmatrix} ns & qu \\ ps & ru \end{pmatrix} \begin{pmatrix} w \\ x \end{pmatrix} = \begin{pmatrix} nsw + qux \\ psw + rux \end{pmatrix}$$

$$f_{XY} = f_{WX} K_X u_1 = \begin{pmatrix} nsw + qux \\ psw + rux \end{pmatrix} \begin{pmatrix} y & z \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = y \begin{pmatrix} nsw + qux \\ psw + rux \end{pmatrix} = \begin{pmatrix} nswy + quxy \\ pswy + ruxy \end{pmatrix}$$

$$f_{XZ} = f_{WX} K_X u_2 = \begin{pmatrix} nsw + qux \\ psw + rux \end{pmatrix} \begin{pmatrix} y & z \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = z \begin{pmatrix} nsw + qux \\ psw + rux \end{pmatrix} = \begin{pmatrix} nswz + quxz \\ pswz + ruxz \end{pmatrix}$$

The final solution is depicted in the figure below:

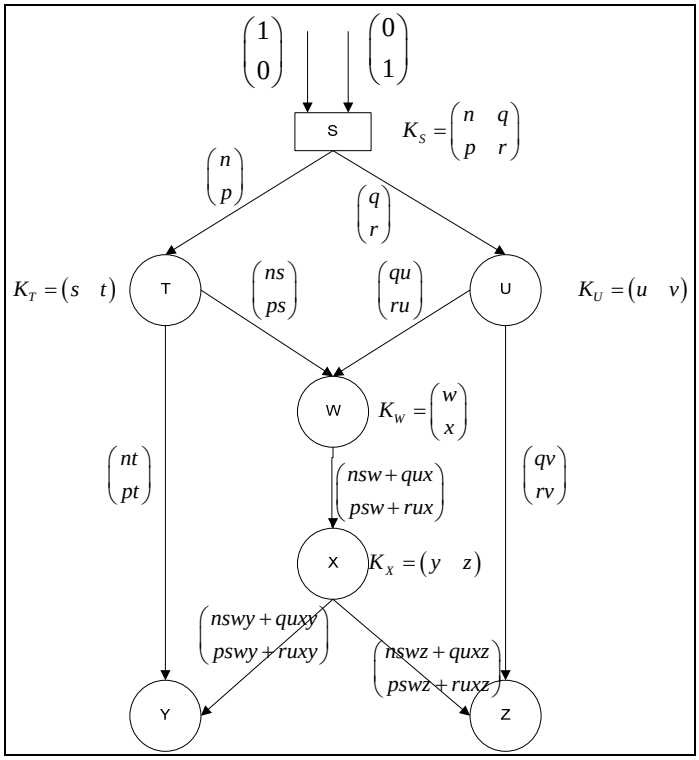


Fig. 2.6 Local/global encoding kernels of a general two-dimensional linear network code

Desirable properties of a Linear Network Code

Data flow with any conceivable coding schemes at an intermediate node abides with the **law of information conservation**: the content of information sent out from any group of non-source nodes must be derived from the accumulated information received by the group from outside. In particular, the content of any information coming out of a non-source node must be derived from the accumulated information received by that node.

As we saw in the first part of this chapter we denote the maximum flow from source S to receiver T as **maxflow(T)**. We know from the **Max-flow Min-Cut theorem** that the information rate received by T cannot exceed maxflow(T). Whether this upper bound is achievable depends on the network topology, the dimension ω, and the coding scheme. There are three special classes of linear network codes, namely linear multicast, linear broadcast and linear dispersion. We will study only linear multicast, but there is a complete study of all linear codes at the tutorial of network coding [7].

Linear Multicast

Let the vectors f_e denote the global encoding kernels in an ω-dimensional F-valued linear network code on an acyclic network. Write $V_T = \langle \{ f_e : e \in \partial^-(T) \} \rangle$,where $\langle \bullet \rangle$ denotes the linear span of a set of vectors. Then, the linear network code qualifies as a **linear multicast** if the following statement holds true: $\dim(V_T) = \omega$ for every non-source node T with maxflow(T) ≥ω.

When the source node S transmits a message of ω data units into the network, a receiving node T obtains sufficient information to decode the message if and only if $\dim(V_T) = \omega$, of which a necessary prerequisite is that maxflow(T)≥ω. Thus, an ω -dimensional linear multicast is useful in multicasting ω data units of information to all those non-source nodes T that meet this prerequisite.

<u>Example 2.4:</u>

The general linear network code in example 2.3 above meets the criterion of a linear multicast for the cases where:

1) f_{TW} and f_{UW} are linearly independent

2) f_{TY} and f_{XY} are linearly independent

3) f_{UZ} and f_{XZ} are linearly independent

Equivalently, the criterion is met if s, t, u,v,y,z, nr − pq, npsw + nrux − pnsw − pqux, and rnsw + rqux − qpsw – qrux are all nonzero. The example 2.2 is a special case with n = r = s = t = u = v = w = x = y = z = 1 and p = q = 0.

We will continue our study for the construction of a network code.

The main theorem in network coding [1] :

Consider a directed acyclic graph $G = (V, E)$ with unit capacity edges, h unit rate sources (imaginary channels) located on the same vertex of the graph and N receivers. Assume that the value of the **min-cut to each receiver** is h. Then there exists a multicast transmission scheme, over a large enough finite field $GF(q)$, in which intermediate network nodes linearly combine their incoming information symbols over $GF(q)$, that delivers the information from the sources simultaneously to each receiver at a rate equal to h.

From the min-cut max-flow theorem that we saw in the first part of this chapter, we know that there exist h edge-disjoint paths between the source and each of the receivers. Thus, if any of the receivers, say, T_j , is using the network by itself, the information from h imaginary channels can be routed to T_j through a set of h edge disjoint paths. When multiple receivers are using the network simultaneously, their sets of paths may **overlap**. The receivers have to **share the network resources**, (e.g., share the overlapping edge capacity or share the access to the edge in time), which leads to **reduced rates**. However, the main network coding theorem tells us that, if we allow intermediate network nodes to not only forward but also **combine** their incoming information flows, then each of the receivers T_j will be getting the information at the same rate as if it had **sole access** to network resources. The theorem additionally claims that it is sufficient for intermediate nodes to perform **linear** operations, additions and multiplications over a finite field $GF(q)$. As we saw before, such transmission schemes are called linear network coding. Thus the theorem establishes the **existence** of linear network codes over some large enough finite field $GF(q)$. To reduce **complexity,** the field $GF(q)$ should be chosen as **small** as possible. In this chapter we don't worry about the choice of order of Galois Field used for network code alphabet, we just use for the size of our alphabet the number of the receivers in our network [13].

Network Code construction

The problem of network code design is to select what linear operations each node of the network performs. A simple algorithm (**random network coding**) is to have each node in the network select **uniformly at random** the coefficients over the field $GF(2^s)$, in a completely independent and decentralized manner [14]. With random network coding there is a certain probability of selecting linearly **dependent** combinations [14]. This probability is related to the field size 2^s. Simulation results indicate that even for small field sizes (for example, *s* = 8) this probability becomes **very small** [15].

Random network coding is used by Avalanche [2]. Avalanche is the name of a proposed peer-to-peer network created by Microsoft, which claims to offer **improved scalability and bandwidth efficiency** compared to existing P2P systems. Avalanche splits the file to be distributed into small blocks. However, instead of the peers simply transmitting the blocks, they transmit random linear combinations of the blocks along with the random coefficients of this linear combination. This technique removes the need for each peer to have complex knowledge of block distribution across the network.

Alternatively, we can use **deterministic algorithms** to design network codes. The polynomial-time algorithm for multicasting in [13], sequentially examines each node of the network, and decides what linear combinations each node performs. Since each node uses fixed linear coefficients, the packets only need to carry the information vector.

Linear Information flow algorithm (LIF) [13]

The LIF is a **greedy algorithm** based on the observation that the choice of coding vectors should preserve the multicast property of the network. Every receiver keeps an hxh matrix T_i, where h is the maximum flow for each receiver constructed by the following algorithm. Starting the algorithm every matrix is the identity. The algorithm sequentially visits the **coding points** (edge e is a **coding point** or **critical edge**, if there exist two or more paths that share e but contain distinct edges before

e) in a topological order and assigns coding vectors to them. The assignment is made in a way that all matrices T_i have **rank** (the number of linearly independed columns) h every time. The algorithm terminates when all critical edges have been processed. After that we have to invert the matrices and multiply them with the incoming information for every receiver.

The following example demonstrates the LIF algorithm:

Consider the classical butterfly network in figure 2.2 ,

In the beginning we have $T_1 = T_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. In this network we have one critical edge (or coding point) which is WX. We have to decide how to linearly combine the vectors $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

If we combine them to be $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, then the matrices become: $T_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $T_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ respectively. Now we must check the rank of the matrices. It is easy to see that $rank(T_1) = rank(T_2) = 2$,that is they are full rank. This terminates the algorithm.

The inverted matrices are:

$T_1^{-1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $T_2^{-1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$.

As we saw from example 2.1 node Y receives b_1 and $b_1 + b_2$. Node Z receives $b_1 + b_2$ and b_2.

The receivers have to decode the received information. This can be done as follows:

Receiver Y:

$$T_1^{-1} \cdot \begin{pmatrix} b_1 \\ b_1 + b_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_1 + b_1 + b_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Receiver Z:

$$T_2^{-1} \cdot \begin{pmatrix} b_1 + b_2 \\ b_2 \end{pmatrix} = \begin{pmatrix} b_1 + b_2 + b_2 \\ b_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Real-world applications tend to use random network coding, because it does not use the network topology (is decentralized) and it is easy to be implemented. But the randomized coding approach may use a larger alphabet than necessary, and there is a possibility that the random choice of coefficients will not meet the multicast criterion. In the case where the choice of coefficients fails to meet the multicast criterion, we have to make another choice and retransmit our information. Of course, as the size of the alphabet increases the probability of failures decreases.

2.3 Summary

In this chapter we gave the minimum amount of theory needed for the basic understanding of the subsequent chapters of the thesis. The interested reader may consult the books that we recommended at the introduction of this chapter and visit the network coding homepage.

Chapter 3 Benefits from Network Coding

3.1 Introduction

In this chapter we will determine the benefits from the application of network coding in specific networks and some generalizations of them. We start our study with two nodes connected to the same access point and continue with more complex topologies. First we will consider networks transmitting in time slots (TDMA) and then we will calculate the gain arising from the application of network coding techniques instead of traditional methods of routing.

3.2 The case of n nodes connected to the same access point

We begin our study with n=2, then we will see the example with n=3 nodes and finally we will generalize our findings to n nodes connected to the same access point.

Each node contains one bit of information (or one information symbol) that wants to transmit to the other nodes of the network through the access point.

n=2

In the figure below we can see our ”network”

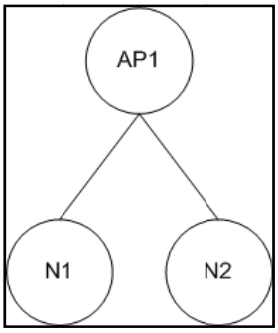


Fig. 3.1 Two nodes connected to an access point

Node N_1 contains info bit a, that wants to transmit to node N_2 and node N_2 has info bit b that wants to transmit it to N_1.

In the first two time slots node N_1 transmits bit a to AP_1 and node N_2 bit b to AP_1 respectively, as it shown in Fig 3.2.

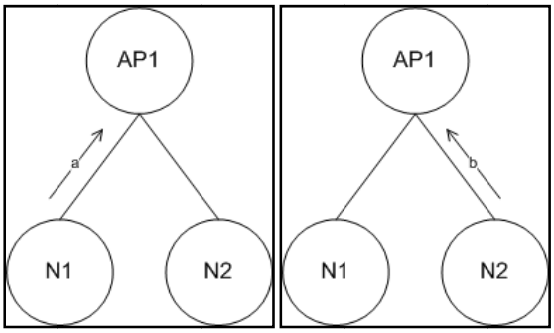


Fig. 3.2 Transmissions of nodes

Now AP_1 has bits a and b and transmits to both N_1 and N_2 with multicast info bit $a \oplus b$

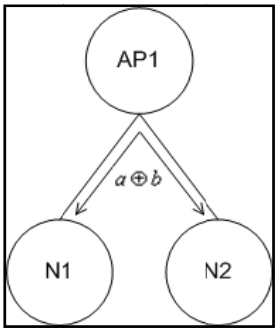


Fig. 3.3 Access points transmits xor info bit

Node N_1 already has bit a, now receives bit $a \oplus b$, after that can compute b because b= $a \oplus (a \oplus b)$. In the same way N_2 obtains bit a, a=$b \oplus (a \oplus b)$. We see that with one multicast transmission from AP_1 to nodes N_1 and N_2 we transmitted the needed information for the two nodes. Our gain is one transmission less (50% gain to downlink bandwidth), in the classical way we would use two transmissions from AP_1 to N_1 (bit b) and from AP_1 to N_2 (bit a) instead of one (bit $a \oplus b$).

In this case network coding looks promising and let's see what happens in cases with more than two nodes connected to the same access point.

Let's examine the case with three nodes.

The structure is the following

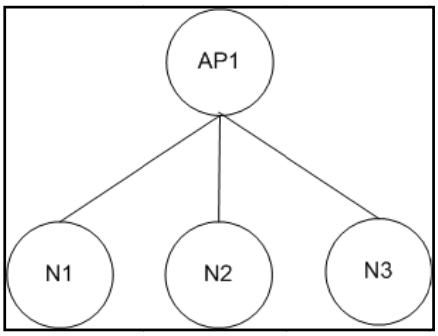


Fig. 3.4 Three nodes connected to access point

In the first three time slots each node uploads its information bit to the access point a, b and c respectively, after that AP_1 encodes bits a, b to $a \oplus b$ and sends it to the nodes. Now nodes N_1 and N_2 have information bits a, b after decoding, but node N_3 has not enough information to decode bits a and b, so stores bit $a \oplus b$ for future use. Access point AP_1 encodes bits b, c to $b \oplus c$ and transmits it. After that transmission, nodes N_1 and N_2 have information bits a, b and c. Let's see what happens to node N_3: after receiving $b \oplus c$ it calculates bit c and using the stored $a \oplus b$,it calculates a and so it has all the information bits now. The above are summarized in the images below.

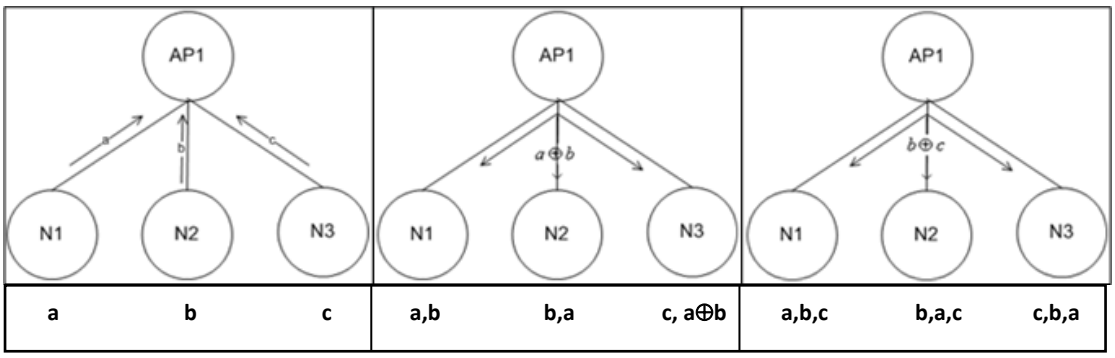


Fig. 3.5 Transmissions for the network of three nodes to an access point

In this case our gain is one transmission less (33% gain to downlink bandwidth). AP_1 transmits two times instead of three.

As the number of nodes increases the gain decreases. We can see that network coding in this case is useful for small number of nodes (two or three). But if the above structure is combined with others to form more complex networks, we will see next that network coding is beneficial.

In the next example we will study a class of networks called tandem networks.

3.3 Tandem networks

Imagine a tandem network with n nodes and n-1 access points.

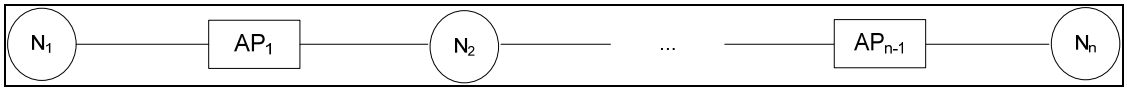


Fig. 3.6 A tandem network

A tandem network with two nodes connected with an access point, is the same network of the previous example with n=2.

In the case of n=3 it is easy to see the transmissions that will take place. Nodes N_1,N_2 and N_3 have bits a, b and c respectively. We have two access points AP_1 and AP_2. In the next figure we summarize the actions and the time slots in which they happen. We assume that some transmissions can be occur simultaneously because there is enough separation between the receiving nodes.

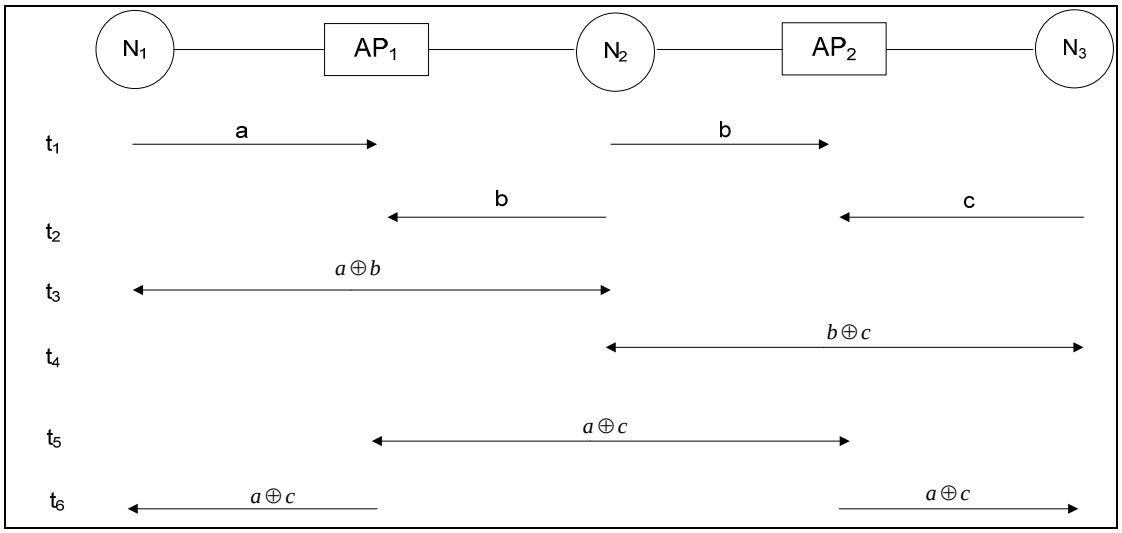


Fig. 3.7 Actions for a tandem network with three nodes and two access points

	N_1	AP_1	N_2	AP_2	N_3
	a		b		c
t_1	a	a	b	b	c
t_2	a	a,b	b	b,c	c
t_3	a,a+b	a,b	b,a+b	b,c	c,c+b
t_4	a,a+b	a,b	b,a+b,b+c	b,c	c,c+b
t_5	a,a+b	a,b,a+c	b,a+b,b+c	b,c,a+c	c,c+b
t_6	a,a+b,a+c	a,b,a+c	b,a+b,b+c	b,c,a+c	c,c+b,a+c

From the previous figure we can see that in time slots t_3, t_4 and t_5 we have one transmission instead of two for each time slot. Without network coding, we need three more time slots to complete our task. So in the case of a tandem network with three nodes and two access points, we avoid three transmissions.

3.4 A case with two access points and two nodes connected to each one

In the figure below we can see the structure. We have four nodes and two access points that are connected. Each access point has two nodes connected to it. Nodes N_1 and N_2 are connected to access point AP_1 and nodes N_3, N_4 to AP_2 respectively.

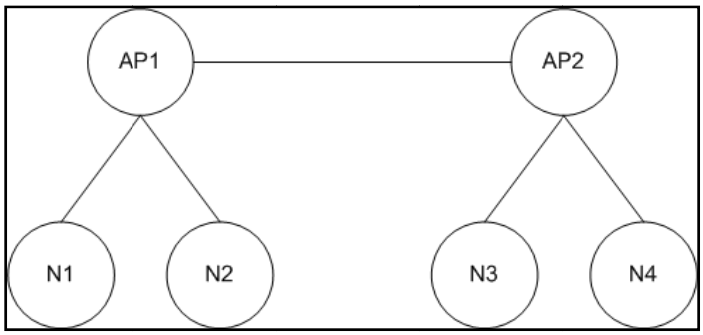


Fig. 3.8 A case with two access points and two nodes connected to each one

In the next table we summarize the actions that take place in our scenario and the bits that each node and access point have in a given time slot

Time	**Action**	**AP_1**	**AP_2**	**N_1**	**N_2**	**N_3**	**N_4**
t_0				a	b	c	d
t_1	$N_1 \rightarrow AP_1$ a $N_3 \rightarrow AP_2$ c	a	c	a	b	c	d
t_2	$N_2 \rightarrow AP_1$ b $N_4 \rightarrow AP_2$ d	a,b	c,d	a	b	c	d
t_3	$AP_1 \rightarrow N_1,N_2$ a+b $AP_2 \rightarrow N_3,N_4$ b+d	a,b	c,d	a,b	a,b	c,d	c,d
t_4	$AP_1 \rightarrow AP_2$ a	a,b	a,c,d	a,b	a,b	c,d	c,d
t_5	$AP_2 \rightarrow AP_1$ c $AP_2 \rightarrow N_3,N_4$ a	a,b,c	a,c,d	a,b	a,b	a,c,d	a,c,d
t_6	$AP_1 \rightarrow AP_2$ b $AP_1 \rightarrow N_1,N_2$ c	a,b,c	a,b,c,d	a,b,c	a,b,c	a,c,d	a,c,d
t_7	$AP_2 \rightarrow AP_1$ d $AP_2 \rightarrow N_3,N_4$ b	a,b,c,d	a,b,c,d	a,b,c	a,b,c	a,b,c,d	a,b,c,d
t_8	$AP_1 \rightarrow N_1,N_2$ d	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d

Discussion:

Nodes N_1, N_2, N_3 and N_4 have information bits a, b, c and d respectively. We want each node in the network to have all information bits. In the first time slot node N_1 sends to access point AP_1 info-bit a, concurrently node N_3 sends to AP_2 info-bit c. In the next time slot nodes

N_2 and N_4 send to AP_1 and AP_2 info-bits b and d respectively. Now, access points AP_1, AP_2 have collected info-bits a, b and c, d respectively. In the next time slot AP_1 transmits to N_1, N_2 info-bit $a \oplus b$ and concurrently AP_2 to N_3, N_4 info-bit $b \oplus d$. Node N_1 decodes b from $a \oplus b$ and N_2 a respectively (same for N_3, N_4). All the other actions are summarized in the table above. In that example we see that network coding doesn't have great advantage over classical method, we gain only two transmission overall at time slot t_3 one for each access point.

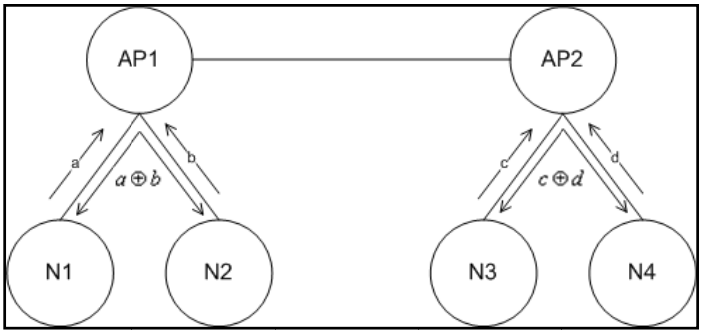


Fig. 3.9 Actions for the case 3.4

3.5 A case with two access points with two nodes each, connected to a server

In the figure below we can see the structure of the network

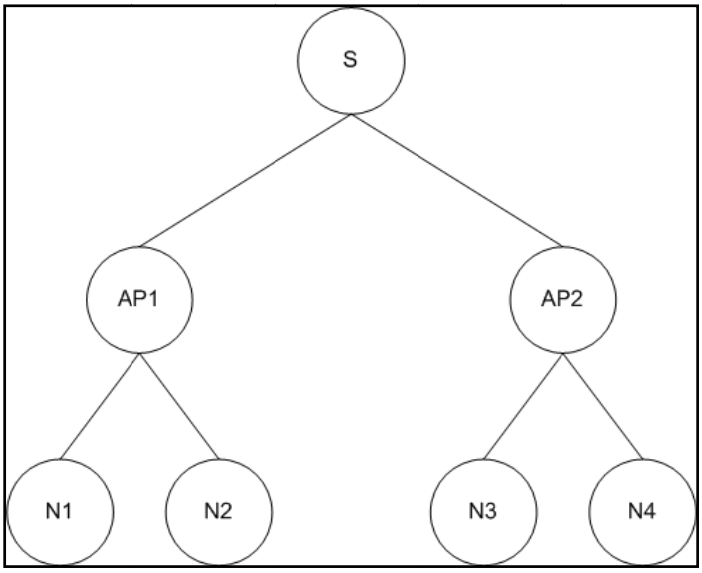


Fig. 3.10 A case with two access points with two nodes each, connected to a server

Nodes N_1, N_2, N_3 and N_4 have information bits a, b, c and d respectively. We want each node in the network to have all information bits.

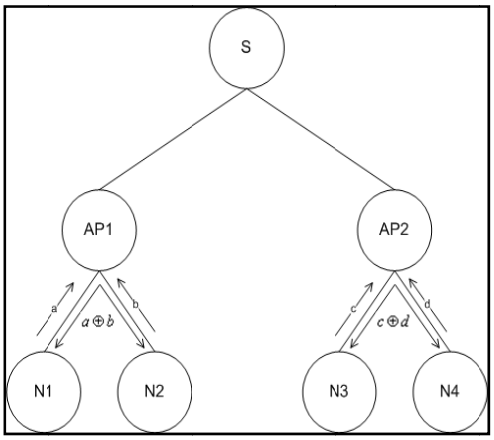
In the next two tables we summarize the actions that take place and the bits that are stored in each node, access point and server respectively.

Time	**Action**
t_0	
t_1	$N_1 \rightarrow AP_1$ a $N_3 \rightarrow AP_2$ c
t_2	$N_2 \rightarrow AP_1$ b $N_4 \rightarrow AP_2$ d
t_3	$AP_1 \rightarrow N_1,N_2$ a+b $AP_2 \rightarrow N_3,N_4$ c+d
t_4	$AP_1 \rightarrow S$ a
t_5	$AP_2 \rightarrow S$ c
t_6	$S \rightarrow AP_1,AP_2$ a+c
t_7	$AP_1 \rightarrow N_1,N_2$ a+c $AP_2 \rightarrow N_3,N_4$ a+c $AP_1 \rightarrow S$ b
t_8	$AP_2 \rightarrow S$ d
t_9	$S \rightarrow AP_1,AP_2$ b+d
t_{10}	$AP_1 \rightarrow N_1,N_2$ b+d $AP_2 \rightarrow N_3,N_4$ b+d

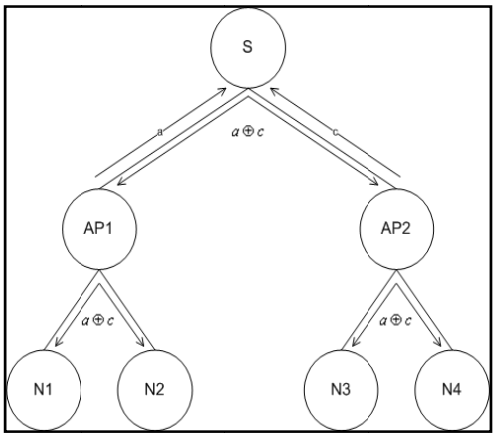
Time	AP_1	AP_2	S	N_1	N_2	N_3	N_4
t_0				a	b	c	d
t_1	a	c		a	b	c	d
t_2	a,b	c,d		a	b	c	d
t_3	a,b	c,d		a,b	a,b	c,d	c,d
t_4	a,b	c,d	a	a,b	a,b	c,d	c,d
t5	a,b	c,d	a,c	a,b	a,b	c,d	c,d
t_6	a,b,c	a,c,d	a,c	a,b	a,b	c,d	c,d
t_7	a,b,c	a,c,d	a,b,c	a,b,c	a,b,c	a,c,d	a,c,d
t_8	a,b,c	a,c,d	a,b,c,d	a,b,c	a,b,c	a,c,d	a,c,d
t_9	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c	a,b,c	a,c,d	a,c,d
t_{10}	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d	a,b,c,d

The figures below illustrate the transmissions that take place.

From time slot t_1 to time slot t_3:



From time slot t_4 to time slot t_7:



The other actions are the same but with transmissions of different information bits as we can see from the tables.

3.6 Conclusion

In this chapter we saw the benefits of applying network coding in some network topologies. This is not of course an exhaustive study but just a beginning.

For further research it is interesting to try to formulate the gain from network coding for some networks (primitive ones) using the parameters of the network (such as access points, connected nodes etc.). After that we could identify sub-networks with known gain properties in larger networks and then, by using them, try to calculate the gain for the whole network. It may be of interest to study the connection of the gains of small networks with the total gain of the network.

Dina Katabi from MIT presented (WiOpt 07 that took place in Lemessos, Cyprus) analog network coding [17], in a way to take advantage of the interference from transmissions in wireless networks. It is known that interference is harmful in general. Wireless networks strive to avoid scheduling multiple transmissions at the same time from nodes that are near, in order to prevent interference. The paper of Katabi [17] adopts the opposite approach. It encourages strategically picked senders to interfere. Instead of forwarding packets, routers forward the interfering signals. The destination leverages network-level information to cancel the interference and recover the signal destined to it. The result is analog network coding because it codes signals not bits. More details for analog network coding are given in the paper.

Chapter 4 A new approach for Network Code generation

In this chapter we present a new algorithm for the construction of a network code for a multicast network. We restrict our study to acyclic directed graphs as models of our network. The algorithm is centralized and assumes knowledge of the topology of our network.

4.1 The algorithm

Before giving the pseudo-code we briefly describe the algorithm.

Consider a directed acyclic graph $G=(V,E)$ where *V* is the vertex set and *E* is the edge set, every edge having unitary weight. Graph G has one source $S \in V$ and n sinks $T_i \in V \; i=1...n$.

The maximum flow from S to i-th sink is h_i. If we want to transmit h information symbols simultaneously to all sink nodes, then $h \le \min_i h_i$ [1]. Also, it is known from Menger's Theorem [9,10,11] that from source S to each sink node there are h edge-disjoint paths and therefore we have in total nh paths.

We use the following notation: $P_{i,j}$ is the path *j* of receiver i, P_i is the set of h edge disjoint paths of receiver i and we denote as: $P_{i,j}=\left\{e_{i,j}^{start},...,e_{i,j}^{end}\right\}$

Recall that $\partial^-(v)=\{e\in E \mid e=(u,v)\}$ is the set of the incoming channels to node v and $\partial^+(v)=\{e\in E \mid e=(v,w)\}$ is the set of outgoing channels

It is easy to see that $e_{i,j}^{start} \in \partial^+(S)$ and $e_{i,j}^{end} \in \partial^-(T_i)$ $\forall\, 1 \le i \le n \; \forall 1 \le j \le h$

$P_i \triangleq \{P_{i,j}: 1 \leq j \leq h\} \; \forall \; 1 \leq i \leq n$

In every path $P_{i,j}$ we assign a global encoding vector $v_{i,j}$. We have to assign in total h linearly independent vectors ($h = \dim(\langle v_{i,j}: 1 \leq j \leq h \rangle)$) to the paths $P_{i,j} \in P_i$. This ensures that every sink can reconstruct the vector space. Assignment of a vector to a path means that we assign this vector to every edge that belongs to the path.

We will process the outgoing edges from source S, to help us find the vectors that we will assign to the paths. After that, we will assign the same vector to the rest of the edges for each path.

It is important to assign the vectors in a way that all paths of the same receiver have h linearly independent vectors and that this condition applies to all receivers.

The above apply to the edge-disjoint paths belonging to **the same receiver**. For the paths belonging to **different receivers** we have two cases; those who **overlap** and those who **do not**. An interesting situation arises when we consider paths of two or more receivers that have common edges. In this case we have to decide what vector to assign to the common edge. Edges that belong to more than one path and do not belong to $\partial^+(S)$ (outgoing links from the source) are called **critical edges**. As we will see below the main problem is to find the global encoding vector that we will assign to these critical edges.

An important step is the discovery of the critical edges. If our network does not have any critical edges then network coding does not offer any benefit to it.

In the following figure we see part of network (graph). The dotted edge is a critical edge because it belongs to paths k and m of the i and j receivers respectively.

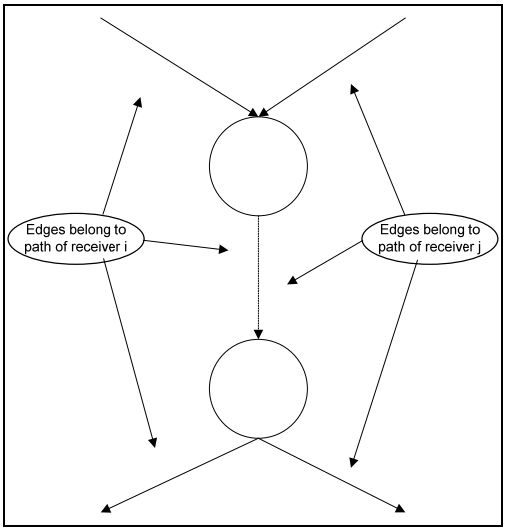


Fig. 4.1 A critical edge

There are two cases for the global encoding vectors $v_{i,k}$ (belonging to T_i) and $v_{j,m}$ (belonging to T_j) that are assigned to the paths $P_{i,k}$ and $P_{j,m}$ sharing a critical edge :

- $v_{i,k} = v_{j,m}$, in this case the assigned vector to the critical edge is $v_{critical} = v_{i,k} (= v_{j,m})$
- $v_{i,k} \neq v_{j,m}$, in this case we have to somehow combine them linearly

After that, the edges following the critical edge (which belong to the overlapping paths) replace their vector with the new encoded vector.

At the end chapter we apply the algorithm to some networks.

Recapitulating the previous description we present the following pseudo code:

We have n sinks T_i and one source S

For every sink T_i

Compute h_i =maxflow(S, T_i)

EndFor

$h = \min_i h_i$

For every sink T_i

Find h edge-disjoint paths for T_i

EndFor

Use edges in $\partial^+(S)$ to initialize the vectors in paths

a) Choose linearly independent vectors for the edge-disjoint paths of the same receiver.

b) Find critical edges and encode the vectors of overlapping paths

c) Replace assigned vectors to edges following critical edges of overlapping paths.

d) If for some sink the incoming edges are not assigned with h linearly independent vectors, then go to a) (with a different choice of vectors for the $\partial^+(S)$ edges).

Note: We did not mention anything about the local encoding kernels, because if we have the global encoding vectors it is easy to find them. This is explained as follows:

Suppose that we have a node with two incoming and two outgoing edges. We know both the global encoding vectors for the incoming and outgoing edges as depicted below:

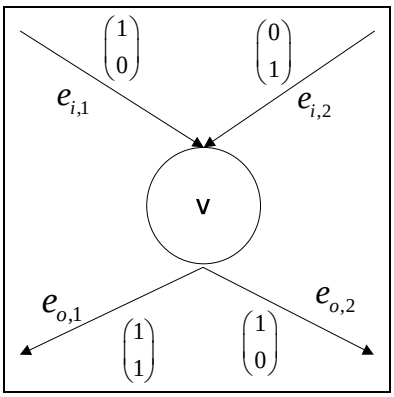


Fig. 4.2 a node with two incoming and two outgoing channels

Node v has a local encoding kernel with matrix K_v with dimension $|\partial^-(v)|\times|\partial^+(v)|$ here K_v is a 2x2 matrix $K_v = \begin{pmatrix} k_{e_{i,1},e_{o,1}} & k_{e_{i,1},e_{o,2}} \\ k_{e_{i,2},e_{o,1}} & k_{e_{i,2},e_{o,2}} \end{pmatrix}$. As we know [Chapter 2], $f_{e_{o,i}} = \sum_{d\in\partial^-(v)} k_{d,e_{o,i}} f_d \quad \forall i \in \partial^+(v)$. Thus we can solve the system and compute K_v in our example $K_v = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.

In the following examples we demonstrate the application of our algorithm

4.2 Examples

Example 4.1: Butterfly network

In our first example, we use the well known butterfly network that is in depicted the figure below.

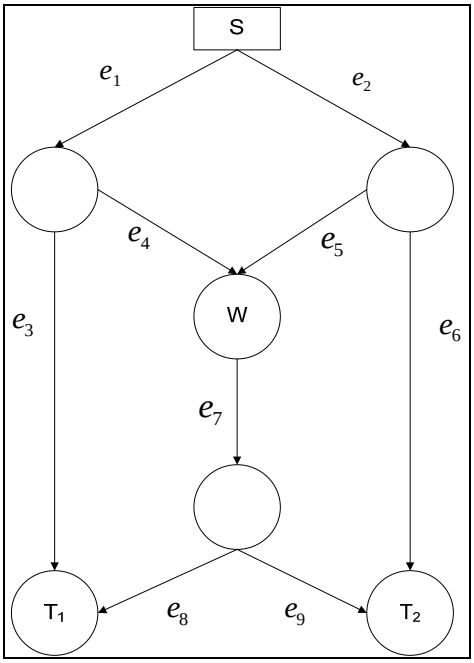


Fig. 4.3 Butterfly network

For this network the maximum flow for each sink is two, $h = h_1 = h_2 = 2$. It is easy to find two edge-disjoint paths for each receiver. In the next two figures we see the paths of receivers T_1 and T_2 respectively.

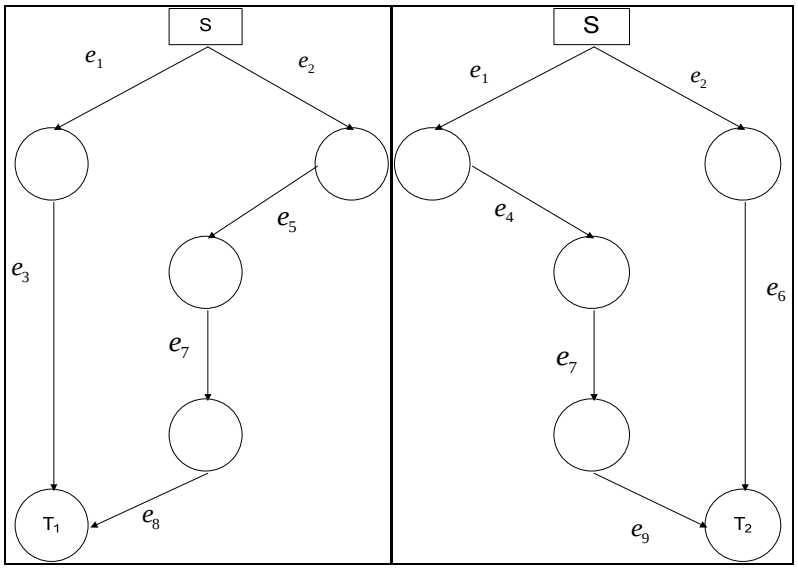


Fig. 4.4 The edge-disjoint paths for the two receivers for butterfly network

The paths for the first receiver are $P_{1,1} = \{e_1, e_3\}$, $P_{1,2} = \{e_2, e_5, e_7, e_8\}$ and for the second are $P_{2,1} = \{e_1, e_4, e_7, e_9\}$, $P_{2,2} = \{e_1, e_6\}$ respectively. It is obvious that we have only one critical edge e_7. Edge e_7 belongs to $P_{1,2}$ and $P_{2,1}$. Next we proceed with edges belonging to $\partial^+(S)$. The vectors that we assign belong to $GF(2)^2$ (see chapter 2). In the next chapter we discuss a new approach for the computation of the order of the Galois Field for a given network.

An orthogonal basis of $GF(2)^2$ is $\left\langle \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\rangle$. There are three vectors in $GF(2)^2$, namely : $\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. We have two edges in $\partial^+(S)$ ={ e_1, e_2 }.

In this case it is easy to assign the vectors for both paths, because we want the vector in e_1 to be linear independent to the vector in e_2. Thus, we assign vectors

$v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ to the edges e_1 and e_2 respectively, and initialize the paths' vectors as follows:

for path ($P_{1,1}$, $P_{2,1}$) $v_{1,1} = v_{2,1} = v_1$, and

for path ($P_{1,2}$, $P_{2,2}$) $v_{1,2} = v_{2,2} = v_2$.

Next we find the critical edges. As we mentioned before in this network there is only one critical edge which can be easily found. The overlapping paths are:

$P_{1,2}$, $P_{2,1}$. In order to transmit all the information through channel e_7 it is necessary to encode information in node W.

We have to combine vectors v_1 and v_2, assign the result to edge e_7 and to the edges following e_7 that belong to the overlapping paths. The encoding here might be a simple XOR operation, in which case the "encoded" or "combined" vector is $v_1 \oplus v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

The solution is depicted in the figure below:

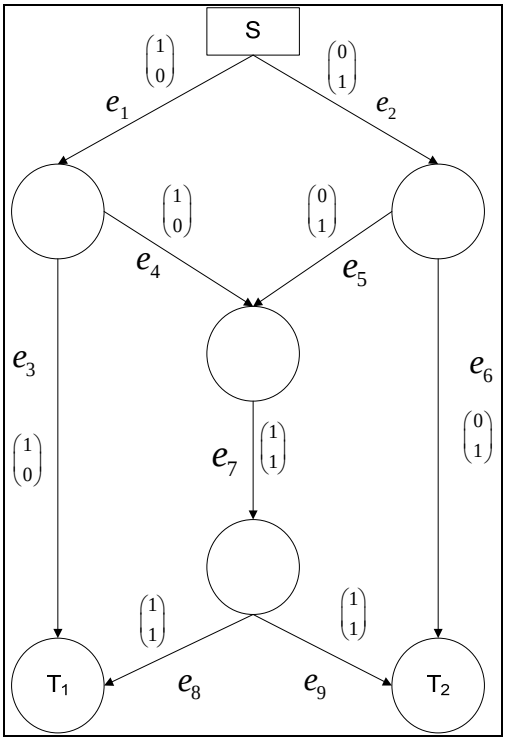


Fig. 4.5 Global encoding kernels for the butterfly network

Finally, sinks T_1 and T_2 receive the information symbols $\{a, a \oplus b\}$, $\{b, a \oplus b\}$ respectively.

<u>Example 4.2 (a multicast network with three receivers)</u>

In the figure below we can see the graph for this example (it consists of two butterfly networks connected in parallel)

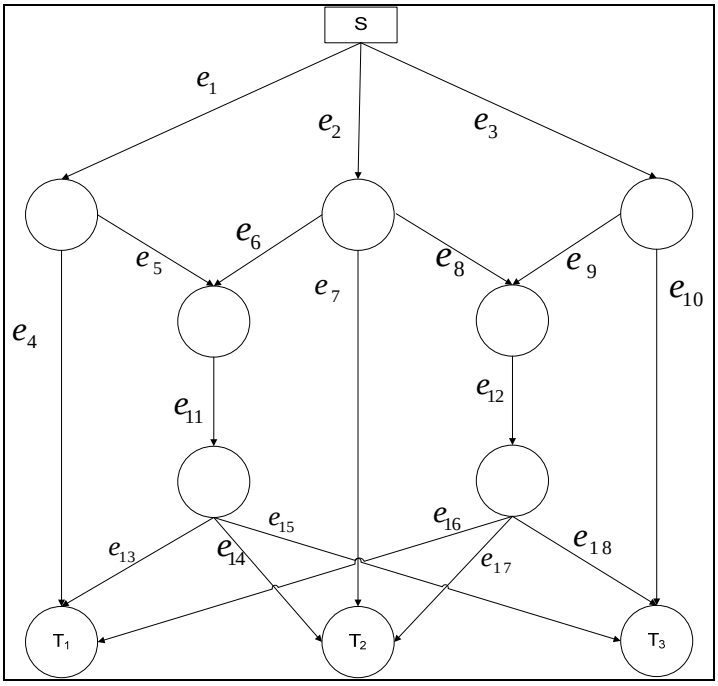


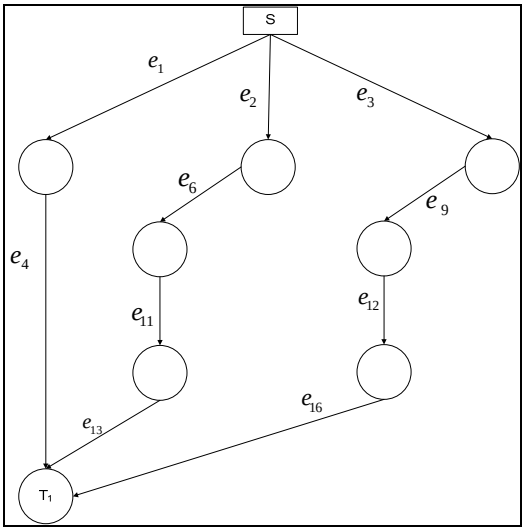
Fig. 4.6 A multicast network with three receivers

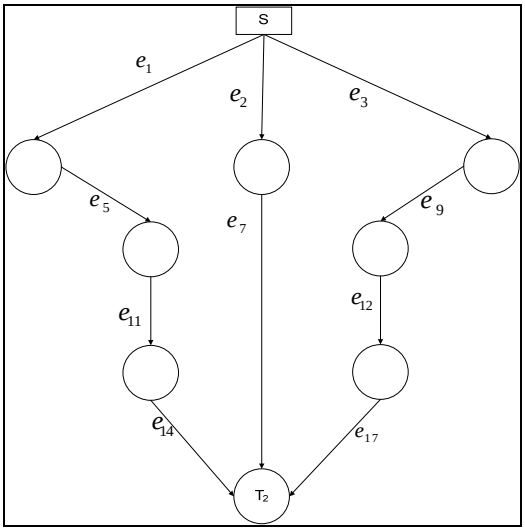
We have three receivers, maximum flow from S to each receiver is three, $h_1 = h_2 = h_3 = 3$. So h=3, and we need to find three edge-disjoint paths for each receiver.

In the images below we see the three edge-disjoint paths for the receivers T_1, T_2 and T_3 respectively:

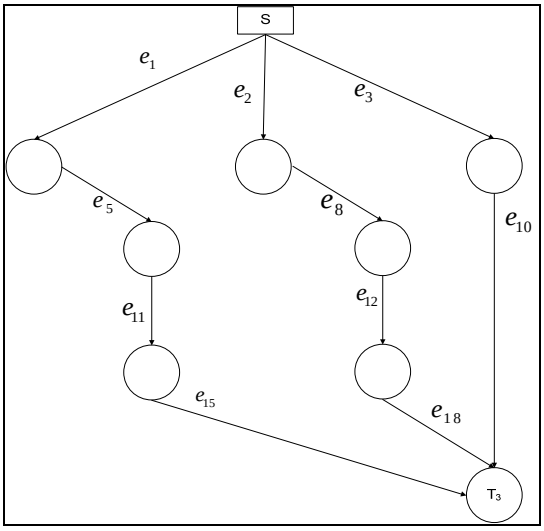
The set of paths for the receivers are:

$$P_1 = \{P_{1,1}, P_{1,2}, P_{1,3}\}, P_2 = \{P_{2,1}, P_{2,2}, P_{2,3}\}, P_3 = \{P_{3,1}, P_{3,2}, P_{3,3}\}$$





$P_{1,1} = \{e_1, e_4\}, P_{1,2} = \{e_2, e_6, e_{11}, e_{13}\}, P_{1,3} = \{e_3, e_9, e_{12}, e_{16}\}$ $P_{2,1} = \{e_1, e_5, e_{11}, e_{14}\}, P_{2,2} = \{e_2, e_7\}, P_{1,2} = \{e_3, e_9, e_{12}, e_{17}\}$



$P_{3,1} = \{e_1, e_5, e_{11}, e_{15}\}, P_{3,2} = \{e_2, e_8, e_{12}, e_{18}\}, P_{3,3} = \{e_3, e_{10}\}$

Fig. 4.7 The paths for the receivers

Now we proceed with the initialization process and we assign vectors to $\partial^+(S)$ edges.

The outgoing edges and the paths they belong to are :

$e_1 : P_{1,1}, P_{2,1}, P_{3,1}$
$e_2 : P_{1,2}, P_{2,2}, P_{3,2}$
$e_3 : P_{1,3}, P_{2,3}, P_{3,3}$

We want the paths to the same receiver to carry linearly independent vectors.

The vectors that we assign belong to $GF(2)^3$

Thus we assign the vectors $\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}$ to edges e_1, e_2, e_3 respectively.

So, all edges of the paths take the vector that is assigned to the first edge of their path. Now we have to find the critical edges of our graph. There are four critical edges e_5, e_9, e_{11}, e_{12}.

But after a more careful examination we can see that only two of them are of interest, because edges e_5, e_9 belong to paths with the same vectors. The "combined" vector will be the common one.

The edges e_{11}, e_{12} belong to paths with different vectors. Here we have to encode the different vectors to a new encoded one. So in order to keep the linear independence among the paths, we sum (XOR) the vectors. The new vectors for edges e_{11}, e_{12} are

$\begin{pmatrix}1\\0\\0\end{pmatrix} + \begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix} + \begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\1\\1\end{pmatrix}$ respectively.

After that we have to replace the vectors of the edges that are following e_{11}, e_{12} and belong to the same paths with them.

As you can see, in this network only two nodes need to use network coding.

The final solution is depicted in the figure below:

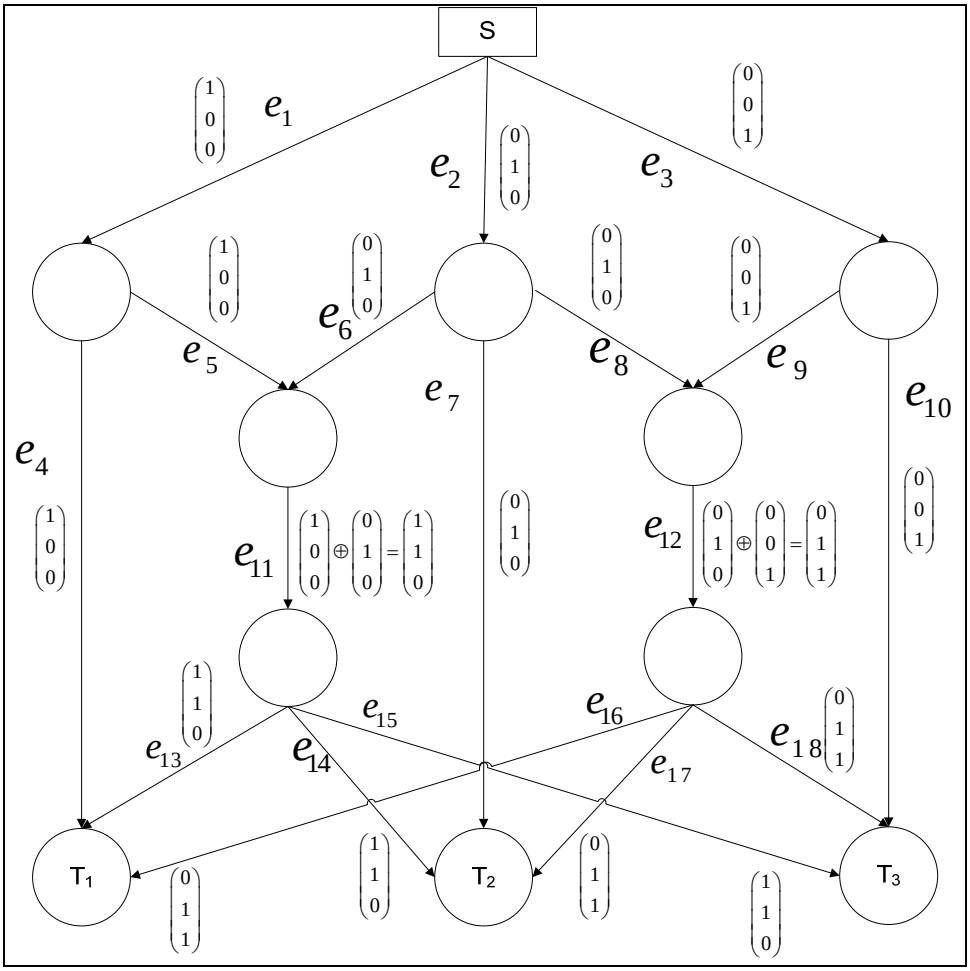


Fig. 4.8 The global encoding vectors

Note: It is easy to see that the same approach can be applied to a n-1 parallel connected butterfly network with n receivers. This network has n-1 critical edges.

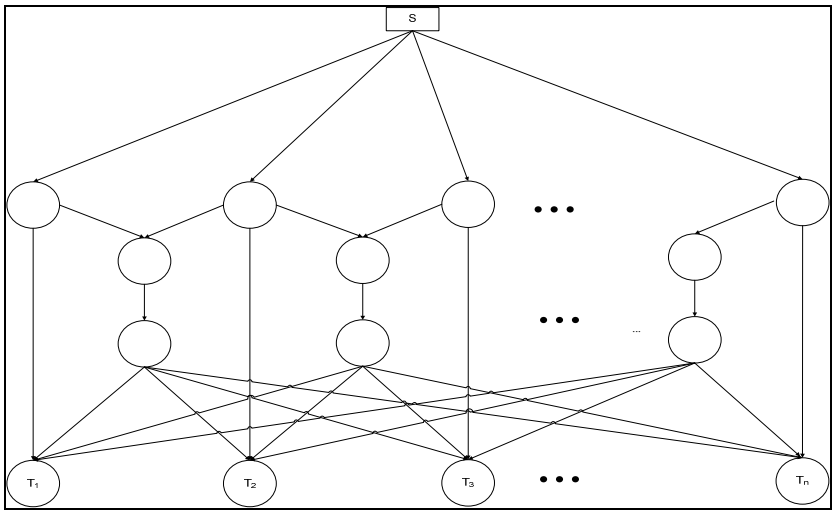


Fig. 4.9 A multicast network with n receivers and n-1 critical edges

Example 4.3 Combination network (4/2)

In the figure below we see the network of this example:

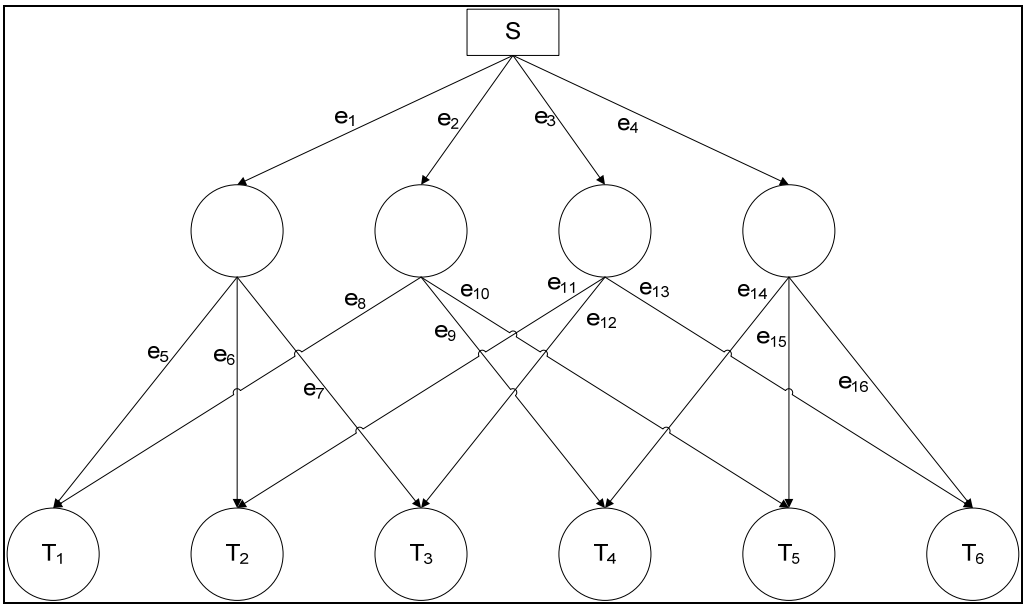


Fig. 4.10 A combination network with six receivers

In this example we have six receivers. The maximum flow from S to each receiver is two, so $h_1 = h_2 = h_3 = h_4 = h_5 = h_6 = 2$.

Thus we can transmit simultaneously to all receivers two information symbols.

As we will see in the next chapter, for this network we need a $GF(3)$ in order to construct our network code. Note that:

$$GF(3)^2 = \{\begin{pmatrix}0\\0\end{pmatrix},\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}2\\0\end{pmatrix},\begin{pmatrix}0\\2\end{pmatrix},\begin{pmatrix}2\\1\end{pmatrix},\begin{pmatrix}1\\2\end{pmatrix},\begin{pmatrix}2\\2\end{pmatrix}\}.$$

In the figures below we can see the two edge-disjoint paths to each receiver respectively.

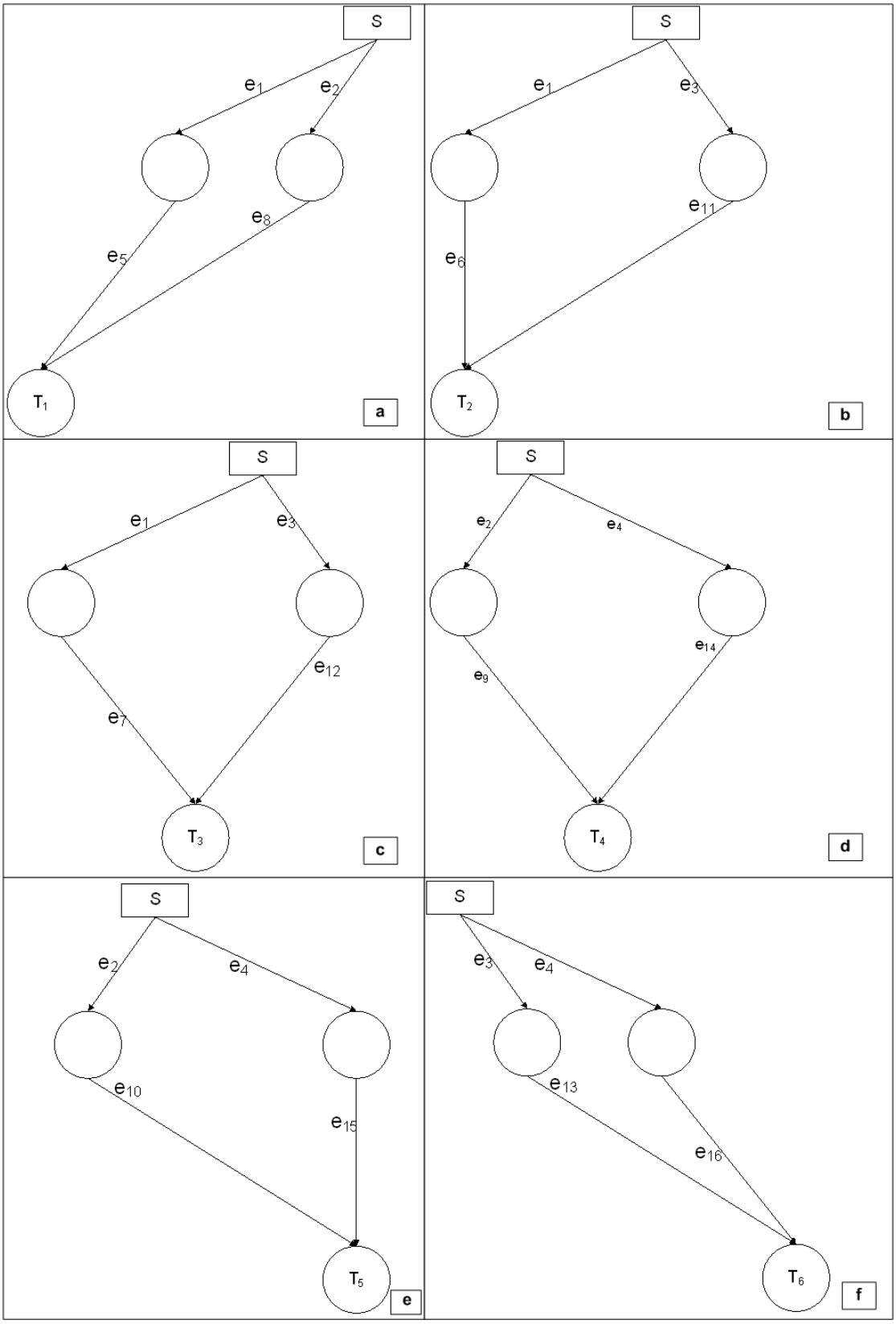


Fig. 4.11 The edge-disjoint paths for the receivers

The paths are:

$P_{1,1} = \{e_1, e_5\}, P_{1,2} = \{e_2, e_8\}$
$P_{2,1} = \{e_1, e_6\}, P_{2,2} = \{e_3, e_{11}\}$
$P_{3,1} = \{e_1, e_7\}, P_{3,2} = \{e_3, e_{12}\}$
$P_{4,1} = \{e_2, e_9\}, P_{4,2} = \{e_4, e_{14}\}$
$P_{5,1} = \{e_2, e_{10}\}, P_{5,2} = \{e_4, e_{15}\}$
$P_{6,1} = \{e_3, e_{13}\}, P_{6,2} = \{e_4, e_{16}\}$

Now we proceed with the initialization process and assign vectors to $\partial^+(S)$ edges.

The outgoing edges and the paths they belong to are:

$e_1 : P_{1,1}, P_{2,1}, P_{3,1}$
$e_2 : P_{1,2}, P_{4,1}, P_{5,1}$
$e_3 : P_{2,2}, P_{3,2}, P_{6,1}$
$e_4 : P_{4,2}, P_{5,2}, P_{6,2}$

We want the paths of the same receiver to carry linearly independent vectors. As you can see edge e_1 must carry a vector linearly independent to the vector assigned to e_2 and to the vector assigned to e_3. Respectively the vector assigned to edge e_2 must be linearly independent with the vectors of edges e_1 and e_4, and so on for the other edges. This is feasible with the following vector assignment:

$$e_1 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}, e_2 : \begin{pmatrix} 1 \\ 1 \end{pmatrix}, e_3 : \begin{pmatrix} 1 \\ 2 \end{pmatrix}, e_4 : \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

All edges in the paths take the vector that is assigned to the first edge of their path. Now we have to find the critical edges of our graph. But in this network there are not any critical edges. Thus in the figure below we can see the final solution of network code for this graph. It is clear that in this case there is no benefit in using network coding

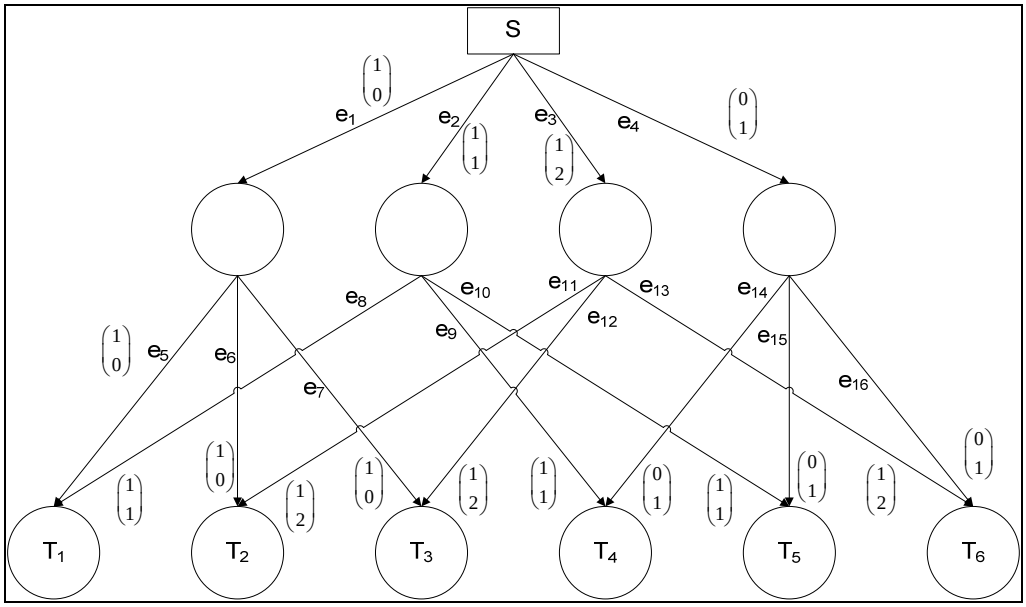


Fig. 4.12 The global encoding vectors for the combination network

In the following example, we will study a network that has more than one set of edge-disjoint paths for the same receiver. We will see how the choice of paths can affect the network code and that in some cases we will not need any coding for our network.

<u>Example 4.4</u>

In the figure below we see the graph of our example:

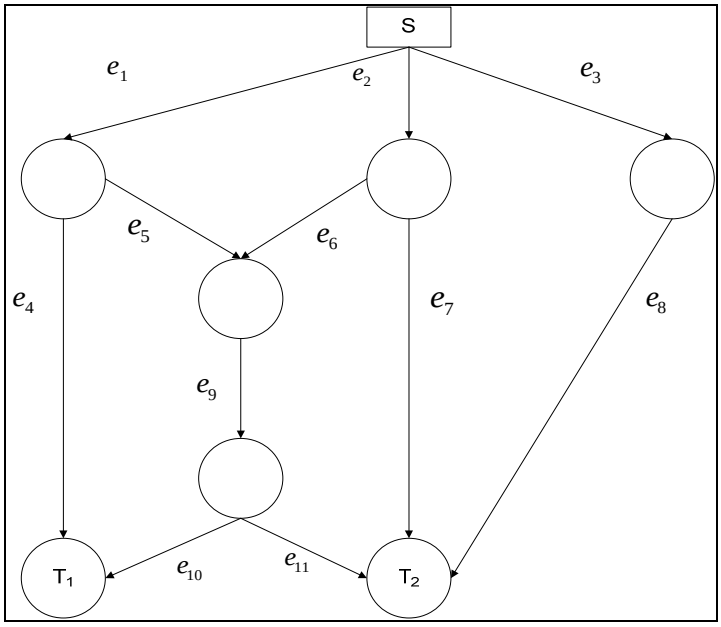


Fig. 4.13 Network of example 4.4

We have two receivers with maximum flow $h_1 = 2$ and $h_2 = 3$, so $h = 2$.

From the theorem of Menger we can find two edge disjoint paths for each receiver. For the receiver T_2 there exist three edge-disjoint paths and thus at least $\binom{3}{2} = 3$, sets of two edge-disjoint paths for T_2. In this network we can find four sets of two edge-disjointed paths to T_2. The paths for receiver T_1 are:

$$\begin{Bmatrix} P_{1,1} = \{e_1, e_4\} \\ P_{1,2} = \{e_2, e_6, e_9, e_{10}\} \end{Bmatrix}$$

As mentioned before, there are four sets of two edge-disjointed paths for T_2:

$$\begin{Bmatrix} P_{2,1} = \{e_2, e_7\} \\ P_{2,2} = \{e_3, e_8\} \end{Bmatrix} \text{or} \begin{Bmatrix} P_{2,1} = \{e_1, e_5, e_9, e_{11}\} \\ P_{2,1} = \{e_2, e_7\} \end{Bmatrix} \text{or} \begin{Bmatrix} P_{2,1} = \{e_1, e_5, e_9, e_{11}\} \\ P_{2,2} = \{e_3, e_8\} \end{Bmatrix} \text{or} \begin{Bmatrix} P_{2,1} = \{e_2, e_6, e_9, e_{11}\} \\ P_{2,2} = \{e_3, e_8\} \end{Bmatrix}$$

So we have to check four schemes of paths.

First case:

We consider the following paths:

$$\left\{\begin{matrix} P_{1,1}=\{e_1,e_4\} \\ P_{1,2}=\{e_2,e_6,e_9,e_{10}\} \end{matrix}\right\} \text{ and } \left\{\begin{matrix} P_{2,1}=\{e_2,e_7\} \\ P_{2,2}=\{e_3,e_8\} \end{matrix}\right\}$$

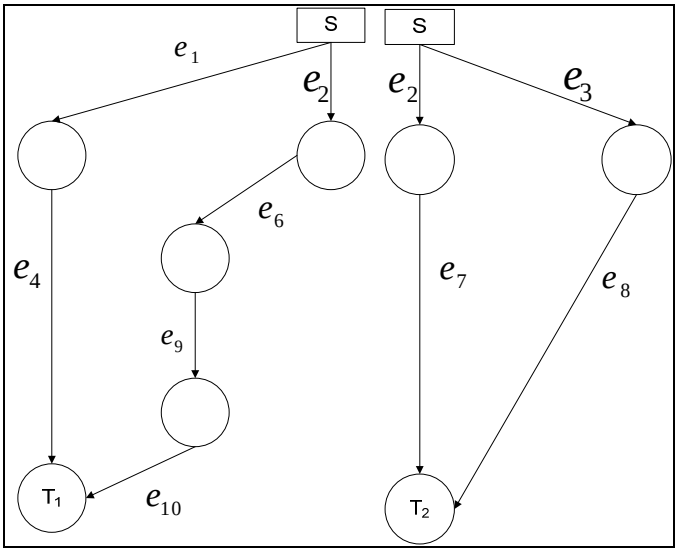


Fig. 4.14 The edge-disjoint paths for the receivers

Now we proceed with the initialization process and we assign vectors to $\partial^+(S)$ edges.

The outgoing edges and the paths they belong to are:

$e_1 : P_{1,1}$
$e_2 : P_{1,2}, P_{2,1}$
$e_3 : P_{2,2}$

We want the paths of the same receiver to carry linearly independent vectors. So we have the following assignment:

$$e_1 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}, e_2 : \begin{pmatrix} 0 \\ 1 \end{pmatrix}, e_3 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

So, all edges in the paths take the vector that is assigned to the first edge of their path. Now we have to find the critical edges of our graph. But in this network with

the previous configuration of paths, there are not any critical edges (that means no coding needed). In the figure below we can see the solution:

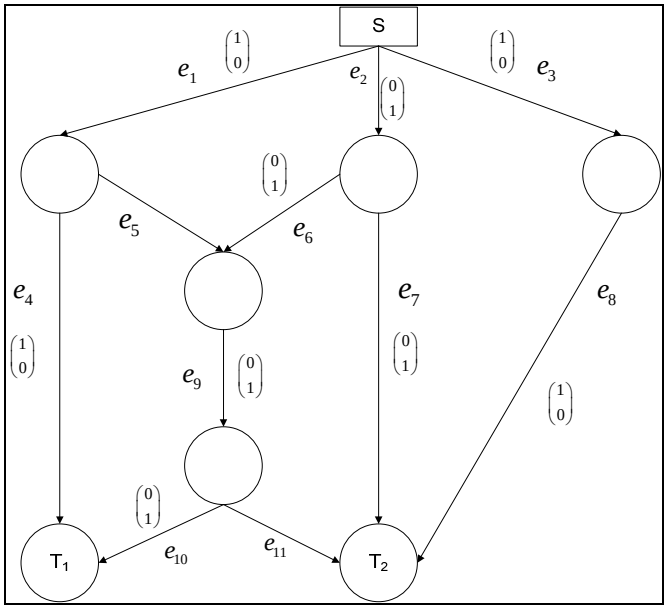


Fig. 4.15 Global encoding vectors for the first case

Second case:

We consider the following paths:

$$\left\{\begin{matrix} P_{1,1} = \{e_1, e_4\} \\ P_{1,2} = \{e_2, e_6, e_9, e_{10}\} \end{matrix}\right\} \text{and} \left\{\begin{matrix} P_{2,1} = \{e_1, e_5, e_9, e_{11}\} \\ P_{2,1} = \{e_2, e_7\} \end{matrix}\right\}$$

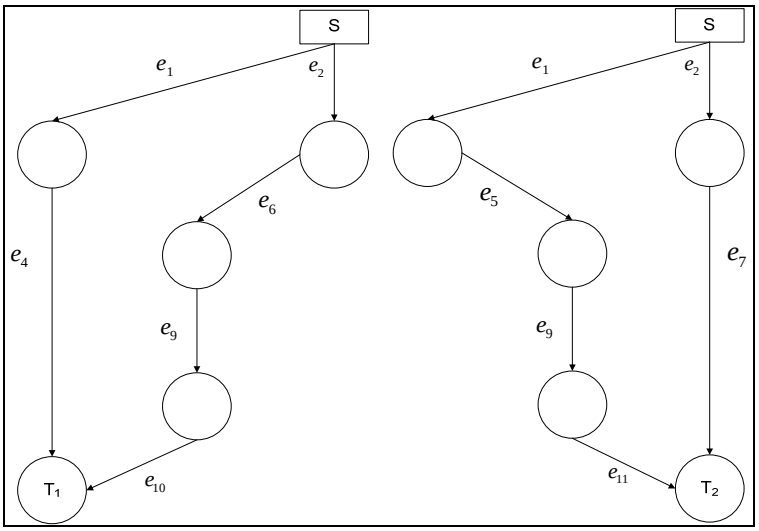


Fig. 4.16 The edge-disjoint paths for the receivers

Now we proceed with the initialization process and we assign vectors to $\partial^{+}(S)$ edges.

The outgoing edges and the paths they belong to are:

$e_1 : P_{1,1}, P_{2,1}$
$e_2 : P_{1,2}, P_{2,2}$
$e_3 : -$

We want the paths of the same receiver to carry linearly independent vectors. So we have the following assignment:

$$e_1 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}, e_2 : \begin{pmatrix} 0 \\ 1 \end{pmatrix}, e_3 : -$$

So, all edges in the paths take the vector that is assigned to the first edge of their path. Now we have to find the critical edges of our graph. In this case we have one critical edge e_9 and we need coding because the incoming vectors are different. The new vector assigned to edge e_9 is $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. After that we have to change the vectors of edges that are after e_9 and belong to the same path.

The final vector assignment is depicted in the figure below:

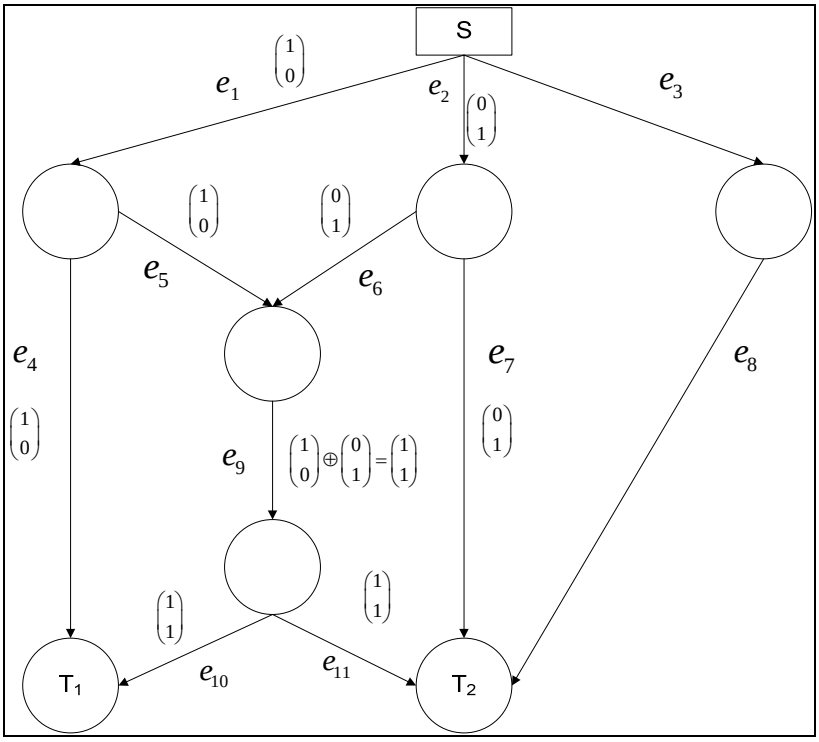


Fig. 4.17 Global encoding vectors for the second case

Third case:

We have the following paths:

$$\left\{\begin{matrix} P_{1,1}=\{e_1,e_4\} \\ P_{1,2}=\{e_2,e_6,e_9,e_{10}\} \end{matrix}\right\} \text{and} \left\{\begin{matrix} P_{2,1}=\{e_1,e_5,e_9,e_{11}\} \\ P_{2,2}=\{e_3,e_8\} \end{matrix}\right\}$$

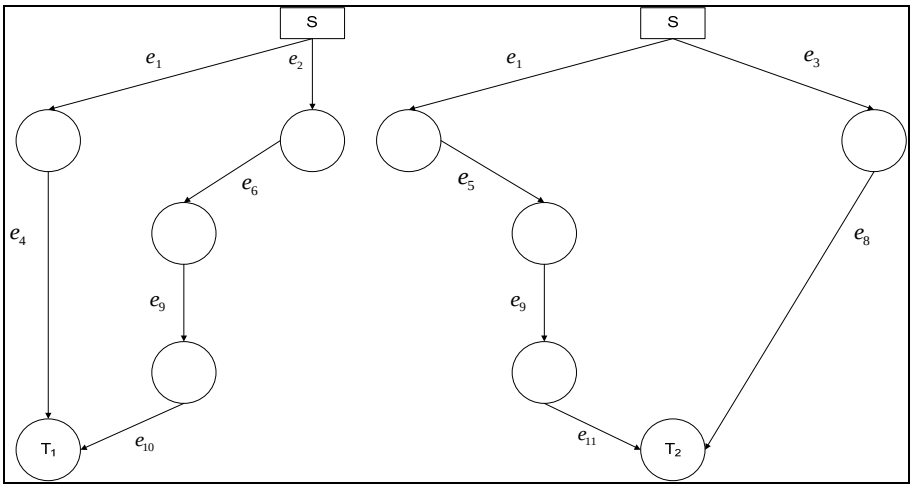


Fig. 4.18 The edge disjoint paths for the receivers

Now we proceed with the initialization process, we assign vectors to $\partial^{+}(S)$ edges. The outgoing edges and the paths they belong to are:

$e_1 : P_{1,1}, P_{2,1}$
$e_2 : P_{1,2}$
$e_3 : P_{2,2}$

We want the paths of the same receiver to carry linearly independent vectors. So we have the following assignment:

$$e_1 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}, e_2 : \begin{pmatrix} 0 \\ 1 \end{pmatrix}, e_3 : \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

So, all edges in the paths take the vector that is assigned to the first edge of their path. Now we have to find the critical edges of our graph. In this case we have one critical edge e_9, we need coding because incoming vectors are different. The new vector assigned to edge e_9 is $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. After that we have to change the vectors of edges that are below e_9 and belong to the same path.

The final solution is depicted in the figure below:

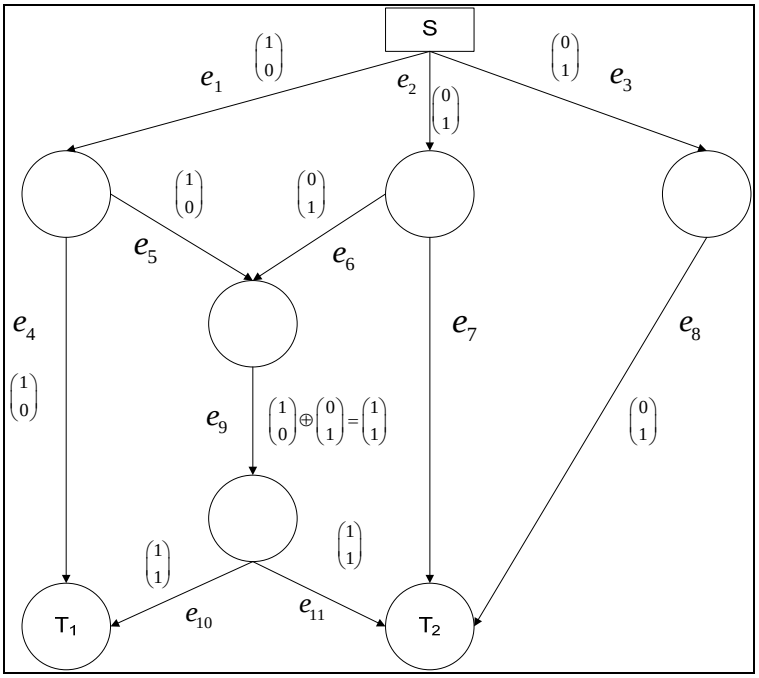


Fig. 4.19 Global encoding vectors for the third case

Fourth case:

We have the following paths:

$$\left\{\begin{matrix} P_{1,1}=\{e_1,e_4\} \\ P_{1,2}=\{e_2,e_6,e_9,e_{10}\} \end{matrix}\right\} \text{and} \left\{\begin{matrix} P_{2,1}=\{e_2,e_6,e_9,e_{11}\} \\ P_{2,2}=\{e_3,e_8\} \end{matrix}\right\}$$

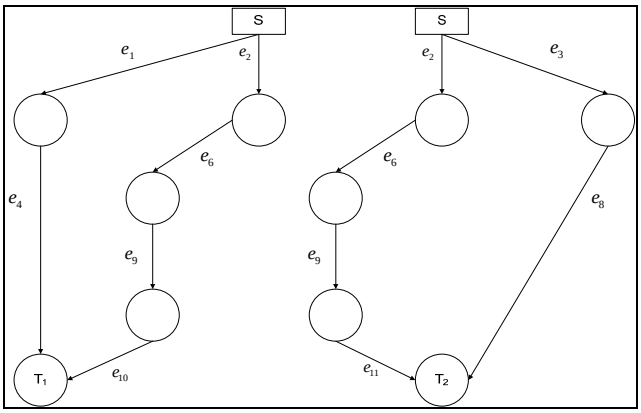


Fig. 4.20 The edge-disjoint paths for the receivers

Now we proceed with the initialization process, we assign vectors to $\partial^+(S)$ edges.

The outgoing edges and the paths they belong to are:

$e_1 : P_{1,1}$
$e_2 : P_{1,2}, P_{2,1}$
$e_3 : P_{2,2}$

We want the paths of the same receiver to carry linearly independent vectors. So we have the following assignment:

$$e_1 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}, e_2 : \begin{pmatrix} 0 \\ 1 \end{pmatrix}, e_3 : \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

So, all edges in the paths take the vector that is assigned to the first edge of their path. Now we have to find the critical edges of our graph. We have e_6 but the vectors are the same, so we do not need coding in this configuration. The solution is depicted below:

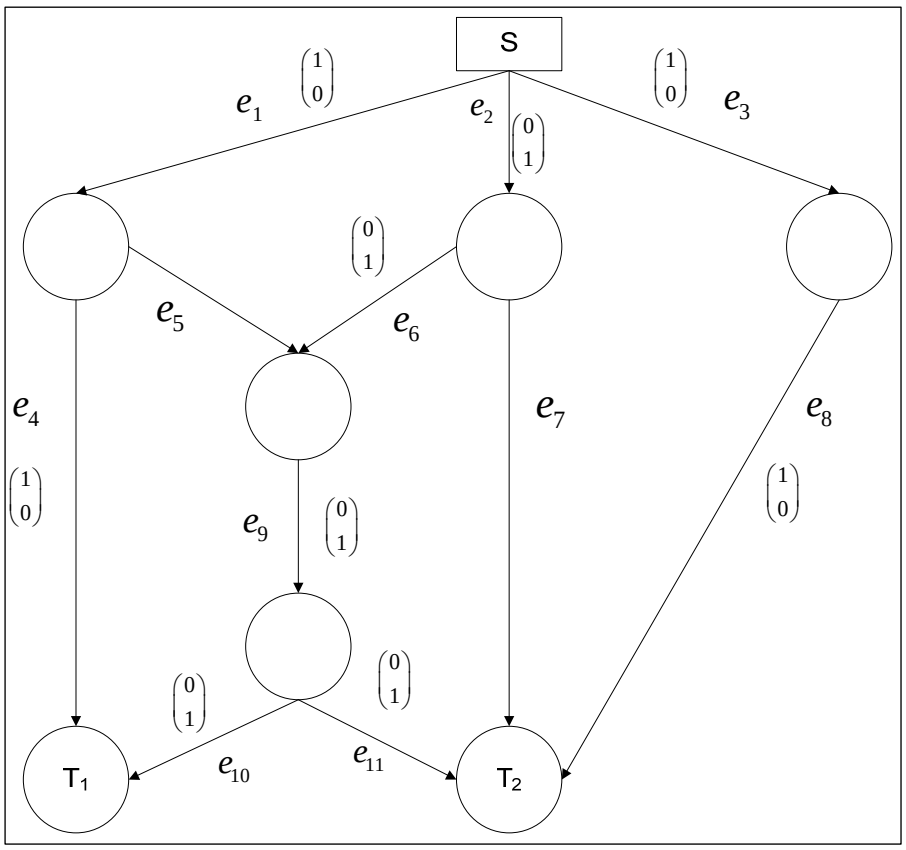


Fig. 4.21 Global encoding vectors for the fourth case

Note: In the network before we checked four configurations of paths. As we saw only two of them need coding.

4.3 Conclusion

From the above analysis we see that only few nodes in a network need to perform network coding, especially the nodes at the start of the critical edges. Knowing the topology of our network (the graph) we are able to decide which nodes need to encode and decode the received information. It is therefore important to explore the selection of paths in order to decrease the complexity and the amount of network coding.

In a multicast network with k receivers, let h_i to be the max flow for receiver i. We know from the theorem of Menger that we can find **exactly** h_i edge-disjoint paths for receiver i. But in a multicast network the total max flow is $h = \min_{1 \le i \le k} h$. That means, that in order to apply any deterministic algorithm for network coding we need exactly h edge-disjoint paths for every receiver. So, we have at least $\binom{h_i}{h} = \frac{h_i!}{h!(h_i - h)!}$ sets of $h$edge-disjoint paths for receiver i. Therefore there are at least $\prod_{i=1}^{k} \binom{h_i}{h}$ schemes of paths. This is a large number.

It is interesting to compare this algorithm with random linear network coding, the most commonly used algorithm for network coding. Our algorithm as we stated in the beginning is centralized (uses the network topology). On the other hand random linear network coding is decentralized.

Chapter 5 Computing the alphabet size

5.1 Introduction

In this chapter we present a technique to find the size of the alphabet of a network code using only $\partial^{+}(S)$ channels. It is known [13] that for a given multicast acyclic network it is sufficient to choose the size of the alphabet to be equal to the number of the receiving nodes. At the end of this chapter we present a "dirty trick" to reduce the size to two for a class of networks.

The size of the alphabet plays an important role because the memory and the computational complexity needed depend on it. The complexity of the operations over finite fields, such as Galois Fields, grows with the field size. For example algorithms over a field of size $q=2^n$ require $O(n^2)$ binary operations. Karatsuba's method [16] requires $O(n^{1.59})$ binary operations. The required storage capabilities at intermediate nodes also depend on the alphabet size.

5.2 Algorithm for alphabet size calculation

From Menger's theorem [9,10,11] it is known that there are h edge-disjoint paths from the source to each receiver. Overall, we have n h paths from source to all receivers (n= number of receivers).

All these paths originate from $\partial^{+}(S)$ channels.

The i-th path for T_t receiver is denoted as $P_{t,I}$. The vector assigned to $P_{t,i}$ is $v_{t,i}$.

In every path of the same receiver we have to assign linearly independent vectors, thus we have to assign h linearly independent vectors to the h paths of every receiver [chapters 2,4]. In this way, we assure that every sink receives the upper bound of information [12,1]

We present the algorithm for the channels that are outgoing from source s

We consider the following two cases for the $|\partial^{+}(S)|$:

1) $|\partial^{+}(S)|=h$, in this case every channel belongs to exactly n paths, one for each receiver. Each channel carries one vector that is linearly independent to the rest *h-1* vectors of the $\partial^{+}(S)$ channels.
2) $|\partial^{+}(S)|>h$, in this case every channel belongs to at most n paths, one or zero for each receiver. Necessarily $|\partial^{+}(S)|-h$ channels carry linearly dependent vectors.

We do not consider the case of $|\partial^{+}(S)|<h$ (does not exist), because then we would have h<h which is not valid.

Every channel belongs to a number of paths (at most one for each receiver, because of the theorem of Menger).

The constraint of our algorithm is that the channels that belong to paths of the same receiver must carry linear independent vectors.

Summarizing our algorithm:

Begin with GF(q),q=2. Dimension of our vector space is h

Find all paths $P_{t,i}$

Find for every edge $e \in \partial^{+}(S)$ the P_e (the set of paths beginning with e)

For every edge $e \in \partial^{+}(S)$

Choose a vector V_e which is linearly independent to the vectors assigned to the other paths of the receivers with path belonging to P_e.

If you cannot assign such a vector increase the order of Galois Field (the next integer that is a power of prime) and start again.

End for

Discussion:

We begin with GF(q), q=2. The dimension of the vector space is h. Choose a vector in order to have linearly independent vectors in different paths of the same receiver. Probably other paths (let us call them $P_{e,}$ the paths that start from edge e) from different receivers traverse the edge e, so we have to choose properly the vector to be linearly independent to vectors of the edges belonging to $\partial^{+}(S)$. If we cannot assign enough linear independent vectors, then we increase q to the next integer that is a power of a prime number and start again. The following two examples illustrate the algorithm:

Examples

We apply our methodology in the following two networks shown in the figures below; the first one is the well known butterfly network. Both networks have maximum flow 2 for each receiver, so the dimension of our vector space is h=2.

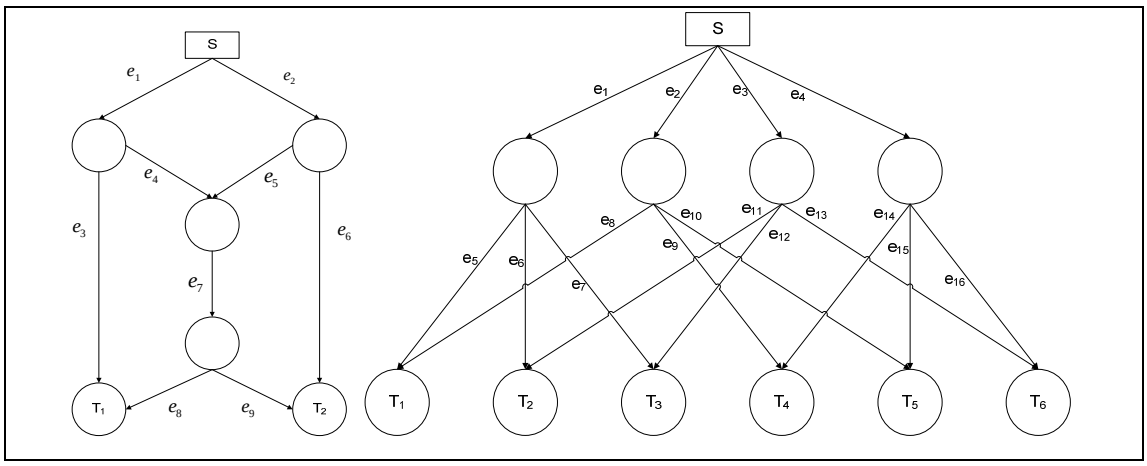


Fig. 5.1 Butterfly and a combination network

For the butterfly network: casein this case $|\partial^{+}(S)|=h$=2. T_1 and T_2 have maximum flow 2. By Menger's Theorem we know that two edge-disjoint paths exist for each receiver. We can reach T_1 by travelling across e_1, e_3 (let us call this path $P_{1,1}$) or e_2, e_5, e_7, e_8 (path $P_{1,2}$). T_2 can be reached by path $P_{2,1}=\{e_1,e_4,e_7,e_8\}$ or $P_{2,2}=\{e_2,e_6\}$. Let us now apply our algorithm. We begin with GF(2) with dimension 2. We observe that edge e_1 belongs to paths $P_{1,1}$ and $P_{2,1}$ and edge e_2 belongs to $P_{1,2}$ and $P_{2,2}$ Therefore the choice of the order of the Galois Field is trivial; we need to assign to the edges e_1 and e_2 linearly independent vectors of the space $GF(2)^2$. We choose the $\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix}$ vectors for e_1 and e_2 respectively. In conclusion: a Galois Field of order 2 is enough for constructing a network code for the butterfly network.

Combination Network: In this case $|\partial^{+}(S)|>h$. T_1,… T_6 have maximum flow 2, so the dimension of vector space is 2.

The paths are:

$P_{1,1}=\{e_1,e_5\}$, $P_{1,2}=\{e_4,e_{14}\}$, $P_{2,1}=\{e_1,e_6\}$, $P_{2,2}=\{e_3,e_{11}\}$
$P_{3,1}=\{e_1,e_7\}$, $P_{3,2}=\{e_2,e_8\}$, $P_{4,1}=\{e_2,e_9\}$, $P_{4,2}=\{e_4,e_{15}\}$
$P_{5,1}=\{e_2,e_{10}\}$, $P_{5,2}=\{e_3,e_{12}\}$, $P_{6,1}=\{e_3,e_{13}\}$, $P_{6,2}=\{e_4,e_{16}\}$

The outgoing channels from S are e_1, e_2, e_3, e_4 .

Edge e_1 belongs to paths $P_{1,1}$, $P_{2,1}$, $P_{3,1}$, e_2 belongs to $P_{3,2}$, $P_{4,1}$, $P_{5,1}$, e_3 to $P_{2,2}$, $P_{5,2}$, $P_{6,1}$ and finally e_4 belongs to $P_{1,2}$, $P_{4,2}$, $P_{6,2}$. We have to assign linearly independent vectors among $P_{i,1}$ and $P_{i,2}$ paths (i=1…6). We begin with GF(2). From the above, we see that e_1 needs to carry a linearly independent vector to e_2 (for T_1 receiver) and e_3 (for T_2 and T_3). Edge e_2 has to carry linearly independent vectors with edges e_1, e_4. Similarly, we work for edges e_3 and e_4 .Totally, we need more than two independent vectors, so we increase the order from two to the next integer that is a power of prime, therefore the order becomes three. We try again with GF(3); we solve the problem by assigning vectors $\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\2\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix}$ (as we saw in the previous chapter) to e_1,e_2,e_3 and e_4 respectively.

5.3 A "dirty trick"

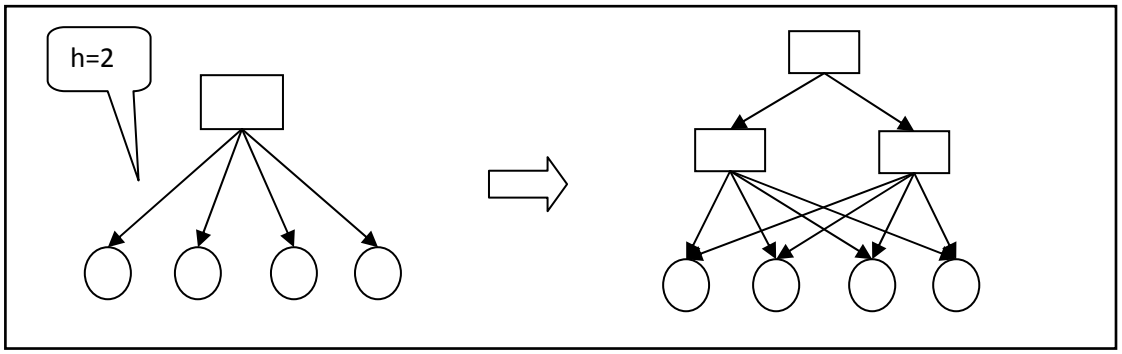
In this part of the chapter, we present a technique to decrease the size of the alphabet (=order of a Galois Field) for a network code. We will explain the technique using the combination network we used in the previous chapter. As we saw in the start of this chapter we have two cases for the number of outgoing from the source channels:

- $|\partial^+(S)|=h$
- $|\partial^+(S)|>h$

Recall that $\partial^{+}(S)$ is the set that describes the outgoing channels from S, and $|\partial^{+}(S)|$ denotes the number of $\partial^{+}(S)$ channels.

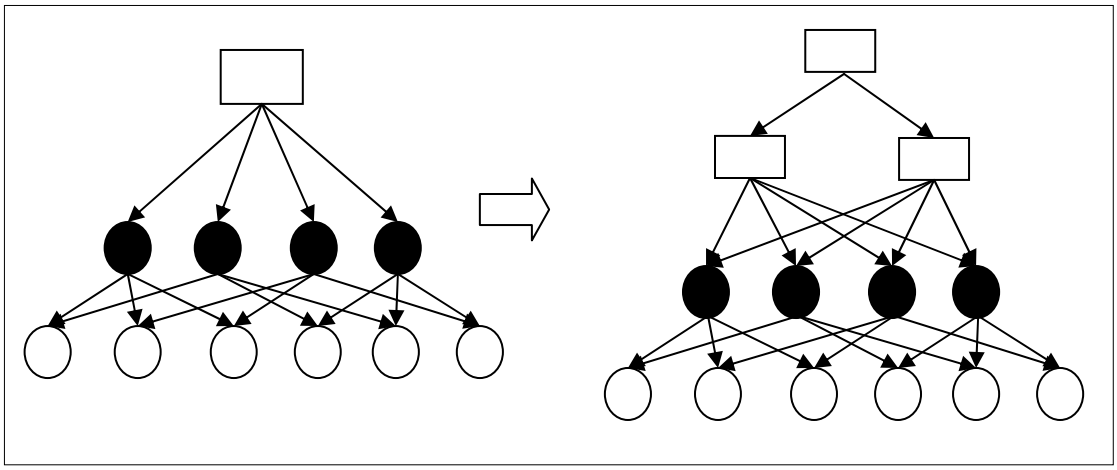
Let us consider the second case. We put h sources (we call them meta-sources) under the source S. Every meta-source "copies" the connectivity of S to the nodes that in topological order are under S, as is shown in the figure below:

In the left side of the figure we see the $\partial^{+}(S)$ channels and we are given that h=2, and that the size of alphabet is three. Doing the transformation in the right we decrease the size to two.



<u>Example</u>

We have the following network



While in the first case we need GF(3) to construct our network code, in the second case we need GF(2). In both cases the dimension of coding vectors is 2.

In the paper of Jaggi etc *"Polynomial time Algorithms for multicast network code construction"*, [13] the author states that we use as order|F| of Galois Field a number satisfying the criterion $\frac{|F|}{|T|} \approx 1$ where $|T|$ is the number of receiver nodes in our network.

In the first network we need a GF(3) to construct the network code, while in the second one, a GF(2) is enough to construct our network code. The gain from this, is that in the first case our vector space has 8 vectors (excluding the zero vector) whereas in the second case only 3.

Note:

Another point here is that the black nodes in the first network have one incoming edge and in the second two. Doing the above transformation in our graph, we need more transmissions from S to next level of nodes of our graph. This is the cost we pay for decreasing the order of our alphabet.

Take into account that the above transformation is for networks that we can choose the order of our alphabet using only $\partial^{+}(S)$ channels.

5.4 Conclusion

In this chapter we saw that the outgoing from the source channels play an important role in network coding. All the information that flows into to the network traverses firstly these channels. The number of $|\partial^{+}(S)|$ channels helps us to choose the order of Galois field to be used for the alphabet of network code. In the end of the chapter we saw a technique to decrease the size of alphabet for a combination network.

For further research it is interesting to see if we can reduce the size of the alphabet to a number lower than the number of receivers for a given network.

Chapter 6 Summary and future work

Closing this thesis, we would like to summarize our results.

In chapter three, we saw the benefits of applying network coding in some network topologies. This is not of course an exhaustive study but a beginning. For further research it is interesting to try to formulate the gain from network coding for some networks using the parameters of the network (such as access points, connected nodes etc.). After that we could identify sub-networks with known gain formulas in larger networks and then, by using them, try to calculate the gain of the whole network. It may be of interest to study the connection of the gains of small networks with the total gain of the network.

In chapter four, we presented an algorithm for the construction of a network code for an acyclic multicast network. The algorithm depends on the knowledge of the topology of the network. We applied the algorithm to some examples. Finally, we see that only few nodes in a network need the ability of network coding. Knowing the topology of our network we are able to decide which nodes need to encode and decode the received information. It is therefore important to explore the selection of paths in order to decrease the complexity and the amount of network coding. It is interesting to compare this algorithm with random linear network coding, the most commonly used algorithm for network coding.

In the previous chapter, we presented a way to find the size of the alphabet of a network code using only outgoing channels from source. In the end of the chapter we saw a technique to decrease the size of alphabet for a combination network. For further research it is interesting to see if we can find tighter bounds for the size of the alphabet of a network code. This is a tough field of network coding.

By the time the present thesis was being written, about eight PhDs theses have been published, the majority of them from Muriel's Medard group at MIT. As well in the

network coding webpage there are about 200 research papers that are related with network coding.

Appendix A: Galois Fields

The purpose of this appendix is to provide the reader with an elementary knowledge of Galois Fields that will aid in the understanding of the material that is covered in the present master's thesis. There are many good textbooks on Galois Fields such as ***"Finite Fields for Computer Scientists and Engineers"*** by *Robert J. McEliece*.

A ***Galois Field*** is a finite field with $\boldsymbol{q=p^n}$ elements where p is a prime number and is denoted by **GF(q)={0,1,…,q-1}**, q is called the ***order*** of the Galois Field. For example GF(2)={0,1} and GF(3)={0,1,2}. More formally a Galois Field of order q is a set of q elements with two binary ***modulo-p*** operations "+" and "x".

In the next two tables we see the binary operations for *GF(2)* and *GF(3)* respectively:

GF(2):

+	0	1
0	0	1
1	1	0

x	0	1
0	0	0
1	0	1

GF(3):

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

x	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Note that the operations are modulo-2 and modulo-3 respectively.

Vector Spaces

Let $V_n = GF(2)^n$ be a n-dimensional vector space. The elements of V_n are called **vectors** and the elements of *GF(2)* are called **scalars** with values 0 and 1.

Example:

Let n=5. The vector space V_5 of all 5-tuples over *GF(2)* consists of 2^5 =32 vectors.

Such as $\begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix},\begin{pmatrix}1\\0\\0\\1\\0\end{pmatrix},\begin{pmatrix}1\\0\\1\\0\\1\end{pmatrix},\begin{pmatrix}1\\1\\1\\0\\0\end{pmatrix},\begin{pmatrix}1\\0\\0\\0\\1\end{pmatrix},\begin{pmatrix}1\\1\\1\\1\\1\end{pmatrix}$.

The vector sum of $\begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix},\begin{pmatrix}1\\0\\0\\1\\0\end{pmatrix}$ is $\begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix}+\begin{pmatrix}1\\0\\0\\1\\0\end{pmatrix}=\begin{pmatrix}1+1\\0\\0\\1\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\1\\0\end{pmatrix}$

A set of vectors $v_1, v_2, ..., v_k$ in a vector space V over a Galois Field F is said to be ***linearly dependent*** if and only if there exists k scalars $a_1, a_2, ..., a_k$ from F, not all zero such that:

$$a_1 v_1 + a_2 v_2 + ... + a_k v_k = 0$$

A set of vectors $v_1, v_2, ..., v_k$ is said to be ***linearly independent*** if it is not linearly dependent. That is, if $v_1, v_2, ..., v_k$ are linearly independent, then:

$$a_1 v_1 + a_2 v_2 + ... + a_k v_k \neq 0 \text{ unless } a_1 = a_2 = ... = a_k = 0 .$$

A set of vectors is said to ***span*** a vector space V if every vector in V is a linear combination of the vectors in the set. In any vector space or subspace there exists at least one set *B* of linearly independent vectors that span the space. This set is called a ***basis*** (or base) of the vector space. The number of vectors in a basis of a vector space is called the ***dimension*** of the vector space.

Consider the vector space $V_n = GF(2)^n$. Let form the following n-tuples:

$$e_0 = (1,0,0,\ldots,0,0)$$
$$e_1 = (0,1,0,\ldots,0,0)$$
$$\ldots$$
$$e_{n-1} = (0,0,0,\ldots,0,1)$$

Then every n-tuple $(a_0, a_1, \ldots, a_{n-1})$ in V_n can be expressed as a linear combination of $e_0, e_1, \ldots, e_{n-1}$ as follows:

$$(a_0, a_1, \ldots, a_{n-1}) = a_0 e_0 + a_1 e_1 + \ldots + a_{n-1} e_{n-1}$$

Therefore $e_0, e_1, \ldots, e_{n-1}$ span the vector space V_n of all n-tuples over $GF(2)$.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow", IEEE Transactions on Information Theory, IT-46, 2000.

[2] C. Gkantsidis, P. R. Rodriguez “Network Coding for Large Scale Content Distribution”, Microsoft Research Technical Report (MSR-TR-2004-80).

[3] T. Ho, R. Koetter, M. Medard, D. Karger and M. Effros, “The Benefits of Coding over Routing in a Randomized Setting”, ISIT 2003

[4] Ning Cai and Raymond. W. Yeung, “Secure Network Coding,” ISIT 2002.

[5] K. Bhattad and K. R. Nayayanan, Weakly secure network coding, Netcod 2005, Italy, April 2005

[6] T. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. R. Karger, “Byzantine Modification Detection in Multicast Networks using Randomized Network Coding”, IEEE International Symposium on Information Theory (ISIT 2004), June 27th-July 2nd, Chicago.

[7] R. W. Yeung, S. -Y. R. Li, N. Cai, and Z. Zhang, “Network Coding Theory,” Foundation and Trends in Communications and Information Theory , vol. 2, nos. 4 and 5, pp. 241 -381, 2005.

[8] Christina Fragouli and Emina Soljanin “Network Coding Fundamentals” Foundation and Trends in Communications and Information Theory , to be published

[9] J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, Amsterdam: North-Holland, 1979.

[10] R. Diestel, Graph Theory, Springer-Verlag, 2000

[11] B. Bollobas, Modern Graph Theory, Springer-Verlag, 2002.

[12] S.-Y. R. Li, R. W. Yeung, and N. Cai. “Linear network coding”. IEEE Transactions on Information Theory , Februray, 2003

[13] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, “Polynomial time algorithms for multicast network code construction,”IEEE Transactions on Information Theory. Submitted July 2003.

[14] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger, “Toward a Random Operation of Networks”, IEEE Transactions on Information Theory, submitted. (2004)

[15] Y. Wu, P. A. Chou, K. Jain, “A comparison of network coding and tree packing,” IEEE International Symposium on Information Theory (ISIT 2004), June 27th-July 2nd, Chicago.

[16] Karatsuba, A. and Ofman, Yu. "Multiplication of Many-Digital Numbers by Automatic Computers." *Doklady Akad. Nauk SSSR* **145**, 293-294, 1962. Translation in *Physics-Doklady* **7**, 595-596, 1963.

[17] Sachin Katti, Shyamnath Gollakota, and Dina Katabi, "Embracing Wireless Interference: Analog Network Coding." MIT Technical Report, 2007.