1 Introduction

Network coding, as a field of study, is young. It was only in 2000 that the seminal paper by Ahlswede, Cai, Li, and Yeung [3], which is generally attributed with the "birth" of network coding, was published. As such, network coding, like many young fields, is characterized by some degree of confusion, of both excitement about its possibilities and skepticism about its potential. Clarifying this confusion is one of the principal aims of this book. Thus, we begin soberly, with a definition of network coding.

1.1 WHAT IS NETWORK CODING?

Defining network coding is not straightforward. There are several definitions that can be and have been used.

In their seminal paper [3], Ahlswede, Cai, Li, and Yeung say that they "refer to coding at a node in a network as *network coding*," where, by coding, they mean an arbitrary, causal mapping from inputs to outputs. This is the most general definition of network coding. But it does not distinguish the study of network coding from network, or multiterminal, information theory – a much older field with a wealth of difficult open problems. Since we do not wish to devote this book to network information theory (good coverage of network information theory already exists, for example, in [27, Chapter 14]), we seek to go further with our definition.

A feature of Ahlswede et al.'s paper that distinguishes it from most network information theory papers is that, rather than looking at general networks where essentially every node has an arbitrary, probabilistic effect on every other node, they look specifically at networks consisting of nodes interconnected by error-free point-to-point links. Thus the network model of Ahlswede et al. is a special case of those ordinarily studied in network information theory, albeit one that is very pertinent to present-day networks – essentially all wireline networks can be cast into their model once the physical layer has been abstracted into error-free conduits for carrying bits.

Another possible definition of network coding, then, is coding at a node in a network with *error-free* links. This distinguishes the function of network coding from that of channel coding for noisy links; we can similarly distinguish the function of network coding from that of source coding by considering the former in the context of independent incompressible source processes. This definition is frequently used and, under it, the study of network coding reduces to a special case of network information theory. This special case was in fact studied well before 2000 (see, for example, [51, 131]), which detracts from some of the novelty of network coding, but we can still go further with our definition.

Much work in network coding has concentrated around a particular form of network coding: *random linear network coding*. Random linear network coding was introduced in [58] as a simple, randomized coding method that maintains "a vector of coefficients for each of the source processes," which is "updated by each coding node." In other words, random linear network coding requires messages being communicated through the network to be accompanied by some degree of extra information – in this case, a vector of coefficients. In today's communications networks, there is a type of network that is widely used, that easily accommodates such extra information, and that, moreover, consists of error-free links: packet networks. With packets, such extra information, or side information, can be placed in packet headers and, certainly, placing side information in packet headers is common practice today (e.g., sequence numbers are often placed in packet headers to keep track of order).

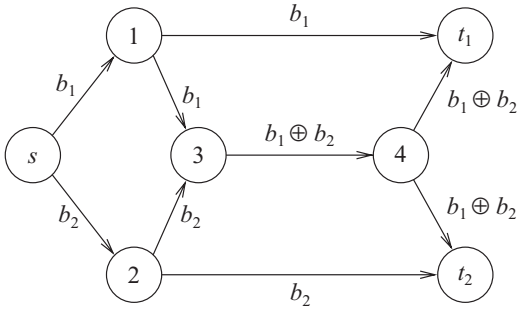
A third definition of network coding, then, is coding at a node in a *packet* network (where data are divided into packets and network coding is applied to the contents of packets), or more generally, coding above the physical layer. This is unlike network information theory, which is generally concerned with coding at the physical layer. We use this definition in this book. Restricting attention to packet networks does, in some cases, limit our scope unnecessarily, and some results with implications beyond packet networks may not be reported as such. Nevertheless, this definition is useful because it grounds our discussion in a concrete setting relevant to practice.

1.2 WHAT IS NETWORK CODING GOOD FOR?

Equipped with a definition, we now proceed to discuss the utility of network coding. Network coding can improve throughput, robustness, complexity, and security. We discuss each of these performance factors in turn.

1.2.1 Throughput

The most well-known utility of network coding – and the easiest to illustrate – is increase of throughput. This throughput benefit is achieved by using packet

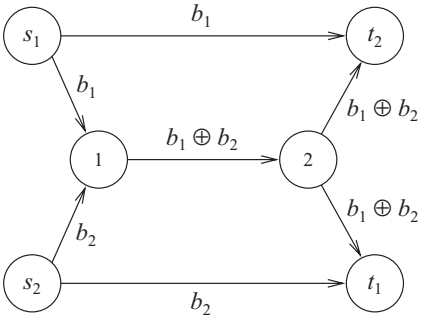


1.1. The butterfly network. In this network, every arc represents a directed link that is capable of carrying a single packet reliably. There are two packets, b_1 and b_2, present at the source node s, and we wish to communicate the contents of these two packets to both of the sink nodes, t_1 and t_2.

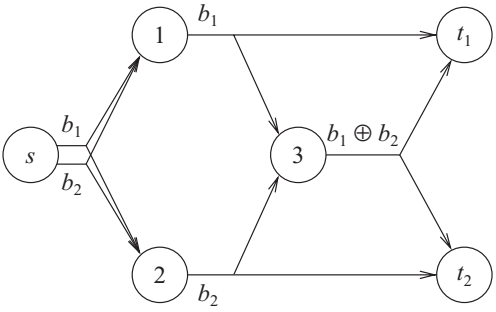
transmissions more efficiently, i.e., by communicating more information with fewer packet transmissions. The most famous example of this benefit was given by Ahlswede et al. [3], who considered the problem of multicast in a wireline network. Their example, which is commonly referred to as the butterfly network (see Figure 1.1), features a multicast from a single source to two sinks, or destinations. Both sinks wish to know, in full, the message at the source node. In the capacitated network that they consider, the desired multicast connection can be established only if one of the intermediate nodes (i.e., a node that is neither source nor sink) breaks from the traditional routing paradigm of packet networks, where intermediate nodes are allowed only to make copies of received packets for output, and performs a coding operation – it takes two received packets, forms a new packet by taking the binary sum, or XOR, of the two packets, and outputs the resulting packet. Thus, if the contents of the two received packets are the vectors b_1 and b_2, each comprised of bits, then the packet that is output is $b_1 \oplus b_2$, formed from the bitwise XOR of b_1 and b_2. The sinks decode by performing further coding operations on the packets that they each receive. Sink t_1 recovers b_2 by taking the XOR of b_1 and $b_1 \oplus b_2$, and likewise sink t_2 recovers b_1 by taking the XOR of b_2 and $b_1 \oplus b_2$. Under routing, we could communicate, for example, b_1 and b_2 to t_1, but we would then only be able to communicate one of b_1 or b_2 to t_2.

The butterfly network, while contrived, illustrates an important point: that network coding can increase throughput for multicast in a wireline network. The nine packet transmissions that are used in the butterfly network communicate the contents of two packets. Without coding, these nine transmissions cannot be used to communicate as much information, and they must be supplemented with additional transmissions (for example, an additional transmission from node 3 to node 4).

While network coding can increase throughput for multicast in a wireline network, its throughput benefits are not limited to multicast or to wireline networks. A simple modification of the butterfly network leads to an example that involves two unicast connections that, with coding, can be established and, without coding, cannot (see Figure 1.2). This example involves *two* unicast connections. For unicast in the lossless wireline networks that have been



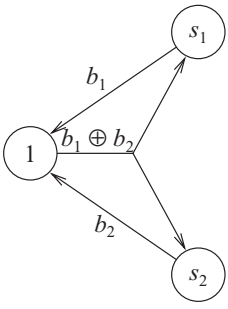
1.2. The modified butterfly network. In this network, every arc represents a directed link that is capable of carrying a single packet reliably. There is one packet b_1 present at source node s_1 that we wish to communicate to sink node t_1 and one packet b_2 present at source node s_2 that we wish to communicate to sink node t_2.



1.3. The wireless butterfly network. In this network, every hyperarc represents a directed link that is capable of carrying a single packet reliably to one or more nodes. There are two packets, b_1 and b_2, present at the source node s, and we wish to communicate the contents of these two packets to both of the sink nodes, t_1 and t_2.

considered so far, a minimum of two unicast connections is necessary for there to be a throughput gain from network coding. As we establish more concretely in Section 2.3, network coding yields no throughput advantage over routing for a *single* unicast connection in a lossless wireline network.

Network coding can also be extended to wireless networks and, in wireless networks, it becomes even easier to find examples where network coding yields a throughput advantage over routing. Indeed, the wireless counterparts of the butterfly network (Figure 1.3) and the modified butterfly network (Figure 1.4) involve fewer nodes – six and three nodes, respectively, as opposed to seven and six. As before, these examples show instances where the desired communication objective is not achievable using routing, but is achievable using coding. These wireless examples differ in that, rather than assuming that packet transmissions are from a single node to a single other node, they allow for packet transmissions



1.4. The modified wireless butterfly network. In this network, every hyperarc represents a directed link that is capable of carrying a single packet reliably to one or more nodes. There is one packet b_1 present at source node s_1 that we wish to communicate to node s_2 and one packet b_2 present at source node s_2 that we wish to communicate to node s_1.

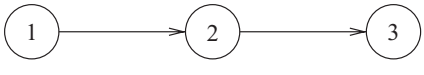
to originate at a single node and end at more than one node. Thus, rather than representing transmissions with arcs, we use *hyperarcs* – generalizations of arcs that may have more than one end node.

The examples that we have discussed so far demonstrate that, even in the absence of losses and errors, network coding can yield a throughput advantage when it is applied either to one or more simultaneous multicast connections or two or more simultaneous unicast connections. This is true both when packets are transmitted only from a single node to a single other node (wireline networks) and when they are transmitted from a single node to one or more other nodes (wireless networks). These examples are, however, seemingly contrived, toy examples, and it is natural to wonder whether network coding can be generalized and, if so, to what end. Much of the remainder of this book will be devoted to generalizing the observations made thus far on network coding to more general settings.

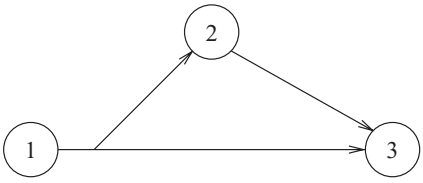
1.2.2 Robustness

1.2.2.1 Robustness to packet losses. But before we proceed, we address an important issue in packet networks, particularly wireless packet networks, that we have thus far neglected: packet loss. Packet loss arises for various reasons in networks, which include buffer overflow, link outage, and collision. There are a number of ways to deal with such losses. Perhaps the most straightforward, which is the mechanism used by the transmission control protocol (TCP), is to set up a system of acknowledgments, where packets received by the sink are acknowledged by a message sent back to the source and, if the source does not receive the acknowledgment for a particular packet, it retransmits the packet. An alternative method that is sometimes used is channel coding or, more specifically, *erasure coding*. An erasure code, applied by the source node, introduces a degree of redundancy to the packets so that the message can be recovered even if only a subset of the packets sent by the source are received by the sink.

Erasure coding is coding applied by the source node. What about coding applied by intermediate nodes? That is, what about network coding? Is network coding useful in combating against packet losses? It is; and the reason it is can be seen with a very simple example. Consider the simple, two-link tandem network shown in Figure 1.5. In this network, packets are lost on the link joining nodes 1 and 2 with probability ε_{12} and on the link joining nodes 2 and 3 with probability ε_{23}. An erasure code, applied at node 1, allows us to communicate information at a rate of $(1 - \varepsilon_{12})(1 - \varepsilon_{23})$ packets per unit time. Essentially



1.5. Two-link tandem network. Nodes 1 and 2 are each capable of injecting a single packet per unit time on their respective outgoing links.



1.6. The packet relay channel. Nodes 1 and 2 are each capable of injecting a single packet per unit time on their respective outgoing links.

we have, between nodes 1 and 3, an erasure channel with erasure probability $1-(1-\varepsilon_{12})(1-\varepsilon_{23})$, whose capacity, of $(1-\varepsilon_{12})(1-\varepsilon_{23})$, can be achieved (or approached) with a suitably designed code. But the true capacity of the system is greater. If we apply an erasure code over the link joining nodes 1 and 2 and another over the link joining nodes 2 and 3, i.e., if we use two stages of erasure coding with full decoding and re-encoding at node 2, then we can communicate information between nodes 1 and 2 at a rate of $1-\varepsilon_{12}$ packets per unit time and between nodes 2 and 3 at a rate of $1-\varepsilon_{23}$ packets per unit time. Thus, we can communicate information between nodes 1 and 3 at a rate of $\min(1-\varepsilon_{12}, 1-\varepsilon_{23})$, which is in general greater than $(1-\varepsilon_{12})(1-\varepsilon_{23})$.

So why isn't this solution used in packet networks? A key reason is delay. Each stage of erasure coding, whether it uses a block code or a convolutional code, incurs some degree of delay because the decoder of each stage needs to receive some number of packets before decoding can begin. Thus, if erasure coding is applied over every link or connection, the total delay would be large. But applying extra stages of erasure coding is simply a special form of network coding – it is coding applied at intermediate nodes. Thus, network coding can be used to provide robustness against packet losses, which can be translated into throughput gains. But what we want from a network coding solution is not only increased throughput: – we want a solution that goes beyond merely applying additional stages of erasure coding – we want a network coding scheme that applies additional coding at intermediate code *without decoding*. In Chapter 4, we discuss how random linear network coding satisfies the requirements of such a coding scheme.

Losses add an additional dimension to network coding problems and, when losses are present, even a single unicast connection suffices for gains to be observed. Losses are very pertinent to wireless networks, and considering losses makes network coding more relevant to wireless applications. Another characteristic of wireless networks that we have discussed is the presence of broadcast links – links that reach more than one end node – and we have yet to combine losses and broadcast links.

In Figure 1.6, we show a modification of the two-link tandem network that we call the packet relay channel. Here, the link coming out of node 1 doesn't only reach node 2, but also reaches node 3. Because of packet loss, however, whether a packet transmitted by node 1 is received by neither node 2 nor node 3, by node 2 only, by node 3 only, or by both nodes 2 and 3 is determined

probabilistically. Let's say packets transmitted by node 1 are received by node 2 only with probability $p_{1(23)2}$, by node 3 only with probability $p_{1(23)3}$, and by both nodes 2 and 3 with probability $p_{1(23)(23)}$) (they are lost entirely with probability $1 - p_{1(23)2} - p_{1(23)3} - p_{1(23)(23)}$). As for packets transmitted by node 2, let's say packets transmitted by node 2 are received by node 3 with probability p_{233} (they are lost entirely with probability $1 - p_{233}$). Network coding, in particular random linear network coding, allows for the maximum achievable throughput in such a setup, known as the min-cut capacity, to be reached, which in this case is $\min(p_{1(23)2} + p_{1(23)3} + p_{1(23)(23)}, p_{1(23)3} + p_{1(23)(23)} + p_{233})$.

This is no mean feat: first, from the standpoint of network information theory, it is not even clear that there would exist a simple, capacity-achieving network code and, second, it represents a significant shift from the prevailing approach to wireless packet networks. The prevailing, routing approach advocates treating wireless packet networks as an extension of wireline packet networks. Thus, it advocates sending information along routes; in this case, either sending information from node 1 to node 2, then to node 3, or directly from node 1 to node 3, or, in more sophisticated schemes, using a combination of the two. With network coding, there are no paths as such – nodes contribute transmissions to a particular connection, but these nodes do not necessarily fall along a path. Hence a rethink of routing is necessary. This rethink results in subgraph selection, which we examine in Chapter 5.

1.2.2.2 Robustness to link failures. Besides robustness against random packet losses, network coding is also useful for protection from non-ergodic link failures. Live path protection, where a primary and a backup flow are transmitted for each connection, allows very fast recovery from link failures, since rerouting is not required. However, this approach doubles the amount of network traffic. By allowing sharing of network resources among different flows, network coding can improve resource usage. For an individual multicast session, there exists, for any set of failure patterns from which recovery is possible with arbitrary rerouting, a static network coding solution that allows recovery from any failure pattern in the set without rerouting [82].

1.2.3 Complexity

In some cases, although optimal routing may be able to achieve similar performance to that of network coding, the optimal routing solution is difficult to obtain. For instance, minimum-cost subgraph selection for multicast routing involves Steiner trees, which is complex even in a centralized setting, while the corresponding problem with network coding is a linear optimization that admits low-complexity distributed solutions. This is discussed further in Section 5.1.1.

Network coding has also been shown to substantially improve performance in settings where practical limitations necessitate suboptimal solutions,

e.g., gossip-based data dissemination [32] and 802.11 wireless ad hoc networking [74], which is discussed in Section 3.5.2.2.

1.2.4 Security

From a security standpoint, network coding can offer both benefits and drawbacks. Consider again the butterfly network (Figure 1.1). Suppose an adversary manages to obtain only the packet $b_1 \oplus b_2$. With the packet $b_1 \oplus b_2$ alone, the adversary cannot obtain either b_1 or b_2; thus we have a possible mechanism for secure communication. In this instance, network coding offers a security benefit.

Alternatively, suppose that node 3 is a malicious node that does not send out $b_1 \oplus b_2$, but rather a packet masquerading as $b_1 \oplus b_2$. Because packets are coded rather than routed, such tampering of packets is more difficult to detect. In this instance, network coding results in a potential security drawback. We discuss the security implications of network coding in Chapter 6.

We have now given a number of toy examples illustrating some benefits of network coding. That these examples bear some relevance to packet networks should be evident; exactly how the principles they illustrate can be exploited in actual settings is perhaps not. We address more general cases using the model that we put forth in the following section.

1.3 NETWORK MODEL

Packet networks, especially wireless packet networks, are immensely complex and, as such, difficult to accurately model. Moreover, network coding is used in such a wide variety of contexts that it is not sensible to always use the same model. Nevertheless, there are common aspects to all of the models that we employ, which we now discuss. The specific aspects of the various models we use are discussed as we encounter them.

As a starting point for our model, we assume that there are a number of *connections*, or *sessions*, that we wish to establish. These connections may be unicast (with a single source node and a single sink node) or multicast (with a single source node and more than one sink node). In a multicast connection, all of the sink nodes wish to know the same message originating from the source node. These connections are associated with packets that we wish to communicate at rates that may or may not be known. Thus, our model ignores congestion control, i.e., our model does not consider having to regulate the rates of connections. We consider congestion control to be a separate problem that is not covered in this book, but is readily incorporated into the model (see e.g. [25, 135]).

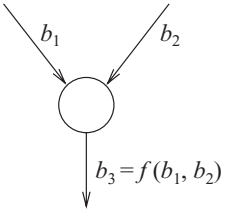
We represent the topology of the network with a directed hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of hyperarcs. A hypergraph

is a generalization of a graph, where, rather than arcs, we have hyperarcs. A hyperarc is a pair (i, J), where i, the start node, is an element of $\mathcal{N}$ and J, the set of end nodes, is a non-empty subset of $\mathcal{N}$. Each hyperarc (i, J) represents a broadcast link from node i to nodes in the non-empty set J. In the special case where J consists of a single element j, we have a point-to-point link. The hyperarc is now a simple arc and we sometimes write (i, j) instead of $(i, \{j\})$. If the network consists only of point-to-point links (as in a wireline network), then $\mathcal{H}$ is a graph, denoted alternatively as $\mathcal{G}$ rather than $\mathcal{H}$. The link represented by hyperarc (i, J) may be lossless or lossy, i.e., it may or may not be subject to packet erasures.

To establish the desired connection or connections, packets are injected on hyperarcs. Let z_{iJ} be the average rate at which packets are injected on hyperarc (i, J). The vector z, consisting of z_{iJ}, $(i, J) \in \mathcal{A}$, defines the rate at which packets are injected on all hyperarcs in the network. In this abstraction, we have not explicitly accounted for any queues. We assume that queueing occurs at a level that is hidden from our abstraction and, provided that z lies within some constraint set Z, all queues in the network are stable. In wireline networks, links are usually independent, and the constraint set Z decomposes as the Cartesian product of $|\mathcal{A}|$ constraints. In wireless networks, links are generally dependent and the form of Z may be complicated (see, for example, [28, 72, 73, 79, 137, 141]). For the time being, we make no assumptions about Z except that it is a convex subset of the positive orthant and that it contains the origin.

The pair $(\mathcal{H}, Z)$ defines a capacitated graph that represents the network at our disposal, which may be a full, physical network or a subnetwork of a physical network. The vector z, then, can be thought of as a subset of this capacitated graph – it is the portion actually under use – and we call it the *coding subgraph* for the desired connection or connections. We assume that the coding subgraph defines not only the rates of packet injections on hyperarcs, but also the specific times at which these injections take place. Thus, the classical networking problems of routing and scheduling are special subproblems of the problem of selecting a coding subgraph.

The examples discussed in the previous section give instances of coding subgraphs, instances where packet injections have been chosen, and the task that remains is to use them as effectively as possible. Perhaps the simplest way of representing a coding subgraph in a lossless network is to represent each packet transmission over some time period as a separate hyperarc, as we have done in Figures 1.1–1.4. We may have parallel hyperarcs as we do in Figure 1.3 (where there are two hyperarcs $(s, \{1, 2\})$), representing multiple packets transmitted and received by the same nodes in a single time period. Coding at a node is shown in Figure 1.7. We call this representation of a subgraph a *static subgraph*. In a static subgraph, time is not represented explicitly, and it appears as though events occur instantaneously. Presumably in reality, there is some delay involved

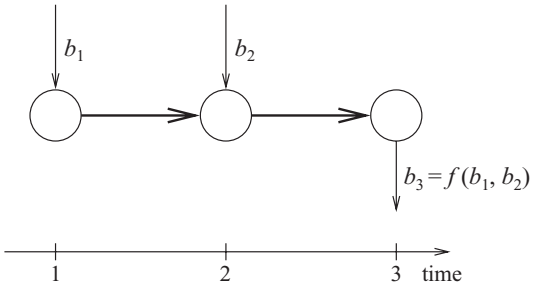


1.7. Coding at a node in a static subgraph. Two packets, b_1 and b_2, are each carried by one of the incoming arcs. The outgoing arc carries a packet that is a function of b_1 and b_2.

in transmitting packets along links, so the output of packets on a link is delayed from their input. Thus, static subgraphs hide some timing details that, though not difficult to resolve, must be kept in mind. Moreover, we must restrict our attention to acyclic graphs, because cyclic graphs lead to the instantaneous feedback of packets. Despite its limitations, static subgraphs suffice for much that we wish to discuss, and they will be used more or less exclusively in Chapter 2, where we deal with lossless networks.

For lossy networks, the issue of time becomes much more important. The network codes that are used for lossy networks are generally much longer than those for lossless networks, i.e., one coding block involves many more source packets. Looked at another way, the time period that must be considered for a network code in a lossy network is much longer than that in a lossless network. Hence, it becomes imperative to examine the interplay of coding and time at a coding node. To do this, we extend static subgraphs to *time-expanded subgraphs*.

A time-expanded subgraph represents not only the injection and reception points of packets, but also the times at which these injections and receptions take place. We draw only successful receptions; hence, in a lossy network, a time-expanded subgraph in fact represents a particular element in the random ensemble of a coding subgraph. Suppose for example that, in Figure 1.7, packet b_1 is received at time 1, packet b_2 is received at time 2, and packet b_3 is injected at time 3. In a time-expanded subgraph, we represent these injections and receptions as shown in Figure 1.8. In this example, we have used integral values of time, but real values of time can just as easily be used. We now have multiple instances of the same node, with each instance representing the node at a different time. Joining these instances are infinite capacity links that



1.8. Coding at a node in a time-expanded subgraph. Packet b_1 is received at time 1, and packet b_2 is received at time 2. The thick, horizontal arcs have infinite capacity and represent data stored at a node. Thus, at time 3, packets b_1 and b_2 can be used to form packet b_3.