Fast Resource Allocation for Network-Coded Traffic — A Coded-Feedback Approach

Chih-Chun Wang and Xiaojun Lin
School of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN 47907, U.S.A.
Email: {chihw,linx}@purdue.edu.

Abstract

In this paper, we develop a fast resource allocation algorithm that takes advantage of intra-session network coding. The algorithm maximizes the total utility of multiple unicast (or multicast) sessions subject to capacity constraints, where packets are coded within each session. Our solution is a primal solution that does not use duality or congestion prices. Thus, it does not require building up queues to achieve the optimal resource allocation. Hence the queueing delay of the packets can be tightly controlled. The existing primal solution in the literature requires a separate graph-theoretic algorithm to find the min-cut of each session, whose complexity grows quadratically with the total number of nodes. In contrast, we provide a new *coded-feedback* approach whose complexity grows only linearly with the total number of nodes. More explicitly, by *letting the ACK/feedback packets on the return paths also carry coding coefficients* as does the forward coded traffic, key network information can be obtained more efficiently, which leads to a fast resource allocation scheme fully integrated with the network coding operation.

1 Introduction

In this paper, we are interested in developing fast resource allocation algorithms that take advantage of intra-session network coding. For multicast sessions, intra-session network coding has been found to offer two benefits over routing. First, the achievable throughput with coding is often larger than that with routing [1]. Further, the maximum throughput can be achieved with high probability by using simple random linear network codes [2]. Second, the resource allocation problem with coding can be modeled as a convex optimization problem that is easy to solve. In contrast, finding the optimal multicast tree with routing is often NP-hard [3, 4].

Although network coding leads to considerable performance improvement for multicast sessions, the majority of the traffic in today's Internet is by unicast. Intra-session network coding does not offer throughput advantage over routing for unicast. Nonetheless, we argue in this paper that, even for unicast traffic, intra-session network coding offers an attractive alternative to routing. We use the following simple and motivational example. Consider the case when there is only a single unicast session on a network (e.g., on a community LAN or a Virtual Private

Network). It is well-known that the maximum throughput of the session is equal to the *min-cut max-flow* (**MCMF**) rate between its source and destination. Traditional Internet protocols often use only a single path. As a result, they are unlikely to attain the min-cut max-flow. Thus, it would be more desirable to use multi-path control protocols. Without network coding, there are two approaches available in the literature to achieve the optimal min-cut max-flow rate. First, we can use a standard graph-theoretic algorithm, such as the push-&-relabel algorithm [5], to find the *max-flow* value of the network, along with the paths that non-coded traffic should be sent. Such graph-theoretic algorithms typically need to be executed separately from the network traffic, and they require additional communication and control overhead. Second, we can apply an adaptive control algorithm, such as the back-pressure algorithm in [6] or the multi-path control algorithm in [7], to achieve the optimal throughput for this single-session case. Both of these algorithms need to build up queues in order to achieve the optimal throughput. Such queues lead to increase in packet delay, which can be difficult to predict and control.

In contrast, network coding provides a surprisingly simple solution to achieve the optimal rate of this single unicast problem. Each intermediate router simply mixes the incoming packets and sends the coded packets to its neighbors. Then, with close-to-one probability, the number of independent packets received by the destination will be equal to the min-cut max-flow value. With simple *precoding* at the source, the destination can then *decode* the packets at near-optimal rate. The advantage of network coding is apparent: The algorithm is simple and it converges instantaneously to the optimal rate (when there is only a single session). Further, *there is no queue build-up and hence the packet delay can be tightly controlled.* Third, the above scheme extends naturally to a single multicast session.

In real networks, multiple unicast (or multicast) sessions must share the network resources. Existing resource allocation algorithms are devised separately from the network coding operations and can be classified into two categories: primal solutions and dual solutions. In a typical dual solution, the queue at each link is interpreted as a price signal, which indicates the level of imbalance between the offered-load and the capacity. The dual solution then updates the resource allocation based on this price signal [3, 8–11]. One of the main disadvantage of dual solutions is that there can be a disconnection between the optimality of the dual variables (i.e., the price) and the optimality of the primal variables (i.e., the resource allocation). First, even if the the price converges, the rate allocation may not converge. An additional step (e.g., using proximal algorithms [9]) is often required to enforce the convergence of the primal variables, which increases the complexity of the solution. Second, before the price converges, the system utility does not necessarily improve in each iteration. In fact, before the algorithm converges, the rate allocation often violates the capacity constraints, which leads to queue-length dynamics that are difficult to predict and control.

In contrast, primal solutions adjust the primal control variables (i.e., the rate allocation) directly within the capacity constraints [12, 13]. Hence, the resulted rate allocation typically improves the system utility (towards the optimal) over each iteration. Further, even before the algorithm converges, the rate-allocation always satisfies the capacity constraints, and hence there is no queue build-up.

However, a key step in the existing primal solution with network coding [12] is to use a separate graph-theoretic algorithm to find the min-cut of the network. To the best of our knowledge, the time-complexity of existing min-cut algorithms in the literature is at least $\mathcal{O}(|V|^2)$, which significantly increases the overall complexity of the solution. In this paper, we develop a much

faster resource allocation algorithm than that of [12]. Specifically, by exploiting and refining a novel concept of *coded feedback* (first introduced in [14,15]), we develop a new and fully distributed algorithm that only takes $\mathcal{O}(|V|)$ time to compute the min-cut. This is an order of degree faster than existing min-cut algorithms[1] (including the $\mathcal{O}(|V|^2)$ coded-feedback min-cut/max-flow algorithms devised in [14, 15]). Our coded-feedback approach computes the min-cut as an integral part of network-coding operation, by coding the feedback packets on the return path. By developing low-complexity and distributed resource allocation schemes working in conjunction with the ultra-efficient coded-feedback algorithm, our solution enjoys the benefits of both primal solutions and the simplicity of network coding with minimal control and communication overhead.

The contributions of this paper are summarized as follows.

- We develop a fast resource allocation algorithm that can quickly allocate the full network capacity and maximize the total network utility for multiple sessions.
- As a key step in the algorithm, we use the novel concept of coded-feedback to find the min-cut of each session. Not only does the coded-feedback approach require lower complexity than the graph-theoretic algorithms used in related works [12], it is also fully integrated with the network coding operation.
- Even for unicast traffic, our algorithm can serve as an attractive alternative to traditional non-coded solutions, such as the back-pressure algorithm [6]. Our algorithm has comparable convergence speed and overhead as the back-pressure algorithm. On the other hand, since our solution does not require the build-up of queues as price signals, the queue-length and the packet delay can be tightly controlled.

2 System Model & The Overall Concept

2.1 System Model

We represent a network by a graph $G = (V, L)$, where V is the set of nodes and L is the set of directed links. We assume that each directed link $l = 1, ..., |L|$ has a fixed capacity R^l. For the sake of simplicity and ease of exposition, we assume that there is a set I of unicast sessions that share the resources in such a network. (In Section 6, we will discuss how the problem formulation and the solution can also be readily extended to the case with multicast sessions.) For each session $i = 1, ..., |I|$, let s_i and d_i denote its source node and destination node, respectively. Let r_i^l denote the capacity of link l that is allocated to session i. Let $\vec{r}_i = [r_i^l, l = 1, ..., |L|]$. From session i's point of view, its packets are transmitted on a sub-network with the same topology $G = (V, L)$ except that the capacity of each link is r_i^l. In addition, to simplify the description of the network coding operation, we assume that each session i only uses an acyclic subgraph $G_i = (V, L_i)$ of G, where the link subset L_i forms an acyclic graph. (we will discuss in Section 6 how to extend the results to the case when sessions do not need to choose these acyclic graphs.) Define a cut C_i for session i to be a subset of links from L_i such that when these links are removed from the

[1]Throughout this paper, we ignore the computation time within a network node because we are mostly interested in the speed of distributed algorithms. Thus, the time-complexity discussion is based on the delay caused by exchanging control messages rather than caused by computation within a node.

subgraph G_i, the source s_i and the destination d_i are disconnected. Given $\vec{r}_i$, define the value $\xi_i(C_i|\vec{r}_i)$ of the cut C_i as the total rate (allocated to session i) of the links that belong to C_i. Let $\hat{\mathcal{C}}_i$ denote the collection of all cuts for session i. Define a minimum-cut (or min-cut) $C_i^{\min}$ for session i as the cut with the smallest value, i.e.,

$$\xi_i(C_i^{\min}|\vec{r}_i) = \min_{C_i \in \hat{\mathcal{C}}_i} \xi_i(C_i|\vec{r}_i). \tag{1}$$

This value $\xi_i(C_i^{\min}|\vec{r}_i)$, which we subsequently denote as $\mathbf{MCMF}_i(\vec{r}_i)$, is also the maximum rate that session i can transfer packets from source s_i to destination d_i, and hence is known as the min-cut max-flow value of session i.

Assume that each session has a utility function $U_i(x_i)$ that is strictly concave and non-decreasing. The utility function $U_i(x_i)$ characterizes the satisfactory level of the user of session i when the service rate that it receives is x_i. We assume that the derivative of $U_i(\cdot)$ is bounded by M_0 for all i. Let $\vec{r} = [\vec{r}_i, i = 1, ..., |I|]$. We are interested in the following utility-maximization problem subject to capacity constraints:

$$\begin{aligned} \max_{\vec{r}=[\vec{r}_i]\geq 0} \quad & \sum_{i=1}^{|I|} U_i(\mathbf{MCMF}_i(\vec{r}_i)) \\ \text{subject to} \quad & \sum_{i=1}^{|I|} r_i^l \leq R^l \text{ for all links } l. \end{aligned} \tag{2}$$

This problem formulation is similar to the one proposed in [12].[2] Next, we first present a high-level overview of our solution methodology, which also has some similarity to the solution of [12]. We then highlight the main step where our solution differs from [12].

2.2 The Overall Solution Methodology

One can show that the objective function in (2) is concave, and hence Problem (2) is a convex optimization problem [12]. We can therefore use a subgradient-ascent method to solve it.[3] Let $f(\vec{x})$ be any concave function defined on a convex set. A vector $\partial f(\vec{x}_0)$ is called a *subgradient* of the function $f(\vec{x})$ at $\vec{x}_0$ if the following holds:

$$f(\vec{x}) - f(\vec{x}_0) \leq \partial f(\vec{x}_0) \cdot (\vec{x} - \vec{x}_0) \text{ for all } \vec{x}. \tag{3}$$

In [12], the authors show that a subgradient of the min-cut max-flow function $\mathbf{MCMF}_i(\vec{r}_i)$ is given by the min-cut. More precisely, with a given $\vec{r}_i$, consider any min-cut $C_i^{\min}$ for session i. Define a vector $\vec{\mu}_i = [\mu_i^l, l \in L]$ such that

$$\mu_i^l = \begin{cases} 1 & \text{if } l \in C_i^{\min} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

[2]For multicast sessions, we can still use the problem formulation (2) by replacing $\mathbf{MCMF}_i(\vec{r}_i)$ with the minimum of the min-cut max-flow value between the source s_i and each destination of a multicast session i [12].

[3]Since the $\mathbf{MCMF}_i(\vec{r}_i)$ is a piecewise-linear function of $\vec{r}_i$, the gradient may not exist everywhere, which is why we use the subgradients.

Then one can show that this vector $\vec{\mu}_i$ is a subgradient of $\mathbf{MCMF}_i(\cdot)$ at $\vec{r}_i$ [12].

Denote the objective function of (2) by

$$F(\vec{r}) = \sum_{i=1}^{|I|} U_i(\mathbf{MCMF}_i(\vec{r}_i)).$$

We can then construct a subgradient $\partial F(\vec{r})$ of $F(\vec{r})$ such that its (i,l)-th component is given by [12]:

$$[\partial F(\vec{r})]_{il} = U_i'(\mathbf{MCMF}_i(\vec{r}_i))\mu_i^l, \tag{5}$$

where $U_i'(\cdot)$ is the derivative of the utility function $U_i(\cdot)$. Let Λ denote the set of $\vec{r}$ that satisfies that capacity constraints, i.e., $\vec{r} \geq 0$, and $\sum_{i=1}^{|I|} r_i^l \leq R^l$ for all links l. We can then use the following constrained subgradient-ascent algorithm to solve (2). Let $\vec{r}(t)$ denote the resource allocation at iteration t, and let $\partial F(\vec{r}(t))$ denote the corresponding subgradient (5) of the objective function at $\vec{r}(t)$. The subgradient-ascent iteration is then given by:

$$\vec{r}(t+1) = [\vec{r}(t) + \alpha \partial F(\vec{r}(t))]_\Lambda, \tag{6}$$

where α is a positive step-size and $[\cdot]_\Lambda$ denotes the projection to the constraint set Λ. Like [12], the above solution is a primal solution, and hence it has the desirable features of primal solutions as discussed in Section 1.

Remark: Since the capacity constraint on each link is independent, the projection operation $[\cdot]_\Lambda$ can easily be carried out independently at each link.

A critical step in the above solution is to find a min-cut $C_i^{\min}$ for each session i. The authors of [12] use the graph-theoretic push-&-relabel (PNR) algorithm [5] to find the min-cut. Although PNR can be implemented in a distributed fashion, it must be executed separately from the network coding operation (see [14] for detailed discussion). Further, even the fully parallel, distributed implementation of PNR still requires $\mathcal{O}(|V|^2)$ communication time, which could significantly slow down the overall convergence. In this paper, we take a different approach based on the concept of coded-feedback, which computes the min-cut in $\mathcal{O}(|V|)$ time. In addition, a highly-desirable feature of our new approach is that the computation is tightly integrated with the network coding operations. Hence, it incurs minimum overhead and is easy to implement.

3 Finding Min-Cut Using Coded Feedback

3.1 A Unit-Capacity Network Model of Network Coding

Network coding operations are performed on a packet-by-packet basis. To better describe the coding operations among different packets, we adopt the following unit-capacity network model, commonly used in the network coding literature, to describe the new coded-feedback-based min-cut algorithm.

For the following, we focus on a given session i and consider a directed acyclic subnetwork[4] $G_i = (V, E_i)$ used by session i. For simplicity, we drop the subscript i whenever there is no

[4]For directed cyclic networks with transmission delay, the time-axis naturally decouples the network into its acyclic counterpart as first illustrated in [16]. More explicitly, by regarding the sequence number of each "packet" as the corresponding topological order of each "unit-capacity edge", any directed cyclic network is converted automatically to the above unit-capacity acyclic network coding model.

source of confusion. We assume that each edge $e \in E$ is of unit-capacity, i.e., it is able to send one packet per unit time. High-capacity links are modelled by multiple parallel edges. We also assume that the delay for each edge is one unit time. Long-delay links are modelled by multi-hop paths with added auxiliary nodes. Let s and d denote the source and destination respectively. By converting links with different capacities to unit-capacity edges, this model characterizes the packet-by-packet behavior of network coding.

Without loss of generality, we may further assume s being the most upstream node and d being the most downstream node. We sometimes label nodes according to their corresponding topological order (i.e. $s = 1$ and $d = |V|$). For the following, we use $\text{In}(v) \subseteq E$ and $\text{Out}(v) \subseteq E$ to denote all edges entering/leaving node v. For notational simplicity, we use finite Galois field $\text{GF}(2^b)$. The new algorithm works for finite fields of any size, including GF(3) used in the illustrative example that follows.

3.2 Algorithmic Description of Network Coding

Following the generation-based practical network coding protocols [17], each packet along a unit-capacity edge e carries an n-dimensional *global coding vector* $m_e = (m_1, \cdots, m_n)$ in its header, such that the coded symbol $X_e \in \text{GF}(2^b)$ in the payload is

$$X_e = m_1 X_1 + \cdots + m_n X_n \tag{7}$$

where X_1 to X_n are the information symbols (also known as non-coded/native symbols). We assume each coding vector m_e being a row vector. For any subset $E' \subseteq E$ of edges, we use the notation $[m_e : e \in E']$ as an $|E'| \times n$ matrix constructed by concatenating $|E'|$ row vectors m_e vertically.

The network coding (linear mixing) operations performed at each intermediate node consist of two sub-routines:

§ Normal Network Coding

1: Run Choose Random Mixing Coefficients
2: Run Compute Coded Traffic

The description of the sub-routine Choose Random Mixing Coefficients is as follows.

§ Choose Random Mixing Coefficients

1: Each node $v \in V \backslash \{s, d\}$ chooses randomly an $|\text{Out}(v)| \times |\text{In}(v)|$ matrix $\Gamma(v) = [\gamma_{w,u}(v)]$ where each entry $\gamma_{w,u}(v)$ is chosen *independently* and *uniformly randomly* from $\text{GF}(2^b)$ for all $(v, w) \in \text{Out}(v)$ and $(u, v) \in \text{In}(v)$.

The second sub-routine Compute Coded Traffic is computed from the most upstream node s toward the most downstream node d.

§ Compute Coded Traffic

1: Source s sends out randomly coded symbols according to (7) along $\text{Out}(s)$, where the coding vector m_e along $e \in \text{Out}(s)$ is chosen uniformly randomly.
2: **for** $v = s + 1, \cdots, d - 1$ **do**

3: $[m_e : e \in \text{Out}(v)] \leftarrow \Gamma(v)[m_{e'} : e' \in \text{In}(v)]$.
4: Node v sends out coded symbols X_e according to (7) along all edges in $\text{Out}(v)$.
5: **end for**

As illustrated in Line 3 of COMPUTE CODED TRAFFIC, matrix $\Gamma(v)$ chosen in CHOOSE RANDOM MIXING COEFFICIENTS is the mixing/transfer coefficients within node v. By choosing $\Gamma(v)$ and $[m_e : e \in \text{Out}(s)]$ randomly, the above two sub-routines are simply an algorithmic way of describing random packet mixing and broadcasting in a practical network coding scheme [17]. It is worth emphasizing that no special hardware is required for the first two subroutines except the standard network-coding capable hardware.

Next, we introduce the novel concept of *coded feedback*, and show how it can identify a min-cut using similar simple coding operations.

3.3 A Coded-Feedback Min-Cut Algorithm

The novel component of the new min-cut algorithm is *the coded feedback message* q_e, which is an n-dimensional row vector sent in the opposite direction of m_e. Namely, for any $e = (u, v) \in \text{In}(v)$, its coded feedback vector q_e is a linear combination of $\{q_{e'} : e' \in \text{Out}(v)\}$. In practice, the coded feedback vector q_e can also be embedded in the header of the acknowledgement packet in a similar way as the coding coefficients m_e for the forward coded traffic. A detailed description of the min-cut algorithm is as follows. (Subsequently, the following algorithm will be referred as the **Main Algorithm**.)

§ **A New Min-Cut Algorithm — The Main Algorithm**
1: Run CHOOSE RANDOM MIXING COEFFICIENTS
2: Run COMPUTE CODED TRAFFIC
3: Run COMPUTE CODED FEEDBACK
4: **return** $\{e \in E : m_e q_e^{\mathrm{T}} = 1\}$

The first two sub-routines are exactly the same as those in NORMAL NETWORK CODING. The only new sub-routine COMPUTE CODED FEEDBACK is described as follows. Let $\text{Rank}(d)$ denote the rank of the vectors $[m_e : e \in \text{In}(d)]$ received by destination d.

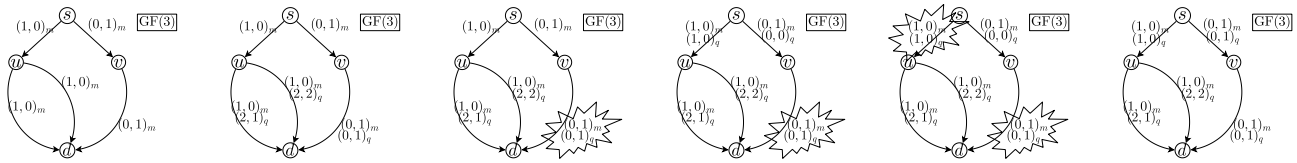
§ COMPUTE CODED FEEDBACK

Starting from the most downstream node d and proceeding back to the most upstream node s, compute the coded feedback vectors as follows.

1: Destination d *randomly* constructs $(n - \text{Rank}(d))$ vectors $\{m_a^* : a \in \{1, \cdots, n - \text{Rank}(d)\}\}$ such that jointly $\{m_e : e \in \text{In}(d)\}$ and $\{m_a^*\}$ span the entire n-dimensional vector space.
2: Destination d *randomly*[5] constructs two sets of vectors $\{q_e : e \in \text{In}(d)\}$ and $\{q_a^* : a \in$

[5]As in a typical random network algorithms, the randomness in Lines 1 and 2 is needed for computing min-cut correctly with high probability. (Refer to Proposition 2 below.) The detailed "random constructions" used in Lines 1 and 2 are described briefly as follows. For Line 1, d arbitrarily chooses the complementing vectors $[m_a^*]$. Then, d randomly selects an $(n - \text{Rank}(d)) \times |\text{In}(d)|$ matrix Γ_d with each entry uniformly and independently distributed. Replace $[m_a^*]$ by $[m_a^*] + \Gamma_d[m_e : e \in \text{In}(d)]$. In this way, the new $[m_a^*]$ is randomly constructed.

For Line 2, d first identifies $\text{Rank}(d)$ independent m_e in $[m_e : e \in \text{In}(d)]$. Without loss of generality, assume



(a) Normal Network Coding. (b) Construct the initial coded feedback $[q_e : e \in \mathrm{In}(d)]$. (c) Identify the first edge with $m_e q_e^{\mathrm{T}} = 1$. (d) Continue the propagation of coded feedback q_e. (e) Identify the second edge with $m_e q_e^{\mathrm{T}} = 1$. (f) The m_e and q_e under Algo.

Figure 1: An illustration of the new min-cut algorithm.

$\{1, \cdots, n - \mathrm{Rank}(d)\}\}$ such that the following matrices

$$\mathbf{q} \triangleq \begin{pmatrix} [q_e : e \in \mathrm{In}(d)] \\ [q_a^* : a \in \{1, \cdots, n - \mathrm{Rank}(d)\}] \end{pmatrix}$$
$$\mathbf{m} \triangleq \begin{pmatrix} [m_e : e \in \mathrm{In}(d)] \\ [m_a^* : a \in \{1, \cdots, n - \mathrm{Rank}(d)\}] \end{pmatrix}$$

satisfy $\mathbf{q}^{\mathrm{T}}\mathbf{m} = \mathbf{I}_n$ where $\mathbf{I}_n$ is an n by n identity matrix and $\mathbf{q}^{\mathrm{T}}$ is the transpose of $\mathbf{q}$.

3: Destination d sends out the newly constructed $\{q_e : e \in \mathrm{In}(d)\}$ along each edge in $\mathrm{In}(d)$.
4: **for** $v = d - 1, \cdots, s + 1$ (i.e. from the most downstream node to the most upstream node) **do**
5: Construct $\mathbf{q}_v \leftarrow [q_e : e \in \mathrm{Out}(v)]$.
6: For those $e \in \mathrm{Out}(v)$ satisfying $m_e q_e^{\mathrm{T}} = 1$, replace the corresponding rows in $\mathbf{q}_v$ by all-zero vectors.
7: $[q_e : e \in \mathrm{In}(v)] \leftarrow \Gamma(v)^{\mathrm{T}}\mathbf{q}_v$.
8: Node v sends q_e along edges in $\mathrm{In}(v)$.
9: **end for**

Finally, the **Main Algorithm** returns the set of all edges satisfying $m_e q_e^{\mathrm{T}} = 1$.

Most of the computation load in the above subroutine is within destination d, which involves computing $\mathbf{q}$ satisfying $\mathbf{q}^{\mathrm{T}}\mathbf{m} = \mathbf{I}_n$. Since during the *decoding stage* of normal network coding, d has already had to find the inverse of the coding matrix $\mathbf{m}$, the computation of the coded feedback $\mathbf{q}$, the left-inverse of $\mathbf{m}$ can be carried out with little extra cost. For intermediate nodes other than s and d, a straightforward linear mixing is performed in Line 7 of Compute Coded Feedback with *the same mixing matrix* $\Gamma(v)$ *used previously for computing the forward messages.* Therefore, one extra hardware requirement is for each intermediate node v to *memorize* the randomly chosen mixing coefficients $\Gamma(v)$ (used to perform previous coding operations). In addition to the memory requirement, the other extra computation is to check the inner product $m_e q_e^{\mathrm{T}}$ and to reset those q_e with $m_e q_e^{\mathrm{T}} = 1$, which induces little hardward/computation cost. In

that it is the last $\mathrm{Rank}(d)$ rows. Then the $\mathbf{m}$ matrix can be decomposed as an $(|\mathrm{In}(d)| - \mathrm{Rank}(d)) \times n$ matrix $\mathbf{m}_1$ and an $n \times n$ invertible matrix $\mathbf{m}_2$. We then construct two corresponding sub-matrices $\mathbf{q}_1$ and $\mathbf{q}_2$ as follows. Destination d first randomly chooses a $\mathbf{q}_1$ with each entry uniformly and independently distributed. Then, it finds the unique $\mathbf{q}_2$ satisfying $\mathbf{q}_2^{\mathrm{T}} = (\mathbf{I}_n - \mathbf{q}_1^{\mathrm{T}}\mathbf{m}_1)\mathbf{m}_2^{-1}$. The newly found $\mathbf{q}_1$ and $\mathbf{q}_2$ are concatenated to form the desired $\mathbf{q}$. The randomly chosen $\mathbf{q}_1$ will ensure the $\mathbf{q}_2$ and the entire $\mathbf{q}$ matrix being randomly constructed.

summary, the subroutine COMPUTE CODED FEEDBACK is of similar complexity to COMPUTE CODED TRAFFIC and can be performed with little extra cost.

We provide an example of this min-cut algorithm in Fig. 1. Consider a simple network as in Fig. 1(a). Suppose there are $n = 2$ uncoded symbols and GF(3) is used for all linear operations. Fig. 1(a) describes the corresponding normal network coding operations, in which nodes u and v simply forwards the coding vectors $(1, 0)$ and $(0, 1)$, or equivalently the corresponding mixing matrices are $\Gamma(u) = (1, 1)^{\mathrm{T}}$ and $\Gamma(v) = 1$. (Note that the coding vectors are illustrated in the figure with subscript m.) Line 1 of COMPUTE CODED FEEDBACK is redundant in this particular example because $\mathrm{Rank}(d) = 2 = n$ and the received vectors $[m_e : e \in \mathrm{In}(d)]$ themselves are spanning the entire n-dimensional space. The desired $\{m_a^*\}$ is thus empty. Fig. 1(b) illustrates one choice of $[q_e : e \in \mathrm{In}(d)]$ such that

$$[q_e : e \in \mathrm{In}(d)]^{\mathrm{T}}[m_e : e \in \mathrm{In}(d)]$$
$$= \begin{pmatrix} 2 & 2 & 0 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbf{I}_2 \text{ in GF(3)}$$

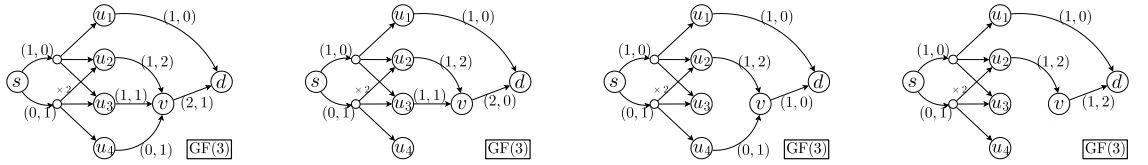
satisfying Line 2 of COMPUTE CODED FEEDBACK. (Note that in this figure the coded feedback vectors q are labeled with subscript q.) Fig. 1(c) shows that edge (v, d) has $m_{(v,d)}q_{(v,d)}^{\mathrm{T}} = 1$. Fig. 1(d) continues the propagation of the coded feedback vectors back to the source s. Since $\Gamma(u) = (1, 1)^{\mathrm{T}}$, $q_{(s,u)} = \Gamma(u)^{\mathrm{T}}[q_e : e \in \mathrm{Out}(u)] = (2, 1) + (2, 2) = (1, 0)$. Since $\Gamma(v) = 1$ and $m_{(v,d)}q_{(v,d)}^{\mathrm{T}} = 1$, we have $q_{(s,v)} = \Gamma(v)^{\mathrm{T}}\mathbf{q}_v = 1 \cdot (0, 0) = (0, 0)$. The final coded traffic and coded feedback vectors are illustrated in Fig. 1(d). There are exactly two edges $\{(s, u), (v, d)\}$ satisfying $m_e q_e^{\mathrm{T}} = 1$ in Fig. 1(e). Our new coded-feedback algorithm then outputs $\{(s, u), (v, d)\}$, which is indeed a minimum cut separating s and d.

Proposition 1 (Linear Running Time) *Assuming the computation time (within a node) is much smaller than the communication time (across different nodes), the proposed min-cut algorithm stops in linear time $\mathcal{O}(|V|)$.*

Note that existing preflow-based min-cut algorithms [5] require $\mathcal{O}(|V|^2)$ communication time for exchanging control messages.

Proof: As COMPUTE CODED TRAFFIC and COMPUTE CODED FEEDBACK process the nodes in their topological order, both sub-routines finish in linear time. Once the feedback vector q_e is computed, each edge e immediately knows whether it belongs to the output edge set or not by checking the inner product of m_e and q_e. The proof of the linear running time is thus complete. *Q.E.D.*

The correctness of the proposed algorithm is intractable for finite fields of small size (e.g. $\mathrm{GF}(2^1)$) due to the intrinsic hardness of the integer programming problem. (Note that we consider the packet-by-packet (edge-by-edge) integer solutions). Take Fig. 2(a) for example. Each edge is capable of carrying one symbol and thus has $r^l = 1$. The $\mathbf{MCMF}(\vec{r})$ of Fig. 2(a) is 2. Fig. 2(a) also depicts one network coding solution in GF(3) where each node performs simple addition of the incoming packets and the vectors along each edges describe the *global coding vectors* that is embedded in the header of the packets. The network coding solution described in Fig. 2(a)



(a) Original Network. (b) Removing (u_4, v). (c) Removing (u_3, v). (d) Removing (u_3, v) and (u_4, v).

Figure 2: An illustration of the non-convex nature of packet-by-packet network coding behavior

achieves the optimal rate. If we remove edge (u_4, v) as in Fig. 2(b), which is equivalent to reduce the rate on (u_4, v) from 1 to 0, the new $\mathbf{MCMF}(\vec{r}) = 2$ remains unchanged but the original coding solution is no longer able to achieve the optimal rate as edges (u_1, T) and (v, T) now carry linearly dependent packets. Similarly, if we remove edge (u_3, v) alone, then the optimal rate cannot be achieved (see Fig. 2(c)). Nonetheless, if we remove two edges (u_3, v) and (u_4, v) simultaneously, then the optimal rate 2 can be again achieved (see Fig. 2(d)). The above examples illustrate that although the graph-theoretic concept $\mathbf{MCMF}(\vec{r})$ is concave with respect to the link-rate-assignment $\vec{r}$, the relationship between $\vec{r}$ and the achievable end-to-end rank/throughput is no longer concave (actually not even monotonic). Since the **Main Algorithm** relies solely on the algebraic concepts such as the rank, the inner product, and the left-inverse, rather than the graph-theoretic concept of **MCMF**, the above irregularity significantly complicates the correctness analysis of the **Main Algorithm**.

Nonetheless, the **Main Algorithm** becomes tractable when a relatively large finite field (say $\mathrm{GF}(2^8)$ or $\mathrm{GF}(2^{16})$ commonly used in practice) is used in conjunction with *random linear network coding* [2].

Proposition 2 (Correctness) *Let $P(success)$ denote the probability that the output of our new min-cut algorithm is a minimum cut separating s and d. For a finite field $\mathrm{GF}(2^b)$, the success probability can be lower bounded by*

$$P(success) \geq \left(1 - 2^{-b}\right)^{|E|} + |E|\left(1 - 2^{-b}\right)^{|E|} - |E|. \tag{8}$$

When the field size $2^b \to \infty$, $P(success)$ tends to one.[6]

Note that the size $q = 2^b$ is exponential to the number of bits b assigned for each symbol and grows exponentially to a large number, say $q = 2^{16}$. Therefore, with close-to-one probability, the new algorithm outputs a minimum cut.

Proof: We will first sketch the main intuition and several key intermediate steps. Then we will provide the detailed proof of Proposition 2. Throughout this proof, we assume, without loss of generality, that the min-cut max-flow value $\mathbf{MCMF} = n$, which can always be satisfied by adding $(n - \mathbf{MCMF})$ auxiliary edges that directly connect s and d.

The main intuition behind this coded feedback approach is as follows. We say that an edge set E' is *parallel* if any edge $e \in E'$ cannot reach any other $e' \in E' \backslash e$. Assume that there exists a min-cut C that is also parallel. Let $\mathbf{m}_C \stackrel{\Delta}{=} [m_{e'} : e' \in C]$. Then the forward coded messages

[6]There was a typo of (8) in our Infocom09 submission.

received by destination d can be expressed as

$$\mathbf{m}_{\mathrm{In}(d)} \triangleq [m_e : e \in \mathrm{In}(d)] = \Gamma \cdot \mathbf{m}_C \tag{9}$$

for some $|\mathrm{In}(d)| \times n$ matrix Γ. We call Γ the transfer matrix from C to $\mathrm{In}(d)$.

Let Algo denote a new algorithm that is identical to Compute Coded Feedback except that Line 6 is removed, i.e. no zero-row-vector substitution. Then under Algo, we have

$$[q_e : e \in \mathrm{In}(v)] \leftarrow \Gamma(v)^{\mathrm{T}}[q_e : e \in \mathrm{Out}(v)]. \tag{10}$$

Note that in (10) the transfer matrix of the feedback vectors q_e is the transpose of the transfer matrix of the forward vectors m_e. We thus have

$$\begin{aligned} \mathbf{q}_C &\triangleq [q_{e'} : e' \in C] \\ &= \Gamma^{\mathrm{T}}[q_e : e \in \mathrm{In}(d)] \triangleq \Gamma^{\mathrm{T}} \cdot \mathbf{q}_{\mathrm{In}(d)}. \end{aligned} \tag{11}$$

The above implies that

$$\mathbf{q}_C^{\mathrm{T}}\mathbf{m}_C = \mathbf{q}_{\mathrm{In}(d)}^{\mathrm{T}}\Gamma\mathbf{m}_C = \mathbf{q}_{\mathrm{In}(d)}^{\mathrm{T}}\mathbf{m}_{\mathrm{In}(d)} = \mathbf{I}_n \tag{12}$$

where the first equality follows from (11), the second equality follows from (9), and the last equality follows from the construction of $\mathbf{q}_{\mathrm{In}(d)}$ in Line 2 of Compute Coded Feedback. Since we assume $\mathbf{MCMF} = n$, both $\mathbf{q}_C$ and $\mathbf{m}_C$ are $n \times n$ square matrices. Hence, (12) implies that $\mathbf{m}_C\mathbf{q}_C^{\mathrm{T}} = \mathbf{I}_n$. Therefore, if any edge e is in the parallel min-cut C, then $m_e q_e^{\mathrm{T}} = 1$. This property can be generalized as follows even when C is not parallel.

- **Property 1:** Under Algo, if removing any edge e reduces the rank of $[m_e : e \in \mathrm{In}(d)]$, then $m_e q_e^{\mathrm{T}} = 1$.

In addition, the converse is also true as stated below.

- **Property 2:** Under Algo, removing any single edge e that has $m_e q_e^{\mathrm{T}} = 1$ will reduce the rank of the coding vectors $[m_e : e \in \mathrm{In}(d)]$.

Both properties have been rigorously proven in [14, 15, 18].

These two properties suggest that we can check whether $m_e q_e^{\mathrm{T}} = 1$ to decide whether an edge e is in a min-cut. Consider the following modified algorithm, which serves as a conceptually more straightforward but implementation-wise less efficient version of the **Main Algorithm**.

---

§ A Modified Min-Cut Algorithm

1: Run Choose Random Mixing Coefficients
2: Run Compute Coded Traffic
3: Run Algo.
4: Let $V_1 \subseteq$ denote the vertex set such that any $v \in V_1$, there exists a path from v to d not using any edge e with $m_e q_e^{\mathrm{T}} = 1$.
5: **return** the edge set $C^{\diamond} \triangleq \{\forall (u, v) \in E : u \in V \backslash V_1, v \in V_1\}$.

As an example, Fig. 1(f) demonstrates the m_e and q_e generated by Algo (i.e. without the zero-vector-substitution in Line 6 of COMPUTE CODED FEEDBACK). There are three edges (s,u), (s,v), and (v,d) that have $m_e q_e^{\mathrm{T}} = 1$. Therefore, the node set V_1 is $V_1 = \{u,d\}$ and the output $C^\diamond$ is $C^\diamond = \{(s,u),(v,d)\}$. Note that this is identical to the output of the proposed **Main Algorithm** (cf. Fig. 1(e)).

Consider the following events: Let A_0 be the event that random network coding achieves the optimal rate **MCMF** (after Lines 1 and 2 of the MODIFIED MIN-CUT ALGORITHM). For any edge e, let A_e denote the event that after the removal of e, random network coding can still achieve the optimal rank **MCMF**. In [2], it is shown that

$$P(A_0) \geq (1-2^{-b})^{|E|} \quad \text{and} \quad P(A_e) \geq (1-2^{-b})^{|E|-1}$$

if e is not in any minimum cut.[7] Let E' denote the set of e that is not in any minimum cut and let C^* denote the min-cut that is the closest to d. (A formal definition that "a min-cut is the closest to d" is provided in Appendix A, along with a proof that such a closest min-cut always exists and is unique.) For the following, we will show that *in the event* $A_0 \cap \bigcap_{e\in E'} A_e$*, the output* $C^\diamond$ *of the* MODIFIED MIN-CUT ALGORITHM *is indeed the min-cut* C^*. By simple probability inequalities, we will then have

$$\begin{aligned} P(\text{success}) &\geq P\left(A_0 \cap \bigcap_{e\in E'} A_e\right) \\ &\geq \left(1-2^{-b}\right)^{|E|} + |E'|\left(1-2^{-b}\right)^{|E|-1} - |E'| \geq (8). \end{aligned}$$

We first provide one important property of C^*, the min-cut that is the *closest* to d, as follows (which is proven in Appendix A). For any $e \notin C^*$ that is reachable from some edge in C^*, e must be in E', i.e., such e must not participate in any other min-cut. With the above property, proving $C^\diamond = C^*$ is a straightforward exploitation of the two properties of Algo as described below.

In the event A_0, removing any edge in C^* will reduce the rate. By Property 1, any edge in C^* must have $m_e q_e^{\mathrm{T}} = 1$. By the construction of $C^\diamond$, for any $(u,v) \in C^\diamond$, there exists a path connecting v and d without using any edge in C^*. If $C^\diamond \neq C^*$, then there must exist a $e^\diamond = (u^\diamond, v^\diamond) \in C^\diamond$ such that $e^\diamond \notin C^*$ and $e^\diamond$ is reachable from some edge in C^*. By the property of the min-cut C^*, any such edge $e^\diamond$ must be in E'. Nonetheless, in the event $\bigcap_{e'\in E'} A_{e'}$, the edge $e^\diamond \in E'$ must have $m_{e^\diamond} q_{e^\diamond}^{\mathrm{T}} \neq 1$ by Property 2 of the **Algo**. This leads to a contradiction as the construction of $C^\diamond$ guarantees that any $e^\diamond \in C^\diamond$ must have $m_{e^\diamond} q_{e^\diamond}^{\mathrm{T}} = 1$. The correctness of the MODIFIED MIN-CUT ALGORITHM is proven.

Lines 4 and 5 of the MODIFIED MIN-CUT ALGORITHM require additional computation of the node/edge sets that can reach d without using any edge e with $m_e q_e^{\mathrm{T}} = 1$, which can be achieved by any SHORTEST PATH or CONNECTED COMPONENT algorithm after setting the "distance" to be infinity for edges with $m_e q_e^{\mathrm{T}} = 1$. The proposed **Main Algorithm** replaces the MODIFIED MIN-CUT ALGORITHM and the need of such a separate SHORTEST PATH algorithm by resetting to zero the very first encountered feedback vectors with $m_e q_e^{\mathrm{T}} = 1$ (Line 6 of COMPUTE CODED

[7] $P(A_e) = 0$ if e is part of a minimum cut.

FEEDBACK). For the following, we will quantify the impact of this new zero-vector substitution when compared to **Main Algorithm**, the results of which constitute a formal proof of Proposition 2.

A formal proof of Proposition 2:

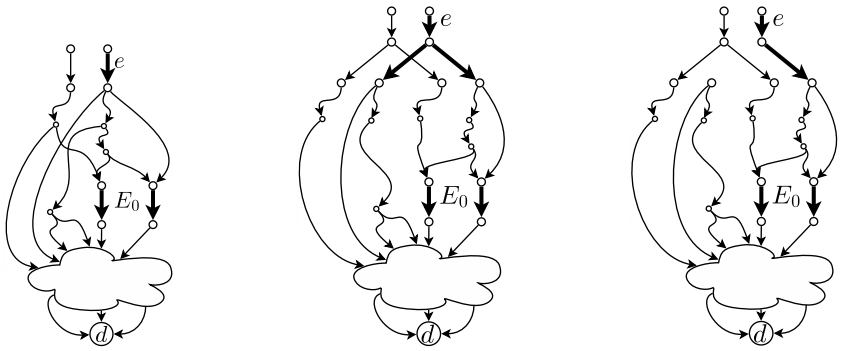
We first define the concept of *intercepted/non-intercepted* impact of a given edge e with respect to an $E_0 \subseteq E$. In the following definition, we consider the case that all edges in E_0 is reachable from e since the concept of being *intercepted* makes sense only to the downstream edges of e. For those E_0 containing non-downstream edges of e, our definition still applies with the following slight modification:

$$\text{Consider a new } E_0' \triangleq \{e' \in E_0 : e' \text{ being donwstream edges of } e\}. \tag{13}$$

Therefore, without loss of generality, we assume all edges in E_0 are downstream edges of e.

For any $e \in E$ and any edge set $E_0 \subseteq E$, (as illustrated in Fig. 3(a)), we can construct an equivalent network by the following steps (as illurstrated in Fig. 3(b)). We first consider an arbitrary cut containing e such that E_0 is in between the cut and destination d. In Fig. 3(a), the cut contains two edges. We use $V_0 \subseteq V$ to denote all the nodes that are reachable from some edges in E_0, which is illustrated by the cloud in Fig. 3(a). Consider all the paths connecting the cut and some node v in V_0 such that these paths use exactly one node in V_0 (i.e. the terminal node v) and do not use any edge in E_0. In Fig. 3(a), there are four such paths connecting the cuts and V_0 without using any edges in E_0. Augment the original network by duplicating those paths and the corresponding coding operations on those paths. Each edge in the cut is split to supply the original network and the new duplicated paths. All the terminal nodes v of the original network are first disconnected from the original network and then reconnected to the new duplicated paths. For example, the four paths in Fig. 3(b) is now connected to the corresponding $v \in V_0$ and the two edges in the cut are split to cover both the original network and the four duplicated paths. Since the duplicated paths carry the same coding operations as those in the original network, Figs. 3(a) and 3(b) are equivalent from the coding perspective. As illustrated in Fig. 3(b), the left branch of the newly split e corresponds to the paths from e to V_0 *not intercepted by* E_0, and the right branch correspond to the paths of e *intercepted* by E_0. We can then define the *intercepted/non-intercepted* impact of e by focusing on the right/left branches of e respectively. For example, Fig. 3(c) illustrates *the removal of the non-intercepted impact of* e as the left (non-intercepted) branch of the outgoing edges of e being removed.

For the following, we use the previous definition of A_0 as the event that random network coding achieves the optimal rate **MCMF**. However, we redefine the events A_e as follows. For any edge $e \notin C^*$ that is reachable from some edge in C^* (or equivalently in between C^* and d), let A_e denote the event that after *the removal of the non-intercepted impact of* e *with respect to* C^*, random network coding can still achieve the optimal rank **MCMF**. See Fig. 4 for illustration. Event A_e denotes the case in which the removal of the non-intercepted thick edge does not change the rank of the network coding session. We then note that removing non-intercepted impact can be viewed as an extension of edge-removal in the duplicated networks. Moreover, by the property of the closest min-cut C^*, any edge $e \notin C^*$ that is reachable from some edge in C^* does not participate in any min-cut. Then there must exist a network coding solution such that after the removal of e, one can still achieve the optimal **MCMF** rate (see Fig. 5(a)). Therefore, with the removal of the left branch (the non-intercepted impact), there exists one solution that achieves



(a) The original network.

(b) An equivalent subnetwork after duplicating the non-intercepted paths.

(c) Removal of the non-intercepted impact of e.

Figure 3: Illustration of the intercepted/non-intercepted impact of e with respect to an edge subset E_0.

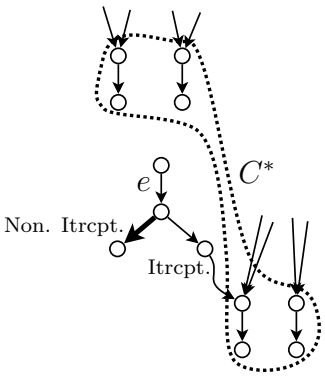
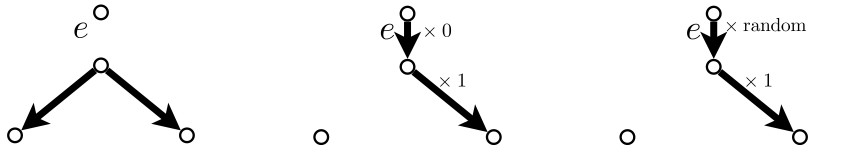


Figure 4: The topological relationship of e and C^*. Edge cut C^* contains four edges. We use the thick edge to denote the non-intercepted impact of e with respect to C^*.

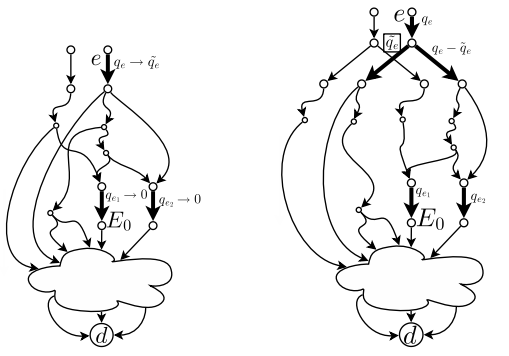


(a) Removal of e in the original network.

(b) Equivalent coding coefficients with the removal of the unintercepted impact (left-branch) of e.

(c) Random coding coefficients with the removal of the unintercepted impact (left-branch) of e.

Figure 5: The random coding arguments for the removal of the non-intercepted impact of e with respect to an edge subset E_0.



(a) Original network with q_f substituted by 0.

(b) Equivalent Network without substitution.

Figure 6: Illustration of the zero-row-vector substitution in Line 6 of the main algorithm.

the optimal **MCMF** rate with the coding coefficient on e being set to "$\times 0$" (see Fig. 5(b)). Hence, by the same random network coding arguments in [2], if we choose the coding coefficient of e randomly (see Fig. 5(c)), we have the following probability lower bound on $P(A_e)$.

$$P(A_e) \geq (1 - 2^{-b})^{|E|}.$$

Redefine E' as the set of $e \notin C^*$ that is reachable from C^*. For the following, we will show that in the event $A_0 \cap \bigcap_{e \in E'} A_e$, the **Main Algorithm** will output the min cut C^*. By simple probability inequalities, we again have

$$\begin{aligned} P(\text{success}) &\geq P\left(A_0 \cap \bigcap_{e \in E'} A_e\right) \\ &\geq \left(1 - 2^{-b}\right)^{|E|} + |E'|\left(1 - 2^{-b}\right)^{|E|-1} - |E'| \geq (8). \end{aligned}$$

We need the following observation before completing the proof.

> **Observation:** Line 6 of the COMPUTE CODED FEEDBACK has the effect of *switching the coded feedback to the one corresponding to the non-intercepted path with respect to the edge subset* E_0, where E_0 contains all edges that have had output $m_e q_e^{\mathrm{T}} = 1$ by the **Main Algorithm**.

Take Fig. 6 for example. Suppose originally edge e is carrying a feedback vector q_e and the two edges e_1 and e_2 in E_0 are carrying q_{e_1} and q_{e_2}. Once the feedback vectors q_{e_1} and q_{e_2} along E_0 are replaced by zero vectors (see Fig. 6(a)), this changes of the feedback vectors will propagate backward to the upstream edges of E_0. Therefore, the feedback vector along e will change from q_e to a new feedback vector, which we term it $\tilde{q}_e$. Let us also consider the intercepted/non-intercepted impact of e with respect to E_0 as depicted in Fig. 6(b). It can be proven that in the new subnetwork of Fig. 6(b), the original edge e is still carrying q_e, the same coded feedback vector of Fig. 6(a) *before the zero-vector substitution*. Moreover, the non-intercepted path of e carries $\tilde{q}_e$, the same coded feedback vector of Fig. 6(a) *after the zero-vector substitution*. Therefore, the

zero-vector substitution is equivalent to switching our focus from q_e on the original edge e to the $\tilde{q}_e$ along non-intercepted path of e. For the following, we will prove that the output of the **Main Algorithm** is C^* by combining Properties 1 and 2 of Algo and the above observation.

Assume the event $A_0 \cap \bigcap_{e\in E'} A_e$ is satisfied. Let e_k denote the k-th edge such that the **Main Algorithm** results in $m_{e_k} q_{e_k}^{\mathrm{T}} = 1$. We will prove by induction that all such e_k must be in C^*. Moreover, e_k must be the k-th closest-to-d edge[8] in C^* for all $k = 1, \cdots, n$.

Consider the initial case $k = 1$: I.e. e_1 being the very first edge that has $m_e q_e^{\mathrm{T}} = 1$ under the **Main Algorithm**. Since the zero-vector substitution in Line 6 of Compute Coded Feedback has not be executed before, e_1 must have $m_{e_1} q_{e_1}^{\mathrm{T}} = 1$ under the normal Algo. The assumption that A_0 is satisfied and Property 1 of **Algo** jointly imply that any $e \in C^*$ have $m_{e_1} q_{e_1}^{\mathrm{T}} = 1$ under normal Algo. Therefore, as the very first such edge, e_1 must be *in between* C^* and d. (One implication of the above statement is that if $e_1 \notin C^*$, then $e_1 \in E'$.) Suppose $e_1 \notin C^*$. Then the impact of e_1 must not be intercepted by C^*. Otherwise, there exists $e \in C^*$ that is a downstream edge of e_1 and has $m_e q_e^{\mathrm{T}} = 1$, which contradicts the construction of e_1 as the first such edge (the one that is the closest to d). If the impact of e_1 is not intercepted by e_1, then the assumption of A_{e_1} is equivalent to saying that the removal of e_1 will not reduce the rank received by d. Nonetheless, this implication contradicts Property 2 of Algo, which says that the removal of any e_1 with $m_{e_1} q_{e_1}^{\mathrm{T}} = 1$ will reduce the received rank. From the above reasoning, we have proven that the very first e_1 must be in C^*. Moreover, e_1 must be the most closest-to-d edge in C^*.

Suppose $e_k \in C^*$ for all $k \leq k_0$ and $e_1, \cdots, e_{k_0}$ are the k_0 closest-to-d edges in C^*. Consider e_{k_0+1}. Note that the zero-vector substitution has been performed on and only on $E_0 \stackrel{\Delta}{=} \{e_1, \cdots, e_{k_0}\}$. Consider an arbitrary edge $e \in C^* \backslash E_0$. The fact that C^* is a min-cut implies that removal of the non-intercepted impact of e with respect to E_0 must reduce the rank received by d. Therefore, after the k_0 times of zero-vector substitution, the output q_e of the **Main Algorithm** must still satisfy $m_e q_e^{\mathrm{T}} = 1$, which is proven by applying Property 1 of **Algo** to the *non-intercepted* path in the duplicated network noted in the **Observation**. Since all $e \in C^* \backslash E_0$ must have $m_e q_e^{\mathrm{T}} = 1$, we must have the following two cases: Either $e_{k_0+1} \in C^*$, or $e_{k_0+1} \in E'$. Namely, e_{k_0+1} must be *in between* C^* *and* d and cannot be strictkly in between s and C^*. Otherwise, the **Main Algorithm** will output some $e \in C^* \backslash E_0$ with $m_e q_e^{\mathrm{T}} = 1$ first before outputting e_{k_0+1}, which contradicts that that e_{k_0+1} is the very next such edge. Since e_{k_0+1} is the very next edge having $m_{e_{k_0+1}} q_{e_{k_0+1}}^{\mathrm{T}} = 1$, if $e_{k_0+1} \in C^*$, then e_{k_0+1} must be the (k_0+1)-th closest-to-d edge in C^*. The proof of induction will then be complete.

The only remaining case is when $e_{k_0+1} \in E'$. In this case, the intercepted/non-intercepted impact of e_{k_0+1} with respect to C^* must be equivalent to the intercepted/non-intercepted impact of e_{k_0+1} with respect to E_0. Otherwise, suppose there is an edge $e \in C^* \backslash E_0$ that also "intercepts" the impact of e_{k_0+1}. Then such e must be a downstream edge of e_{k_0+1}. The facts that all $e \in C^* \backslash E_0$ have $m_e q_e^{\mathrm{T}} = 1$ and e being a downstream edge of e_{k_0+1} contradict the assumption that e_{k_0+1} is the very next edge having $m_{e_{k_0+1}} q_{e_{k_0+1}}^{\mathrm{T}} = 1$. Following this reasoning, the assumption of $A_{e_{k_0+1}}$ is equivalent to that the removal of the non-intercepted impact of e_{k_0+1} with respect to E_0 will not reduce the rank received by d. By Property 2, this statement implies that after the k_0 times of zero-vector substitution, we must have $m_{e_{k_0+1}} q_{e_{k_0+1}}^{\mathrm{T}} \neq 1$, which contradicts the

[8]We can use an arbitary but fixed topological order of the edges that is consistent with the upstream/downstream relationship.

construction of e_{k_0+1}. Therefore $e_{k_0+1} \in C^*$ and the proof of induction is complete.

Q.E.D.

4 Combining Coded Feedback with Subgradient-Ascent

In this section, we will combine the coded-feedback method in Section 3 and the subgradient-ascent algorithm in (6) to develop a unified solution to Problem (2). Compared with the solution in [12] that uses a standard graph-theoretic method (i.e., PNR) to find the min-cut, the use of the coded-feedback method in our solution introduces two new issues when interacting with the subgradient-ascent algorithm. First, the coded-feedback method can only operate on graphs with unit edge-capacity. However, the subgradient-ascent iteration (6) produces rate-allocations that are real numbers. Hence, we need a way to switch between integers and real numbers. Second, there is a non-zero probability that the coded-feedback method will not find the min-cut in a given iteration. We will need a way to account for this probability.

In this section, we use the following technique to address these issues. We choose an integer D as a common denominator. We then choose the time unit such that a link with rate 1 can transmit D packets per unit time. For each $\vec{r}(t)$ (whose components are real numbers), we construct arbitrarily another vector $\tilde{\vec{r}}(t) \in \Lambda$ such that each of its components $\tilde{r}_i^l(t)$ is an integer multiple of $1/D$ and its difference from r_i^l is less than $1/D$. For each session i, when we form the unit-capacity network model for network coding (see Section 3.1), we convert each link $l \in L_i$ with allocated rate $r_i^l(t)$ to $D\tilde{r}_i^l(t)$ number of edges. Thus, each unit-capacity edge corresponds to a link rate of $1/D$, and can transmit one packet per unit time. Intra-session network coding and coded-feedback operations are carried out on this unit-capacity network model. Let $\text{Rank}_i(\tilde{\vec{r}}_i(t))$ denote the rank collected by the destination d_i of session i through this unit-capacity sub-network for session i. Then with high probability, $\mathbf{MCMF}_i(\tilde{\vec{r}}_i(t)) = \text{Rank}_i(\tilde{\vec{r}}_i(t))/D$. Once the coded-feedback algorithm returns a subset $\tilde{C}_i$ of *edges* (which with high probability is the min-cut for the unit-capacity sub-network for session i), we then construct a subset of *links* $C_i \subset L_i$ from the *edge* subset $\tilde{C}_i$ as follows. For any link $l \in L_i$, if $\tilde{r}_i^l(t) = 0$, then include l in C_i. If $\tilde{r}_i^l(t) > 0$ and if all the $D\tilde{r}_i^l(t)$ unit-capacity edges that link l maps to belong to $\tilde{C}_i$, then include l in C_i. For other cases, exclude l from C_i. The reason behind the above construction is as follows. If $\tilde{C}_i$ is indeed a min-cut for the unit-capacity sub-network of session i, and if a link l maps to an edge $e \in \tilde{C}_i$, then all other edges that link l maps to must also belong to $\tilde{C}_i$. Hence, the subset C_i constructed above must be a min-cut of the sub-network $G(V, L_i)$ of session i where the capacity of each link l is given by $\tilde{r}_i^l$. We can then form the vector $\tilde{\vec{\mu}}_i(t)$ as in (4), and calculate a vector $\partial\tilde{F}(\vec{r}(t))$ such that its (i,l)-th component is given by:

$$[\partial\tilde{F}(\vec{r}(t))]_{il} = U_i'(\text{Rank}_i(\tilde{\vec{r}}_i(t)))\tilde{\mu}_i^l. \tag{14}$$

Note that this vector $\partial\tilde{F}(\vec{r}(t))$ can be viewed as an approximation to the subgradient $\partial F(\vec{r}(t))$ given in (5). Finally, we replace $\partial F(\vec{r}(t))$ by $\partial\tilde{F}(\vec{r}(t))$ in the subgradient-ascent iteration (6), i.e.,

$$\vec{r}(t+1) = [\vec{r}(t) + \alpha\partial\tilde{F}(\vec{r}(t))]_\Lambda. \tag{15}$$

Unfortunately, as a consequence of the above techniques, the vector $\partial\tilde{F}(\vec{r}(t))$ is no longer a subgradient of the objective function $F(\vec{r})$ at $\vec{r}(t)$. Hence, the iteration (15) is not precisely a subgradient-ascent algorithm. However, as we show next, if D is large and if the probability with which the coded-feedback method returns a min-cut of the unit-capacity sub-network is close to 1, then the above iteration will result in rate allocation that is close to optimal. Let Ψ denote the set of solutions to the original problem (2). Let $F^* = F(\vec{r}^*)$ for any $\vec{r}^* \in \Psi$.

Proposition 3 *For any $\epsilon > 0$, there exist $\alpha_0, \delta_0, D_0 > 0$ such that for all $\alpha \leq \alpha_0$, $\delta \leq \delta_0$ and $D \geq D_0$, if the coded-feedback algorithm can output the min-cut for the unit-capacity sub-network of each session i with probability no less than $1-\delta$, then starting from any initial rate-allocation vector $\vec{r}(0)$, the iteration (15) will produce rate-allocation vectors that satisfy the following:*

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \mathbf{E}[F^* - F(\tilde{\vec{r}}(t))] \leq \epsilon.$$

Remark: Recall that $F(\tilde{\vec{r}}(t))$ corresponds to the total system utility when network-coding is applied to the unit-capacity sub-network generated by the discretized rate-allocation vector $\tilde{\vec{r}}(t)$. Clearly, $F(\tilde{\vec{r}}(t)) \leq F^*$ for all t. Hence, Proposition 3 implies that the rate-allocation produced by iteration (15) will be close to optimal in an averaging sense. Further, if the system is stationary and ergodic in the limit when $t \to \infty$, then Proposition 3 implies that, when t is large, we will have,

$$\mathbf{E}[F(\tilde{\vec{r}}(t))] \geq \mathbf{E}[F^*] - \epsilon. \tag{16}$$

Proof: For any $\vec{r}^* \in \Psi$, define

$$\Delta(t) \triangleq ||\vec{r}(t+1) - \vec{r}^*||^2 - ||\vec{r}(t) - \vec{r}^*||^2.$$

We would like to calculate the conditional expectation $\mathbf{E}[\Delta(t)|\vec{r}(t)]$. Note that from (15), we have,

$$\begin{aligned} ||\vec{r}(t+1) - \vec{r}^*||^2 &\leq ||[\vec{r}(t) + \alpha\partial\tilde{F}(\vec{r}(t))] - \vec{r}^*||^2 \\ &= ||\vec{r}(t) - \vec{r}^*||^2 + 2\alpha\partial\tilde{F}(\vec{r}(t))(\vec{r}(t) - \vec{r}^*) \\ &\quad + \alpha^2||\partial\tilde{F}(\vec{r}(t))||^2. \end{aligned} \tag{17}$$

There are two cases.

Case 1: The coded feedback algorithm returns the min-cut of the unit-capacity sub-network of each session i correctly. In this case, $\mathbf{MCMF}_i(\tilde{\vec{r}}_i(t))$ must be equal to $\text{Rank}_i(\tilde{\vec{r}}_i(t))$. Hence, $\partial\tilde{F}(\vec{r}(t))$ must be a subgradient of $F(\cdot)$ at $\tilde{\vec{r}}(t)$. From the definition of the subgradient (3), we then have,

$$\partial\tilde{F}(\vec{r}(t))(\vec{r}^* - \tilde{\vec{r}}(t)) \geq F(\vec{r}^*) - F(\tilde{\vec{r}}(t)).$$

Further, recall that each component of $\tilde{\vec{r}}(t)$ differs from the corresponding component of $\vec{r}(t)$ by at most $1/D$. Since the derivative of $U_i(\cdot)$ is bounded by M_0, we must have $||\partial\tilde{F}(\vec{r}(t))|| \leq M_0|I||L|$ for all t. Hence,

$$|\partial\tilde{F}(\vec{r}(t))[\vec{r}(t) - \tilde{\vec{r}}(t)]| \leq \frac{M_0(|I||L|)^2}{D}$$

for all t. Using (17), we thus have,

$$\begin{aligned} & ||\vec{r}(t+1)-\vec{r}^*||^2 \\ \leq\ & ||\vec{r}(t)-\vec{r}^*||^2 + 2\alpha\partial\tilde{F}(\vec{r}(t))(\tilde{\vec{r}}(t)-\vec{r}^*) \\ & +2\alpha\partial\tilde{F}(\vec{r}(t))(\vec{r}(t)-\tilde{\vec{r}}(t)) + \alpha^2||\partial\tilde{F}(\vec{r}(t))||^2 \\ \leq\ & ||\vec{r}(t)-\vec{r}^*||^2 - 2\alpha[F(\vec{r}^*)-F(\tilde{\vec{r}}(t))] \\ & +2\alpha\frac{M_0(|I||L|)^2}{D} + \alpha^2(M_0|I||L|)^2. \end{aligned}$$

Case 2: if the coded-feedback algorithm does not compute the min-cut correctly, we will still have $||\partial\tilde{F}(\vec{r}(t))|| \leq M_0|I||L|$. Further, note that $||\vec{r}(t)-\vec{r}^*|| \leq |I|\sum_{l=1}^{L} R^l \triangleq M_1$. Using (17) again, we have,

$$\begin{aligned} & ||\vec{r}(t+1)-\vec{r}^*||^2 \\ \leq\ & ||\vec{r}(t)-\vec{r}^*||^2 + 2\alpha M_0|I||L|M_1 + \alpha^2(M_0|I||L|)^2. \end{aligned}$$

Combining the two cases, and using the assumption that the coded-feedback algorithm can return the min-cut correctly with probability no smaller than $1-\delta$, we then have,

$$\begin{aligned} & \mathbf{E}[\Delta(t)|\vec{r}(t)] \\ \leq\ & -2\alpha(1-\delta)[F(\vec{r}^*)-F(\tilde{\vec{r}}(t))] \\ & +2\alpha M_0|I||L|\left(\delta M_1 + \frac{|I||L|}{D}\right) + \alpha^2(M_0|I||L|)^2. \end{aligned}$$

Take another expectation on both sides with respect to $\vec{r}(t)$, and summing over $t = 0, 1, ..., T$, we obtain

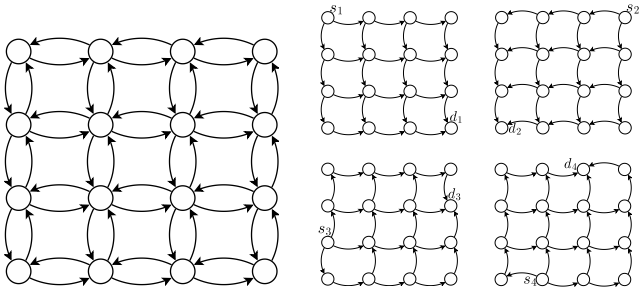
$$\begin{aligned} & \mathbf{E}[||\vec{r}(T+1)-\vec{r}^*||^2 - ||\vec{r}(0)-\vec{r}^*||^2] \\ \leq\ & -2\alpha(1-\delta)\sum_{t=0}^{T}\mathbf{E}[F(\vec{r}^*)-F(\tilde{\vec{r}}(t))] \\ & +2T\alpha M_0|I||L|\left(\delta M_1 + \frac{|I||L|}{D}\right) + T\alpha^2(M_0|I||L|)^2. \end{aligned}$$

Since $\mathbf{E}[||\vec{r}(T+1)-\vec{r}^*||^2] \geq 0$, dividing both sides by $2\alpha(1-\delta)T$, and letting $T \to \infty$, we have

$$\begin{aligned} & \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T}\mathbf{E}[F(\vec{r}^*)-F(\tilde{\vec{r}}(t))] \\ \leq\ & M_0|I||L|\left[\frac{\delta M_1}{1-\delta} + \frac{|I||L|}{(1-\delta)D}\right] + \frac{\alpha}{2(1-\delta)}(M_0|I||L|)^2. \end{aligned}$$

The result then follows by choosing δ, D and α such that the right-hand-side is less than ϵ.

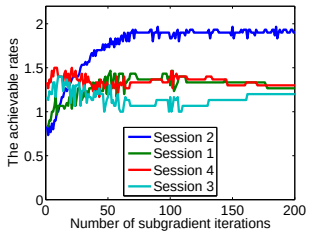
Q.E.D.



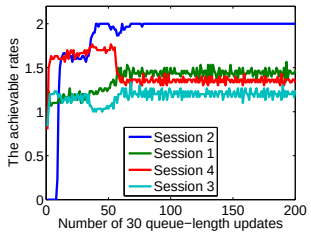
(a) The directed cyclic network

(b) Acyclic subgraphs for individual sessions

Figure 7: The network with four coexisting sessions.



(a) The New Coded-Feedback Scheme



(b) The back-pressure scheme with packet-by-packet queue-length updates.

Figure 8: Time evolution of the proposed coded-feedback scheme compared with the back-pressure scheme.

5 Numerical Experiments

We simulate our solution on a network of 16 nodes (see Fig. 7(a)). There are 48 directed links, each with a rate of 1. There are four co-existing unicast sessions (s_1, d_1) to (s_4, d_4) competing for the resources and each session chooses an acyclic subgraph as shown in Fig. 7(b). As discussed in Section 4, we choose the common denominator $D = 30$. This means that a link with rate 1 can transmit $D = 30$ packets per unit time, and can be converted to $D = 30$ unit-capacity edges when we construct the sub-networks for network coding. Every unit time, one set of coding coefficients are transmitted from the source nodes to the destination nodes, and the feedback coefficients are transmitted from the destination nodes to the source nodes. We then use (15) to update the rate-allocation vector $\vec{r}(t)$ once every unit-time. Other parameters of the system include: The number of information symbols to be mixed together at the source (often called the generation size [17]) is $n = 60$, the finite field size is GF(2^8), the utility function is $U_i(x) = \log(x + 10)$ for $i = 1, \cdots, 4$, and the step size is $\alpha = 1$.

Fig. 8(a) illustrates the time evolution of the proposed scheme with respect to the number of sub-gradient updates. We have also solved the optimization problem (2) offline, and found that the optimal rate-allocation is given by $(x_1^*, x_2^*, x_3^*, x_4^*) = (\frac{4}{3}, 2, \frac{4}{3}, \frac{4}{3})$. As shown in Fig. 8(a), our proposed solution with coded-feedback reaches 90% of the optimal rate-allocation in less than

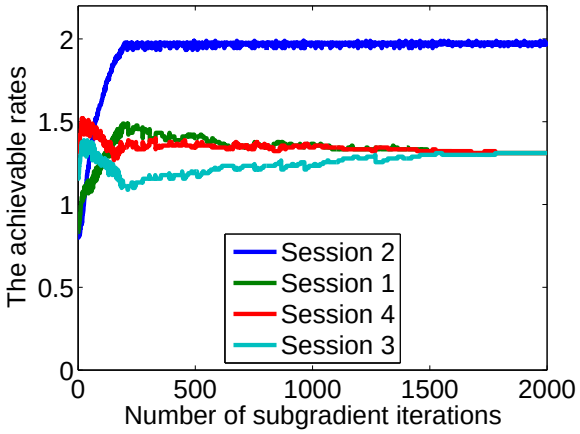


Figure 9: Time evolution for a set of less practical values of parameters, which on the other hand has a smaller gap to the optimal rates.

130 iterations. In our simulation, we choose the step-size $\alpha = 1$ such that it is the largest α that can still reach 90% of the optimal rate-allocation. We say such choice of α as having the *fastest 90% convergence*. Note that by sacrificing the speed of convergence (decreasing the value of α) and by using finer granularity (increasing the values of n and D), our algorithm can approach the optimal rates $(\frac{4}{3}, 2, \frac{4}{3}, \frac{4}{3})$. Fig. 9 describes the convergence of the our fast resource allocation algorithm but using a different set of parameters: $D = 90$, $n = 180$, and $\alpha = 0.3$. This is a less practical set of parameter values as in practice [19] the generation size is usually between n =30–60. On the other hand, this choice of parameters reduces the gap to the optimal solution $(x_1^*, x_2^*, x_3^*, x_4^*) = (\frac{4}{3}, 2, \frac{4}{3}, \frac{4}{3})$ as can be seen in Fig. 9.

To compare with a non-coded solution, we have simulated the rate-allocation algorithm in [6,20] that solves a similar utility-maximization problem using the back-pressure algorithm. In the rest of the section, we will simply refer to it as the back-pressure scheme. For a fair comparison, we restrict the back-pressure scheme to also search among the acyclic subgraphs predefined in Fig. 7(b). In addition, we let the back-pressure scheme update its queue-length and rate-allocation every 1/30 time. (This is because the back-pressure scheme may update the control every time one packet is transmitted, which takes 1/30 time under our setting.) In Fig. 8(b), we show the time-evolution with the back-pressure scheme, where each unit on the x-axis is equal to 30 queue-length updates (i.e., equal to the time for one iteration in Fig. 8(a)). Next, we briefly discuss some features of our scheme and compare it with the back-pressure scheme.

5.1 Convergence Speed and Overhead of Feedback

Our first observation is that the solution in this paper has comparable convergence speed as the back-pressure algorithm. (Note that in Fig. 8(b) the step-size is also chosen to have the *fastest 90% convergence*.) The overhead of the two algorithms is also comparable. In the coded-feedback algorithm, one feedback coefficient is sent on every edge. Hence, a link of rate 1 sends 30 feedback coefficients per unit-time. In the back-pressure scheme, since each update (every 1/30 time) requires the feedback of queue-length information, each link also sends 30 queue-

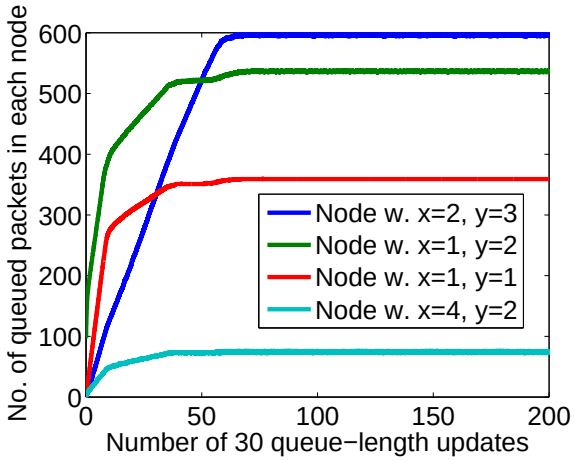


Figure 10: The number of queued packets in four representative nodes under the back-pressure scheme in Fig. 8(b) with packet-by-packet queue-length updates.

length updates per unit-time. We note however that the feedback frequency in our proposed solution (i.e., once every unit-time) is much slower than that of back-pressure (i.e., once every 1/30-time). Hence, our proposed solution could be more attractive when there are constraints on the feedback frequency.

5.2 Controlling the Queue Length

A key advantage of our proposed solution with coded-feedback is that the queue-length can be tightly controlled. In fact, since each link needs to store at most 30 packets before applying the coding operation, it only needs a buffer of 30 packets. In contrast, in our numerical experiments of the back-pressure algorithm in Fig. 8(b), the number of queued packets exceeds 500 at 7 of the total 16 nodes. Fig. 10 describes the queue-length evolution of the back-pressure algorithm for four representative nodes. We denote each representative node by its x and y coordinates. For example, the first representative node with $x = 2$ and $y = 3$ is the node that is at the intersection of the 2nd column from the left and the 3rd row from the bottom. As seen in Fig. 10, different nodes will have different queue lengths. We observe that under the back-pressure scheme, the queue lengths vary significantly from node to node and many of them are much larger than the buffer sizes required by our network-coding-based solution.

In general, the queue-length dynamics under the back-pressure scheme depend on many factors, such as the step-size, the network topology and the traffic pattern. Hence, the queue-length (and correspondingly the packet delay) is much harder to control. For instance, if one decreases the step size in a rate-allocation algorithm based on back-pressure (e.g., in [20]), although the eventual rate-allocation will be closer to the optimal, the queue-length will typically increase inverse-proportionally to the step-size.[9] In comparison, a highly-desirable feature of our proposed solution is that the queue-length is decoupled from the step-size α. Hence, one can tune the precision of the eventual rate-allocation without increasing the queue-length or packet delay.

[9]Note that even with the technique of virtual queues [20], the physical queue length is still highly correlated with the step size.

This thus gives an extra degree of freedom for designing efficient control schemes, especially for delay-sensitive applications.

5.3 Quick Start

We also observe that our proposed solution with coded-feedback achieves better rate-allocation than the back-pressure scheme *at the beginning of the iterations*. This "quick start" feature is achieved by the following. If a given link l is shared by I^l sessions, we can simply let each competing session grab $1/I^l$ share of the available capacity R^l and set $r_i^l(0) = \frac{R^l}{I^l}$ as the initial rate-allocation. As a result, it is guaranteed that each session can achieve at least $\frac{1}{\max_l I^l}$ of the optimal rate-allocation *at the very beginning of the iteration*. (Note that this lower bound $\frac{1}{\max_l I^l}$ is quite conservative, and in practice the ratio is usually much higher.) Moreover, suppose after a certain amount of time, one additional session would like to join the network. Again, the new session can grab its fair share of the link capacities at the beginning of the iterations and start transmission right-away.

In Fig. 8(a), the above "quick-start" feature ensures that, at the beginning of the iterations, the initial receiving rates are $(\frac{5}{6}, \frac{5}{6}, \frac{7}{6}, \frac{4}{3})$, which is (62.5%, 41.7%, 87.5%, 100%) of the optimal rates $(\frac{4}{3}, 2, \frac{4}{3}, \frac{4}{3})$. In Fig. 8(a), the coded-feedback algorithm ensures that Sessions 3 and 4 retain ≈70–90% of the optimal rates throughout the iterations while gradually increasing the rates for Sessions 1 and 2. For comparison, in Figs. 8(b), *Session 2 is not able to receive any packets in the first* 8×30 *iterations under the back-pressure scheme.* These results indicate that the back-pressure scheme does not have this "quick-start" feature.

6 Discussions

In this section, we discuss extensions of our results to multicast sessions and to cyclic networks.

6.1 Multicast Sessions

To model multicast sessions, we associate each session i with a source node s_i and a set J_i of destination nodes $d_{ij}, j = 1, ..., |J_i|$. Given the rate allocation $\vec{r}_i$ for the sub-network i, let $\mathbf{MCMF}_i(\vec{r}_i, d_{ij})$ denote the min-cut max-flow value between the source s_i and the destination d_{ij}. With intra-session network coding, the maximum rate at which the source can send to *all* destination nodes is equal to $\mathbf{MCMF}_i(\vec{r}_i) \triangleq \min_{j \in J_i} \mathbf{MCMF}_i(\vec{r}_i, d_{ij})$. Note that the problem formulation (2) remains the same with this new definition of $\mathbf{MCMF}_i(\vec{r}_i)$.

The subgradient-ascent algorithm can also be used, except that when computing a subgradient, we need to use the notion of a "critical cut" [12]. A destination node j is *critical* if $\mathbf{MCMF}_i(\vec{r}_i, d_{ij}) = \mathbf{MCMF}_i(\vec{r}_i)$, i.e., its min-cut max-flow value is equal to the minimum among all destinations. A min-cut $C_i^{\min}$ between the source s_i and such a critical destination j is called a *critical cut*. Once a critical cut $C_i^{\min}$ is found, the authors of [12] then use (4) and (5) to form a subgradient of the objective function $F(\vec{r})$. They then use the subgradient-ascent algorithm (6) to compute the optimal rate-allocation.

A similar methodology can also be used in our coded-feedback approach. As in Section 4, we first replace the rate allocation $\vec{r}_i$ by the discretized value $\tilde{\vec{r}}_i$ with granularity $1/D$. We then

search for the critical-cut for the sub-network with the rate allocation $\tilde{\vec{r}}_i$. To find out which destination is critical, we use a technique introduced in [12]. Specifically, when the source is sending out packets, these packets are coded from a set of uncoded symbols with a rank greater than $\mathbf{MCMF}_i(\tilde{\vec{r}}_i)D$. Note that with high probability, the critical destination will receive a rank equal to $\mathbf{MCMF}_i(\tilde{\vec{r}}_i)D$, while a destination that is not critical will receive a rank greater than $\mathbf{MCMF}_i(\tilde{\vec{r}}_i)D$. Hence, this technique enables the identification of the critical destination through the outcome of the network coding operation. Once a critical destination is identified, the coded-feedback algorithm can then start from this destination to find the critical cut. We can then use (14) and (15) to update the rate-allocation. This network-coding-based search for the critical destination returns the critical destination with probability close-to-one (but not exactly one). Therefore, an additional small amount of uncertainty is introduced during this step. Using similar proof techniques, we can upper bound the impact of this uncertainty and derive parallel convergence results as in Proposition 3.

6.2 Cyclic Graphs

In our system model in Section 2.1, we assume that each session i chooses an acyclic subgraph $G_i = (V, L_i)$ from the original network graph. In fact, our solution can be generalized such that one does not need to pre-choose such an acyclic subgraph. We use the following technique, which is standard in the network-coding literature to convert a cyclic graph $G = (V, L)$ to an acyclic graph [16]. The key idea is to introduce the notion of time/delay. Assume for the sake of simplicity that each link has unit delay. For a given integer K, we can make $K + 1$ copies of the set V of nodes. We index these copies by time $0, 1, ..., K$. Then, we make K copies of each directed link l. Each copy of link l has the same capacity R^l. For each $k = 1, 2, ..., K$, the k-th copy of the directed link l connects the corresponding transmitting node at time $k - 1$ to the receiving node at time k. Intuitively, such a construction models the delay at the original directed link l. Further, the copy of node n at time $k - 1$ is also connected to the copy of node n at time k with infinite capacity, which models the "memory" at node n. With these constructions, the resulted graph must be acyclic because the links always go in the direction of time. It has been shown that, for both unicast and multicast sessions, as long as K is large, the maximum rate supported by the above acyclic graph will approach that of the original (and potentially cyclic) graph [16].

Our coded-feedback-based solution can be applied to such an acyclic network graph by expanding the set of control variables. Specifically, we can associate a primal variable $\vec{r}^{l,k} = [r_i^{lk}], k = 1, ..., K$ for each copy of the directed link l, where r_i^{lk} is the capacity allocated to session i on the k-th copy of link l. Each copy of the primal variable $\vec{r}^{l,k}, k = 1, ..., K$, must satisfy the capacity constraint that $\sum_{i=1}^{|I|} r_i^{lk} \leq R^l$. Once $\vec{r}^k, k = 1, ..., K$ are chosen, the network coding and coded feedback operations can be applied to the acyclic subgraph constructed above for each session i. Finally, the computation of the approximate subgradient-vector (14) and the approximate subgradient-ascent iteration (15) can also be naturally extended to this acyclic subgraph.

7 Conclusion

We have developed a new fast resource allocation algorithm based on the novel concept of *coded-feedback*. *Our solution performs coding operations on both the forward and the return directions.* Coding on the forward direction eliminates the need of building up queues. In conjunction with a primal subgradient-ascent algorithm, this leads to a scheme with easily controllable packet delay and with a desirable *quick-start* feature. Coding on the feedback direction provides an ultra-efficient $\mathcal{O}(|V|)$-time method of finding the min-cut of a network, which is an order of degree faster than existing $\mathcal{O}(|V|^2)$-time min-cut algorithms. These features, enabled by the tight integration of network coding and coded feedback, are beneficial for both multicast and unicast sessions. As shown in our numerical results, even for unicast sessions, our solution can serve as an attractive alternative to the existing non-coded resource allocation algorithms.

A The Closest Minimum Cut

Consider the unit-capacity, directed acyclic network model $G = (V, E)$ discussed in Section 3.1. We can have the following definition. Let (E', E'') denote a partition of the edge set E such that E'' contains the edges that participate in at least one min-cut and $E' = E \backslash E''$ denotes the edges that do not participate in any min-cut. For any min-cut C, we denote the edges $E_{C\to d}$ that are *strictly in between* C *and* d by

$$E_{C\to d} \stackrel{\Delta}{=} \{e \in E\backslash C : e \text{ is reachable from some edge in } C \text{ and } d \text{ is reachable from } e\} \qquad (18)$$

Definition 4 (The Closest Min-Cut) *A min-cut C^* is the closest to destination d if $E_{C^*\to d} \cap E'' = \emptyset$.*

Lemma 5 *There exists a unique C^* that is the closest to d.*

Proof: We first prove the existence. For any min-cut C, consider the corresponding value $g(C) = |E_{C\to d} \cap E''|$. Choose a min-cut C_a that minimizes $g(C)$. As $g(C) \geq 0$ being of integer value, such a minimizing C_a always exists. The existence of C^* is thus equivalent to proving $g(C_a) = 0$.

Suppose $g(C_a) > 0$. There exists an edge e_b and a min-cut C_b such that $e_b \in E_{C_a\to d}$ and $e_b \in C_b$. Since $|C_a| = |C_b| = \mathbf{MCMF}$, by the min-cut max-flow theorem, there exists a set of **MCMF** edge-disjoint paths connecting s and d. We use P_1, P_2, to $P_{\mathbf{MCMF}}$ to denote such edge-disjoint paths. Since C_a must *cut* these **MCMF** paths, each edge in C_a must use exclusively one of these paths. We can thus label the edges in C_a as $e_{a,1}$ to $e_{a,\mathbf{MCMF}}$ depending on their corresponding paths. Similarly, we can label the edges in C_b as $e_{b,1}$ to $e_{b,\mathbf{MCMF}}$. For any i, we define the order between $e_{a,i}$ and $e_{b,i}$ as follows. We say $e_{a,i} < e_{b,i}$ if $e_{a,i}$ is a strictly upstream edge of $e_{b,i}$. Intuitively, $e_{a,i} \leq e_{b,i}$ implies that either $e_{a,i} < e_{b,i}$ or $e_{a,i} = e_{b,i}$. Symmetrically, we can define $e_{a,i} > e_{b,i}$ and $e_{a,i} \geq e_{b,i}$. Following the above order, we let $e_{a\vee b,i}$ denote the *maximum* of the two edges $e_{a,i}$ and $e_{b,i}$.

Without loss of generality, we assume that it is $e_{b,1}$ that satisfies $e_{b,1} \in E_{C_a\to d}$, which also implies that $e_{a\vee b,1} = e_{b,1} > e_{a,1}$. Consider a new edge set $C_{a\vee b} = \{e_{a\vee b,1}, \cdots, e_{a\vee b,\mathbf{MCMF}}\}$ of **MCMF** edges. Since each path P_i is used by exactly one edge in a min-cut, we have $e_{a,i} \neq e_{b,j}$

for any $i \neq j$. Therefore, $C_{a\vee b}$ contains **MCMF** distinct edges. Shortly after, we will show that $C_{a\vee b}$ is a cut. Since $|C_{a\vee b}| = \mathbf{MCMF}$, $C_{a\vee b}$ must be a min-cut. We then notice that by the above construction

$$E_{C_{a\vee b}\to d} \subseteq E_{C_a\to d}.$$

Since $e_{b,1} = e_{a\vee b,1}$ belongs to $E_{C_a\to d} \cap E''$ but not to $E_{C_{a\vee b}\to d}$, we have $g(C_{a\vee b}) \leq g(C_a) - 1$, which contradicts the assumption that C_a minimizes $g(C)$. Therefore $g(C_a) = 0$ and the proof of existence is complete. The above construction can also be used to prove the uniqueness. Suppose C_a^* and C_b^* are two closest min-cuts and we label their edges $e_{a,i}^*$ and $e_{b,i}^*$ as described previously. Since each path P_i is used by exactly one edge in a min-cut, we also have $e_{a,i}^* \neq e_{b,j}^*$ for any $i \neq j$. Therefore, for any $i = 1, \cdots, \mathbf{MCMF}$, edge $e_{b,i}^*$ must not be a strictly downstream edge of $e_{a,i}^*$. Otherwise, $e_{b,i}^*$ belongs to $E_{C_a^*\to d}$, which contradicts the assumption $g(C_a^*) = 0$. Symmetrically, $e_{a,i}^*$ must not be a strictly downstream edge of $e_{b,i}^*$. Therefore $e_{a,i}^* = e_{b,i}^*$ for all $i = 1, \cdots, \mathbf{MCMF}$ and we have $C_a^* = C_b^*$.

We use contradiction to show that $C_{a\vee b}$ is indeed a cut. Suppose not. There must exist a path P^* connecting s and d without using any edge in $C_{a\vee b}$. Since C_a is a min-cut, there must exist at least one edge $e_a \in C_a$ such that P^* uses e_a. Similarly, since C_b is a min-cut, there must exist at least one edge $e_b \in C_b$ such that P^* uses e_b. Among all such e_a and e_b, choose the one that P^* meets the last before finally arriving d. Without loss of generality, assume it is e_{a,i_0} that P^* meets the last, which uses the i_0-th edge-disjoint path P_{i_0}. Since P^* does not use any edge in $C_{a\vee b}$, $e_{a,i_0} \in P^*$ implies that $e_{a,i_0} < e_{b,i_0} = e_{a\vee b,i_0}$. Consider a new s-to-d path $sP_{i_0}e_{a,i_0}P^*d$ consisting of two path segments. The first segment is from s to e_{a,i_0} via path P_{i_0} and the second segment is from e_{a,i_0} to d via path P^*. Since P_{i_0} uses no edges in C_b other than e_{b,i_0}, and since $e_{a,i_0} < e_{b,i_0}$, the first path segment $sP_{i_0}e_{a,i_0}$ does not use any $e_{b,j}$ for $j = 1, \cdots, \mathbf{MCMF}$. Since e_{a,i_0} is the very last edge in $C_a \cup C_b$ that P^* meets, the second path segment $e_{a,i_0}P^*d$ does not use any $e_{b,j}$ for $j = 1, \cdots, \mathbf{MCMF}$. As a result, the combined new path $sP_{i_0}e_{a,i_0}P^*d$ does not use any $e_{b,j}$ for $j = 1, \cdots, \mathbf{MCMF}$, which contradicts the assumption that C_b is an edge cut. By contradiction, $C_{a\vee b}$ must be an edge cut. The proof is complete.

Q.E.D.

References

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, pp. 1204–1216, July 2000.

[2] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inform. Theory*, vol. 52, no. 10, pp. 4413–4430, October 2006.

[3] D. S. Lun, N. Ratnakar, M. Médard, R. Koetter, D. R. Karger, T. Ho, E. Ahmed, , and F. Zhao, "Minimum-cost multicast over coded packet networks," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2608–2623, June 2006.

[4] C. Fragouli, J. Widmer, and J.-Y. L. Boudec, "Network coding: an instant primer," *ACM SIGCOM Computer Communication Review*, 2006.

[5] A. Goldberg and R. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM*, vol. 35, no. 4, pp. 921–940, October 1988.

[6] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, December 1992.

[7] X. Lin and N. B. Shroff, "Utility Maximization for Communication Networks with Multipath Routing," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 766–781, May 2006.

[8] T. Ho and H. Viswanathan, "Dynamic algorithms for multicast with intra-session network coding," in *43rd Allerton Annual Conference on Communication, Control, and Computing*, Monticello, IL, September 2005.

[9] Y. Wu and S.-Y. Kung, "Distributed Utility Maximization for Network Coding Based Multicasting: A Shortest Path Approach," *IEEE J. Select. Areas Commun.*, vol. 24, no. 8, pp. 1475–1488, August 2006.

[10] L. Chen, T. Ho, S. H. Low, M. Chiang, and J. C. Doyle, "Optimization Based Rate Control for Multicast with Network Coding," in *Proc. of IEEE INFOCOM*, Anchorage, Alaska, May 2007, pp. 1163–1171.

[11] A. Khreishah, C.-C. Wang, and N. Shroff, "An optimization-based rate control for communication networks with inter-session network coding," in *Proc. 27th IEEE Conference on Computer Communications (INFOCOM)*. Phoenix, USA, April 2008.

[12] Y. Wu, M. Chiang, and S.-Y. Kung, "Distributed utility maximization for network coding based multicasting: A critical cut approach," in *Proc. 2nd Workshop on Network Coding, Theory, & Applications (NetCod)*. Boston, Massachusetts, April 2006.

[13] Y. Xi and E. M. Yeh, "Distributed Algorithms for Minimum Cost Multicast with Network Coding," in *43rd Allerton Annual Conference on Communication, Control, and Computing*, Monticello, IL, September 2005.

[14] C.-C. Wang, "Pruning network coding traffic by network coding — a new max-flow algorithm," in *Proc. IEEE Int'l Symp. Inform. Theory*. Toronto, Canada, July 2008.

[15] ——, "A coded-feedback approach for constructing minimum-cost multicast network codes," in *Proc. 45th Annual Allerton Conf. on Comm., Contr., and Computing*. Monticello, Illinois, USA, September 2008.

[16] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inform. Theory*, vol. 49, no. 2, pp. 371–381, February 2003.

[17] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41st Annual Allerton Conf. on Comm., Contr., and Computing*. Monticello, IL, October 2003.

[18] C.-C. Wang, "Pruning network coding traffic by network coding — a new class of max-flow algorithms," *IEEE Trans. Inform. Theory*, submitted and under review.

[19] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the air: Practical wireless network," in *Proc. ACM Special Interest Group on Data Commun. (SIGCOMM)*, 2006.

[20] X. Lin and N. B. Shroff, "Joint Rate Control and Scheduling in Multihop Wireless Networks," in *Proceedings of the IEEE Conference on Decision and Control*, Paradise Island, Bahamas, December 2004.