REDUCED-COMPLEXITY NETWORK CODING FOR MULTICASTING OVER AD HOC NETWORKS

Yunnan Wu, Sun-Yuan Kung

Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544.

ABSTRACT

Network coding generalizes the conventional routing paradigm by allowing nodes to mix information received on its incoming links to generate information to be transmitted to other nodes. As a result, network coding improves throughput, resource efficiency, robustness, manageability, etc, in wired and wireless ad hoc networks. In particular, it was established that network coding can achieve the maximum rate for multicasting information from a source node to multiple destination nodes. The objective of this work is to show how to achieve the aforementioned multicast capacity with lower processing/implementation complexity than what was proposed in the literature.

We classify the links in a network into two categories: 1) links entering relay nodes, and 2) links entering destinations. We show the same multicast capacity can be achieved by applying (non-trivial) network coding only on the links entering relay nodes. In other words, links entering destinations will only require routing, which leads to a saving in the processing/implementation complexity. The novelty of this work lies in a new algorithm, its proof of correctness, and a complexity analysis.

1. INTRODUCTION

Consider the problem of information multicast, namely transmitting common information from a source node s to a set of destination nodes T, in a network represented by a directed graph $G = (V, E)$ with unit-capacity edges. Conventionally, end-to-end information transmission is done by *routing*, i.e., letting nodes in the network store-and-forward information. Recently, in [1], Ahlswede et al. showed that the *multicast capacity*, which is defined as the maximum multicast rate from s to T, can be achieved with *network coding*, while it cannot be achieved in general by routing. Network coding allows a node to "mix" information, i.e., generate output symbols by computing certain functions of the symbols it received.

Network coding is highly applicable to real packet networks. For example, previous work [2] presented a prototype system for practical network coding in packet networks, using distributed random linear network coding with buffering. The system achieves throughput close to capacity with low delay, and is robust to random packet loss and delay as well as to changes to network topology or capacity.

An increasingly important application domain of network coding is mobile ad hoc networks. By having random mixture packets self-orchestrate multiple paths, network coding offers built-in error protection and adaptivity to topology changes due to joins, leaves, node or link failures, congestion, etc; by employing a flooding-type delivery, network coding can be implemented in a distributed fashion easily, whereas the creation and maintenance of distribution trees incurs notable signalling overhead, especially in a dynamic network, such as a mobile ad hoc network. In fact, these properties render network coding potentially useful for unicasting in mobile ad hoc networks as well.

The benefits of network coding come with a price of extra processing/implementation complexity. This work aim at complexity reduction techniques, which are useful for practical deployment of network coding, especially in low-end networks such as wireless ad hoc networks.

2. OVERVIEW OF MAIN RESULTS

For a destination $t \in T$, let $C_G(s,t)$ denote the minimum cardinality of an s-t-cut, where an s-t-cut is an edge set

$$\{e \in E : \mathrm{tail}(e) \in X, \mathrm{head}(e) \in V - X\} \qquad (1)$$

with $s \in X$ and $t \in V - X$ (an edge pointing from v to w is said to have $\mathrm{tail}(e) = v$ and $\mathrm{head}(e) = w$.) According to Menger's Theorem (Max-Flow = Min-Cut), $C_G(s,t)$ is equal to the maximum number of edge-disjoint s-t-paths (paths from s to t). Since the capacity (cardinality) of any s-t-cut is an upper-bound on the achievable rate for information transmission from s to t, $C_G(s,T) \equiv \min_{t \in T} C_G(s,t)$ is an upper-bound on the *multicast capacity*.

For the extreme case where $T = V - \{s\}$ (all nodes other than s are destinations), Edmonds [3] showed the maximum number of edge-disjoint spanning trees rooted at s is equal to $C_G(s,T)$. In other words, the broadcast capacity can be achieved by routing information with $C_G(s,T)$ trees.

Some conjectures have been made in the graph theory literature regarding possible generalizations of Edmonds' theorem to the case where there exist *Steiner nodes* (nodes other than the source s and the destinations T); however,

these conjectures have been disproved. For example, Lovász [4] gave the graph in Fig. 1 to show there are no two edge-disjoint trees connecting s with T although $C_G(s,T)=2$.

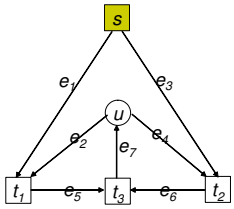


Fig. 1. An example graph G_1 [4] with $T=\{t_1,t_2,t_3\}$.

In this paper we constructively prove a generalization of Edmonds' Theorem. Without loss of generality, assume s has no incoming edges. Then the edges in G are classified into two categories: 1) edges entering Steiner nodes, which we call *Steiner edges*, and 2) edges entering destination nodes, which we call *non-Steiner edges*. We introduce a graph transformation called *hardwiring*. As illustrated in Fig. 3(c), hardwiring an edge e to edge f with $\mathrm{head}(f)=\mathrm{tail}(e)$ establishes a single predecessor for e; after this operation, the information on e has to come from f. Now we are ready to state the main theorem as follows.

Theorem 1 (Hardwiring Non-Steiner Edges) *Given a directed graph G, a source node s, and destination nodes T, all the non-Steiner edges of G can be* hardwired *while preserving connectivity, i.e., in the resulting graph $\ddot{G}$, $C_{\ddot{G}}(s',T)=C_G(s,T)$.*

For the example G_1 in Fig. 1, Fig. 2(b) gives a possible final graph $\ddot{G}_1$ satisfying the condition in Theorem 1.

Apart from being a generalization of Edmonds' Theorem, Theorem 1 is useful in reducing the complexity of a network coding system. Ahlswede et al. [1] showed that the multicast capacity is $C_G(s,T)$ and it can be achieved by network coding. Theorem 1 leads to the following generalization of Ahlswede et al's theorem: the multicast capacity $C_G(s,T)$ can be achieved by performing network coding on Steiner edges and traditional routing, as a degenerate form of coding, on non-Steiner edges [5]. Since coding is more complex than routing, this leads to a complexity saving.

We now use Theorem 1 and Ahlswede et al's theorem to prove this generalization of Ahlswede et al's theorem. First, according to Ahlswede et al's Theorem, there exists a network coding solution achieving a multicast rate $C_G(s,T)$ in $\ddot{G}$. In Fig. 2(b), a capacity-achieving network coding solution for $\ddot{G}_1$ is given by showing next to each edge the information on it. The source generates two symbols x_1 and x_2 in every time unit. x_1 is forwarded to t_1 and t_3; x_2 is forwarded to t_2 and t_3. Then t_3 mixes the received information by computing $x_1\oplus x_2$ ($\oplus$ is XOR), which is further forwarded to t_1 and t_2. Thus t_1 receives x_1 and $x_1\oplus x_2$ and can recover x_1 and x_2; similarly, t_2 can also recover x_1 and x_2. Hence the multicast capacity of 2.0 is achieved.

Since $\ddot{G}$ fixes all connections of non-Steiner edges in G, a network coding solution on $\ddot{G}$ maps directly into a network coding solution on G, which satisfies the additional constraint that the information on each non-Steiner edge is only produced by routing instead of mixing. This can be easily verified on the example in Fig. 2(b).

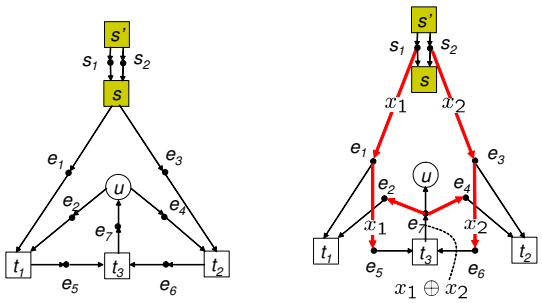
This result (coding on Steiner edges, routing on others) limits the edges where coding needs to be applied. It also leads to potential benefits in facilitating the construction of a capacity-achieving coding solution, since the search space is now smaller. For example, it may ease the task of deciding the operating field size.

In this paper, we establish Theorem 1 by proving the correctness of a procedure that repeatedly hardwires a non-Steiner edge to a Steiner edge or another non-Steiner edge that has already been hardwired. The proof consists of two parts: 1) proving that each step is connectivity-preserving, 2) proving that the procedure finishes hardwiring all the non-Steiner edges. The main challenge lies in the second part. For this part, we make use of a wire deletion operation that is more general than the hardwiring operation. Deleting a wire entering an edge e breaks one of its connections with the predecessor edges, whereas hardwiring e simultaneously breaks all but one connections with the predecessor edges. We prove by induction that all wires entering the non-Steiner edges that have not been hardwired can be deleted without losing the connectivity. The induction is performed in a special way: assuming that *any* subset of k wires across the cut can be deleted without losing the connectivity, we prove that any subset of $k+1$ wires across the cut can be deleted without losing the connectivity.

3. SOME GRAPH TRANSFORMATIONS

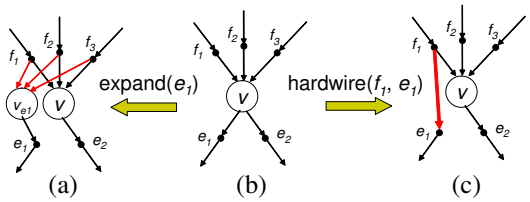
For ease in notation, let $h\equiv C_G(s,T)$. First, we add to G a new vertex s' and h *source edges*, $s_1,\ldots,s_h$; the resulting graph is denoted by $G[s,h]$. These source edges correspond to the h sub-streams of source information, each with unit rate. Next we introduce for each edge of $G[s,h]$ a new node, shown as a dot in Fig. 2(a), which "divides" the edge into two "halves". Denote the resulting graph by $\dot{G}$. Since there is a one-to-one mapping between a dot in $\dot{G}$ and an edge in $G[s,h]$, we refer to a dot in $V(\dot{G})$ (the vertex set of $\dot{G}$) by the name of the corresponding edge in $G[s,h]$. These dots play roles as the "connecting points" for the hardwiring operation.

As illustrated in Fig. 3(a), the operation $\texttt{expand}(e_1)$ introduces a new vertex v_{e_1} that has incoming edges from f_1, f_2, f_3 and an outgoing edge to e_1. Intuitively, this vertex v_{e_1} can be viewed as a duplicate of $v=\mathrm{tail}(e_1)$ that provides information only for e_1. We use the name *wire* to refer to edges such as (f_1,v_{e_1}), (f_2,v_{e_1}), (f_3,v_{e_1}), since they correspond to connections between two edges in $G[s,h]$. Then, we introduce the operation `deleteWire`: after expanding e_1, $\texttt{deleteWire}(f_2,e_1)$ deletes (f_2,v_{e_1}), hence



(a) (b)

Fig. 2. (a) The "dotted" version $\dot{G}_1$ obtained by subdividing all edges of $G_1[s,2]$. (b) A possible resulting graph $\ddot{G}_1$, after all non-Steiner edges have been hardwired.



(a) (b) (c)

Fig. 3. The expanding and hardwiring operation.

breaking the connection between f_2 and e_1.

Next we introduce a *hardwiring opeation*. As illustrated in Fig. 3(c), the operation $\texttt{hardwire}(f_1, e_1)$ deletes edge (v, e_1) and adds edge (f_1, e_1), where $v \equiv \text{tail}(e_1) \in V(G[s,h])$. Note that $\texttt{hardwire}(f_1, e_1)$ in Fig. 3(c) is equivalent to deleting all wires except (f_1, v_{e_1}) in Fig. 3(a). This operation enforces that the information on e_1 will only come from a single predecessor f_1, whereas previously the information on e_1 may be a mixture of the information on f_1, f_2, f_3.

4. A CONSTRUCTIVE PROOF OF THEOREM 1

While Theorem 1 has appeared in [5], here we provide a new constructive proof with a more direct algorithm.

Algorithm 1 performs a sequence of hardwiring operations on the graph $\dot{G}$ (as illustrated in Fig. 2(a)) to eventually arrive at a final graph $\ddot{G}$ (as illustrated in Fig. 2(b)), in which all non-Steiner edges E_1 have been hardwired. We use the notation $\dot{G}$ to refer to the "current" version $\dot{G}$, as the algorithm proceeds. Let $E_R \subseteq E_1$ denote the set of non-Steiner edges that have not been hardwired in the current $\dot{G}$. The procedure always checks if there exists a non-Steiner edge $e \in E_R$ that can be hardwired to a predecessor edge $f \in E_0 + E_1 - E_R$, i.e., a Steiner edge or a non-Steiner edge that has been hardwired, while preserving the required connectivity h to $\text{head}(e)$. If so, hardwire e to f and remove e from E_R. The algorithm stops when such a qualifying pair (f, e) cannot be found.

4.1. The Pivoting Lemma

Before proceeding to the main proof, we show a lemma [5], which is an easy consequence of Menger's Theorem (Max-Flow = Min-Cut). Fig. 4 illustrates the conditions in Lemma 1. We call the node v in Lemma 1 a *pivot*.

Algorithm 1 A Hardwiring Algorithm

1: $E_R := E_1$;
2: **while** $\exists (f,e), f \in E_0 + E_1 - E_R, e \in E_R, \text{head}(f) = \text{tail}(e)$ such that after performing $\texttt{hardwire}(f,e)$, there are still h edge-disjoint paths to $\text{head}(e)$. **do**
3: $\quad \texttt{hardwire}(f,e)$;
4: $\quad E_R := E_R - \{e\}$;
5: **end while**

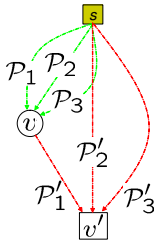


Fig. 4. The conditions in Lemma 1.

Lemma 1 (Pivoting Lemma [5]) *In a directed graph G, consider three vertices s, v, and v', where s is not adjacent to either v or v'. Suppose there are h edge-disjoint s-v-paths $\mathcal{P}_1, \ldots, \mathcal{P}_h$ and there are h edge-disjoint paths $\mathcal{P}'_1, \ldots, \mathcal{P}'_h$, where $\mathcal{P}'_1$ is from v to v' and $\mathcal{P}'_2, \ldots, \mathcal{P}'_h$ are from s to v'. Then there are h edge-disjoint paths from s to v'.*

4.2. Proof that Algorithm 1 is connectivity-preserving

First we prove that Algorithm 1 preserves the required edge-connectivity h to the destinations. Consider a generic iteration of while-loop in Algorithm 1. Let $\dot{G}$ and $\dot{G}'$ denote the graphs before and after executing $\texttt{hardwire}(f,e)$. Assuming the existence of h edge-disjoint paths to each destination $t \in T$ in $\dot{G}$, we need to prove that there exist h edge-disjoint paths to each destination $t \in T$ in $\dot{G}'$.

Due to the choice of wire (f,e), there are h edge-disjoint paths to $\text{head}(e)$ in $\dot{G}'$. For a destination $t \neq \text{head}(e)$, denote a set of h edge-disjoint paths to t in $\dot{G}$ by $\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}$. If e is not used in any of these paths, then $\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}$ are h edge-disjoint s'-t-paths in $\dot{G}'$. If e is used in one of these paths, say $\mathcal{P}_{t1}$, then $\mathcal{P}_{t1}$ has the following form

$$\mathcal{P}_{t1} = s \to \cdots \to e \to \text{head}(e) \to \cdots \to t. \qquad (2)$$

This is because in Algorithm 1 an edge can only be hardwired to an edge in $E_0 + E_1 - E_R$. Thus when $e \in E_R$ is hardwired, there is no edge hardwired to e and hence e has only one outgoing edge to $\text{head}(e)$. Then we can apply the pivoting lemma to show $C_{\dot{G}'}(s', t) \geq h$, using $\text{head}(e)$ as the pivot, the sub-path of $\mathcal{P}_{t1}$ from $\text{head}(e)$ to t as $\mathcal{P}'_1$, and $\mathcal{P}_{t2}, \ldots, \mathcal{P}_{th}$ as $\mathcal{P}'_2, \ldots, \mathcal{P}'_h$.

4.3. Proof that Alg. 1 hardwires all non-Steiner edges

It remains to prove when Algorithm 1 stops, all the non-Steiner edges have been hardwired. We prove this by contradiction. Let E_R refer to the resulting set of non-Steiner edges that have not been hardwired. Suppose $E_R \neq \emptyset$.

We then *expand* all the edges in E_R. This allows us to apply the wire deletion operation, which has finer control granularity than the hardwiring operation. Denote the resulting graph by $\dot{G}$. Consider the cut

$$\delta^{\text{in}}(X) \equiv \{(u,v) \in E(\dot{G}) : u \in V(\dot{G}) - X, v \in X\}, \quad (3)$$
$$X = E_R \cup \{v_e : e \in E_R\}. \quad (4)$$

This is illustrated in Fig. 5, where all the edges that have not been hardwired reside in the right hand side.

We will prove that after deleting $\delta^{\text{in}}(X)$, there are still h edge-disjoint s'-t-paths, $\forall t \in T$. This will lead to a contradiction since it implies Algorithm 1 should have proceeded hardwiring these edges.

We establish this result by inductively proving that deleting any subset of $\delta^{\text{in}}(X)$ consisting of k wires preserves the connectivity h. For $k = 0$, this is trivially true. Assuming this assertion is true for k, we now prove that after deleting an arbitrary subset $W \subseteq \delta^{\text{in}}(X)$ with $|W| = k+1$, there exist h edge-disjoint s'-t-paths, $\forall t \in T$. We prove the existence of h edge-disjoint paths first for destinations in

$$T_1 \equiv \{t \in T | \exists (f, v_e) \in W, \text{head}(e) = t\}, \quad (5)$$

and then for destinations in $T - T_1$, using the destinations in T_1 as pivots.

For any $t \in T_1$, choose a wire $(f, v_e) \in W$ with $\text{head}(e) = t$. By inductive assumption, there exist h edge-disjoint s'-t-paths $\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}$ in $\dot{G} - W + (f, v_e)$. If (f, v_e) is used in none of these paths, then $\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}$ are h edge-disjoint s'-t-paths in $\dot{G} - W$. If (f, v_e) is used in one of these paths, then it should have been possible to hardwire e to f in Algorithm 1. Thus a contradiction.

For any $t \in T - T_1$, pick an arbitrary wire $(f, v_{e_0}) \in W$. By inductive assumption, there exist h edge-disjoint s'-t-paths $\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}$ in $\dot{G} - W + (f, v_{e_0})$. If (f, v_{e_0}) is used in none of these paths, then $\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}$ are h edge-disjoint s'-t-paths in $\dot{G} - W$. If (f, v_{e_0}) is used in one of these paths, say $\mathcal{P}_{t1}$, then $\mathcal{P}_{t1}$ has the following form

$$\mathcal{P}_{t1} = s \to \cdots \to f \to \mathbf{v_{e_0}} \to \mathbf{e_0} \to \cdots \to \mathbf{v_{e_n}} \to \mathbf{e_n} \to \text{head}(e_n) \to \cdots \to t. \quad (6)$$

Note that the sub-path shown in boldface enters X via v_{e_0} and leaves X via e_n.

If $\text{head}(e_n) \in T_1$, then there exist h edge-disjoint paths to $\text{head}(e_n)$ in $\dot{G} - W$, according to our earlier discussions. Then we can apply the pivoting lemma to show $C_{\dot{G}-W}(s', t) \geq h$, using $\text{head}(e_n)$ as the pivot, the sub-path of $\mathcal{P}_{t1}$ from $\text{head}(e_n)$ to t as $\mathcal{P}'_1$, and $\mathcal{P}_{t2}, \ldots, \mathcal{P}_{th}$ as $\mathcal{P}'_2, \ldots, \mathcal{P}'_h$.

If $\text{head}(e_n) \notin T_1$, let m be the largest index in $\{0, \ldots, n\}$ such that $\text{head}(e_m) \in T_1$. This is well-defined since $(f, v_{e_0}) \in W$ and hence $\text{head}(e_0) \in T_1$. Since $\text{head}(e_{m+1}) \notin T_1$, none of the wires entering $v_{e_{m+1}}$ in $\dot{G}$ has been deleted in $\dot{G} - W$. Hence $v_{e_{m+1}}$ is essentially a duplicate of $\text{head}(e_m)$ in $\dot{G} - W$ (see Fig. 3). Then we can apply the pivoting lemma to show $C_{\dot{G}-W}(s', t) \geq h$, using $v_{e_{m+1}}$ as the pivot.

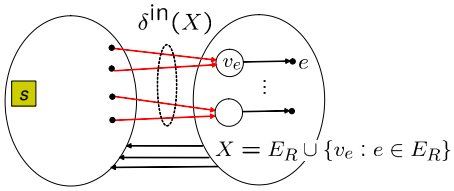


Fig. 5. A cut separating E_R from s.

4.4. Complexity

First note that we only need to test each pair (f, e) once, to check whether after `hardwire`(f, e), there are h edge-disjoint paths to $\text{head}(e)$. If the test result is negative, then testing `hardwire`(f, e) later will give the same answer: e should not be hardwired to f. Further observe that for a given $e \in E_R$, testing the pairs

$$\{(f, e) | \text{head}(f) = \text{tail}(e), f \in E_0 + E_1 - E_R\} \quad (7)$$

can be combined. First, expand e and delete the wires $\{(f', v_e)\}$ for all $f' \in E_R$. Then, a maximum flow algorithm can be used to test if there exist h edge-disjoint paths to $\text{head}(e)$. If the answer is no, then the pairs in (7) are all disqualified. If the answer is yes, then e can be hardwired to its predecessor edge in the h edge-disjoint paths.

Using a preprocessing technique in [6], a subgraph D of G can be found satisfying $C_D(s, T) = C_G(s, T)$ and the in-degree of each destination is h in D. Thus, for a given $e \in E_1$ with $\text{tail}(e) \in T$, the number of wires (f, e) to be tested is bounded by h. For a given $e \in E_1$ with $\text{tail}(e) \in V(G) - T$, all the wires $\{(f, e) | \text{head}(f) = \text{tail}(e)\}$ can be tested together. Thus the complexity is $O(h \cdot |E_1| \cdot T_{MF})$, where T_{MF} is the time to find a maximum flow in $\dot{G}$.

5. REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Information Theory*, vol. IT- 46, no. 4, pp. 1204-1216, July 2000.

[2] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," *41st Allerton Conf. Comm., Ctrl. and Computing*, Oct. 2003.

[3] J. Edmonds, "Edge-disjoint branchings," in *Combinatorial Algorithms*, ed. R. Rustin, Academic Press, NY, 1973.

[4] L. Lovász, "Connectivity in digraphs," *Journal of Combinational Theory B*, vol. 15, pp. 174-177, 1973.

[5] Y. Wu, K. Jain, and S.-Y. Kung, "A unification of Menger's and Edmonds' graph theorems and Ahlswede et al's network coding theorem," in *42nd Allerton Conf. Comm., Control and Computing*, Monticello, IL, Oct. 2004, Invited paper.

[6] P. Tong and E. L. Lawler, "A faster algorithm for finding edge-disjoint branchings," *Information Processing Letters*, 17(2):73-76, Aug. 1983.