A Unification of Menger's and Edmonds' Graph Theorems and Ahlswede et al's Network Coding Theorem

Yunnan Wu*, *Student Member, IEEE,* Kamal Jain†,

Sun-Yuan Kung*, *Fellow, IEEE.*

Abstract

The *multicast capacity* is the maximum rate that a sender can communicate common information to a set of receivers in a network (represented by a directed graph). A fundamental theorem in graph theory by Menger determines the unicast capacity from a sender to a receiver; another fundamental theorem in graph theory by Edmonds determines the broadcast capacity from a sender to all other nodes. In both extreme cases, the multicast capacity can be achieved by routing, i.e., replicating or forwarding, information. A recent work by Ahlswede et al established that the multicast capacity can be achieved by performing network coding, while in general it cannot be achieved by routing.

In this paper, we present a statement that unifies Menger's Theorem, Edmonds' Theorem, and Ahlswede et al's Theorem. Specifically, we show the multicast capacity can be achieved by performing conventional routing on *non-Steiner edges* (edges entering receivers) and (non-trivial) network coding on *Steiner edges*. To enforce these coding constraints, we introduce a graph transformation called "hardwiring"; hardwiring an edge means restricting it to connect with at most one of its predecessor edges. We present algorithms that "hardwires" all non-Steiner edges while preserving the required connectivity from the sender to each receiver.

Index Terms

This work was presented in part at the Allerton Conference on Communications, Control, and Computing, Monticello, IL, Sept. 2004 [1].

*Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544. Tel.:(609)258-3780. Fax:(609)258-3745. {yunnanwu, kung}@princeton.edu.

†Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399 USA. kamalj@microsoft.com

Multicast, network coding, network information flow, connectivity, flow, capacity.

I. Introduction

Consider the problem of information multicast, namely transmitting common information from a sender to a set of receivers, in a communication network. We are interested in maximizing the multicast throughput. We assume the communication network is represented by a directed graph $G = (V, E)$ with all edges having unit bit-rate; an example graph G_1 is given in Figure 1. For each ordered vertex pair (v, w), multiple edges are generally allowed so as to reflect the bit-rate differences of the communication links. Let the sender be $s \in V$ and the receivers be $T \subseteq V \backslash \{s\}$. A vertex that is neither the sender s nor a receiver in T is called a *Steiner node*.

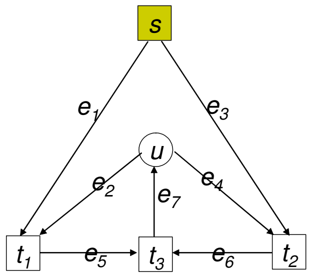

Fig. 1. An example graph G_1. The sender s is shown with a filled square, the receivers t_1, t_2, t_3 are shown with unfilled squares, and the only Steiner node u is shown with a circle. It can be checked that $C_{G_1}(s, t_1) = C_{G_1}(s, t_2) = C_{G_1}(s, t_3) = 2$ and $C_{G_1}(s, u) = 1$. Therefore, $C_{G_1}(s, T) = 2$ and $C_{G_1}(s, V \backslash \{s\}) = 1$. This example was first given by Lovász [2] to show that there are no two edge-disjoint Steiner trees connecting s with T although $C_{G_1}(s, T) = 2$.

One extreme case of this problem is when the receiver set T is a singleton $\{t\}$. A fundamental theorem by Menger [3] gives the maximum unicast rate from s to t.

Menger's Theorem on Packing Edge-Disjoint Paths (1927) [3]: *In a directed graph G, there are h edge-disjoint s-t-paths (i.e., paths from s to t) if and only if after deleting any $h - 1$ edges t is still reachable from s.*

Stated alternatively, the maximum number of edge-disjoint s-t-paths is equal to the minimum capacity (cardinality, denoted by $|\cdot|$) of an s-t-cut,

$$C_G(s, t) \equiv \min_{X:\ s \in X,\ t \in V \backslash X} |\{e \in E : \text{tail}(e) \in X, \text{head}(e) \in V \backslash X\}|, \tag{1}$$

where an s-t-cut is an edge set $\{e \in E : \mathrm{tail}(e) \in X, \mathrm{head}(e) \in V \backslash X\}$ with $s \in X$ and $t \in V \backslash X$ (an edge pointing from v to w is said to have $\mathrm{tail}(e) = v$ and $\mathrm{head}(e) = w$.). The minimum s-t-cut sets an upper-bound on the achievable rate for information transmission from s to t, which can be achieved by *routing*, i.e., replicating and forwarding, information along the $C_G(s,t)$ edge-disjoint paths. For general T, the follow quantity $C_G(s,T)$ is an upper-bound on the multicast rate from s to T:

$$C_G(s,T) \equiv \min_{t \in T} C_G(s,t). \tag{2}$$

In Figure 1, $C_{G_1}(s,T) = 2$ and thus the maximum achievable multicast rate from s to T is less than or equal to 2.0.

Another extreme case is when the receiver set T is $V \backslash \{s\}$, or equivalently, there are no Steiner nodes. Another fundamental theorem by Edmonds gives the maximum broadcast rate from s to all other nodes.

Edmonds' Theorem on Packing Spanning Trees (1972) [4]: *In a directed graph $G = (V,E)$, there are h edge-disjoint spanning trees rooted at $s \in V$ if and only if $h \leq C_G(s, V \backslash \{s\})$.*

In other words, the maximum rate for broadcasting information from s to all other nodes can be achieved by routing information with the $C_G(s, V \backslash \{s\})$ trees. In Figure 1, $C_{G_1}(s, V \backslash \{s\}) = 1$.

Recently, the concept of *network coding* has evolved as an interesting generalization of the traditional routing paradigm. With network coding, each node in the network is allowed to perform arbitrary operations on the information received on its incoming edges to produce information to be transmitted on its outgoing edges. In their pioneering theoretical work on network coding [5], Ahlswede *et al* demonstrated that it is in general suboptimal to restrict the interior nodes to perform only routing. Conversely, the *multicast capacity*, which is defined as the maximum rate that a sender can communicate common information to a set of receivers, is $C_G(s,T)$ and it can be achieved with network coding. Shortly afterwards, Li, Yeung, and Cai [6] showed that it is sufficient for the encoding functions at the interior nodes to be linear. Koetter and Médard [7] gave an algebraic characterization of linear encoding schemes and prove existence of linear time-invariant codes achieving the multicast capacity. The following statement summarizes these results.

Network Coding Theorems (2000-2003) [5]–[7]: *In a directed graph $G = (V, E)$ with unit capacity edges, the multicast capacity, which is defined as the maximum rate that a sender s can communicate common information to a set of receivers T, is $C_G(s, T)$ and it can be achieved with (linear time-invariant) network coding.*

We now use the example graph G_1 to briefly explain the meaning of linear time-invariant (LTI) network coding. Assume the network operates as a synchronous system with a discrete time index n running from 0 to $+\infty$. The sender s generates two information symbols (e.g., two bits), x_{1n} and x_{2n}, in the n-th time unit. Let the z-transform ($D = z^{-1}$) of the two streams of symbols be $x_1(D) \equiv \sum_{n=0}^{+\infty} x_{1n}D^n$ and $x_2(D) \equiv \sum_{n=0}^{+\infty} x_{2n}D^n$. Assume it takes one time unit for a symbol to propagate through an edge. Let $G[s, h]$ be the graph obtained by adding to G a new vertex s' and h *source edges*, $s_1, \ldots, s_h$. Figure 2(left) shows $G[s, 2]$. Figure 2(left) illustrates a LTI network coding solution achieving the multicast capacity. The information flowing on each edge is shown beside each edge. The i-th source edge s_i carries the i-th stream of symbols $x_i(D)$; this represents how information is originally injected to the network. As shown in Figure 2(left), $x_1(D)$ flows on e_1 and is then forwarded to e_5 after a unit delay, resulting in $x_1(D)D$ flowing on e_5; similarly, $x_2(D)D$ flows on e_6. With LTI network coding, the information flowing on an edge is generally allowed to be a causal linear combination of the information received on its incoming predecessor edges. For example, the received information from e_5 and e_6 is combined at t_3 to produce $[x_1(D) + x_2(D)]D^2$, which flows on e_7. Then, the mixed stream $[x_1(D) + x_2(D)]D^2$ is further forwarded to e_2 and e_4, after a unit delay. With this coding solution, receiver t_1 receives $x_1(D)D$ from e_1 and $[x_1(D)+x_2(D)]D^4$ from e_2; it can then recover both source streams $x_1(D)$ and $x_2(D)$ after certain delay. Similarly, it can be checked that receiver t_2 and t_3 can recover the source information. This confirms that this LTI coding solution achieves the multicast capacity.

Now let us take a more abstract look at how network coding is applied at each edge in this simple example. Figure 2(right) shows the *coding relations* among the edges, determined by the LTI network coding solution of Figure 2(left). We draw a dashed arrow from an edge f to e if $\text{head}(f) = \text{tail}(e)$ and f is involved in the linear combination producing the information flowing on e. Given a LTI coding solution, for each edge $e \in E$, the edges with dashed arrows pointing to e are said to be the *coding predecessors* of e. For example, the information flowing on e_5 is produced as a delayed version of the information flowing on e_1; thus e_5 has only one coding predecessor e_1. If an edge e has more than one coding predecessors, then the information flowing

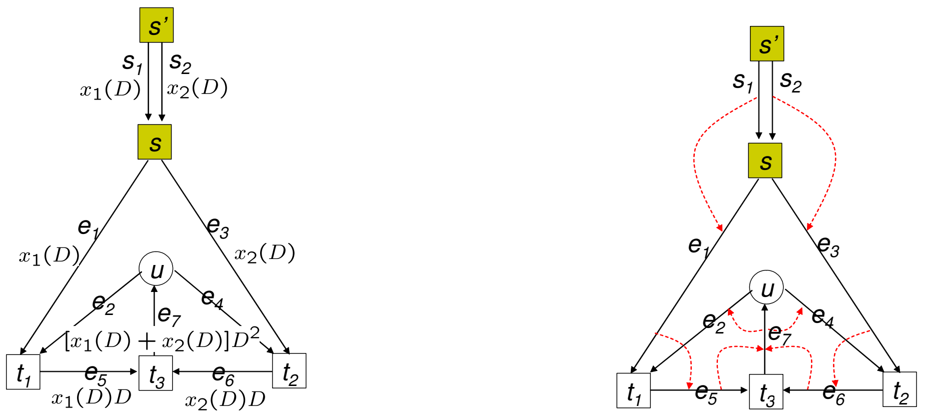

Fig. 2. (Left) An example linear time-invariant network coding solution achieving the multicast capacity. The graph shown is $G[s,2] \equiv (V \cup \{s'\}, E \cup \{s_1, s_2\})$. (Right) Coding relations among the edges, illustrated by dashed arrows.

on e is produced by "mixing" information on the coding predecessors; on the other hand, if an edge e has only one coding predecessor, say f, then we say that f is the *hardwired predecessor* of e. Intuitively speaking, network coding generalizes routing by allowing information to be mixed. By now, we have already known that when there is a single receiver or when all nodes other than the the sender are receivers, no mixing is necessary to achieving the multicast capacity. In this capacity-achieving LTI coding solution for G_1, which has one Steiner node, information mixing is only performed at e_7.

What distinguishes e_7 from other edges? In this example, only e_7 is entering a Steiner node and all other edges are entering receivers. We introduce the notion of *Steiner edges*. An edge entering a Steiner node is called a *Steiner edge* and an edge entering a receiver $t \in T$ is called a *non-Steiner edge*. Without essential loss of generality, we assume sender s has no incoming edges. Thus the edge set can be partitioned into Steiner edges and non-Steiner edges. Our main result of this paper is the following statement.

Statement 1 (A Unifying Statement):
In a directed graph $G = (V, E)$ with unit capacity edges, the multicast capacity from sender s to receivers T is $C_G(s, T)$ and it can be achieved with (linear time-invariant) network coding subject to the following constraints:

- *Each non-Steiner edge has at most one coding predecessor.*
- *Each edge has at most $|T|$ coding predecessors.*

This statement unifies Menger's Theorem [3], Edmonds' Theorem [4], and the network coding theorems [5]–[7]. Let us now explain why these are special cases of this statement. Let $h = C_G(s,T)$. First, let $T = \{t\}$. In this case, there is a capacity-achieving LTI network coding solution, where each edge has at most one coding predecessor according to the second constraint. Then the information flowing on each edge e in the network is $y_e(D) = \alpha(D)x_i(D)$ for some $i \in \{1,\ldots,h\}$. We remove all edges with $y_e(D) = 0$ from $G[s,h]$. Think of a coloring of the edges in the resulting sub-graph of $G[s,h]$ with h color indices $\{1,\ldots,h\}$: if $y_e(D) = \alpha(D)x_i(D)$, then edge e is assigned the i-th color. In the capacity-achieving solution, receiver t has incoming edges spanning all h colors. By backtracking from receiver t, we can find h edge-disjoint s-t-paths. This establishes Menger's Theorem [3] as a special case. Next, let $T = V\backslash\{s\}$. In this case, there is no Steiner edge and thus there is a capacity-achieving LTI network coding solution, where each edge has at most one coding predecessor according to the first constraint. Then the argument above can be applied again to show that by backtracking from each receiver $t \in T$, we can obtain h edge-disjoint spanning trees rooted at s. This establishes Edmonds' Theorem [3] as a special case.

As a generalization of known network information flow results, the unifying statement has the following implications. On the theoretical side, this result advances our understanding on the very structural features of graphs that render network coding critical to achieving the maximum throughput. On the practical side, this result implies that in a multicast system employing network coding such as in [8]–[11], the design of the "last hops" (i.e., edges entering receivers) may be simplified without compromising the optimal throughput.

Now let us briefly describe our proof of the unifying statement, which states that there exists capacity-achieving network coding solutions subject to additional coding constraints. The second constraint in the unifying statement is actually an easy corollary of known network coding results. We include it mainly for the sake of completeness. Hence we will focus on establishing the first constraint. To do so, we will prove a graph theoretic statement:

Statement 2 (Existence of Structurally Constrained Paths): *Given a directed graph G, a sender s, and receivers T, there are $C_G(s,T)$ edge-disjoint s-t-paths, $\forall t \in T$, satisfying the constraints that if a non-Steiner edge e is used in any of these $|C_G(s,T)| \cdot |T|$ paths, its predecessor has to be a fixed edge called the hardwired predecessor of e.*

We will use this graph theoretic statement in conjunction with the network coding theorems to establish the unifying statement. Thus, the problem of showing the existence of a network coding solution subject to the coding constraints is transformed into one of showing the existence of paths subject to structural constraints. We will enforce these structural constraints by a series of graph transformations that eventually results in a graph where all non-Steiner edges in the original graph have been *hardwired*. Hardwiring a non-Steiner edge e (to one of its predecessors) essentially enforces that information mixing, corresponding to non-trivial network coding, cannot be used to generate the information flowing on e.

More specifically, our proof of this graph theoretic statement is accompanied by a polynomial-time constructive procedure. Visit the non-Steiner edges in an arbitrary fixed order. When an non-Steiner edge e is visited, *hardwire* it to one of its predecessors and possibly *re-hardwire* some edges that have already been hardwired, while *preserving connectivity*, i.e., ensuring that in the resulting graph, there still exist $C_G(s,T)$ edge-disjoint paths to each receiver $t \in T$.

The rest of this paper is organized as follows. In Section I-A, we introduce some notations and terminologies. We state and prove our main theorem in Section II. Interestingly, based on the graph theoretic statement, we also find a second constructive procedure that hardwires all non-Steiner edges one by one without re-hardwiring. This is presented in Section III. Some discussions are made in Section IV. Finally, a conclusion is drawn in Section V.

A. Notations and Terminologies

For notational convenience, let $h \equiv C_G(s,T)$. Let $E_0 \equiv \{e : \text{head}(e) \in V(G[s,h]) - T\}$ and $E_1 \equiv \{e : \text{head}(e) \in T\}$ denote the set of Steiner edges and non-Steiner edges, respectively. Note that the source edges $s_1, \ldots, s_h$ are treated as Steiner edges as a notational convention. Having assumed sender s has no incoming edges in G, $E(G[s,h]) = E_0 + E_1$.

For a graph G, $V(G)$ and $E(G)$ denote its vertex set and edge set, respectively. For a directed graph $G = (V,E)$, let the edges going out of a vertex subset $X \subseteq V$ to its complement set $V \backslash X$ be $\delta^{\text{out}}(X) \equiv \{e \in E : \text{tail}(e) \in X, \text{head}(e) \in V \backslash X\}$. Let $\delta^{\text{in}}(X) \equiv \delta^{\text{out}}(V \backslash X)$. We often do not distinguish one-element vertex sets from its single element, e.g., $\delta^{\text{out}}(v) \equiv \delta^{\text{out}}(\{v\})$. We use subscripts (e.g., $\delta^{\text{out}}_G(X)$) to specify the graph G if necessary.

An *s-t-path* $\mathcal{P}$ in a directed graph $G = (V,E)$ is an alternating sequence

$$s = v_0, e_1, v_1, \ldots, e_n, v_n = t, \qquad v_i \in V, e_i \in E, i = 1, \ldots, n$$

and the defining edges and vertices are distinct. When no ambiguity can arise, we may treat it as a sequence of vertices and write

$$\mathcal{P} = s \to v_1, \ldots, \to t. \tag{3}$$

In a graph $G = (V, E)$, *subdividing* an edge $e \in E$ with a vertex u means replacing e with two edges $(\text{tail}(e), u)$ and $(u, \text{head}(e))$ where u is a new vertex.

II. A Constructive Proof

A. Some Graph Transformations

Let us begin by introducing some operations on the graph. Figure 3(left) gives an example graph $G_2[s, h]$, which is used to help illustrate the constructive proof.

As a preparatory step, we subdivide all edges of $G[s, h]$ and denote the resulting graph by $\dot{G}$. Figure 3(right) shows $\dot{G}_2$. As illustrated in Figure 3(right), each edge e of $G[s, h]$ is "split" into two "halves" by a new node, shown as a dot. 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 $V(\dot{G})$ by the name of the corresponding edge in $G[s, h]$. The vertex set of $\dot{G}$ is thus $V(G[s, h]) \cup E(G[s, h])$. An edge e of $G[s, h]$ from v to w is now replaced by two edges, (v, e) and (e, w).

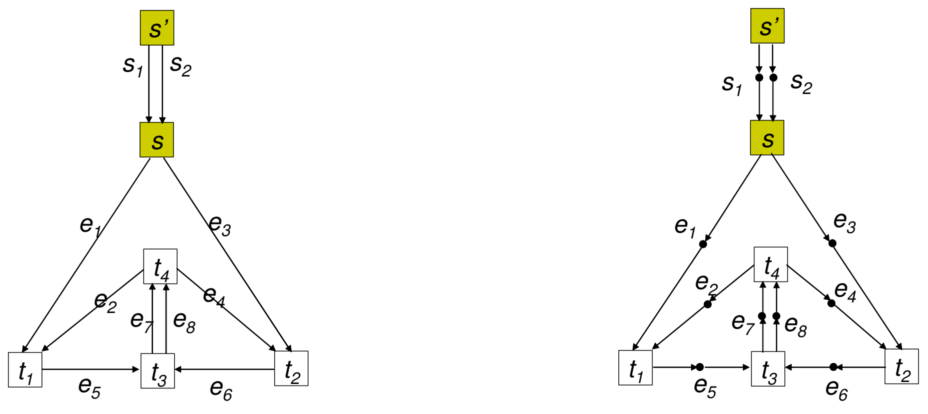

Fig. 3. (Left) An example graph $G_2[s, 2]$. (Right) The "dotted" version $\dot{G}_2$ obtained by subdividing all edges of $G_2[s, 2]$.

Next, we introduce a *hardwiring opeation*. As illustrated in Figure 4, the operation $\texttt{hardwire}(f_1, e_1)$ replaces edge (v, e_1) with edge (f_1, e_1), where $v \equiv \text{tail}(e_1) \in V(G[s, h])$. This operation essentially restricts e_1 to only forward information received from f_1. It enforces a structural constraint in Statement 2: whenever e_1 is used in one path, its predecessor has to be a fixed edge (f_1 in this case). We use the name *hardwire* to denote an edge (f_1, e_1) after hardwiring e_1 with f_1, since earlier the potential predecessor edges of edge e_1 are f_1, f_2, f_3, whereas now the only predecessor edge of e_1 is f_1. The hardwires are highlighted in the figures with thicker lines. As a degenerated case of the hardwiring operation, the operation $\texttt{hardwire}(\texttt{NULL}, e_1)$ simply deletes (v, e_1). After the operation $\texttt{hardwire}(f_1, e_1)$ (or $\texttt{hardwire}(\texttt{NULL}, e_1)$), we say the hardwired predecessor of e_1 is f_1 (or NULL) and write $\texttt{wiredPred}(e_1) = f_1$ (or $\texttt{wiredPred}(e_1) = \texttt{NULL}$).

We also introduce a *restoring operation*, which reverts the hardwiring operation back.

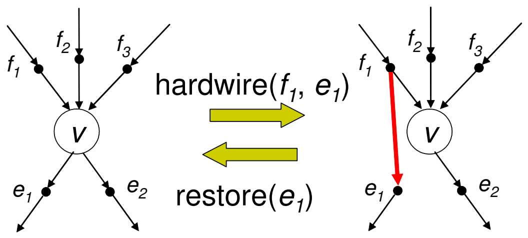

Fig. 4. The hardwiring operation and restoring operation.

Since each hardwiring operation enforces one constraint of Statement 2, Statement 2 is equivalent to the following Theorem 1.

Theorem 1 (Hardwiring Non-Steiner Edges):
Given a directed graph G, a sender s, and receivers T, all the non-Steiner edges of G can be hardwired while preserving connectivity, i.e., in the resulting graph $\ddot{G}$, there are $C_G(s,T)$ edge-disjoint s'-t-paths, $\forall t \in T$.

For the example graph $G_2[s,h]$, Figure 5 gives a possible final graph $\ddot{G}_2$ satisfying the conditions in Theorem 1.

B. Proof of the Unifying Statement with Theorem 1 and Network Coding Theorems

We now prove the unifying statement using Theorem 1 and the network coding theorems. According to Theorem 1, in the graph $\ddot{G}$, there are $C_G(s,T)$ edge-disjoint s'-t-paths, $\forall t \in T$. Thus, according to the network coding theorems, there exists a (linear time-invariant) network coding solution, achieving a rate $C_G(s,T)$ for multicasting from s' to t. Note that every dot in

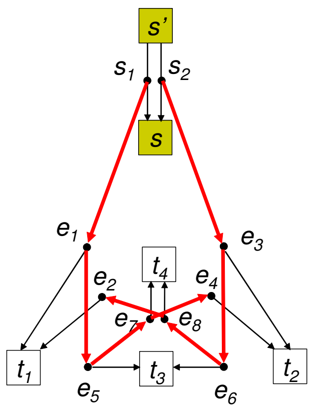

Fig. 5. A possible resultant graph $\ddot{G}_2$, after all non-Steiner edges have been hardwired.

$\ddot{G}$ has in-degree 0 or 1 and edge-disjoint s'-t-paths in $\ddot{G}$ map into edge-disjoint s-t-paths in G. Because of the structural relationship between $\ddot{G}$ (see Figure 5 for an example) and $\dot{G}$ (or G), it can be seen that a LTI network coding solution on $\ddot{G}$ maps directly into a LTI network coding solution on G; for a detailed explanation of LTI network coding, please refer to the Appendix. Furthermore, such a LTI network coding solution satisfies the constraint that each non-Steiner edge has only one coding predecessor, since in $\ddot{G}$, each dot corresponding to a non-Steiner edge $e \in E_1$ has been hardwired to one of its predecessors. In other words, the structure of $\ddot{G}$ rules out the possibility of performing non-trivial information mixing to produce the information flowing on the non-Steiner edges. This establishes that the multicast capacity $C_G(s, T)$ can be achieved by LTI network coding subject to the constraint that each non-Steiner edge has only one coding predecessor.

The second constraint of the unifying statement is used to incorporate Menger's Theorem as a special case. To establish that a LTI network coding solution exists subject to both constraints, we need some extra steps that transforms $\ddot{G}$ into a graph G''. We then show that G'' has a network coding solution without coding constraints, which maps into a network coding solution of G satisfying the coding constraints.

We introduce two more graph operations. As illustrated in Figure 6, 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 $v = \mathsf{tail}(e_1)$ 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

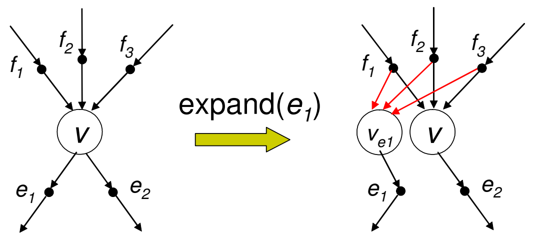

Fig. 6. The expanding operation.

they correspond to connections between two edges in $G[s, h]$. Then, we introduce the operation `deleteWire`: after expanding e_1, the operation `deleteWire`(f_2, e_1), i.e., deleting (f_2, v_{e_1}), breaks the connection between f_2 and e_1. Note that the operation `hardwire`(f_1, e_1) can be regarded as expanding e_1 followed by deleting all but one wire (f_1, v_{e_1}).

Denote by G' the resulting graph after expanding all the Steiner edges. In G', there are still $C_G(s, T)$ edge-disjoint s'-t-paths, $\forall t \in T$. Then, we remove from G' all the wires that have not been used in these paths; denote the resulting graph by G''. According to the network coding theorems, there exists a (linear time-invariant) network coding solution that achieves a rate $C_G(s, T)$ for multicasting from s' to t in G''. This solution maps into a network coding solution for G, satisfying the coding constraints.

C. A Hardwiring Procedure

Up to now, the problem of proving the unifying statement has been turned into one of proving Theorem 1, which states that all the non-Steiner edges can be hardwired while preserving connectivity. In this section, we present a hardwiring procedure described in Algorithm 1. Algorithm 1 performs a sequence of operations on the graph $\dot{G}$ (as illustrated in Figure 3(right)) to eventually arrive at a final graph $\ddot{G}$ (as illustrated in Figure 5), in which all non-Steiner edges E_1 have been hardwired. With slight abuse of notation, the notation $\dot{G}$ will be used to refer to the "current" version $\dot{G}$, as the algorithm proceeds.

The basic framework of the proposed hardwiring procedure is as follows. Visit the non-Steiner edges in an arbitrary fixed order. When e is visited, hardwire it to one of its predecessors. We want to ensure that after each step, there are still h edge-disjoint s'-t-paths, $\forall t \in T$. According to this procedure, a generic scenario to consider is that at certain point, some non-Steiner edges

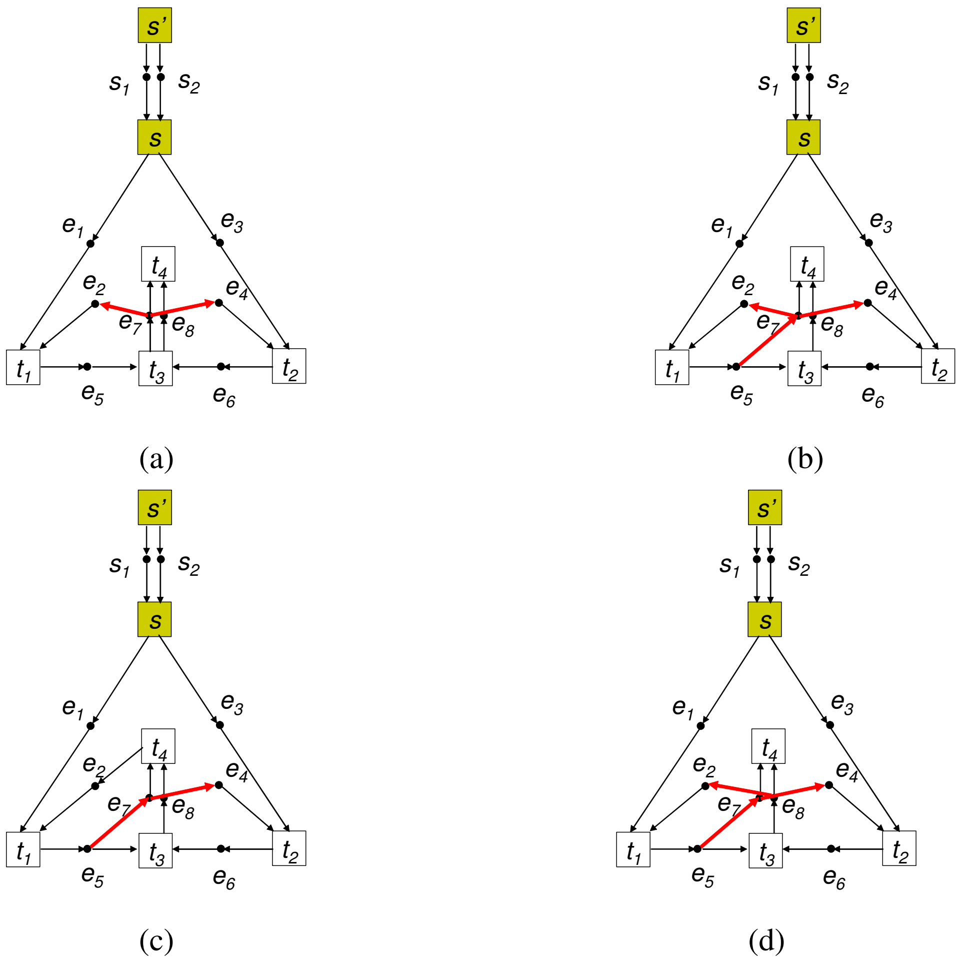

Fig. 7. (a) An example scenario during the execution of Algorithm 1. (b) The graph after hardwiring e_7 with e_5. (c) The graph after restoring e_2. (d) The graph after hardwiring e_2 with e_8.

have already been hardwired and another non-Steiner edge needs to be newly hardwired. The question then is how to newly hardwire this non-Steiner edge while preserving the required connectivity to all receivers.

Consider an example scenario shown in Figure 7(a), where e_2 has been hardwired to e_7, e_4 has been hardwired to e_7, and e_7 needs to be newly hardwired. It can be verified that in the current graph $\dot{G}$, there are $h = 2$ edge-disjoint s'-t-paths, $\forall t \in T$. Suppose e_7 is hardwired to e_5, then there no longer exist $h = 2$ edge-disjoint s'-t_1-paths. Suppose e_7 is hardwired to e_6, then there no longer exist $h = 2$ edge-disjoint s'-t_2-paths. To cope with this difficulty, we propose to re-hardwire some of the non-Steiner edges that were hardwired earlier. Note that e_7 is the root of a tree comprising the hardwires (e_7, e_2) and (e_7, e_4). First, we hardwire e_7 such that there are h edge-disjoint paths to $\mathrm{head}(e_7) = t_4$. This can be done by finding h edge-disjoint paths to $\mathrm{head}(e_7) = t_4$ in the current $\dot{G}$ and then identify the predecessor edge of e_7 as the hardwired predecessor of e_7. As shown in Figure 7(b), hardwiring e_7 with e_2 meets this requirement. Then, we will re-hardwire e_2 such that there are h edge-disjoint paths to $\mathrm{head}(e_2) = t_1$. Note that re-hardwiring can be decomposed into two operations: $\texttt{restore}(e_2)$ and $\texttt{hardwire}(f, e_2)$, where f will be selected from one of the predecessors of e_2. Figure 7(c) shows the current $\dot{G}$ after the operation $\texttt{restore}(e_2)$. It can be checked that there exist $h = 2$ edge-disjoint s'-t_1-paths in Figure 7(c); this observation will be generalized later in Lemma 1. In these two paths, e_2 is used after e_8, and therefore, after hardwiring e_2 with e_8, there still exist $h = 2$ edge-disjoint s'-t_1-paths, as shown in Figure 7(d). In a similar way, we can try to re-hardwire e_8 such that there are $h = 2$ edge-disjoint s'-t_2-paths; it turns out that e_8 will remain hardwired with e_7 and thus $\dot{G}$ remain unchanged. It can be further checked that there are $h = 2$ edge-disjoint s'-t-paths, $\forall t \in T$, in Figure 7(d). To summarize, a revised procedure that comprises three sub-steps:

1) hardwire e_7 to ensure connectivity for $\mathrm{head}(e_7) = t_4$;
2) re-hardwire e_2 to ensure connectivity for $\mathrm{head}(e_2) = t_1$;
3) re-hardwiring e_4 to ensure connectivity for $\mathrm{head}(e_4) = t_2$,

has been seen to ensure the required connectivity to all the receivers.

Algorithm 1 Hardwiring Non-Steiner Edges

```
1:  for $e \in E_1$ do
2:      Topologically sort the dots in the tree of hardwires rooted at $e$ as $e^0 = e, e^1, \ldots, e^K$;
3:      Delete from $\dot{G}$ the wires in this tree of hardwires rooted at $e$;
4:      for $k = 0, \ldots, K$ do
5:          restore($e^k$);
6:          $t := \mathrm{head}(e^k)$;
7:          $[\mathcal{P}_{t1}, \ldots, \mathcal{P}_{th}]$ := edgeDisjointPaths($\dot{G}, s', t, h$);
8:          wiredPred($e^k$) := predecessor($\mathcal{P}_{t1} \cup \ldots \cup \mathcal{P}_{th}, e^k$);
9:          hardwire(wiredPred($e^k$), $e^k$);
10:     end for
11: end for
```

The general procedure is more formally described in Algorithm 1. Line 2 assumes that in the current $\dot{G}$, e is the root of a tree of hardwires. Let P denote the *graph of hardwires*, which is derived from the current $\dot{G}$ as the sub-graph consisting only of the hardwires. The vertex set of P can be assumed to be $E(G[s,h])$. Since e has not been hardwired, the in-degree of e is 0 in P. Further note that every node in P has in-degree 0 or 1, since there can never be more than one hardwires ending at a given dot. These observations imply that e is the root of a tree of hardwires, or in the degenerated case, e is an isolated node in P. Line 2 enumerates the dots in the tree of hardwires rooted at e according to a topological order as $e^0 = e$, …, e^K. In the inner for-loop, we visit $e^0 = e$, …, e^K in this partial order. When e^k is visited, we restore the edge $(\mathrm{tail}(e^k), e^k)$ back to $\dot{G}$. Line 7 assumes that there are h edge-disjoint paths from s' to $\mathrm{head}(e^k)$; this assertion will be proven in the next subsection. Next, e^k is hardwired to the predecessor of e^k in the h paths. If e^k is not used in any of the h paths, then $\texttt{wiredPred}(e^k)$ is set to $\texttt{NULL}$.

One might note a subtle difference between Algorithm 1 and the earlier walk-through for the example graph. In Algorithm 1, we delete the wires in the tree rooted at e in P in Line 3, restore the connection between $\mathrm{tail}(e^k)$ and e^k in Line 5, and finally hardwire e^k in Line 9. Line 3 is introduced for the ease of argument in the proof. Since each execution of the inner for-loop can only add a wire to the graph, once the required connectivity to a receiver t is established after a certain execution of the inner for-loop, subsequent executions will not decrease the connectivity to t.

D. *Proof of Correctness*

1) The Pivoting Lemma: Before proceeding to the main proof, we show a lemma, which is an easy consequence of Menger's Theorem. Figure 8 illustrates the conditions in Lemma 1.

Lemma 1 (Pivoting Lemma): In a directed graph $G = (V, E)$, consider three vertices s, v, and v', where s is not adjacent to either v or v'. Suppose there are h edge-disjoint paths $\mathcal{P}_1$, …, $\mathcal{P}_h$, from s to v and there are h edge-disjoint paths $\mathcal{P}'_1$, …, $\mathcal{P}'_h$, 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' in G.

Proof: Consider any s-v'-cut $\delta^{\mathrm{out}}(X)$ with $s \in X$ and $v' \in V - X$. If $v \in V - X$, then the existence of h edge-disjoint s-v-paths implies that $|\delta^{\mathrm{out}}(X)| \geq h$. If $v \in X$, then the existence of h edge-disjoint paths $\mathcal{P}'_1$, …, $\mathcal{P}'_h$ going from X to $V - X$ implies that $|\delta^{\mathrm{out}}(X)| \geq h$. Thus any s-v'-cut contains at least h edges. By Menger's Theorem, there are h edge-disjoint s-v'-paths. ■

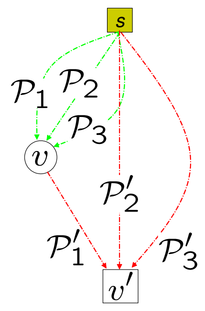

Fig. 8. The conditions in Lemma 1.

We call the node v in Lemma 1 a *pivot* and the path $\mathcal{P}'_1$ a *partial path* since it is not from s to v'.

2) Inductive Proof: We now inductively prove that each execution of the outer for-loop in Algorithm 1 is *connectivity-preserving*. In the following, we discuss one execution of the outer for-loop where non-Steiner edge e is currently visited.

Before executing Line 2, denote the current graph $\dot{G}$ by $\dot{G}_0$. By inductive assumption, $\dot{G}_0$ has h edge-disjoint s'-t-paths, $\forall t \in T$. Denote a set of h edge-disjoint s'-t-paths in $\dot{G}_0$ by $\mathcal{P}^0_{t1}$, …, $\mathcal{P}^0_{th}$. Denote the current `wiredPred`(e^k), $k = 0, \ldots, K$ by `wiredPred`$_0(e^k)$, $k = 0, \ldots, K$. After exiting from the inner for-loop, denote the graph $\dot{G}$ by $\dot{G}_1$. We need to prove that $\dot{G}_1$ has h edge-disjoint s'-t-paths, $\forall t \in T$.

For each receiver $t \in T$, there are three cases

1. $e \notin V(\mathcal{P}^0_{t1} \cup \ldots \cup \mathcal{P}^0_{th})$.
2. $e \in V(\mathcal{P}^0_{t1} \cup \ldots \cup \mathcal{P}^0_{th})$ and $\exists \tau \in \{0, \ldots, K\}, \text{head}(e^\tau) = t$.
3. $e \in V(\mathcal{P}^0_{t1} \cup \ldots \cup \mathcal{P}^0_{th})$ and $\nexists \tau \in \{0, \ldots, K\}, \text{head}(e^\tau) = t$.

For case 1, e is not used in any of the paths $\mathcal{P}^0_{t1}$, …, $\mathcal{P}^0_{th}$. Note that this implies that none of $e^0, \ldots, e^K$ is used in $\mathcal{P}^0_{t1}$, …, $\mathcal{P}^0_{th}$ because any s'-e^k-path must go through the root of the tree, e. An example of this case is t_3 in Figure 7(a). Since Line 3-Line 9 can only change `wiredPred`(e^k), $k = 0, \ldots, K$, $\mathcal{P}^0_{t1}$, …, $\mathcal{P}^0_{th}$ continue to serve as h edge-disjoint s'-t-paths in $\dot{G}_1$.

For case 2, e is used in one of the paths $\mathcal{P}^0_{t1}$, …, $\mathcal{P}^0_{th}$ and t is the head of a certain edge e^τ in the tree rooted at e. Without loss of generality, assume $e \in \mathcal{P}^0_{t1}$. An example of this case is

t_1 in Figure 7(a). Note that $\mathcal{P}^0_{t1}$ must be in the following form

$$\mathcal{P}^0_{t1} = s' \rightarrow, \ldots, \rightarrow \mathbf{e} \rightarrow, \ldots, \rightarrow \mathbf{e}^\tau \rightarrow t, \tag{4}$$

where the part of the path that intersects with the tree has been shown in bold face. For example, in Figure 7(a), $e = e_7$, $e^\tau = e_2$, and

$$\mathcal{P}^0_{t1} = s' \rightarrow s_2 \rightarrow s \rightarrow e_3 \rightarrow t_2 \rightarrow e_6 \rightarrow t_3 \rightarrow \mathbf{e_7} \rightarrow \mathbf{e_2} \rightarrow t_1, \tag{5}$$

$$\mathcal{P}^0_{t2} = s' \rightarrow s_1 \rightarrow s \rightarrow e_1 \rightarrow t_1. \tag{6}$$

We cover this case by proving that for $k = 0, \ldots, K$, after e^k is visited in the inner for-loop, there exists h edge-disjoint paths from s' to $t = \text{head}(e^k)$. As mentioned earlier, since each execution of the inner for-loop can only add a wire to the graph, once the required connectivity to a receiver t is established after a certain execution of the inner for-loop, subsequent executions will not decrease the connectivity to t.

We prove this by induction over k, $k = 0, \ldots, K$. First consider $k = 0$. Since deleting the wires in the tree of hardwires rooted at e does not affect $\mathcal{P}^0_{t1}, \ldots, \mathcal{P}^0_{th}$ for $t = \text{head}(e)$, there are h edge-disjoint paths from s to $t = \text{head}(e)$ when $\texttt{edgeDisjointPaths}(\dot{G}, s', t, h)$ is executed. Thus we can hardwire e and guarantee there are h edge-disjoint paths from s to t. For $k > 0$, let $t' = \text{head}(\texttt{wiredPred}_0(e_k))$. Since $\texttt{wiredPred}_0(e_k) \in E_1$, $t' \in T$. By assumption, there are h edge-disjoint paths to $\text{head}(e^{k'})$, $\forall k' \in \{0, \ldots, k-1\}$. Since the order of $e^0, \ldots, e^K$ is a partial order, there are h edge-disjoint paths to t' in particular. We then apply Lemma 1 with $v = t'$, $v' = \text{head}(e_k) \in T$, $\mathcal{P}'_i = \mathcal{P}^0_{ti}$, $i = 2, \ldots, h$ and

$$\mathcal{P}'_1 = t' \rightarrow e^k \rightarrow \text{head}(e^k). \tag{7}$$

For example, in Figure 7(c),

$$\mathcal{P}'_1 = t_4 \rightarrow e_2 \rightarrow t_1 \tag{8}$$

$$\mathcal{P}'_2 = s' \rightarrow s_1 \rightarrow s \rightarrow e_1 \rightarrow t_1. \tag{9}$$

This establishes that there are h edge-disjoint paths from s to $t = \text{head}(e_k)$ when executing $\texttt{edgeDisjointPaths}(\dot{G}, s', t, h)$. Thus we can hardwire e^k and guarantee there are h edge-disjoint paths from s to $\text{head}(e^k)$.

For case 3, e is used in one of the paths $\mathcal{P}^0_{t1}$, …, $\mathcal{P}^0_{th}$ and t is not the head of any edge e^k in the tree rooted at e. An example is t_5 in Figure 9(left). Assume $e \in \mathcal{P}^0_{t1}$. Let the last intersection

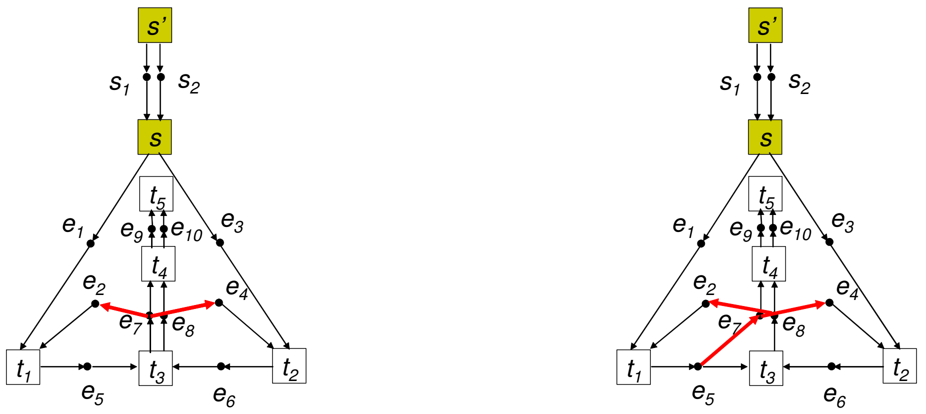

Fig. 9. (left)An example receiver in case 3 is t_5. (right) The graph after (re)-hardwiring e_7, e_2, and e_4. The existence of $h = 2$ edge-disjoint s'-t_5-paths is guaranteed by the pivoting lemma.

of $\mathcal{P}^0_{t1}$ with $\{e^0 = e, \ldots, e^K\}$ be e^τ. Then $\mathcal{P}^0_{t1}$ is in the following form

$$\mathcal{P}^0_{t1} = s' \to, \ldots, \to \mathbf{e} \to, \ldots, \to \mathbf{e}^\tau \to \text{head}(e^\tau) \to, \ldots, \to t, \tag{10}$$

For example, in Figure 9(left),

$$\mathcal{P}^0_{t1} = s' \to s_1 \to s \to e_1 \to t_1 \to e_5 \to t_3 \to \mathbf{e_7} \to t_4 \to e_9 \to t_5, \tag{11}$$

$$\mathcal{P}^0_{t2} = s' \to s_2 \to s \to e_3 \to t_2 \to e_6 \to t_3 \to e_8 \to t_4 \to e_{10} \to t_5. \tag{12}$$

Note that the immediate successor vertex of e^τ in $\mathcal{P}^0_{t1}$ is $\text{head}(e^\tau) \in T$ since there are no other edges hardwired to e^τ, which is a leaf in the tree of hardwires rooted at e. We then apply the pivoting lemma with $v = \text{head}(e^\tau)$, $v' = t$, $\mathcal{P}'_i = \mathcal{P}^0_{ti}$, $i = 2, \ldots, h$ and $\mathcal{P}'_1$ being the sub-path of $\mathcal{P}^0_{t1}$ that goes from $\text{head}(e^\tau)$ to t. This establishes that there are h edge-disjoint s-t-paths.

E. Complexity

First, the number $h = C_G(s, T) = \min_{t\in T} C_G(s, t)$ can be found in $O(|T| \cdot h \cdot |E|)$, using Ford and Fulkerson's augmenting path method; see, e.g., [12]. For $i = 1, \ldots$, find the i-th augmenting path for all $t \in T$. Stop when the $h + 1$-th augmenting path does not exist for some $t \in T$.

The running time of Algorithm 1 is dominated by the time required for the procedure calls of `edgeDisjointPaths`$(\dot{G}, s', t, h)$. Each such procedure call finds h edge-disjoint paths from s' to t in $\dot{G}$. Since $\dot{G}$ has at most $2|E|$ edges, `edgeDisjointPaths`$(\dot{G}, s', t, h)$ can be done in $O(h \cdot |E|)$ with an augmenting path method for finding maximum flow.

The outer for-loop of Algorithm 1 is called $|E_1|$ times. Each time a non-Steiner edge $e \in E_1$ is visited, the non-Steiner edges in the tree of hardwires rooted at e are (re-)hardwired. By a slight modification of Algorithm 1, in any tree of P, the corresponding receivers (heads) of the non-Steiner edges can be made distinct. Then, the number of non-Steiner edges in the tree is less than or equal to $|T|$. More specifically, we can perform one extra check before adding a hardwire $(\texttt{wiredPred}(e^k), e^k)$ in Algorithm 1. Assume by induction that the associated receivers of non-Steiner edges in the tree of P containing $\texttt{wiredPred}(e^k)$ are distinct. Let the root of this tree be e'. If this tree already contains an edge e'' with $\mathrm{head}(e'') = \mathrm{head}(e^k) = t \in T$, then we do not need to add the hardwire $(\texttt{wiredPred}(e^k), e^k)$. This is because we can replace the sub-path in $\mathcal{P}_{t1} \cup \ldots \cup \mathcal{P}_{th}$ from e' to e^k and then to t by the path from e' to e'' and then to t and still have h edge-disjoint paths.

Algorithm 1 can be applied with any arbitrary order of the non-Steiner edges E_1. To reduce the computational cost, we should try to sort the non-Steiner edges in an order that results in a small number of calls to $\texttt{edgeDisjointPaths}(\dot{G}, s', t, h)$. Let G_T denote the sub-graph of G induced by the receivers T, i.e., $E(G_T) = \{e \in G | \, \mathrm{head}(e) \in T, \mathrm{tail}(e) \in T\}$. If G_T is acyclic, we can sort the non-Steiner edges by a partial order. With this order, each $e \in E_1$ can be *guaranteed* to be an isolated vertex in P when it is visited, for any possible outcome of $\texttt{edgeDisjointPaths}(\dot{G}, s', t, h)$. Therefore, re-hardwiring is avoided if G_T is acyclic. If G_T is cyclic, we can decompose it into strongly connected components (SCC). It is well-known that if each SCC is contracted into one summary node, the resulting coarse-level graph is acyclic. Thus, we can sort the non-Steiner edges by a partial order of SCCs so as to reduce the complexity of re-hardwiring. Furthermore, we can apply some greedy heuristics at run-time. For example, we can choose a non-Steiner edge with the least number of descendants. This technique may be applied to decide the ordering inside each SCC.

Therefore, the overall complexity of hardwiring all non-Steiner edges with Algorithm 1 is $O(|E_1| \cdot |T| \cdot h \cdot |E|)$. If G_T is acyclic, then the overall complexity of hardwiring all non-Steiner edges is $O(|E_1| \cdot h \cdot |E|)$.

III. A Direct Hardwiring Procedure

Algorithm 1 visits the non-Steiner edges in an arbitrary order. When e is visited, it is newly hardwired and some other non-Steiner edges may be re-hardwired. A natural question to ask is whether re-hardwiring is indeed necessary. Figure 7(a) showed that re-hardwiring could be necessary if Algorithm 1 is used with an arbitrary order of non-Steiner edges. However, this does not rule out the possibility of performing hardwiring "more carefully" to avoid reaching situations such as in Figure 7(a).

We now present a direct hardwiring algorithm that avoids re-hardwiring. Let $E_R \subseteq E_1$ denote the set of non-Steiner edges that have not been hardwired in the current $\dot{G}$. The algorithm is given in Algorithm 2. 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 edge-connectivity h to $\mathrm{head}(e)$. If so, hardwire e to f and remove e from E_R. The algorithm stops when such a qualifying wire (f, e) cannot be found.

Algorithm 2 A Direct Hardwiring Algorithm

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

A. *Proof of Correctness*

In Algorithm 2, an edge $e \in E_R$ can only be hardwired to an edge $f \in E_0 + E_1 - E_R$. Thus, when $e \in E_R$ is hardwired, there are no other edges hardwired to e. If after hardwire(f, e), there are h edge-disjoint paths to $\mathrm{head}(e)$, then there are h edge-disjoint paths to any receiver $t \in T$, according to the proof of Algorithm 1's correctness (note that case 2 will never occur with Algorithm 2).

It remains to prove that when Algorithm 2 stops, all the non-Steiner edges have been hardwired. Let $\dot{G}$ refer to the resulting graph when Algorithm 2 stops and E_R refer to the resulting set of non-Steiner edges that have not been hardwired. We prove this by contradiction. Suppose $E_R \neq \emptyset$. Then, for each $e \in E_R$, we replace two edges $(\mathrm{tail}(e), e)$ and $(e, \mathrm{head}(e))$ by one edge from

$\mathrm{tail}(e)$ to $\mathrm{head}(e)$, which we denote by the same symbol e. Denote the resulting graph by $\bar{G}$. Note that in $\bar{G}$, the predecessor edge of an edge $e \in E_R$ can be either another edge in E_R or an edge in the form $(f, \mathrm{tail}(e))$, where $f \in E_0 + E_1 - E_R$. Figure 10 illustrates this.

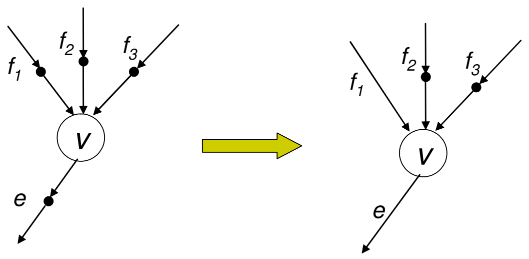

Fig. 10. The transformation from $\dot{G}$ to $\bar{G}$. Assume $f_1, e \in E_R$, $f_2, f_3 \in E_0 + E_1 - E_R$.

Applying Theorem 1 with $\bar{G}$ as the original graph G, we know that all the non-Steiner edges can be hardwired while preserving the required connectivity h to all the receivers. If according to this hardwiring solution, an edge $e \in E_R$ is hardwired to an edge $(f, \mathrm{tail}(e)) \in E(\bar{G})$, $f \in E_0 + E_1 - E_R$, then the operation `hardwire`(f, e) would have satisfied the test in Algorithm 2 and thus the algorithm should have been able to proceed further. Thus a contradiction. If according to this hardwiring solution, all edges in E_R are hardwired to edges in E_R, then the set E_R cannot be reached by s'. Thus the entire set E_R can be deleted from $\dot{G}$, or equivalently, hardwired to `NULL`, while preserving the required connectivity. Thus a contradiction.

B. Complexity

Algorithm 2 can have different implementations, depending on how the candidate wires (f, e) are enumerated and tested. For example, we can control the procedure to maximally grow one tree of hardwires before growing another one. Next, we present some observations that lead to low-complexity implementations.

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 $\mathrm{head}(e)$. 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

$$\{(f, e) | \, \mathrm{head}(f) = \mathrm{tail}(e), f \in E_0 + E_1 - E_R\} \tag{13}$$

can be combined: in $\dot{G}$, disallow the connections between e and its predecessor edges in E_R and then try to find h edge-disjoint paths to $\mathrm{head}(e)$.

Based on these observations, we give a possible implementation of Algorithm 2 in Algorithm 3. In Algorithm 3, each edge in E_R has an associated boolean property, active(e), indicating whether the set (13) contains a pair that has not been tested. Initially, only the non-Steiner edges originated from s or a Steiner node are active. Then, we always consider the active edges in E_R to see if the set (13) contains a pair that can be hardwired while preserving connectivity. If the answer is yes, then e is hardwired and removed from E_R. In addition, this activates the successor edges of e in E_R, since the pair (e, e'), $e' \in E_R$, $\mathrm{head}(e) = \mathrm{tail}(e')$, has not been tested. If the answer is no, then e becomes inactive since the pairs in the current (13) have been tried.

Algorithm 3 A Possible Implementation of Algorithm 2

```
1:  $E_R := E_1$;
2:  active($e$) :=true, for all $e : \mathrm{tail}(e) \in V - T, \mathrm{head}(e) \in T$;
3:  active($e$) :=false, for all $e : \mathrm{tail}(e) \in T, \mathrm{head}(e) \in T$;
4:  **while** $\exists e : e \in E_R$ && active($e$) :=true **do**
5:      expand($e$);
6:      deleteWire($f', e$), for all $f' : f' \in E_R, \mathrm{head}(f') = \mathrm{tail}(e)$;
7:      $[\mathcal{P}_1, \ldots, \mathcal{P}_{h'}] :=$ edgeDisjointPaths($\dot{G}, s', \mathrm{head}(e), h$);
8:      restore($e$);
9:      **if** $h' == h$ **then**
10:         wiredPred($e$) := predecessor($\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_h, e$);
11:         hardwire(wiredPred($e$), $e$);
12:         $E_R := E_R - e$;
13:         active($e'$):=true, for all $e' : \mathrm{head}(e) = \mathrm{tail}(e'), e' \in E_R$;
14:     **else**
15:         active($e$):=false;
16:     **end if**
17: **end while**
```

Tong and Lawler [13] showed that a preprocessing step can be used to reduce the complexity for finding the maximum number of edge-disjoint spanning trees. They showed that given a directed graph $G = (V, E)$ and a root vertex s, a subgraph D that is the union of $C_G(s, V - s)$ edge-disjoint spanning trees rooted at s, can be found in time $O(C_G(s, V - s) \cdot |V| \cdot |E|)$. This preprocessing is as follows. For each non-root vertex v, find $C_G(s, V - s)$ edge-disjoint s-v-paths and delete the edges entering v that is on none of these paths. This preprocessing can also be applied to the current context to find a subgraph D of G with $C_D(s, T) = C_G(s, T)$ and the in-degree of each receiver t is $h = C_G(s, T)$. For each receiver $t \in T$, find h edge-disjoint s-t-paths and delete the edges entering t that is on none of these paths.

With this preprocessing, the in-degree of each receiver t is $h = C_G(s, T)$. Thus, for a given $e \in E_1$ with $\text{tail}(e) \in T$, i.e., edges going from a receiver node to another receiver node, the number of pairs (f, e) to be tested is bounded by h

$$|\{(f, e)|\, \text{head}(f) = \text{tail}(e)\}| \leq h. \tag{14}$$

During the execution of Algorithm 3, these edges may switch between active and inactive state for at most h times. Thus, `edgeDisjointPaths`$(\dot{G}, s', \text{head}(e), h)$ will be called at most h times. For a given $e \in E_1$ with $\text{tail}(e) \in V - T$, i.e., edges going from a non-receiver node to a receiver node, all the pairs $\{(f, e)|\, \text{head}(f) = \text{tail}(e)\}$ will be tested in one call of `edgeDisjointPaths`$(\dot{G}, s', \text{head}(e), h)$. Therefore, the total number of calls of `edgeDisjointPaths` is $O(h \cdot |E_1|)$. Thus, the overall complexity is $O(h \cdot |E_1| \cdot h \cdot |E|)$.

IV. Discussions

A. Splitting Off Operation

Interestingly, the hardwiring operation introduced in this paper can be viewed as an asymmetric relaxation of a well-known *splitting off* operation in graph theory. In a directed graph, *splitting off* a pair of edges e_1, f_1 with $\text{head}(f_1) = \text{tail}(e_1)$ means that we replace e_1 and f_1 by a new edge $\bar{e}$ with $\text{tail}(\bar{e}) = \text{tail}(e), \text{head}(\bar{e}) = \text{head}(f)$. Figure 11 illustrates this operation on $\dot{G}$.

The splitting-off operation was introduced by Lovász for undirected graphs [14]. Later Mader [15] derived some powerful results concerning splitting while preserving the local edge-connectivity in undirected graphs. Mader [16] also proved a splitting result for directed graphs. Under certain conditions, the splitting off operation is connectivity-preserving. The splitting off operation proves to be a very useful tool for establishing results about graph connectivity, especially for undirected

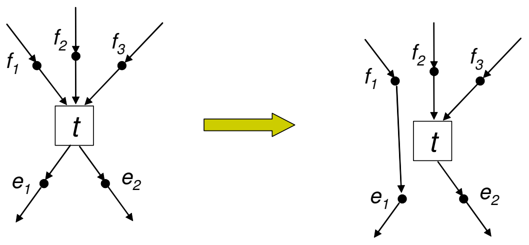

Fig. 11. The splitting off operation

graphs. For example, it has been used to derive a 2-factor approximation algorithm for packing Steiner trees in undirected graphs [17] and a generalized version of Edmonds' Theorem [18]. However, in the presence of Steiner nodes, it seems unclear whether the splitting off operation can be applied to prove Theorem 1, since its prior applications often require some global connectivity conditions to be fulfilled.

In comparison, the splitting off operation does not allow one-to-many local connection between an edge and its successors, whereas the hardwiring operation does. We look forward to future uses of this hardwiring operation to establish stronger results than those by the splitting off operation.

V. Conclusion

In this paper, we presented a statement that unifies Menger's Theorem on maximally packing edge-disjoint paths, Edmonds' Theorem on maximally packing edge-disjoint spanning trees, and Ahlswede et al's network coding theorem on maximizing the achievable multicast rate. We defined *Steiner edges* as edges entering Steiner nodes, denoted by a set E_1. We showed the multicast capacity can be achieved by performing routing on *non-Steiner edges* and coding on *Steiner edges*.

We introduced a *hardwiring* operation: Hardwiring an edge e means restricting e to connect with at most one of its predecessor edges. We proved that all non-Steiner edges can be hardwired while preserving the required connectivity to each receiver $t \in T$. Our proof is accompanied by an algorithm that visits the non-Steiner edges one by one in an arbitrary order and in each step, newly hardwires a non-Steiner edge and possibly re-hardwires some non-Steiner edges visited earlier. This algorithm runs in time complexity $O(|E_1| \cdot |T| \cdot h \cdot |E|)$ for general graphs, where $h = C_G(s, T)$. For graphs in which the subgraph induced by receivers T is acyclic, the algorithm runs in time $O(|E_1| \cdot h \cdot |E|)$. We also presented a direct hardwiring algorithm that avoids re-hardwiring, which runs in $O(h \cdot |E_1| \cdot h \cdot |E|)$ for general graphs.

Since Edmonds' Theorem [4] is a special case of the unifying statement, our proof gives a constructive proof to Edmonds' Theorem, which is different from other existing proofs and/or algorithms of Edmonds' Theorem [4], [13], [19]–[23].

ACKNOWLEDGEMENT

The authors would like to thank Dr. Philip A. Chou and Dr. László Lovász for the helpful discussions with them.

REFERENCES

[1] Y. Wu, K. Jain, and S.-Y. Kung. A unification of edmonds' graph theorem and ahlswede etc's network coding theorem. In *Proc. 42nd Allerton Conf. Communication, Control and Computing*, Monticello, IL, October 2004. Invited paper.

[2] L. Lovász. Connectivity in digraphs. *Journal of Combinational Theory B*, 15:174–177, 1973.

[3] K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:95–115, 1927.

[4] J. Edmonds. Edge-disjoint branchings. In *Combinatorial Algorithms, ed. R. Rustin*, pages 91–96, Academic Press, NY, 1973.

[5] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Information Theory*, IT-46(4):1204–1216, July 2000.

[6] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Trans. Information Theory*, IT-49(2):371–381, February 2003.

[7] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Networking*, 11(5):782–795, October 2003.

[8] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proc. 41st Allerton Conf. Communication, Control and Computing*, Monticello, IL, October 2003.

[9] Y. Wu, P. A. Chou, and K. Jain. Practical network coding. Technical report, Microsoft Research. In preparation.

[10] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *Proc. Int'l Symp. Information Theory*, Yokohama, Japan, June 2003. IEEE.

[11] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger. On randomized network coding. In *Proc. 41st Allerton Conf. Communication, Control and Computing*, Monticello, IL, October 2003.

[12] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume B. Springer, 2003.

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

[14] L. Lovász. On some connectivity properties of eulerian graphs. *Acta Math. Hungar.*, 28:129–138, 1976.

[15] W. Mader. A reduction method of edge-connectivity in graphs. In *Ann. Discrete Math.*, volume 3, pages 145–164, 1978.

[16] W. Mader. Konstruktion aller n-fach kantenzusammenhängenden digraphen. *Eupropean J. Combin.*, 3:63–67, 1982.

[17] K. Jain, M. Mahdian, and M. R. Salavatipour. Packing steiner trees. In *Proc. 14th Symp. on Discrete Algorithms (SODA)*. ACM-SIAM, 2003.

[18] J. Bang-Jensen, A. Frank, and B. Jackson. Preserving and increasing local edge-connectivity in mixed graphs. *SIAM J. Discrete Math.*, 8(2):155–178, 1995.

[19] R. E. Tarjan. A good algorithm for edge-disjoint branching. *Information Processing Letters*, 3(2):51–53, November 1974.

[20] L. Lovász. On two minimax theorems in graph theory. *Journal of Combinational Theory B*, 21:96–103, 1976.

[21] A. Frank. Kernel systems of directed graphs. *Acta Sci. Math.*, 41:63–76, 1979.

[22] W. Mader. On n-edge-connected digraphs. In *Ann. Discrete Math.*, volume 17, pages 439–441, 1983.

[23] H. Gabow. A matroid approach to finding edge connectivity and packing arboresences. In *Proc. 23rd ACM Symp. Theory of Computing*, pages 112–122, 1991.

APPENDIX

A. Linear Time-Invariant Network Codes

The concept of linear time-invariant (LTI) network coding was first introduced by Li, Yeung, and Cai [6]. In [7], Koetter and Médard proved existence of LTI codes achieving the multicast capacity. In the following, we explain the meaning of LTI network coding and review related results.

Before the explanation, we first review the notion of *line graph*. Let $L(G[s,h])$ be the *(directed) line graph* of $G[s,h] = (V \cup \{s'\}, E \cup \{s_1, \ldots, s_h\})$, which has vertex set $E(G[s,h]) = E \cup \{s_1, \ldots, s_h\}$ and edge set

$$\{(e', e) \in E(G[s,h]) \times E(G[s,h]) : \mathrm{head}(e') = \mathrm{tail}(e)\}\,.$$

For example, the line graph $L(G_1[s,h])$ for Figure 2(left) is shown in Figure 12(left). Vertex-disjoint paths in $L(G[s,h])$ map into edge-disjoint paths in $G[s,h]$.

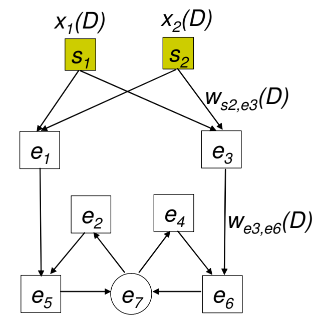

Fig. 12. The line graph $L(G[s,h])$ of $G[s,h] = (V \cup \{s'\}, E \cup \{s_1, \ldots, s_h\})$. Each Steiner edge is shown with a circle; each non-Steiner edge is shown with an unfilled square; each source edge is shown with a filled square.

Definition 1 (Linear Time-Invariant Network Coding):

Let $\mathbb{Q}(\mathbb{F}, D)$ be the field of rational functions of D (ratio of two polynomials) with coefficients in field $\mathbb{F}$, where

D carries the meaning of a unit delay. A rational function $a(D)$ is *realizable* if $a(0)$ is well-defined. Let $\mathbb{Q}^+(\mathbb{F}, D)$ denote the set of realizable rational functions.

A *linear time-invariant network coding assignment* is an assignment of *local encoding operators* in $\mathbb{Q}^+(\mathbb{F}, D)$ to the edges in $L(G[s,h])$. For $(u,v) \in E(L(G[s,h]))$, the assigned local encoding operator is denoted by $w_{u,v}(D)$.

Let the source stream of information be $\underline{x}(D) = [x_1(D), \ldots, x_h(D)]$, where $x_i(D)$ denotes the z-transform ($D = z^{-1}$) of a stream of symbols in field $\mathbb{F}$. The i-th source vertex s_i in $L(G[s,h])$ injects information $x_i(D)$ into the network. Assume a fixed ordering of the edges so that we can refer to edge e for $e = 1, \ldots, |E|$. Let $\underline{y}(D) = [y_1(D), \ldots, y_{|E|}(D)]$, where $y_e(D)$ denotes the the information flowing on edge $e \in E$. Assume each edge $e \in E$ has a unit propagation delay. The meaning of local encoding operators is that the information flowing on edge e is a linear combination of the information flowing on its incoming neighbors $N_L^-(e)$:

$$y_e(D) = \sum_{e' \in N_L^{\text{in}}(e)} w_{e',e}(D) D y_{e'}(D) + \sum_{s_i \in N_L^{\text{in}}(e)} w_{s_i,e}(D) x_i(D). \tag{15}$$

Putting the equations (15) for all $e \in E$ together, an assignment of LTI codes can be described as the following linear system

$$\underline{y}(D) = D\underline{y}(D)\mathcal{A}(D) + \underline{x}(D)\mathcal{B}(D), \tag{16}$$

where $\mathcal{A}(D) \in \mathbb{Q}^+(\mathbb{F}, D)^{|E|\times|E|}$ is the *(local) encoding matrix* and $\mathcal{B}(D) \in \mathbb{Q}^+(\mathbb{F}, D)^{h\times|E|}$ is the *input matrix*. Note the both $\mathcal{A}(D)$ and $\mathcal{B}(D)$ have structural restrictions due to the connectivity in $L(G[s,h])$. The (e', e) entry of $\mathcal{A}(D)$, $w_{e',e}(D)$, can be nonzero only if $e' \in N_L^{\text{in}}(e)$; the (i, e) entry of $\mathcal{B}(D)$, $w_{s_i,e}(D)$, can be nonzero only $\text{tail}(e) = s$. Note that we use the `mathcal` font, e.g., $\mathcal{A}(D)$, to indicate the structural restrictions that certain entries of the matrix have to be zero. Let

$$\mathcal{W}(D) \equiv \begin{bmatrix} \mathcal{A}(D) \\ \mathcal{B}(D) \end{bmatrix}, \tag{17}$$

whose allowed nonzero entries correspond to the nonzero entries in the adjacency matrix of $L(G[s,h])$.

Next let us examine how the signals $\underline{x}(D)$ are recovered at the receivers T for a linear time-invariant code assignment $\mathcal{W}(D)$ achieving multicast throughput h. From (16), $\underline{y}(D)$ is related to $\underline{x}(D)$ as

$$\underline{y}(D) = \underline{x}(D)\mathcal{B}(D)\left(\mathbf{I} - \mathcal{A}(D)D\right)^{-1}. \tag{18}$$

Define $\mathbf{Y}(D) \equiv \left[\mathcal{B}(D)\left(\mathbf{I} - \mathcal{A}(D)D\right)^{-1} \ \mathbf{I}_h\right]$, with the columns corresponding to vertices in $L(G[s,h])$. Define $S \equiv \{s_1, \ldots, s_h\}$. Denote the column of $\mathbf{Y}(D)$ corresponding to $v \in E \cup S$ by $\underline{Y}_v(D)$, which is said to be the *global encoding vector* at vertex v since it describes the cumulative effect of the encoding operations performed in the network on this vertex. Let $\mathcal{C}_t(D) \in \mathbb{Q}(\mathbb{F}, D)^{|E|\times h}$ be a *recovery matrix* for receiver $t \in T$ such that the signal is recovered after certain delay l_t

$$\underline{y}(D)\mathcal{C}_t(D) = \underline{x}(D)\mathcal{B}(D)\left(\mathbf{I} - \mathcal{A}(D)D\right)^{-1}\mathcal{C}_t(D) = \underline{x}(D)D^{l_t}. \tag{19}$$

Note that $\mathcal{C}_t(D)$ also has structural restrictions: the e-th row of $\mathcal{C}_t(D)$ can be non-zero only if $\text{head}(e) = t$. In addition, $\mathcal{C}_t(D)$ is required to be realizable. For a given $\mathcal{A}(D)$ and $\mathcal{B}(D)$, if there exists such a solution $\mathcal{C}_t(D)$, then we say that the source signals are *recoverable* at receiver t, or receiver t can recover the source signals. Note further that this recoverability condition can be checked easily based on the given $\mathcal{A}(D)$ and $\mathcal{B}(D)$: receiver t can recover the source signals if and only if the linear space of global encoding vectors observed by receiver t has dimension h, i.e., $\dim\left(\text{span}\{\underline{Y}_e(D),\ \text{for all } e : \text{head}(e) = t\}\right) = h$.

Summarizing the discussions above, a capacity achieving code assignment gives realizable matrices $\mathcal{A}(D)$ and $\mathcal{B}(D)$ that comply with their respective structural restrictions, such that all receivers can recover the source signals. Koetter and Médard [7] proved existence of linear time-invariant (LTI) codes achieving the multicast capacity. This is given in the following Theorem 2.

Theorem 2 (Existence of LTI Codes on Edges [7]):
In G, there exist a field $\mathbb{F}$ and $\mathcal{A}(D) = \mathcal{A} \in \mathbb{F}^{|E|\times|E|}$ and $\mathcal{B}(D) = \mathcal{B} \in \mathbb{F}^{h\times|E|}$ that comply with their respective structural restrictions due to the connectivity, such that all receivers can recover the source signals if and only if $h \leq C_G(s,T)$. Moreover, the field $\mathbb{F}$ can be chosen to be the Galois field $GF(2^m)$ for a sufficiently large m.