# Efficient Interlayer Network Codes for Fair Layered Multicast Streaming

Joerg Widmer, *Senior Member, IEEE, ACM*, Andrea Capalbo, Antonio Fernández Anta, *Senior Member, IEEE, ACM*, and Albert Banchs, *Senior Member, IEEE*

***Abstract*—Multilayer video streaming allows to provide different video qualities to a group of multicast receivers with heterogeneous receive rates. The number of layers received (and thus the receive rate) determines the quality of the decoded video stream. For such layered multicast streaming, network coding provides higher capacity than multicast routing. Network coding can be performed within a layer or across layers, and in general, interlayer coding outperforms intralayer coding. An optimal solution to a network-coded layered multicast problem may require decoding of the network code at interior nodes to extract information to be forwarded. However, decoding consumes resources and introduces delay, which is particularly undesirable at interior nodes (the routers) of the network. In this paper, we thus focus on the interlayer network coding problem without decoding at interior nodes. We show that the problem is NP-hard and propose a heuristic algorithm for rate allocation and coding based on the Edmonds–Karp maximum flow algorithm. We prove that our algorithm ensures decodability of the information received and provides some fairness properties. Finally, we perform extensive simulations and show that our algorithm may even outperform other heuristics that do require decoding at interior nodes.**

***Index Terms*—Interlayer network coding, layered multicast, network coding, video streaming.**

## I. Introduction

LARGE-SCALE video streaming is rapidly gaining in popularity. In case all receivers of a video stream are able to receive the same bit rate (and hence video quality), IP multicast is a suitable solution [2], an approach used in today's IP-TV broadcasting systems. When multicasting to receivers with heterogeneous reception requirements or receive rates (multirate multicast), there are two common approaches to efficiently distribute video streams. With both approaches, the video is distributed over several multicast flows, and the number of flows a receiver obtains determines the video quality. Layered coding splits the stream into multiple dependent layers. The base layer provides a basic video quality, and each additional enhancement layer received refines this quality [3]. A higher enhancement layer can only be decoded if the base layer and all lower enhancement layers are already available. In contrast, with multiple description coding, the video is split into multiple independently decodable substreams [4]. While multiple description coding offers more flexibility in terms of layers received and hence is very robust to loss of some layers, layered coding provides higher coding efficiency and, for this reason, is much more widespread. The most prominent example of a layered codec is the Scalable Video Coding (SVC) extension to H.264 that offers temporal, spatial, and signal-to-noise ratio (SNR) scalability and thus supports very fine-grained video quality adaptation [3].

For both single-rate as well as multirate multicast, network coding [5] provides capacity gains over plain multicast routing. For single-rate multicast, the upper bound on throughput is given by the minimum of the maximum flows from the sender to each receiver, and network coding achieves this bound [5]. With multirate multicast, throughput for each receiver is upper-bounded by that receiver's maximum flow, but solutions that achieve this bound do not always exist [6], i.e., the optimum solution may provide less than the maximum flow to some receivers. Network coding solutions for single-rate multicast networks can be found in polynomial time [7], whereas there exist no polynomial-time algorithms that solve the multirate multicast problem [8]–[10].

In multirate multicast, network coding can be performed within a layer (intralayer coding) or across layers (interlayer coding), and in general, the latter outperforms the former [11]. Furthermore, an optimum solution to a network-coded layered multicast problem may require decoding of the network code at interior nodes to extract information to be forwarded. However, decoding at interior nodes brings several disadvantages. Decoding network-coded data requires computationally heavy Gaussian elimination. The data also needs to be stored in its entirety before decoding is possible. This introduces a delay until all information required to decode is available at the node, in addition to the delay introduced by the Gaussian elimination operation itself. Such complexity and delay are particularly undesirable at interior nodes (the routers) of the network, where processing overhead is one of the main bottlenecks.

Manuscript received March 27, 2013; revised January 31, 2014; accepted April 02, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Markopoulou. Date of publication June 05, 2014; date of current version August 14, 2015. This work was supported in part by the European Commissions Seventh Framework Programme (FP7-ICT-2009-5) under Grant Agreement No. 258053 (MEDIEVAL project), Comunidad de Madrid Grant S2009TIC-1692, Spanish MICINN Grant TEC2011-29688-C02-01, and the National Natural Science Foundation of China under Grant 61020106002. The paper is an extended version of "Rate allocation for layered multicast streaming with inter-layer network coding," which was presented at the Mini-Conference track of the IEEE International Conference on Computer Communications (INFOCOM), Orlando, FL, USA, March 25–30, 2012.

J. Widmer and A. Fernández Anta are with IMDEA Networks Institute, 28918 Madrid, Spain (e-mail: Joerg.Widmer@imdea.org).

A. Capalbo is with Altran, 28022 Madrid, Spain.

A. Banchs is with IMDEA Networks Institute, 28918 Madrid, Spain, and also with the University Carlos III of Madrid, 28911 Madrid, Spain.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2326523

In this paper, we thus focus on the interlayer network coding problem without decoding at interior nodes. After a brief survey of related work (Section II), we show that the problem is NP-hard (Section III) and propose a heuristic solution of polynomial complexity that is based on the Edmonds–Karp max-flow algorithm (Section IV). We prove that our algorithm ensures that each receiver can decode as many layers as correspond to the minimum max-flow of any of the receivers (Section V). In addition, we show that our algorithm fulfills some fairness criterion, in a sense that increasing the number of layers decoded at a certain receiver to a value $\ell$ may only reduce the number of layers decoded at other receivers that have more than $\ell$ layers, and may at most reduce them to $\ell$ layers. This property is critical for the design of the algorithm since it allows for an iterative processing of receivers. We further analyze the complexity of our algorithm and show that it is moderately more complex than the max-flow algorithm itself, which is used by other heuristics [11]. We perform extensive simulations and show that our algorithm may even outperform other heuristics that *do* require decoding at interior nodes while at the same time using fewer network resources. Specifically in large topologies, our algorithm achieves higher receive rates (Section VI).

## II. Related Work

We first discuss related work on multirate multicast with intralayer network coding and then interlayer coding schemes. Intralayer network coding is conceptually simpler than interlayer network coding. Nevertheless, the general intralayer coding problem with a fixed set of layers is NP-hard [10]. Integer linear programs to optimally solve the intralayer coding problem have been proposed, for example, in [10], [12], and [13]. A basic heuristic with polynomial-time complexity was proposed in [14]. Receivers are grouped into subsets of receivers that support the same rates. The rate of the base layer is selected such that it is supported by all multicast receivers, and a multicast network code for the base layer is constructed using the approach of [7]. The rate of the second layer is set such that it can be received by the group of receivers with the second lowest rate, given the remaining capacity, and a multicast network code for the second layer is constructed. Hence, each layer is transported in its own multicast tree. The procedure is repeated until all receiver groups are served or all capacity is used, and thus may create as many layers as there are distinct receive rates. Lakshminarayana and Eryilmaz [15] provide a more sophisticated solution based on the decomposition of the intralayer coding problem into that of rate allocation for subsessions and the information distribution over these subsessions, which are then solved separately using efficient low-complexity heuristics.

For interlayer network coding, there is comparatively less existing work. While NP-hardness results exist, they apply to problem versions that are slightly different from the one considered in our paper. Shao *et al.* [16] study the rainbow network flow problem for multiple description coding, where receivers try to obtain as many different flows (or colors, hence "rainbow") as possible. Multiple description coding allows receivers to make use of any layer they receive, which makes it easier to find efficient network codes than for layered coding with its layer dependencies. They show that deciding if there is a solution of this problem in which each receiver can decode a number of layers equal to its max-flow is NP-hard. Their proof works for the case of intralayer coding and can easily be extended to interlayer coding.

Király and Kovács [9] study interlayer network coding for layered coding, as in our problem. However, their results do not directly apply to our problem due to differences in the assumptions used. In their work, the specification of the problem assigns to each receiver a number of layers that it expects to receive. Firstly, they prove that deciding if a feasible solution that fulfills the expectations of all the receivers exists is NP-hard. This result does not apply to our problem, since it assumes that *interior nodes that are not receivers can decode*. Secondly, they show that in a system with only two layers, it is NP-hard to fully serve all the receivers that require two layers and maximize the number of satisfied receivers among those that demand only one layer. Since we require that *all* receivers are able to decode at least the first layer, also this result is not directly applicable to our problem. The authors further propose heuristic algorithms for the two-layer and three-layer version of the problem.

Dumitrescu *et al.* [17] provide a centralized algorithm for multirate interlayer network coding for an arbitrary number of layers. The algorithm is based on integer linear programming with exponential complexity.

A practical approach to interlayer network coding for overlay mesh networks is presented in [18]. Here, each node locally determines the fraction of packets received for each set of layers that provides the lowest video distortion. This approach abstracts from the actual network topology and assumes that packets from different parents are linearly independent (with sufficiently high probability).

The work most closely related to ours is presented by Kim *et al.* [11], where two simple but efficient heuristics for interlayer network coding are proposed. For the sake of comparison, in the following we explain these algorithms in more detail.

The algorithms assume the network is a directed acyclic graph. For both algorithms, it is first necessary to determine the max-flow to each receiver, using one of the well known max-flow algorithms [19]. The max-flow of a receiver corresponds to the minimum cut between the source and that receiver. The algorithms then propagate the maximum layer constraints given by the max-flows of the receivers up toward the source, but they differ in how this is done. In the first algorithm called *Min-Req*, each receiver propagates its max-flow value to its parent node or nodes. A parent waits until it hears from all its children and then propagates the minimum of the max-flow values it received to its own parent nodes, and so on. The source then sends linear combinations on its outgoing links coded over as many layers as is allowed by the maximum layer constraint. This is illustrated in Fig. 1, where receivers $t_1$ and $t_4$ with a max-flow of 1 constrain nodes $d$ and $e$ to the first layer, which in turn constrain all of the upstream interior nodes, and hence all receivers only obtain the first layer.

The rationale for propagating up the minimum is that to generate a linear combination, an interior node can simply code over

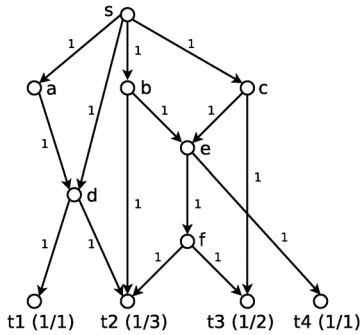



Fig. 1. Example of *Min-Req* algorithm.

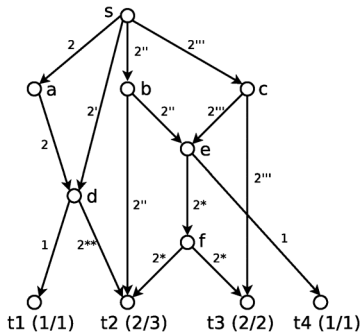



Fig. 2. Example of *Min-Cut* algorithm.

all incoming links. Due to the layer constraints, none of the incoming links will carry a linear combination that is undecodable because the rate required to decode it exceeds the max-flow of a receiver further below in the graph. Note that a node generates a different linear combination for each outgoing link.

In the second algorithm called *Min-Cut*, an interior node propagates up its own max-flow value instead of the minimum of its children's max-flow values, if the own max-flow is larger than the minimum of the children. In this case, the node also has to decode the network-coded layers it receives. This is necessary in order to generate linear combinations containing only a subset of the received layers for those of its children that have lower max-flow values. In Fig. 2, both interior nodes $d$ and $e$ have a max-flow of 2 and receive two different linear combinations coded over the first two layers. Hence, these nodes can decode to extract layer 1 for receivers $t_1$ and $t_4$, while forwarding combinations over two layers to $t_2$ and $t_3$, allowing them to decode the first two layers.

Both algorithms are simple and can be implemented in a distributed manner. Network coding opportunities are exploited implicitly. While the *Min-Req* algorithm does not require decoding at interior nodes, its performance in topologies with heterogeneous max-flow values may be low. The *Min-Cut* algorithm fares better in such topologies as long as interior nodes have sufficiently high max-flow values, but it does require decoding at interior nodes. In addition, the *Min-Cut* algorithm not only needs to determine the max-flow values of the receivers as does *Min-Req*, but also of all interior nodes in the network. Furthermore, both algorithms transport traffic on *all* upstream edges of the receivers, independent of whether they are needed for the multicast or not. This not only enforces lower layers on edges where higher layers could be transported (as in the example on path $(s, a, d, t_2)$), but more importantly, wastes capacity that may prohibit their use in networks with statistical multiplexing where links are shared with other flows. In comparison, the solution we propose in this paper substantially outperforms the *Min-Req* algorithm, while at the same time, it does not require any decoding at interior nodes and hence has lower overhead than the *Min-Cut* algorithm.

## III. Problem Description

### A. Model

We consider single-source multicast on a directed acyclic graph $G(V, E)$ with nodes $V$ and edges $E$. We denote the source node by $s \in V$, and the set of receivers by $T = \{t_1, t_2, \ldots, t_r\} \subset V$. Furthermore, let $In(v)$ be the set of incoming edges of node $v$, and $Out(v)$ be set of outgoing edges of node $v$. The source multicasts a stream of up to $k$ layers, $L_1, \ldots, L_k$. Due to the properties of the layered coding, layer $L_i$ is only useful to a receiver if the receiver also receives layers $L_1, \ldots, L_{i-1}$. To simplify the notation, we assume each layer only consists of a single packet. All results directly extend to layers with multiple packets and to the case where layers are split into multiple generations and coding over layers only happens within the same generation. Let $y(u, v) = \sum_{j=1}^{i} g_j L_j$ denote the coded layer combination that is sent on edge $(u, v)$ with coding coefficients $g_j$. We have that

$$y(u, v) \in \langle \cup_{(w,u)\in In(u)} y(w, u) \rangle \qquad \forall u \in V \setminus T \setminus \{s\}$$

where $\langle Y \rangle$ denotes the linear subspace spanned by the set of vectors $Y$.

For simplicity of exposition, we assume that edges have unit capacity and layers have unit rate, as in [11]. Since our algorithm is directly based on the Edmonds–Karp max-flow algorithm, it is straightforward to extend it to nonunitary edge capacities and nonunitary layer rates.

The problem under consideration is to maximize the sum of the received rates of all receivers, under the following fairness constraint: Increasing the number of layers decoded at a certain receiver to a value $\ell$ may only reduce the number of layers decoded at other receivers that have more than $\ell$ layers, and may at most reduce them to $\ell$ layers (see Section V-A).

In our model, data of layer $L_i$ is always combined with data from all lower layers, i.e., whenever the source needs to inject some random linear combination of layer $L_i$ on an outgoing edge, it does so by sending a random linear combination of all layers $L_j$, $j \in \{1, \ldots, i\}$. This constraint does not change the solution space, as shown in the following theorem.

*Theorem 1:* Consider any solution in which, for some $(s, u) \in Out(s)$ and some $b < i$, $y(s, u) = \sum_{j=1}^{i} g_j L_j$ with $g_i \neq 0$ and $g_b = 0$. If the field from which the coefficients $g_j$ are chosen is large enough, this solution can be transformed into a solution with coefficients $g_j \neq 0, \forall j \leq i$, that achieves the same rate allocation at all the receivers.

*Proof:* The new solution has $s$ sending a new combination $y'(s, u) = \sum_{j=1}^{i} g'_j L_j$, so that $g'_j = g_j$, for $j \neq b$, and a new value $g'_b \neq 0$. Consider a receiver $t \in T$ that uses combinations that contain $y(s, u)$ for decoding in the original solution. In the

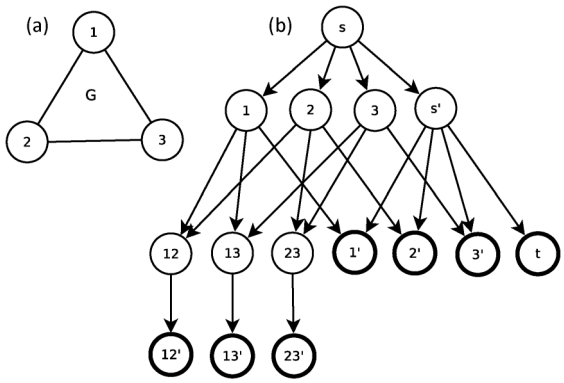



Fig. 3. Sample reduction of the proof of Theorem 2: (a) graph $G$ instance of the minimum vertex cover problem and (b) instance of the network-coding multicast problem obtained from $G$.

new solution, these combinations will have parameter $g'_b$. (Note that the rest of the parameters of the combinations will be the same as before.) Let us consider any layer $\ell \geq i$ that $t$ is able to decode with the original solution. To do so, it solves a system with $m \geq \ell$ linearly independent equations. Replacing in that system $g_b$ by $g'_b$, the system can still be solved if the system's matrix $M$ has a nonzero determinant. Solving for $g'_b$ the equation obtained by setting $|M| = 0$ yields one single value that cannot be used for $g'_b$ to maintain the solvability of the system.

The number of possible receivers is bounded by $|V|$ and the maximum number of systems (i.e., unusable values for $g'_b$) at a receiver is bounded by $|E|$. Then, if the field from which the coefficients are chosen is large enough, $s$ can choose a suitable value for $g'_b$ so that the claim of the theorem holds. If $g'_b$ is chosen randomly, choosing a large enough field guarantees decodability with high probability. ■

Note that Theorem 1 only holds for the case where decoding at interior nodes is not allowed (which is the case for the problem we focus on in this paper).

### B. NP-Hardness

We now show that the problem considered in this paper is NP-hard for any number of layers larger than one.

*Theorem 2:* Consider a network-coding multicast problem with two layers and interlayer coding. Finding a solution to the problem in which all receivers decode layer $L_1$, and the number of receivers that decode $L_2$ is maximized, is NP-hard.

*Proof:* We show that this problem is NP-hard by reduction from the minimum vertex cover problem, which is NP-hard [20]. Consider an instance of minimum vertex cover given by a graph $G(V, E)$. The following network-coded multicast system is obtained from $G$, as shown in Fig. 3(b) for the example graph of Fig. 3(a).

In addition to the source node $s$, the system initially has two more nodes $s'$ and $t$. There is a link from $s$ to $s'$ and a link from $s'$ to $t$. For each vertex $v \in V$, nodes $v$ and $v'$ are added to the multicast system, and links are added from $s$ to $v$, from $s'$ to $v'$, and from $v$ to $v'$. Then, for each edge $(u, v) \in E$, nodes $uv$ and $uv'$ are added to the multicast system, and links are added from $u$ and $v$ to $uv$ and from $uv$ to $uv'$. The set of receivers is formed by nodes $t$, $v'$, and $uv'$, marked in bold in the example graph in Fig. 3(b).

A solution to the above problem that maximizes the number of receivers with two layers solves the minimum vertex cover problem. To see this, observe that to allow $uv'$ to decode layer $L_1$, it must receive it from $uv$, which must get it directly either from $v$ or from $u$ (which, in turn, must receive it from $s$). Then, the set of nodes $v \in V$ such that $v$ receives $L_1$ from $s$ forms a vertex cover of $G$.

Furthermore, observe that $t$ must obtain layer $L_1$ from $s'$. Hence, the nodes $v'$ that receive layer $L_2$ do so from their counterpart $v$. The number of such nodes $v'$ is maximized when the vertex cover in minimized. ■

The NP-hardness proof is an adaptation of the proof in [9] for the problem with two layers. The result can be generalized to a higher number of layers, so that the number of receivers that receive the highest layer is maximized, provided all receivers receive the lower layers. This extension only requires to increase the number of links to $s'$, $t$, and $v'$, so that the missing highest layer is transported via the nodes outside the vertex cover. Note that for the special case of two receivers with two-layer multicast, it is always possible to achieve the max-flow of both receivers as shown in [6, Corollary 2].

*Corollary 1:* Finding a max-min fair allocation is NP-hard.

It can also be shown that the decision problem of whether a feasible solution exists is NP-hard.

## IV. Algorithm

In the following, we present our heuristic for the interlayer network coding problem. We first provide an intuition for the algorithm with the help of the example topology shown in Fig. 4 and then discuss the algorithm together with the pseudocode in more detail.

Our *Multi-Layer Max-Flow* algorithm first determines the max-flow from the source to each receiver using any suitable max-flow algorithm [19]. It then processes the receivers in ascending order of their max-flow values. This ensures that resources are first allocated to receivers with low max-flows before allocating resources to higher max-flow receivers. Our heuristic thus aims to provide an allocation that is close to the optimal fair allocation discussed in Section III.

The mechanism to allocate resources to a receiver is similar to the original Edmonds–Karp max-flow algorithm that repeatedly uses breadth-first search (BFS) [20] to find paths with spare capacity.[1] However, allocating paths to a receiver imposes constraints on the layer combinations that can be transported over that path. The coded combinations must include sufficiently high layers so that they bring useful information to a receiver, but must not include layers that are too high and would therefore be undecodable given the other layer combinations received. These constraints have to be taken into account when subsequently allocating resources to other receivers that may use the same paths. Our layer constrained version of BFS, *Layered BFS*, starts the search at the receivers and proceeds upstream to the source. This allows to find existing flows that can be reused.

[1]In our model where we assume unit capacity edges and layer rates, these paths are edge disjoint.

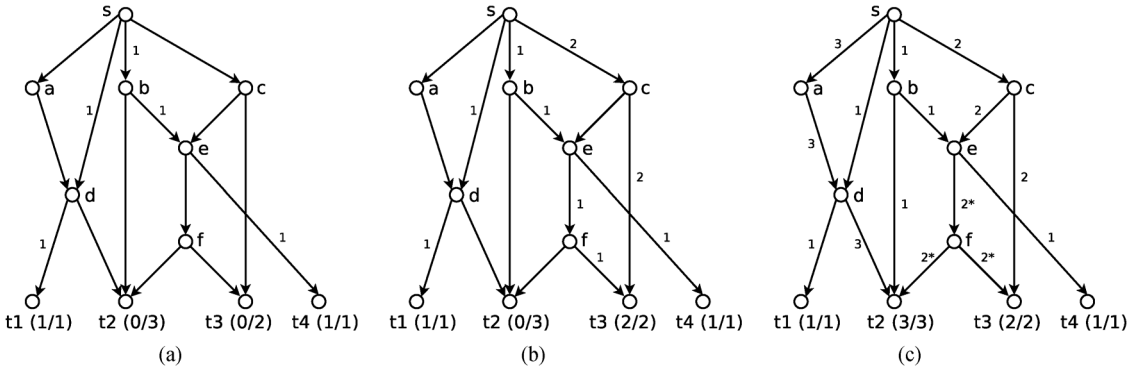



Fig. 4. Different stages of the *Multi-Layer-MaxFlow* algorithm: layer assignment per link and rate achieved versus maximum flow per receiver. (a) Flow assignment: max-flow 1 receivers. (b) Flow assignment: max-flow 2 receivers. (c) Flow assignment: max-flow 3 receiver.

### *A. Example of the Multi-Layer Max-Flow Algorithm*

In the example in Fig. 4(a), first receivers $t_1$ and $t_4$ with a max-flow of 1 use *Layered BFS* to find shortest paths to obtain $L_1$ directly from the source. Next, the max-flow 2 receiver $t_3$ needs to find a path to a flow coded over $L_1$ *and* $L_2$. If such a path is not found, the receiver would attempt to at least find a path to $L_1$. As shown in Fig. 4(b), a suitable $L_2$-path $(s, c, t_3)$ is found. The receiver then has to find $L_1$ *or* a combination of $L_1$ and $L_2$. After traversing the two edges to reach node $e$, the mechanism traverses the remaining path to $s$ before exploring any new edges since the mechanism prioritizes reusing previously established paths over unused edges. Backtracking from the source establishes path $(s, b, e, f, t_3)$, where edges $(s, b)$ and $(b, e)$ are shared with receiver $t_4$ and new capacity only needs to be assigned to edges $(e, f)$ and $(f, t_3)$. Note that the latter edges have a maximum layer constraint of 2, but in the current configuration would carry only layer $L_1$.

A shared edge can also be traversed to augment the existing flow on that edge, in case the flow is already received by the receiver via a different path, or the flow contains layers that are too low to be useful at this stage of the search. Such a shared edge traversal implies network coding across flows since the existing flow on that edge was assigned by another receiver and thus cannot be replaced. In the example, network coding on a shared edge occurs when assigning paths to the max-flow 3 receiver $t_2$. The receiver first searches for a combination of $L_1$, $L_2$, and $L_3$, which it obtains directly from the source via a new path $(s, a, d, t_2)$, as shown in Fig. 4(c). It then needs a combination of the first two or all three layers. It first explores the edges $(b, t_2)$ and $(f, t_2)$. Edge $(s, b)$ cannot be traversed to find layer $L_2$ or above since $t_4$ enforced a maximum layer constraint of 1 on that edge. While $L_1$ is also transported on edge $(e, f)$, the maximum layer constraint on that edge is 2, established by receiver $t_3$, which is able to decode $L_1$ and $L_2$ given its flow assignment. Therefore, the *Layered BFS* can continue via the shared edge and subsequently finds a suitable combination of $L_1$ and $L_2$ at node $c$ via edge $(c, e)$. Node $e$ thus forms a linear combination of $L_1$ received from $b$ and the combination of $L_1$ and $L_2$ received from $c$ and forwards it along edge $(e, f)$. This alters the linear combination received on the subtree below edge $(e, f)$, but the layer constraints ensure that this does not change decodability. Receiver $t_3$ will now receive a linear combination over the first two layers, marked $2^*$ in Fig. 4(c), which is linearly independent from the combination of layers $L_1$ and $L_2$ received via edge $(c, t_3)$.

In this example, the *Multi-Layer Max-Flow* algorithm is able to achieve the max-flows for all receivers. It may not be surprising that it performs better than the *Min-Req* algorithm shown in Fig. 2, which always propagates the minimum of the max-flows and thus makes it difficult for nodes with high max-flow to achieve high rates. However, it is noteworthy that it also outperforms the *Min-Cut* algorithm in Fig. 2, which incurs a higher complexity as it requires decoding at interior nodes. Furthermore, both *Min-Req* and *Min-Cut* transport traffic on all upstream edges of the receivers. This wastes capacity and establishes unnecessary constraints on edges that could otherwise carry higher layers (e.g., in the example on path $(s, a, d, t_2)$).

### *B. Algorithm Description*

Let us consider a specific receiver $t$ with a max-flow of $maxFlow(t)$. The algorithm first tries to find paths from source $s$ to receiver $t$ that allow the receiver to decode $\ell_{\max} = maxFlow(t)$ layers. To this end, it not just has to find $\ell_{\max}$ such paths that are edge disjoint, but needs to ensure that the layer combinations $y_1, y_2, \ldots, y_{\ell_{\max}}$ received via these paths do not include any layers higher than $\ell_{\max}$, i.e.,

$$y_\ell = \sum_{j=1}^{\ell_{\max}} g_{\ell,j} L_j, \qquad \ell = 1, \ldots, \ell_{\max} \tag{1}$$

and that these combinations are linearly independent. A necessary condition for linear independence is that there exists some permutation $\pi$ of $1, 2, \ldots, \ell_{\max}$ such that

$$g_{\pi(\ell),\ell} \neq 0. \tag{2}$$

This ensures that the matrix formed by the linear combinations is at least lower triangular (i.e., there is one layer combination coded over at least the first layer, another coded over at least the first two layers, etc.). Since the source sends out linearly independent combinations on all its links of the paths to a receiver and these paths are edge disjoint, the received combinations have a very high probability of being linearly independent when interior nodes perform random linear network coding [21]. If (1) and (2) hold, we say that the receiver satisfies $\ell_{\max}$-*decodability*.

Note that while the paths of a receiver are mutually edge disjoint, they may overlap with paths of other previously processed receivers. In fact, to avoid wasting network resources, it is desirable to reuse existing paths as much as possible, rather than establishing new paths. Wherever paths transporting different flows overlap, network coding among those flows is necessary. On all the paths used by a receiver, per-edge layer constraints of that receiver's $\ell_{\max}$ value are introduced to ensure that (1) holds. This means that $\ell_{\max}$ is an *upper bound* on the highest layer included in the layer combinations transported over that edge and receivers processed later on cannot use this edge to transport combinations of higher layers.

The algorithm starts by trying to find a (partially) existing or new path for $y_\ell$, with $\ell = \ell_{\max}$, i.e., a path that can transport combinations including layer $\ell_{\max}$ (and all lower layers) using *Layered BFS*. *Layered BFS* starts at the receiver and includes all incoming edges of that receiver that have a layer constraint of $\ell_{\max}$ or higher in the list of candidate edges to be explored. The algorithm first explores edges having a layer constraint of exactly $\ell_{\max}$ to reuse existing paths of other receivers as much as possible. If no such edges exist, unused edges without layer constraints will be explored. Only if no such edges exist either will the algorithm explore edges that have layer constraints higher than $\ell_{\max}$. Using the latter edges lowers their layer constraints, which in the worst case could cause another previously processed downstream receiver to decode fewer layers. Since receivers are processed in ascending order of max-flow, such lowering of layer constraints is rare (and can only happen in connection with lowering a receiver's $\ell_{\max}$ as explained further below).

Whenever an edge $(u, v)$ is explored, all incoming edges $(w, u)$ of node $u$ are included in the list of candidate edges to be explored, and the BFS continues until a path to the source is found. If such a path for layer combination $\ell_{\max}$ is found, the algorithm successively tries to find paths for the remaining $y_\ell, \ell = \ell_{\max} - 1, \ldots, 1$. Once all the paths are found, the algorithm introduces the corresponding layer constraints on the edges that are used by that receiver and proceeds with the next receiver. Existing layer constraints may, however, prevent the algorithm from finding all of these paths. In this case, the algorithm sets $\ell_{\max} = maxFlow(t) - 1$ and again checks for decodability for the receiver to provide it with a lower number of decodable layers. The algorithm decrements $\ell_{\max}$ until $\ell_{\max}$-decodability is fulfilled.

This is the case for the max-flow 3 receiver $t_2$ in the example in Fig. 5(a). There are existing layer constraints of 1 on the incoming edges of nodes $b$ and $e$, as well as a constraint of 3 on edges $(a, c)$ and $(c, t_1)$, introduced by previously processed receivers. The initial BFS with $\ell = \ell_{\max} = 3$ first explores edges to nodes $c$, $d$, and $e$, which in turn causes edges $(a, c)$ and $(b, d)$ to be entered in the list of candidate edges. Of these, $(a, c)$ is explored first, since reusing an existing flow takes precedence over using additional resource on $(b, d)$, as shown in Fig. 5(b). However, the next BFS with $\ell = 2$ fails due to the layer constraints of 1 on the incoming edges of nodes $b$ and $e$. Consequently, $t_2$'s layer constraint of 3 on edge $(c, t_2)$ is removed, and the algorithm starts over with $\ell_{\max}$ reduced to 2. After exploring edges to nodes $c$, $d$, and $e$, the BFS again enters edges $(a, c)$ and $(b, d)$ in the list of candidate edges, but this time $(b, d)$ is explored before $(a, c)$ to avoid lowering the layer constraint of the latter. However, $b$ does not have any suitable incoming edges, which leaves $(a, c)$ as the only remaining edge to explore. A path to the source is found, and layer constraints of 2 are introduced on edges $(a, c)$ and $(c, t_2)$. The final BFS with $\ell = 1$ explores edges $(d, t_2)$ and $(e, t_2)$ and finds an existing layer 1 flow via the latter. Note that this introduces a layer constraint of 2 on edge $(e, t_2)$, but in fact only a layer-1 flow is transported on that edge.

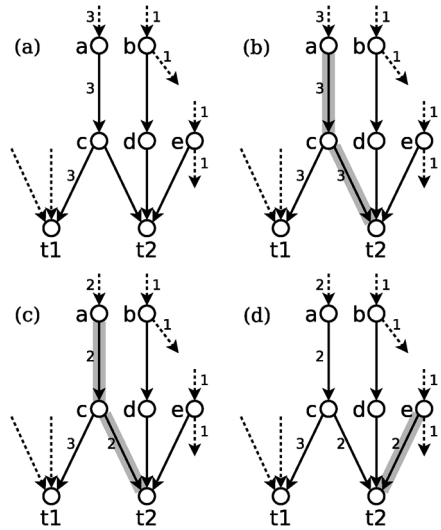



Fig. 5. Layer constrained BFS for receiver $t_2$. (a) Existing layer constraints. (b) BFS with $\ell_{\max} = 3$ fails. (c) BFS with $\ell_{\max} = 2$ succeeds, lowering a layer constraint. (d) A constraint for the second flow of $t_2$ is introduced.

### C. Reverse Edge Traversal

Whenever the original Edmonds–Karp algorithm finds an additional path with spare capacity using BFS, this path may not be part of the optimal resource allocation that allows a receiver to achieve its max-flow. It may thus have to be modified when further paths are found in subsequent BFSs. To this end, whenever a path is established that generates flow on an edge $(u, v) \in E$, a corresponding negative flow is assigned to a virtual reverse edge $(v, u)$ [20].[2] In case a subsequent BFS finds a path through such a reverse edge, the flow originally assigned on the forward edge $(u, v)$ is removed.

Our *Multi-Layer Max-Flow* algorithm proceeds in the same way if $(u, v)$ is only used by the current receiver. However, edge $(u, v)$ may be a shared edge in which case the flow on that edge is required by another receiver and cannot be removed. The BFS may nevertheless traverse the reverse edge to find a suitable flow, which is then combined with the existing flows *in the subtree below the shared edge* $(u, v)$. This mechanism is best explained using the well-known butterfly topology shown in Fig. 6.

Both receivers $t_1$ and $t_2$ have a max-flow of 2. Let us assume that first receiver $t_1$ finds two edge disjoint paths to the source, along which two different combinations of $L_1$ and $L_2$ are transported. These are marked as 2 and 2′ in Fig. 6(a). For receiver $t_2$, the BFS will first find path $(t_2, e, c, b, s)$ since it contains only

[2]Note that here $G$ is a directed acyclic graph, and thus $(v, u) \notin E$.

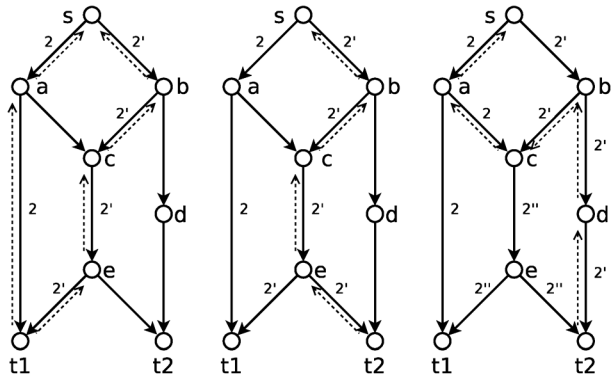



Fig. 6. Reverse edge traversal. (a) Receiver $t_1$. (b) Receiver $t_2$, step 1. (c) Receiver $t_2$, step 2.

```
1  procedure Layer Assignment(G, s, T)
2  ∀(u,v) ∈ E : F_(u,v) ← 0, M_(u,v) ← ∞
3  for each t_i ∈ T
4  |   maxFlow(t_i) ← Compute Max Flow(s, t_i)
5  end for
6  for m ← 1 to max_{t_i∈T} maxFlow(t_i)
7  |   for each t_i such that maxFlow(t_i) = m
8  |   |   (F^i, M^i) ← Layered Edmonds-Karp(s, t_i)
9  |   end for
10 |   for each (u,v) ∈ E such that M^i_(u,v) < ∞
11 |   |   F_(u,v) ← max{F_(u,v), F^i_(u,v)}
12 |   |   M_(u,v) ← min{M_(u,v), M^i_(u,v)}
13 |   end for
14 end for
```

Fig. 7. *Layer Assignment* mechanism.

one new edge and three shared edges, whereas the other possible paths require two new edges. Hence, this path is established as shown in Fig. 6(b), with the corresponding negative flow on the virtual reverse edges. These reverse edges can now be traversed in subsequent breadth first searches. Note that without this, receiver $t_2$ would not be able to find a second path to transport a layer combination that is linearly independent of the first one. A second BFS can now find another suitable path for $t_2$ by traversing edges $(d, t_2)$, $(b, d)$, the virtual reverse edge $(c, b)$, and then $(a, c)$ and $(s, a)$ as shown in Fig. 6(c). If receiver $t_2$ were the only receiver in the network using $(b, c)$, this would cause the flow on edge $(b, c)$ to be removed, and two different combinations of the first two layers would be transported via two edge disjoint paths $(s, a, c, e, t_2)$ and $(s, b, d, t_2)$. However, since edge $(b, c)$ is shared with receiver $t_1$, the flow cannot be removed. Instead, the flow remains but is now also transported via the path $(b, d, t_2)$. At the same time, the suitable flow that was found by BFS via edges $(a, c)$ and $(s, a)$ is transported via the shared edge $(c, e)$. This requires combining it with the flow that was previously assigned to that edge, creating a new linear combination (marked $2''$ in the figure) over the first two layers that is transported along the existing subtree and received by both receivers $t_1$ and $t_2$. Note that network coding occurs between the flow on $(b, c)$, corresponding to the reverse edge, and the new flow on the edge $(a, c)$ that *follows* the reverse edge in the breadth first search. The outgoing edge for this coded combination is $(c, e)$, the original outgoing edge for the flow from $(b, c)$. This results in the common network coding solution for the butterfly topology [Fig. 6(c)].

In case the topology had no node $d$ but a direct edge $(b, t_2)$, the BFS for $t_2$ may first reach the source via $b$ and establish the path $(s, b, t_2)$. The second BFS for $t_2$ would proceed to $s$ via nodes $e$, $c$, and $a$, thereby directly traversing the shared edge $(c, e)$. Again this introduces network coding on the shared edge $(c, e)$ and the resulting layered network code is the same as before.

### D. Pseudocode

The *Multi-Layer Max-Flow* algorithm discussed above is implemented in three different mechanisms. The *Layer Assignment* mechanism processes receivers in ascending order of their max-flow values and maintains the global layer constraints. *Layered Edmonds-Karp* is a multilayer version of the Edmonds-Karp maximum flow algorithm that finds the highest $\ell_{\max}$ for a receiver as well as the corresponding set of paths that ensure $\ell_{\max}$-decodability, given the existing layer constraints. Finally, the *Layered BFS* mechanism is used by *Layered Edmonds-Karp* for the layer constrained breadth first search to find a path with specific layer constraints from a receiver to the sender. The actual network coding of layers is done in the *Coding Stage*, taking into account the previously established layer constraints.

*1) Layer Assignment (Fig. 7):* The mechanism determines the receivers' max-flow values (lines 3–5) and then calls *Layered Edmonds–Karp* for each receiver in ascending order of max-flows. *Layered Edmonds–Karp* returns the flow allocation $F^i$ of receiver $t_i$ as well as maximum layer constraints $M^i$. With these, the global per-edge flow allocation $F$ and the maximum layer constraints $M$ are updated (lines 10–13).

*2) Layered Edmonds–Karp (Fig. 8): Layered Edmonds–Karp* searches for a set of paths for a receiver $t_i$ that provide $\ell_{\max}$-decodability, starting with $\ell_{\max} = maxFlow(t_i)$ (line 2). To this end, it searches for paths to transport a flow with a maximum layer between $\ell_{\min}$ and $\ell_{\max}$, starting with $\ell_{\min} = \ell_{\max}$, then $\ell_{\min} = \ell_{\max} - 1$ (line 26), and so on, until all the $\ell_{\max}$ paths are found. If at any point *Layered BFS* fails to return a layer-$\ell$ path $P$ with $\ell_{\min} \leq \ell \leq \ell_{\max}$, i.e., flag *found* is false, the receiver is not able to decode $\ell_{\max}$ layers given the existing layer constraints. In that case, all existing temporary flows $F^i$ and maximum layer constraints $M^i$ are removed, and *Layered Edmonds–Karp* reduces $\ell_{\max}$ by one. This continues until an $\ell_{\max}$-*decodable* path allocation is found. Note that any receiver will at least be able to receive $L_1$, as shown later, and hence the mechanism always terminates.

Whenever *Layered BFS* returns with a valid path $P$ (line 7), the path is backtracked from $s$ to $t_i$, and $\ell_{\min}$ is decremented (lines 8–27). A path entry $P(u)$ contains the tuple of next-hop $v$ following node $u$, as well as the corresponding layer constraint $\ell$ for edge $(u, v)$. The corresponding flow $F^i_{(v,u)}$ and layer constraint $M^i_{(v,u)}$ are updated (lines 17–23). In case a normal edge is traversed, the corresponding values are set, whereas in case of a reverse edge, the layer constraint and the flow is removed. This continues until the path terminates at the receiver $t_i$.

A node also maintains a so-called *in-out* table that maps incoming to outgoing edges (for simplicity, this is omitted from

```
1  procedure Layered Edmonds-Karp(s, t_i)
2  for ℓ_max ← maxFlow(t_i) down to 1
3  |  ∀(u,v) ∈ E : F^i_(u,v) ← 0, M^i_(u,v) ← ∞
4  |  found = FALSE
5  |  ℓ_min ← ℓ_max
6  |  do
7  |  |  (P, found, update) ← Layered BFS(s, t_i, ℓ_min, ℓ_max)
8  |  |  if found then
9  |  |  |  u ← s
10 |  |  |  while u ≠ t_i
11 |  |  |  |  (v, ℓ) ← P(u)
12 |  |  |  |  if update((u,v)) then
13 |  |  |  |  |  ∀(u', v') upstream of u that carry a flow
14 |  |  |  |  |  that contributes to (u,v) :
15 |  |  |  |  |  |  M^i_(u',v') ← min{M^i_(u',v'), ℓ}
16 |  |  |  |  end if
17 |  |  |  |  if (u,v) ∈ Out(u) then
18 |  |  |  |  |  M^i_(u,v) ← ℓ
19 |  |  |  |  |  F^i_(u,v) ← 1
20 |  |  |  |  else // reverse edge
21 |  |  |  |  |  M^i_(v,u) ← ∞
22 |  |  |  |  |  F^i_(v,u) ← 0
23 |  |  |  |  end if
24 |  |  |  |  u ← v
25 |  |  |  end while
26 |  |  |  ℓ_min ← ℓ_min − 1
27 |  |  end if
28 |  while (ℓ_min ≥ 1) ∧ found
29 |  if found then
30 |  |  return (F^i, M^i)
31 |  end if
32 end for
```

Fig. 8. *Layered Edmonds–Karp* mechanism.

the pseudocode). A path $(w, u, v)$ creates an entry $(w, u) \mapsto (u, v)$ at node $u$. $(w, u)$ may map to multiple outgoing edges, and multiple incoming edges may map to $(u, v)$. The mechanism further checks whether a special marker $update$ is set for edge $(u, v)$ on the path (lines 12–16). This indicates that the maximum layer constraint on edge $(u, v)$ had to be reduced. As a consequence, the maximum layer constraints on all upstream edges *that contribute to* $(u, v)$ have to be updated as well. The edges can easily be identified through the *in-out* tables maintained at the nodes.

*3) Layered BFS (Fig. 9):* The mechanism performs breadth-first search to find a path from $t_i$ to $s$ with matching layer constraints. It maintains a priority queue $Q$ to store tuples of nodes to be visited next, together with their priority,[3] as well as a mapping $P : u \mapsto (v, \ell)$ to store next-hop nodes for backtracking and the corresponding maximum layer constraint for edge $(u, v)$.

The mechanism removes the first node $v$ from the queue (line 7). If $v$ is the source, then a suitable path was found and is returned (line 9). Otherwise, the mechanism explores all forward edges that are not yet used for receiver $t_i$ (lines 13–28) and all virtual reverse edges that are used for $t_i$ (lines 29–37). An edge $(u, v)$ that is not used for $t_i$ has a path entry $P(u) = (\perp, \cdot)$, where $\perp$ signifies a next-hop is not set and $\cdot$ means any value, and a flow value $F^i_{(u,v)}$ of 0. Forward edges that are entirely unused by any receiver, i.e., also $F_{(u,v)} = 0$, are enqueued with a priority of $\delta(v) + 1$ (lines 16 and 17) since additional

[3]The lower the value, the higher the priority.

```
1  procedure Layered BFS(s, t_i, ℓ_min, ℓ_max)
2  ∀v ∈ V : P(v) ← (⊥, ∞), δ(v) ← ∞, update(v) ← FALSE
3  δ(t_i) ← 0;
4  Q ← ∅ // priority queue
5  enqueue(Q, (t_i, δ(t_i))) // enqueue t_i with highest priority 0
6  while Q ≠ ∅ do
7  |  v ← dequeue(Q)
8  |  if v = s then
9  |  |  return (P, TRUE, update)
10 |  end if
11 |  (·, ℓ) ← P(v)
12 |  ℓ ← min{ℓ, ℓ_max}
13 |  for each (u,v) ∈ In(v) such that
14 |  |  (P(u) = (⊥, ·)) ∧ (F^i_(u,v) = 0)
15 |  |  if F_(u,v) = 0 then
16 |  |  |  P(u) ← (v, ℓ); δ(u) ← δ(v) + 1
17 |  |  |  enqueue(Q, (u, δ(u)))
18 |  |  else if ℓ_min ≤ M^i_(u,v)
19 |  |  |  if ℓ ≥ M^i_(u,v) then
20 |  |  |  |  P(u) ← (v, M^i_(u,v)); δ(u) ← δ(v)
21 |  |  |  |  enqueue(Q, (u, δ(u)))
22 |  |  |  else if ℓ = ℓ_max then
23 |  |  |  |  P(u) ← (v, ℓ_max); δ(u) ← δ(v) + |E|
24 |  |  |  |  update(u) ← TRUE
25 |  |  |  |  enqueue(Q, (u, δ(u)))
26 |  |  |  end if
27 |  |  end if
28 |  end for
29 |  for each (v,u) ∈ Out(v) such that
30 |  |  (P(u) = (⊥, ·)) ∧ (F^i_(v,u) > 0)
31 |  |  if F_(v,u) = 0 then
32 |  |  |  P(u) ← (v, ℓ_max); δ(u) ← δ(v) − 1
33 |  |  else
34 |  |  |  P(u) ← (v, M^i_(u,v)); δ(u) ← δ(v)
35 |  |  end if
36 |  |  enqueue(Q, (u, δ(u)))
37 |  end for
38 end while
39 return (∅, FALSE, update)
```

Fig. 9. *Layered BFS* mechanism.

capacity would have to be allocated on a new edge. Shared edges can only be traversed if they support a sufficiently high layer (line 18) and are enqueued with the same priority $\delta(v)$, i.e., at the head of the priority queue, if the current maximum layer constraint $M^i_{(u,v)}$ can be kept (line 19). In this case, no additional capacity is used. If, however, the layer constraint has to be reduced since $\ell < M^i_{(u,v)}$, this means that other receivers downstream of $(u, v)$ may no longer be able to decode up to $M^i_{(u,v)}$ but only up to $\ell$ layers. Therefore, this is only allowed in case $\ell = \ell_{\max}$, i.e., $\ell_{\max}$ was already reduced and the receiver is trying to obtain a *maximum layer* that is lower than that of the downstream receivers $M^i_{(u,v)}$. The corresponding node is enqueued last with a low priority of $\delta(v) + |E|$ to ensure that any other option that does not involve reducing a layer constraint is explored first (line 23). The rationale is that setting $M^i_{(u,v)}$ to $\ell$ *may* reduce the number of layers decodable at other downstream receivers to $\ell$, and at the same time, it is necessary to allow the current receiver to obtain $\ell$ layers (since $\ell = \ell_{\max}$). Therefore, reducing the layer constraint results in a resource allocation that is closer to the max-min fair allocation that our heuristic aims to obtain. In this case, the marker $update$ is set for $(u, v)$ to indicate that the maximum layer constraints on upstream edges that contribute to $(u, v)$ may have to be updated

by *Layered Edmonds–Karp* in case $(u, v)$ is on the path that is ultimately chosen.

When traversing reverse edges that are only used by $t_i$ (lines 31 and 32), the priority is set to $\delta(v) - 1$ since capacity on an edge is freed, and the layer constraint is reset to $\ell_{\max}$. Traversing shared reverse edges leaves the priority $\delta(v)$ unchanged (line 34), and the maximum layer constraint is reset to that of the shared edge $M^i_{(u,v)}$ since network coding occurs after the reverse edge.

*4) Coding Stage:* After the *Layer Assignment* mechanism completes, the source sends out linear combinations over as many layers as is allowed by the layer constraints on the outgoing edges

$$y(s,v) = \sum_{l=1}^{M_{(s,v)}} g_l^{(s,v)} L_l \qquad \forall (s,v) \in Out(s). \tag{3}$$

The source further needs to ensure that the combinations it sends out over each edge are linearly independent. An interior node $u$ of the network codes according to its *in-out* table

$$y(u,v) = \sum_{w:(w,u)\mapsto(u,v)} g_{\text{local}}^{(w,u)} y(w,u) \tag{4}$$

where $g_{\text{local}}^{(w,u)}$ are local coding coefficients (as opposed to the global coding coefficients used in (1) and (2); see [7]).

### E. Distributed Implementation

While we describe the algorithm as a centralized algorithm, it can be implemented in a distributed manner (albeit with somewhat more complexity and coordination overhead than the algorithms in [11]). After the first phase to determine the max-flows of all receivers, the source broadcasts a message to indicate rounds, and in each round, receivers with the same min-cut value locally run *Layered Edmonds–Karp*. State information ($F^i$, $M^i$, $P$, etc.) is maintained at interior nodes on a per-edge, per-receiver basis. For the *Layered BFS* that starts at a receiver and then spreads through the network, it is necessary to implement queue $Q$ in a distributed manner. To this end, *Layered BFS* messages are forwarded to neighboring nodes after a time delay that corresponds to the three different priorities (reuse existing link, add a new link, or overwrite a layer constraint). Additionally, in case messages with higher priority arrive later, they are allowed to overwrite previously established paths and layer constraints of the same receiver.

## V. Analysis

We first prove some basic properties of the algorithm and then analyze decodability and fairness properties. Due to space constraints, we only include sketches of the proofs.

Without loss of generality, in the following we assume $maxFlow(t_1) \le maxFlow(t_2) \le \cdots \le maxFlow(t_r)$. This implies that in the *Layer Assignment* mechanism, receivers are processed in the order $t_1, t_2, \ldots, t_r$. Let us denote by $\hat{F}^i$ and $\hat{M}^i$ the values of vectors $F$ and $M$ after processing receiver $t_i$ in *Layer Assignment*. Initially, $\hat{F}^0_e = 0$ and $\hat{M}^0_e = \infty$ for all edges $e$. Also, recall that $M^i$ and $F^i$ are receiver $t_i$'s layer constraints and flow allocation returned by *Layered Edmonds–Karp* $(s, t_i)$. We claim that, when processing a receiver $t_i$ with some assignment of the vectors $\hat{F}^{i-1}$ and $\hat{M}^{i-1}$, the following property holds.

*Lemma 1:* When the call *Layered Edmonds–Karp* $(s, t_i)$ executes the return statement (line 30 in the corresponding pseudocode in Fig. 8), it has found and processed a set of $\ell_i \le maxFlow(t_i)$ edge-disjoint paths $P^i_j$ from $s$ to $t_i$ so that we have the following:

1) For every edge $e \in P^i_j$, either $e$ was not used by any previously processed receiver (i.e., $\hat{F}^{i-1}_e = 0$) or it was used, but the maximum layer assigned to $e$ was no smaller than $j$ (i.e., $\hat{M}^{i-1}_e \ge j$),
2) Every edge $e \in P^i_j$ is assigned a maximum layer that is no smaller than $j$ and no larger than $\ell_i$ (i.e., $j \le M^i_e \le \ell_i$).
3) For every edge $e$ shared by a path $P^i_j$ and a path $P^k_x$ of a previously processed receiver $t_k$, $k < i$, every edge $e' \in P^k_x$ that precedes $e$ in $P^k_x$ is assigned a maximum layer $M^i_{e'} = \min\{\hat{M}^{i-1}_{e'}, \ell_i\}$. Additionally, $M^i_e = \min\{\hat{M}^{i-1}_e, \ell_i\}$.

*Proof:* As previously described, the call completes when for some value $\ell_{\max} \in \{1, \ldots, maxFlow(t_i)\}$ the *Layered BFS* mechanism has been executed $\ell_{\max}$ times and has found $\ell_i = \ell_{\max}$ suitable disjoint paths. Each call to *Layered BFS* has a different value $\ell_{\min} \in \{1, \ldots, \ell_{\max}\}$. Let $P^i_j$ be the path found by *Layered BFS* when called with $\ell_{\min} = j$. It can be observed in the code of *Layered BFS* that $P^i_j$ only uses edges $e$ with $\hat{F}^{i-1}_e = 0$ or $\hat{M}^{i-1}_e \ge j$ (Property 1). Additionally (Property 2), the maximum layer associated to an edge $e \in P^i_j$ (and later assigned to $M^i_e$) is never smaller than $j$ and never larger than $\ell_i = \ell_{\max}$ (see lines 19–25 of Fig. 9). Finally (Property 3), for any edge $e$ previously used, its maximum layer in vector $M^i$ is set if: 1) it is in one of the paths found; or 2) it is an edge upstream in a path $P^k_x$ from an edge $e \in P^k_x$ that also belongs to some path $P^i_j$. In either case, the value of $M^i_e$ is set to $\min\{\hat{M}^{i-1}_e, \ell_i\}$ (see lines 23–25 of Fig. 9 and lines 12–16 of Fig. 8). ■

We now show that all the calls to *Layered Edmonds–Karp* execute the return statement.

*Lemma 2:* Let $c = \min_{e \in E}\{\hat{M}^{i-1}_e\} \ge 1$. Then, *Layered Edmonds–Karp* $(s, t_i)$ will execute the return statement with a set of $\ell_i \ge \min\{c, maxFlow(t_i)\}$ paths.

*Proof:* Assume *Layered Edmonds–Karp* $(s, t_i)$ does not execute the return statement. Then, it eventually executes an iteration of the for loop with $\ell_{\max} = \min\{c, maxFlow(t_i)\}$. Observe that, in this iteration, all edges satisfy the property that $\hat{F}^{i-1}_e = 0$ or $\hat{M}^{i-1}_e \ge j$, for all $j \in \{1, \ldots, \ell_{\max}\}$, and hence all paths in the network are suitable to be found. There are at least $\ell_{\max}$ edge-disjoint paths from $s$ to $t_i$, and under no layer restrictions, *Layered Edmonds–Karp* works like the Edmonds–Karp algorithm. Hence, $\ell_{\max}$ edge-disjoint paths will be found, the do-while loop will complete with $\ell_{\min} = 0$ and $found =$ TRUE, and the return statement will be executed. This contradicts the assumption, and hence the return statement is always executed with some set of paths. ■

### A. Fairness

Observe that when processing $t_i$, the number of layers that some receiver $t_k$ can decode may be reduced, if $t_k$ was

processed before $t_i$. The following theorem shows a certain level of fairness in this reduction.

*Theorem 3:* Consider that the processing of receiver $t_i$ allows it to decode $\ell_i$ layers. In this process, no receiver $t_k$, $k < i$, has its number of decodable layers reduced below $\ell_i$ (if it is reduced at all).

*Proof:* Assume that before processing $t_i$, receiver $t_k$ was able to decode $\ell_k$ layers. Then, with the network code according to constraints $\hat{M}^k$ there are $\ell_k$ paths $P_j^k$ from $s$ to $t_k$, so that the combinations sent on these paths ensure $\ell_k$-decodability at $t_k$. If $\ell_k \leq \ell_i$, from Property 3 of Lemma 1, the maximum layer constraints of the edges of these paths do not change, and decodability at $t_k$ remains the same after processing $t_i$. On the other hand, if $\ell_k > \ell_i$, the maximum layer of an edge $e \in P_j^k$ can be reduced when processing $t_i$, but only to $M_i(e) = \min\{\hat{M}_e^{i-1}, \ell_i\}$. Then, there is a subset of size $\ell_i$ of the paths $P_j^k$ that guarantee $\ell_i$-decodability at $t_k$. ■

The above theorem thus shows that the proposed algorithm provides a form of fairness similar to max-min fairness, as it only reduces the number of layers of a receiver if this allows increasing the number of layers of a receiver that is worse off.

### B. Decodability

We now show that after processing the paths found for receiver $t_i$ in *Layer Assignment*, it can decode $\ell_i$ layers.

*Lemma 3:* If the network coding described in Section IV-D.4 is used according to the constraints $\hat{M}^i$, then $t_i$ satisfies $\ell_i$-decodability.

*Proof:* In the *Coding Stage*, $s$ will send over each path $P_j^i$ a linearly independent combination. These combinations reach $t_i$ via edge disjoint paths, maybe coded with other combinations. If one of the $\ell_i$ combinations is coded with another combination at some interior node, Property 3 of Lemma 1 guarantees that the latter has at most layer $\ell_i$. Then, from Property 2, the combinations received at $t_i$ satisfy $\ell_i$-decodability. ■

Building on the previous results, it can be shown that the proposed algorithm satisfies some minimal decodability properties. The following theorem proves that *Layer Assignment* guarantees a minimum number of layers that can be decoded.

*Theorem 4:* Let $mf = \min_{t \in T}\{maxFlow(t)\}$. All receivers are able to decode layers $L_1, \ldots, L_{mf}$.

*Proof:* For $t_1$, we have $mf = maxFlow(t_1)$. Since $\hat{M}_e^0 = \infty$ for all $e$, Lemma 2 shows that *Layered Edmonds–Karp* $(s, t_1)$ executes the return statement after finding $mf$ paths for $t_1$. Lemma 3 guarantees that the $mf$ paths can be used to decode layers $L_1, \ldots, L_{mf}$. Applying induction, assume that all receivers $t_1, \ldots, t_{i-1}$ are able to decode at least layers $L_1, \ldots, L_{mf}$. According to Lemma 2, *Layered Edmonds–Karp* $(s, t_i)$ returns after finding $\ell_i \geq maxFlow(t_i) \geq mf$ paths when processing receiver $t_i$. From Lemma 3, layers $L_1, \ldots, L_{\ell_i}$ can be decoded at $t_i$. Since $\ell_i \geq mf$, after processing receiver $t_i$ every receiver $t_1, \ldots, t_{i-1}$ can still decode at least layers $L_1, \ldots, L_{mf}$ according to Theorem 3. ■

Note that the above theorem ensures that, in the worst case where some receivers have a max-flow of 1, all receivers will at least be able to decode layer 1. Thus, the theorem provides a lower bound on the performance of the algorithm.

### C. Comparison With Previous Approaches

*Theorem 5:* The *Multi-Layer Max-Flow* algorithm always performs better or at the same level as *Min-Req* in terms of fairness.

*Proof:* As long as a receiver $t_i$ running our algorithm does not impose a layer constraint below $maxFlow(t_i)$ on an edge, other receivers will see the same constraints as *Min-Req* on some links and no constraints on other links. With less constraints, receivers can achieve at least the same number of layers, and possibly more. Thus, in this case we will outperform *Min-Req*. In the following, we address the case in which this does not hold for at least one receiver.

Let us consider the first receiver $t_i$ that is only able to achieve $\ell_i < maxFlow(t_i)$ layers by imposing a constraint $M_e^i = \ell_i$ on an edge $e$ (i.e., receiver $t_i$ is not able to achieve its max-flow given the prior layer constraints and therefore aims to impose lower constraints). With *Min-Req*, this receiver would receive fewer than $M_e^i$ layers. Until this point, all receivers had only imposed their max-flow value as constraints on some edges. Hence, receiver $t_i$ will see the same constraints as *Min-Req* on some links and no constraints on other links. If it could achieve $M_e^i$ layers without imposing a constraint below $maxFlow(t_i)$, it would not impose this constraint. This means that *Min-Req*, which does not impose such a constraint, cannot possibly decode $M_e^i$ layers.

A receiver $t_j$ that is processed afterwards will see some of the links with the same constraints as *Min-Req*, others with a constraint of $\ell_i$, and others with no constraint. Such a receiver will either: 1) be able to decode $\ell_i$ or more layers; 2) decode $\ell_j < \ell_i$ layers with our algorithm and the same $\ell_j$ with *Min-Req*; or 3) decode number of layers $\ell_j < \ell_i$ layers with our approach and a smaller number with *Min-Req*. Similar reasoning applies to any receiver processed later on. Let receiver $t_k$ be the lowest layer receiver processed so far for which condition 3) applies when processing receiver $t_m$. Then, one of the following holds: 1) receiver $t_m$ will be able to decode $\ell_k$ or more layers; 2) it will decode $\ell_m < \ell_k$ layers with our algorithm and the same $\ell_j$ with *Min-Req*; or 3) it will decode $\ell_m < \ell_k$ layers with our approach and a smaller number with *Min-Req*. Note that, according to Theorem 3, any receiver $t_i$ processed before $t_m$ will not decode fewer than $\ell_m$ layers only because of constraints imposed by $t_m$.

Let us denote by $n$ the receiver with the smallest number of layers for which case 3) applies when the execution of the algorithm is terminated. From the above, we have that this receiver decodes $\ell_n$ layers with our approach and fewer with *Min-Req*, while all the receivers with fewer layers decode the same number of layers with *Min-Req* and our approach. Therefore, our approach outperforms *Min-Req* according to the max-min fairness criterion. ■

Comparing algorithm *Multi-Layer Max-Flow* to *Min-Cut*, there are networks in which *Min-Cut* performs worse despite decoding at interior nodes, as illustrated in Figs. 2 and 4. However, as expected, there are cases in which *Min-Cut* outperforms our algorithm. This can be observed, for example,

in Fig. 11 in Section IV. A large number of receivers impose constraints on each others' paths, reducing the performance of our algorithm. *Min-Cut* instead can decode and give to each receiver the appropriate layers.

### D. Complexity Analyses

There are two parts in the centralized version of algorithm *Multi-Layer Max-Flow* that share the largest fraction of the time complexity: the initial computation of the maximum flow for each receiver and the flow assignment to each of them. The first part (lines 3–5 in the *Layer Assignment* mechanism) involves $|T|$ executions of a max-flow algorithm.

The second part involves a call to *Layered Edmonds–Karp* for each receiver. *Layered Edmonds–Karp* $(s, t_i)$ implies $O(maxFlow(t_i)^2)$ iterations of the do-while loop (lines 7–27 of *Layered Edmonds–Karp*), involving a call to *Layered BFS* and operations for updating of $M^i$ and $F^i$. Each execution of *Layered BFS* processes each link of the network at most once. However, it uses a priority queue to store vertices of the graph. Since at most $|V|$ vertices will ever be stored, each operation on the queue has complexity $O(\log |V|)$. Then, the complexity of each call to *Layered BFS* is bounded by $O(|E| + |V| \log |V|)$. The update of the values $M^i$ and $F^i$ after each call to *Layered BFS* (lines 8–27) may imply $O(|V|)$ iterations of the while loop (lines 10–25). In each iteration, the $M^i_{(u',v')}$ values of up to $|E|$ links $(u', v')$ may have to be updated. Hence, the complexity of the call *Layered Edmonds–Karp* $(s, t_i)$ is bounded by $O(|V|(|E| + \log |V|)maxFlow(t_i)^2)$. Let us define $maxFlow_{\max} = \max_{t \in T}\{maxFlow(t)\}$. Since it is unrealistic to have $|E| < \log |V|$, the complexity of the second part of *Layer Assignment* becomes $O(|T||V||E|maxFlow^2_{\max})$.

Therefore, the time complexity of the *Layer Assignment* algorithm can be bounded as $O(|T|(C_{mf} + |V||E|maxFlow^2_{\max}))$, where $C_{mf}$ is the complexity of the max-flow algorithm used. It is realistic to assume that the graphs considered are sparse, in which case $C_{mf} = O(|V||E|)$ [19]. Hence, the time complexity of *Layer Assignment* becomes $O(|T||V||E|maxFlow^2_{\max})$. On its hand, the complexity of the *Min-Req* and *Min-Cut* algorithms is dominated by the initial phase in which the maximum flow for each receiver (in *Min-Req*) or each node (in *Min-Cut*) is computed. This leads to complexities $\Theta(|T||V||E|)$ and $\Theta(|V|^2|E|)$, respectively, using the lower bound of $C_{mf} = \Omega(|V||E|)$ [22]. Then, the complexity of the *Layer Assignment* algorithm is larger than that of *Min-Req* by an $O(maxFlow^2_{\max})$ factor. (The value $maxFlow_{\max}$ is expected to be small in a practical setting.) The factor with respect to *Min-Cut* is $O(|T|maxFlow^2_{\max}/|V|)$, which may be $o(1)$ in practical settings.

Following the same lines as the time complexity analysis of the centralized version, it can be derived that the signaling complexity of distributed *Multi-Layer Max-Flow* is $O(|T|(S_{mf} + |V||E|maxFlow^2_{\max}))$ messages, where $S_{mf}$ is the signaling complexity of the max-flow algorithm used. The complexities of *Min-Req* and *Min-Cut* are $O(|T|S_{mf})$ and $O(|V|S_{mf})$ messages, respectively. Let us assume a state of the art maximum-flow algorithm is used [23], whose signaling complexity is $S_{mf} = O(|V||E|)$. Then, the distributed version of the algorithm has a signaling complexity that is only a factor $O(maxFlow^2_{\max})$ larger than that of *Min-Req*, and a factor $O(|T|maxFlow^2_{\max}/|V|)$ away from that of *Min-Cut*.

## VI. Simulation Results

For the performance analysis, we implement our *Multi-Layer Max-Flow* heuristic (called *ML-MaxFlow* in the legend of the graphs) as well as the *Min-Req* and *Min-Cut* algorithms from [11] in MATLAB. As main performance metrics, we use the average rate achieved by receivers as well as the fraction of nodes that achieve their max-flow value given in percent (similar to [11]). In addition, we measure network load in terms of number of links used by the algorithms as a fraction of the total number of links of the network topology. For the coding, we use random linear network coding over a sufficiently large finite field $GF(2^{10})$. Simulation results for each setting of parameters are averaged over 1000 runs. For each value, we also plot the 95% confidence interval for the average. Topologies are generated by randomly establishing links between the nodes of the network, ensuring that the resulting topology is connected and free of cycles. For the number of links, we set $|E| = \gamma|V|$ and use $\gamma = 3.7$ unless stated otherwise.[4] The algorithms are run on the same set of random topologies to avoid any bias due to unfavorable topologies for one particular algorithm.

### A. Different Network Sizes

We first analyze the impact of varying the network size. Fig. 10 shows the performance of the different algorithms for networks of 20–320 nodes. Given the constant ratio of number of edges to number of nodes of 3.7, the average max-flow value of the receivers increases slightly from 4.1 to 4.9 as the network size increases, as shown in Fig. 10(a). Note that there may not exist any network coding solution that achieves this max-flow. The 95% confidence intervals for the average rate are also shown, but since they amount to less than 1% of value of the average, they are barely visible in the plot.

The rates achieved by the *Min-Cut* and *ML-MaxFlow* algorithms are similar, in particular for small networks. As the network size (and thus the average max-flow value) increases, *ML-MaxFlow* makes effective use of the larger number of available paths between the source and the receivers, and the algorithm consistently achieves a rate above 90% of the max-flow value. Also, the rate of *Min-Cut* improves as it is increasingly likely to have high max-flow interior nodes throughout the network that prevent layer requests from low max-flow receivers from propagating all the way to the source. However, its rate remains below *ML-MaxFlow*, and the rate gap grows slightly larger for larger networks. For all network sizes, both algorithms are able to serve all or almost all receivers with a max-flow value of up to four at their full rate. However, *ML-MaxFlow* is better able to find paths to transport high layers to high max-flow receivers as the average max-flow increases. For *Min-Cut*, transporting high-layer flows all the way to the receivers requires that there are sufficiently many high max-flow

[4]Note that the authors of [11] restrict themselves to basic scenarios with only two to three layers and few receivers, whereas in our scenarios, we do not restrict the number of layers and we also explore larger topologies. As the algorithms' performance depends to a large degree on the structure of the topology, our results partly differ from the ones reported in [11].

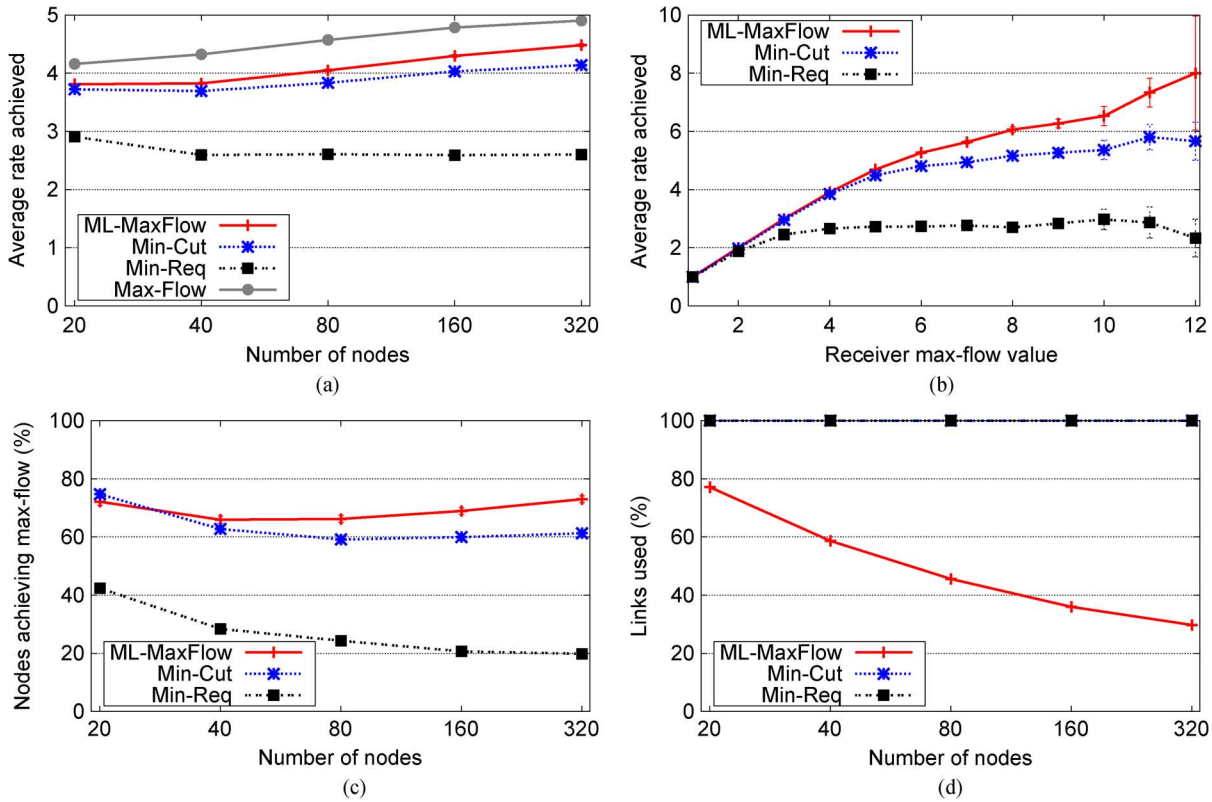



Fig. 10. Different network sizes of $|V| \in \{20, 40, 80, 160, 320\}$ nodes, for $|T| = 10$ receivers. (a) Average rate of the receivers. (b) Average rate per max-flow value for $|V| = 160$, $|T| = 10$. (c) Percentage of nodes that achieve their max-flow. (d) Fraction of links used for the network code.

interior nodes close to those receivers, and that the lower layers can be generated at interior nodes close to low max-flow receivers to prevent their requests from propagating too far up in the network. To illustrate this further, Fig. 10(b) shows the average rate of receivers grouped according to their max-flow values for the network with 160 nodes. As can be seen, the overall rate gap between *Min-Cut* and *ML-MaxFlow* is entirely due to *ML-MaxFlow* serving high-max flow receivers with a higher rate. The same holds true for all the other network sizes, thus the higher the number of high max-flow receivers, the larger the rate gap. The corresponding graphs (not shown) have a shape similar to Fig. 10(b).

The *Min-Req* algorithm achieves a substantially lower rate that even *decreases* as the network size increases. With a higher number of paths between source and receivers and a longer average path length, it becomes more and more likely that paths of low max-flow and high max-flow receivers overlap at some point along the path. Consequently, *Min-Req* propagates low-max flow requests on most of the paths and the rate achieved by receivers with max-flow larger or equal to four stays constant, as can be seen in Fig. 10(b). (Note that few of the random topologies have receivers with max-flow values of 11 and 12. Hence, those rate averages are largely dependent on the corresponding topologies, and the drop for *Min-Req* for a max-flow of 12 is only due to the low number of samples.)

When looking at the fraction of receivers that achieve their max-flow value in Fig. 10(c), the performance differences between the three algorithms are more pronounced. As expected, *Min-Req* performs poorly. Since this algorithm propagates the minimum layer request a node receives, most of the layer combinations on the links contain layers one to three, whereas the majority of the receivers have max-flows in the range of three to five. Both *ML-MaxFlow* and *Min-Cut* perform much better. Between the two algorithms, *ML-MaxFlow* serves a higher fraction of high max-flow receivers with a rate equivalent to their max-flow value and hence increasingly outperforms *Min-Cut* as the network size increases.

Interestingly, for the network with only 20 nodes, *ML-MaxFlow* achieves a higher average rate than *Min-Cut*, but with *Min-Cut*, a higher fraction of receivers achieve their max-flow. The smaller the interior of the network compared to the number of receivers, the more beneficial *Min-Cut*'s decoding at interior nodes becomes, which allows such nodes to locally generate high- as well as low-layer flows. As there are only nine interior nodes (as well as one source and 10 receivers), there are too few edge disjoint paths available for *ML-MaxFlow*, and *Min-Cut* does provide a slight advantage.

In addition to not requiring decoding at interior nodes and a performance on par with that of the *Min-Cut* algorithm, *ML-MaxFlow* also uses substantially fewer links as shown in Fig. 10(d). For a small topology of 20 nodes, less than 80% of the links are used, and as the network size increases, this fraction drops to only 30%, despite the increase in average rate. Due to their design, both *Min-Req* and *Min-Cut* always utilize all available links in the network, independently of the network size and the number of receivers.

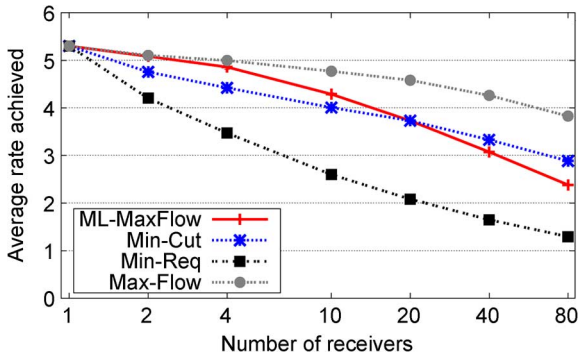



Fig. 11. Average rate of the receivers for different numbers of receivers.

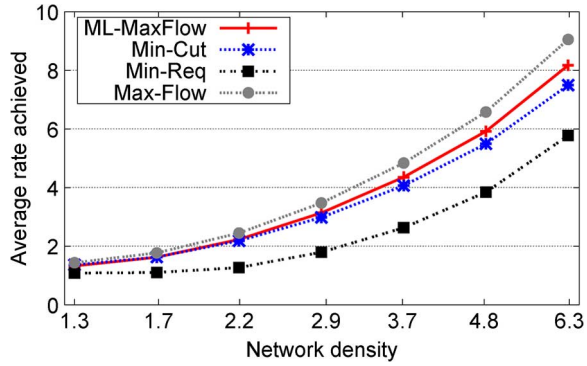



Fig. 12. Average rate of the receivers for different edge densities $|E| = \gamma|V|$.

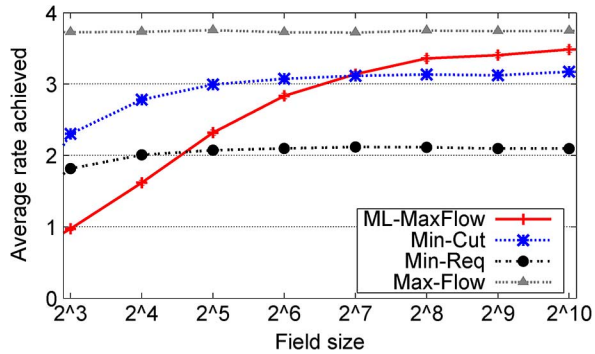



Fig. 13. Average rate of the receivers for different field sizes.

### B. Varying the Number of Receivers

We next explore the impact of varying the number of receivers in a network with $|V| = 160$ nodes. Since the total number of nodes is fixed, the higher the number of receivers, the lower the number of interior nodes. As a result, the average max-flow value drops from 5.3 to 3.8 as the number of receivers increases from 1 to 80.[5]

From Fig. 11, we observe that for a low number of receivers, the *ML-MaxFlow* algorithm outperforms the *Min-Cut* algorithm in terms of achieved rate, but when the number of receivers is a significant fraction of the total number of nodes, the opposite is true. In the latter case, paths of high max-flow and low max-flow receivers frequently overlap. Since without decoding, low max-flow receivers enforce low-layer constraints on the whole path from the receivers to the sender, few available paths for high max-flow receivers remain, despite *ML-MaxFlow* assigning only the necessary number of paths. In contrast, in a sufficiently dense network, the *Min-Cut* algorithm maintains high maximum layer constraints throughout the core of the network and only needs to decode close to the receivers to deliver exactly the right number of layers to each. Here, the additional flexibility with respect to layer assignments provided by the decoding at interior nodes pays off. In the extreme, when half of all nodes are receivers, *ML-MaxFlow* only achieves 80% of *Min-Cut*'s rate (rate of 2.4 versus 2.9). As in the previous experiment, both algorithms outperform the *Min-Req* algorithm by a large margin.

In terms of network utilization, with an increasing number of receivers, *ML-MaxFlow* uses more and more of the links, ranging from 25% of links used for the networks with one single receiver to 60% of links used for networks with 80 receivers (compared to 100% for *Min-Cut* and *Min-Req*).

### C. Different Network Densities

The relative performance of the algorithms remains similar for different nodes densities. In Fig. 12, we show the performance when varying the number of edges $|E| = \gamma|V|$, $\gamma \in \{1.3, 1.7, 2.2, 2.9, 3.7, 4.8, 6.3\}$. Both *ML-MaxFlow* and *Min-Cut* continue to have very similar performance with a slight performance advantage for *ML-MaxFlow* as the network density increases, and both outperform *Min-Req*. At the same time, link utilization (not shown) for *ML-MaxFlow* drops from 80% down to 34% as the node density increases since additional edges in parts of the network with low max-flow receivers remain unused. The smaller rate difference to *Min-Req* for very low network densities is due to the increased homogeneity of the network. Since we verify that the network is connected and each interior node has at least one incoming and outgoing edge, only few edges remain that can be assigned randomly. Since all algorithms achieve at least rate 1, the performance gap is small.

### D. Impact of Field Size

In the previous scenarios, we use a field size of $GF(2^{10})$ to isolate the performance of the layer allocation mechanisms from the impact of field size. In this section, we now analyze the impact of the field size.

The performance results given in Fig. 13 are for the worst case where a layer consists only of a single packet. Here, the redundant links used by *Min-Req* and *Min-Cut* may in fact provide additional useful linear combinations, in case some other linear combinations happen to be linearly dependent due the small field size. Nevertheless, for the most common practical field size of $GF(2^8)$, even for the extreme case of a layer size of one packet, the performance degradation of *ML-MaxFlow* is negligible. More importantly, these considerations are mainly relevant for very small layer sizes. For common layer sizes on the order of tens or hundreds of packets, the probability of noninnovative packets for a given field size is much lower [24], and thus the performance degradation, specifically for *ML-MaxFlow*, is much less pronounced.

[5]Although the total number of links remains unchanged, they are concentrated on fewer nodes, and it is more likely that some interior nodes have more links than can be used by the receivers below them. Thus, fewer of the links contribute to the max-flow.

## VII. Conclusion

In this paper, we investigated rate optimization for multirate multicast with interlayer network coding. We first proved NP-completeness of the problem. We then designed a heuristic algorithm that does not require decoding at interior nodes of the network and delivers a rate of at least the minimum of all the receivers' max-flow values to each receiver. Furthermore, the algorithm has some fairness property in the sense that allocating resources to a receiver so that it decodes a certain number of layers may only reduce the number of layers of receivers that decode more layers. We performed a complexity analysis and show that the complexity of our flow assignment is only moderately larger than the complexity of the Edmonds–Karp maximum flow algorithm. Through extensive simulations, we demonstrated that our algorithm vastly outperforms an existing heuristic that does not require decoding at interior nodes and even achieves multicast rates comparable to a heuristic *with* decoding at interior nodes. At the same time, our algorithm uses far fewer network resources.

## References


[1] J. Widmer, A. Capalbo, A. Fernández Anta, and A. Banchs, "Rate allocation for layered multicast streaming with inter-layer network coding," in *Proc. IEEE INFOCOM (Mini-Conf. Track)*, Orlando, FL, USA, Mar. 2012, pp. 2796–2800.

[2] R. Jain, "I want my IPTV," *IEEE Multimedia*, vol. 12, no. 3, p. 96, Jul. 2005.

[3] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.

[4] Y. Wang and A. Reibman, "Multiple description coding for video delivery," *Proc. IEEE*, vol. 93, no. 1, pp. 57–70, Jan. 2005.

[5] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[6] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Nov. 2003.

[7] S. Jaggi *et al.*, "Polynomial time algorithms for multicast network code construction," *IEEE Trans. Inf. Theory*, vol. 51, no. 6, pp. 1973–1982, Jun. 2005.

[8] X. Wu, B. Ma, and N. Sarshar, "Rainbow network flow of multiple description codes," *IEEE Trans. Inf. Theory*, vol. 54, no. 10, pp. 4565–4574, Oct. 2008.

[9] Z. Király and E. R. Kovács, "A network coding algorithm for multilayered video streaming," in *Proc. IEEE NetCod*, 2011, pp. 1–7.

[10] A. Gopinathan and Z. Li, "Optimal layered multicast," *Trans. Multimedia Comput., Commun., Appl.*, vol. 7, no. 2, Feb. 2011.

[11] M. Kim, D. Lucani, X. Shi, F. Zhao, and M. Medard, "Network coding for multi-resolution multicast," in *Proc. IEEE INFOCOM*, San Diego, CA, USA, Mar. 2010, pp. 1810–1818.

[12] J. Zhao, F. Yang, Q. Zhang, Z. Zhang, and F. Zhang, "LION: Layered overlay multicast with network coding," *IEEE Trans. Multimedia*, vol. 8, no. 5, pp. 1021–1032, Oct. 2006.

[13] C. Xu, Y. Xu, C. Zhan, R. Wu, and Q. Wang, "On network coding based multirate video streaming in directed networks," in *Proc. IEEE IPCCC*, Apr. 2007, pp. 332–339.

[14] N. Sundaram, P. Ramanathan, and S. Banerjee, "Multirate media streaming using network coding," in *Proc. 43rd Annu. Allerton Conf. Commun., Control, Comput.*, Sep. 2005.

[15] S. Lakshminarayana and A. Eryilmaz, "Multirate multicasting with intralayer network coding," *IEEE/ACM Trans. Netw.*, vol. 21, no. 4, pp. 1256–1269, Aug. 2013.

[16] M. Shao, X. Wu, and N. Sarshar, "Rainbow network flow with network coding," in *Proc. NetCod*, Jan. 2008, pp. 1–6.

[17] S. Dumitrescu, M. Shao, and X. Wu, "Layered multicast with inter-layer network coding," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 442–449.

[18] N. Thomos, J. Chakareski, and P. Frossard, "Prioritized distributed video delivery with randomized network coding," *IEEE Trans. Multimedia*, vol. 13, no. 4, pp. 776–787, Aug. 2011.

[19] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988.

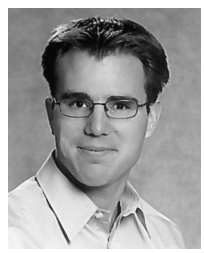
[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, Sep. 2009.

[21] T. Ho *et al.*, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[22] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *J. ACM*, vol. 45, no. 5, pp. 783–797, Sep. 1998.

[23] T. L. Pham, M. Bui, I. Lavallée, and S. H. Do, "An adaptive distributed algorithm for the maximum flow problem in the underlying asynchronous network," in *Proc. IEEE RIVF*, 2006, pp. 187–194.

[24] C. Cooper, "On the distribution of rank of a random matrix over a finite field," *Random Struct. Algor.*, vol. 17, no. 3–4, pp. 197–212, 2000.

**Joerg Widmer** (M'06–SM'10) received the M.S. and Ph.D. degrees in computer science from the University of Mannheim, Mannheim, Germany, in 2000 and 2003, respectively.

He is a Research Professor with IMDEA Networks Institute, Madrid, Spain. From 2005 to 2010, he was Manager of the Ubiquitous Networking Research Group, DOCOMO Euro-Labs, Munich, Germany, leading several projects in the area of mobile and cellular networks.

Dr. Widmer is a Senior Member of the Association for Computing Machinery (ACM).

**Andrea Capalbo** received the B.Sc. and M.Sc. degrees in computer engineering from Politecnico di Torino, Turin, Italy, in 2007 and 2009, respectively.

He worked as a Research Assistant with IMDEA Networks Institute, Madrid, Spain, from 2011 to 2012. He currently works as a consultant for Altran, Madrid, Spain, in the VPN Solutions Design & Technology Department of Telefonica International Wholesale Services.

**Antonio Fernández Anta** (M'98–SM'02) received the M.Sc. and Ph.D. degrees in computer science from the University of Louisiana, Lafayette, LA, USA, in 1992 and 1994, respectively.

He is a Research Professor with IMDEA Networks Institute, Madrid, Spain. Previously, he was a faculty member with the Universidad Rey Juan Carlos, Madrid, Spain, and the Universidad Politécnica de Madrid, Madrid, Spain, where he received an award for his research productivity.

Dr. Fernández Anta has been a Senior Member of the Association for Computing Machinery (ACM) since 2007.

**Albert Banchs** (M'04–SM'12) received the M.Sc. degree in telecommunications engineering and Ph.D. degree in telematics engineering from the Polytechnic University of Catalonia, Barcelona, Spain, in 1997 and 2002, respectively.

He was a Visiting Researcher with the International Computer Science Institute (ICSI), Berkeley, CA, USA, in 1997. He worked for Telefonica I+D, Barcelona, Spain, in 1998, and for NEC Europe, Ltd., Heidelberg, Germany, from 1998 to 2003. He has been with the University Carlos III of Madrid, Madrid, Spain, since 2003. Since 2009, he also has a double affiliation as Deputy Director of IMDEA Networks Institute, Madrid, Spain.