Rate Allocation for Layered Multicast Streaming with Inter-Layer Network Coding

Joerg Widmer*, Andrea Capalbo*†, Antonio Fernández Anta*, Albert Banchs*†

* Institute IMDEA Networks, Madrid, Spain
† University Carlos III of Madrid, Spain
Email: {firstname.lastname}@imdea.org

***Abstract*—Multi-layer video streaming allows to provide different video qualities to a group of multicast receivers with heterogeneous receive rates. The number of layers received 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 (intra-layer) or across layers (inter-layer), and in general inter-layer coding outperforms intra-layer 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 inter-layer network coding problem without decoding at interior nodes. We propose a heuristic algorithm for rate allocation and code assignment based on the Edmonds-Karp maximum flow algorithm and perform simulations that show that our algorithm may even outperform other heuristics that do require decoding at interior nodes.**

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 bitrate (and hence video quality), IP multicast is a suitable solution [1], an approach used in today's IP-TV broadcasting systems. When multicasting to receivers with heterogeneous receive rates, layered coding allows to distribute the stream over several multicast flows such that the higher the number of flows (and thus the overall rate) a receiver obtains, the better the video quality at that receiver. The base layer provides a basic video quality and each additional enhancement layer received refines this quality [2]. A higher enhancement layer can only be decoded if the base layer and all lower enhancement layers are already available.

Multicast with network coding [3] provides capacity gains over plain multicast routing. While network coding achieves the maximum flow (max-flow) [4] to each receiver for single-rate multicast, solutions that achieve the same for the case of multi-rate multicast do not always exist [5], i.e., an optimal solution may provide less than the max-flow to some receivers. Network coding solutions for single-rate multicast networks can be found in polynomial time [6], whereas the network coded multi-rate multicast problem is NP-hard [7].

In multi-rate multicast, network coding can be performed within a layer (intra-layer coding) or across layers (inter-layer coding). Intra-layer network coding is conceptually simpler than inter-layer network coding. A basic approach for intra-layer network coding was proposed in [8]. Receivers are grouped into subsets 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 [6]. The rate of the second layer is set such that, given the remaining capacity, it can be received by the group of receivers with the second lowest rate. Then a multicast network code for the second layer is constructed, and so on. 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.

In general, inter-layer coding outperforms intra-layer coding, but few practical heuristics for the inter-layer network coding problem exist. The two most prominent heuristics were presented in [9]. Both algorithms first determine the max-flow (equivalent to the minimum cut) value to each receiver, using some well known max-flow algorithm (e.g., Ford-Fulkerson). 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*, the receivers propagate as rate requirement their max-flow value to their parent nodes. A node waits until it heard from all its children and then propagates the minimum of the max-flow values to its parents, and so on. (Parents and children are the next nodes toward the source and toward the receivers.) The source then sends linear combinations on its outgoing links coded over as many layers as is allowed by the maximum layer constraints of those links. The rationale for propagating up the minimum is that an interior node can simply code over all incoming links to generate a linear combination for an outgoing edge, without running the risk that a downstream receiver may not be able to decode. In the second algorithm called *Min-Cut*, an interior node propagates up its own max-flow value instead of the minimum value of its children, in case that value is larger than the minimum of the children's max-flows. This allows the interior node to decode up to a number of layers corresponding to its own max-flow

This research was supported in part by the European Commissions Seventh Framework Programme (FP7-ICT-2009-5) under grant agreement n. 258053 (MEDIEVAL project), Comunidad de Madrid grant S2009TIC-1692, Spanish MICINN grants TIN2008-06735-C02-01 and TEC2011-29688-C02-01, and National Natural Science Foundation of China grant 61020106002.

value. The decoded layers can then be recombined to serve linear combinations coded over different numbers of layers to different children in contrast to the first algorithm.

While both algorithms are simple and can be implemented in a distributed manner, they have some disadvantages. The *Min-Req* algorithm does not require decoding at interior nodes but since it propagates minimum max-flow values, its performance in topologies where receivers have heterogeneous max-flow values may be low. The *Min-Cut* algorithm fares better in such topologies as long as some of the interior nodes have sufficiently high max-flow values, but does require decoding at interior nodes. Decoding involves computationally heavy Gaussian elimination and storing data in its entirety before decoding is possible. The resulting complexity and delay are particularly undesirable at interior nodes (the routers) of the network, where processing overhead is one of the main bottlenecks. 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 but, more importantly, wastes capacity and thus may prohibit their use in networks where links are shared with other flows.

In this paper, we focus on the *inter-layer network coding problem without decoding at interior nodes*. We propose a heuristic solution of polynomial complexity that is based on the Edmonds-Karp max-flow algorithm. Instead of propagating traffic along all the links of the network, each receiver runs modified version of Edmonds-Karp max-flow algorithm to find up to a number of paths that correspond to the max-flow from to the source to the receiver. Paths have layer constraints so that information carried on those paths is not coded over more layers than can be decoded at downstream receivers. While paths are node disjoint on a per-receiver basis, paths of different receivers can overlap, as long as the layer constraints of the involved receivers are compatible. Network coding (but no decoding) is required at interior nodes where paths merge. We analyze the performance of our algorithm through simulations and show that it outperforms other heuristics that do not require decoding at interior nodes, and in many cases even reaches similar or better performance than algorithms that do require decoding at interior nodes. At the same time, our algorithm uses far fewer network resources.

II. Algorithm

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 $T = \{t_1, t_2, \ldots, t_r\} \subset V$. Further, 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_l is only useful to a receiver, if the receiver also receives layers $L_1, \ldots, L_{l-1}$. Let $y(u,v) = \sum_{j=1}^{l} g_j L_j$ denote the coded layer combination that is sent on edge (u, v) with coding coefficients g_j. We have that $y(u,v) \in \langle \cup_{(w,u)\in In(u)} y(w,u) \rangle$ $\forall u \in V \setminus T \setminus \{s\}$ 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 [9]. Since our algorithm is directly based on the Edmonds-Karp max-flow algorithm, it is straightforward to extend it to non-unitary edge capacities and non-unitary layer rates.

The problem under consideration is to maximize the sum of the receive rates of all receivers, under a max-min fairness constraint. A given allocation is max-min fair if it is not possible to increase the number of layers received by any receiver t_i, without reducing the number of layers of another receiver t_j to less than that of receiver t_i. In a connected graph with unit capacity edges as in our model, this ensures that each receiver is at least able to receive the base layer L_1. In our model, the source sends data of a layer always combined with data from all lower layers.

Our heuristic first determines the max-flow from the source to each receiver using the Edmonds-Karp algorithm. It then processes the receivers in ascending order of their max-flow values. This ensures that capacity is first allocated to receivers with low max-flows. The assignment is closely related to the notion of max-min fairness. For the assignment of layers to flows, we design a modified multi-layer version of the Edmonds-Karp maximum flow algorithm called *MultiLayer-MaxFlow*. The algorithm in turn uses a layer constrained version of breadth first search *LayeredBFS*. *LayeredBFS* tries to find one additional edge disjoint path with specific layer constraints for the current receiver. We first present the pseudocode and then give an example of the algorithm.

A. Algorithm Pseudocode

For simplicity we describe the algorithm as a centralized algorithm, but its distributed implementation is straightforward.

1) LayerAssignment (Fig. 1): The algorithm executes *MultiLayer-MaxFlow* for each receiver in ascending order of max-flows. *MultiLayer-MaxFlow* returns the set of the i-th receiver's flow allocation F_i on edge disjoints paths, as well as the set of maximum layer constraints M_i that ensure decodability of the information received through these paths. With these, the global flow allocation F and maximum layer constraints M are updated (Lines 11-12).

2) MultiLayer-MaxFlow (Fig. 2): The algorithm tries to find paths from source s to receiver t that allow the receiver to decode the maximum number of layers ℓ_{max}, given the existing flow assignments (starting with $\ell_{max} = maxFlow(t)$). This requires finding ℓ_{max} edge-disjoint paths that deliver linear combinations of the first ℓ_{max} layers. To ensure linear independence, a receiver may receive at most l combinations coded over the first l layers, $1 \leq l \leq \ell_{max}$. This implies the following minimum layer constraints for the linear combinations $y_1, \ldots, y_{\ell_{max}}$:

$$y_{\ell_{min}} = \sum_{j=1}^{\ell} g_j L_j, \ \ell_{min} = 1, \ldots, \ell_{max}, \ \ell \in [\ell_{min}, \ell_{max}] \quad (1)$$

```
1  algorithm LayerAssignment(G, s, T)
2  ∀(u,v) ∈ E : F((u,v)) ← 0, M((u,v)) ← ∞
3  for each t_i ∈ T
4  |   maxFlow(t_i) ← EdmondsKarp(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) ← MultiLayer-MaxFlow(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. 1. Layer assignment algorithm

```
1  procedure MultiLayer-MaxFlow(s, t)
2  for ℓ_max ← maxFlow(t) down to 1
3  |   ∀(u,v) ∈ E : F_i((u,v)) ← 0, M_i((u,v)) ← ∞
4  |   ℓ_min ← ℓ_max
5  |   do
6  |   |   (P, found, update) ← LayeredBFS(s, t, ℓ_min, ℓ_max)
7  |   |   if found then
8  |   |   |   u ← s
9  |   |   |   while u ≠ t
10 |   |   |   |   (v, ℓ) ← P(u)
11 |   |   |   |   if update(u) then
12 |   |   |   |   |   ∀(u', v') upstream of u that carry a flow
13 |   |   |   |   |   that contributes to (u, v) :
14 |   |   |   |   |   |   M_i((u', v')) ← min{M_i((u', v')), ℓ}
15 |   |   |   |   end if
16 |   |   |   |   if (u, v) ∈ Out(u) then
17 |   |   |   |   |   M_i((u,v)) ← ℓ
18 |   |   |   |   |   F_i((u,v)) ← 1
19 |   |   |   |   else // reverse edge
20 |   |   |   |   |   M_i((v,u)) ← ∞
21 |   |   |   |   |   F_i((v,u)) ← 0
22 |   |   |   |   end if
23 |   |   |   |   u ← v
24 |   |   |   end while
25 |   |   |   ℓ_min ← ℓ_min − 1
26 |   |   end if
27 |   while (ℓ_min ≥ 1) ∧ found
28 |   if found then
29 |   |   return (F_i, M_i)
30 |   end if
31 end for
```

Fig. 2. MultiLayer-MaxFlow algorithm

When assigning flows, our algorithm first searches for a flow with $\ell_{min} = \ell_{max}$, then for $\ell_{min} = \ell_{max} - 1$, and so on. If *LayeredBFS* fails to return a layer-ℓ path P with $\ell_{min} \leq \ell \leq \ell_{max}$ (and thus $found = \text{FALSE}$), the receiver will not be be able to decode ℓ_{max} layers. In that case, *MultiLayer-MaxFlow* reduces ℓ_{max} by one and all existing temporary flows F_i and maximum layer constraints M_i are removed (Lines 2-3).

Whenever *LayeredBFS* returns with a valid path, ℓ_{min} is decremented and the path is backtracked from s to t. F_i and M_i are updated on all the edges of the path (Lines 16-22). In addition, nodes maintain a table that maps incoming to outgoing edges (for simplicity, this is not shown in the pseudo-code). A two-hop path segment $(w,u),(u,v)$ creates

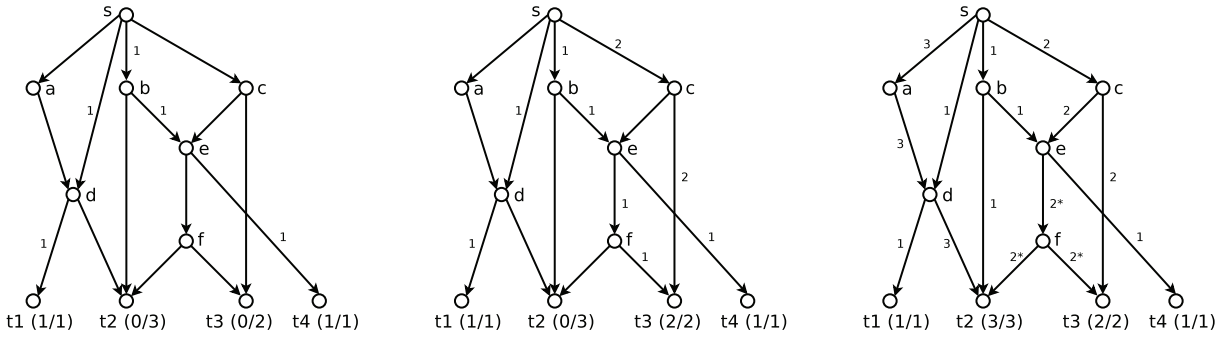
```
1  procedure LayeredBFS(s, t, ℓ_min, ℓ_max)
2  ∀v ∈ V : P(v) ← (⊥, ∞), δ(v) ← ∞, update(v) ← FALSE
3  δ(t) ← 0;
4  Q ← ∅ // priority queue
5  enqueue(Q, (t, δ(t))) // enqueue t with highest priority 0
6  while Q ≠ ∅ do
7  |   v ← dequeue(Q)
8  |   if v = s then
9  |   |   return (P, TRUE, update)
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((u,v))
19 |   |   |   if ℓ ≥ M((u,v)) then
20 |   |   |   |   P(u) ← (v, M((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((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. 3. Layered BFS algorithm

an entry $(w,u) \mapsto (u,v)$ at the local table of node u. (w,u) may map to multiple outgoing edges, and multiple incoming edges may map to (u,v). The algorithm also checks whether a special flag $update(u)$ is set for a node u on the path. 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 (Lines 11-15). The edges can easily be identified through the *in-out* tables maintained at the nodes.

3) LayeredBFS (Fig. 3): The algorithm performs breadth first search to find a path from a receiver t to source s with matching layer constraints. It uses a priority queue Q to store nodes to be visited next, as well as a mapping $P : u \mapsto (v, \ell)$ to store next hop nodes for backtracking and the corresponding maximum layer constraint ℓ for edge (u,v).

The algorithm removes the first node from the queue and explores all forward edges that are not yet used by t (Lines 13-28) and virtual reverse edges that are used by t (Lines 29-37). (Note that $P(u) = (\perp, \cdot)$ marks an unvisited node.) Forward edges that are entirely unused are enqueued with a cost of



(a) Flow assignment: max-flow 1 receivers (b) Flow assignment: max-flow 2 receivers (c) Flow assignment: max-flow 3 receiver

Fig. 4. Different stages of the *MultiLayer-MaxFlow* algorithm: layer assignment per link and rate achieved vs. maximum flow per receiver

$\delta(v)+1$, since 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 can be used at cost 0 if the current maximum layer constraint $M((u,v))$ can be kept (Line 19). If, however, the constraint has to be reduced since $\ell < M((u,v))$, this means that other receivers downstream of (u,v) may no longer be able to decode up to $M((u,v))$ but only up to ℓ layers (Line 22). This is only allowed in case no other options exist. Only when $\ell = \ell_{max} < M((u,v))$, i.e., the node is trying to obtain less layers than $M((u,v))$, is the corresponding node enqueued, and it is enqueued last with a high cost of $\delta(v)+|E|$. The marker $update(u)$ is set to indicate that the maximum layer constraints on edges upstream of u may have to be updated by *MultiLayer-MaxFlow*.

As in the original Edmonds-Karp algorithm, whenever capacity is assigned to a flow on edge $(u,v) \in E$, a corresponding negative flow is assigned to a virtual reverse edge (v,u) [4]. In case the *LayeredBFS* algorithm finds a suitable path through such a reverse edge, the flow originally assigned on the forward edge (u,v) is removed, if (u,v) is only used by the current receiver. If however, edge (u,v) is a shared edge, the flow on that edge is required by another receiver and cannot be removed. The *LayeredBFS* 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*. When traversing reverse edges that are only used by t (Lines 31-32), the cost is set to $\delta(v)-1$ since capacity on an edge is freed, and the layer constraint is reset to ℓ_{max}. Traversing shared reverse edges leaves the cost $\delta(v)$ unchanged, and the maximum layer constraint is reset to that of the shared edge $M((u,v))$, since network coding occurs after the reverse edge.

4) Code assignment: After the *LayerAssignment*, the source sends (linearly independent) combinations on each outgoing link, coded over as many layers as is allowed by the link's layer constraints.

$$y(s,v) = \sum_{l=1}^{M((s,v))} g_l^{(s,v)} L_l, \quad \forall (s,v) \in Out(s) \tag{2}$$

An interior node u of the network codes according to its *in-out* table.

$$y(u,v) = \sum_{w:(w,u)\mapsto(u,v)} g^{(w,u)} y(w,u) \tag{3}$$

B. Example of the Algorithm

As in the original Edmonds-Karp algorithm, we repeatedly perform a breadth-first search to find augmenting paths to the source. However, the *LayeredBFS* starts the search at the receivers and proceeds upstream to the source. This allows to find existing flows that can be reused, as explained later. In the example in Fig. 4(a), first receivers t_1 and t_4 with a max-flow of 1 find shortest paths to obtain L_1 directly from the source.

The max-flow 2 receiver t_3 first 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. In the example in Fig. 4(b), a suitable L_2-path (s,c,t_3) is found. The receiver then uses *LayeredBFS* to find an edge-disjoint path carrying L_1 or a combination of L_1 and L_2. The algorithm may traverse edges that are already in use by other receivers at no cost, as long as the layer constraints on that path are not reduced. After traversing the two edges to reach node e, the algorithm traverses the remaining path to s before exploring any new edges that are unused as of yet. 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).

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 that only L_1 can be transported. 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 *LayeredBFS* can continue via the shared edge and subsequently finds a suitable combination of

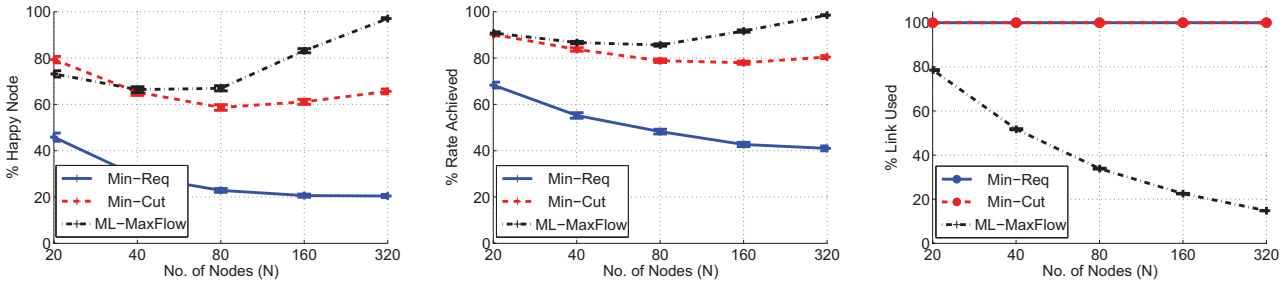


Fig. 5. Fraction of nodes that achieve their max-flow (left), normalized rate (middle), and link usage (right) for different network sizes and 10 receivers

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). Note that while this alters the linear combination received on the subtree below edge (e, f), i.e., also receiver t_3 will now receive a linear combination over the first two layers rather than just layer 1, this does not change decodability at t_3.

In this example, the *MultiLayer-MaxFlow* algorithm achieves the max-flows for all receivers, whereas neither the *Min-Req* and *Min-Cut* algorithms nor the intra-layer approach manage to deliver all three layers to receiver t_2.

III. Simulation Results

For the performance analysis, we implement our *MultiLayer-MaxFlow* heuristic (called *ML-MaxFlow* in the legend of the graphs) as well as the *Min-Req* and *Min-Cut* algorithms from [9] in Matlab. While we do not show results for intra-layer coding and Steiner-tree packing, we note that in the simulations shown in [9] these algorithms have a performance similar to the *Min-Req* algorithm and always perform worse than the *Min-Cut* algorithm. For ease of comparison, we use the same metrics of [9] of "Happy Nodes", the fraction of nodes that achieve their max-flow value, and "Rate Achieved", the average of actual rate achieved normalized by the max-flow value. In addition, we measure network load in terms of number of links used vs. total number of links. For the coding we use random linear network coding over a finite field $GF(2^{10})$. Simulation results are averaged over 1000 runs. We generate random, connected, cycle-free topologies with $|E| = 3.7|V|$ links.

Fig. 5 shows the performance of the different algorithms for networks of $20 - 320$ nodes (including 95% confidence intervals). All algorithms perform relatively well for small networks since the low max-flow values are easier to achieve. The disadvantage of the *Min-Req* algorithm becomes apparent as soon as the network size increases. The fraction of paths that intersect with low max-flow receivers increases as well, and the fraction of happy nodes that achieve their max-flow, as well as the normalized rate achieved both decrease. The *Min-Cut* algorithm performs significantly better. While the normalized rate achieved is relatively stable as the number of nodes increases, the fraction of happy nodes in fact increases. As the network becomes more homogeneous with increasing size, more low max-flow receivers manage to achieve the max-flow but this is compensated by the loss of normalized rate at some high max-flow receivers that no longer manage to obtain the higher layers they require. For *ML-MaxFlow*, both the fraction of happy nodes as well as the normalized rate increase with a network size beyond 80 nodes. Similarly to the *Min-Cut* algorithm, more low max-flow receivers achieve their max-flow. Since only the necessary number of paths is allocated to low max-flow receivers, high max-flow receivers can exploit the remaining paths to obtain higher layers. *ML-MaxFlow* also uses much fewer links than *Min-Req* and *Min-Cut* which both use 100% of the links. 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 15%.

IV. Conclusions

In this paper we investigated rate optimization for multi-rate multicast with inter-layer network coding. We 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. We show that our algorithm outperforms an existing heuristic that does not require decoding at interior nodes and even provides similar or better multicast rates than a heuristic *with* decoding at interior nodes, while using fewer network resources.

References

[1] R. Jain, "I want my IPTV," *IEEE Multimedia*, vol. 12, no. 3, Jul. 2005.
[2] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Scalable Video Coding*, vol. 17, no. 9, Sep. 2007.
[3] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, Sep. 2009.
[5] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Trans. on Networking*, vol. 49, no. 11, Nov. 2003.
[6] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, 2005.
[7] X. Wu, B. Ma, and N. Sarshar, "Rainbow network flow of multiple description codes," *IEEE Transactions on Information Theory*, vol. 54, no. 10, pp. 4565–4574, 2008.
[8] N. Sundaram, P. Ramanathan, and S. Banerjee, "Multirate media stream using network coding," *43rd Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2005.
[9] M. Kim, D. Lucani, X. Shi, F. Zhao, and M. Medard, "Network coding for multi-resolution multicast," in *IEEE Infocom*, San Diego, CA, Mar. 2010.