CodedStream: Live Media Streaming with Overlay Coded Multicast

Jiang Guo, Ying Zhu, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 3G4
Canada

ABSTRACT

Multicasting is a natural paradigm for streaming live multimedia to multiple end receivers. Since IP multicast is not widely deployed, many application-layer multicast protocols have been proposed. However, all of these schemes focus on the construction of multicast trees, where a relatively small number of links carry the multicast streaming load, while the capacity of most of the other links in the overlay network remain unused. In this paper, we propose *CodedStream*, a high-bandwidth live media distribution system based on end-system overlay multicast. In CodedStream, we construct a k-redundant multicast graph (a directed acyclic graph) as the multicast topology, on which *network coding* is applied to work around bottlenecks. Simulation results have shown that the combination of k-redundant multicast graph and network coding may indeed bring significant benefits with respect to improving the quality of live media at the end receivers.

Keywords: Network Coding, Overlay Multicast, Media Streaming, Multiple Description Coding

1. INTRODUCTION

When distributing streaming media to multiple receivers, it is natural to utilize multicast, as opposed to naive all-unicast from the source to all the receivers. Due to the lack of a widely available IP multicast service at the network layer in backbone networks, recent research (*e.g.*,[1–3]) has examined the feasibility and trade-offs of implementing multicast services in the application layer. In these studies, a multicast tree rooted at the source is formed, with the receivers as members of the multicast group. Each node in the tree transmits the received stream to each of its children using unicast. However, previous proposals have failed to address the following two problems. First, a tree is vulnerable to node departures or link failures, especially since overlay nodes are far from being dedicated servers, and may join, leave or fail in a transient manner. Second, the underlying topology resource is rather under-utilized, as only a relatively small number of links are utilized due to the tree structure. In this case, only a single path is assigned to each receiver, leaving many potentially high-quality alternative paths in the overlay untapped.

In this paper, we propose a novel approach, *CodedStream*, to significantly improve the performance of distributing live media to multiple receivers. We deviate from the conventional view that data can only be replicated and forwarded by overlay nodes. Rather, as end systems, these overlay nodes also have the full capability of encoding and decoding data. We apply the mechanism of *network coding*[4–6] in a subset of relaying overlay nodes, with the aim to increase throughput while keeping overhead low. In addition, we also depart from the traditional wisdom that the *multicast topology needs to be a tree* from source to receivers, in which, from the source to each receiver, there exists only one path; rather, we seek to construct a k-redundant *multicast graph* (a directed acyclic graph) as the multicast topology, in which network coding is applied. However, the fundamental trade-off involved in this solution is that, introducing redundant paths may lead to higher *link stress*, *i.e.,* higher number of paths passing through the same physical network link. This will likely result in bottlenecks at the high-stress links, inflicting negative effects on the performance that could potentially be severe enough to defeat the purpose of having multiple redundant paths. To resolve this issue, we resort to the advantages of network coding, as well as a meticulously designed algorithm for multicast topology construction.

E-mail: {jguo, yz, bli}@eecg.toronto.edu

Our strategy is built on two cornerstones: (1) Network coding in the k-redundant multicast graph which provides k redundant paths from the source to each of the receivers; and (2) *Multiple Description Coding* (MDC).[7,8] We propose to encode the live media source into k separate streams, or *descriptions*, and transmit one description down each of the k paths. With such a design, CodedStream is able to achieve significant performance improvement at the receivers. Meanwhile, CodedStream also offers improved robustness to node failures and sudden node departures, since any subset of these k descriptions can be received and decoded into a stream with distortion (with respect to the original stream).

The key challenge in the design of CodedStream is to construct the k-redundant multicast graph that minimizes link stress and bottlenecks, on which we apply network coding. Network coding requires specific overlay topologies that conform to specific patterns, such multicast graphs might not exist in the overlay networks that consist of only the source node and the receivers; therefore, we recruit cooperative high-bandwidth relay nodes in the overlay network as *core* nodes , which do not belong to the multicast group. We note that such a pool of cooperative nodes is the price we pay to exploit the power of network coding and to significantly increase streaming throughput. The problem of provisioning incentives to encourage such cooperation in peer-to-peer systems has been addressed in complementary work (*e.g.*,[9]), and is beyond the scope of this paper.

The remainder of this paper is organized as follows. Sec. 2 briefly reviews the concept of network coding. Sec. 3 presents preliminaries towards the CodedStream design. Sec. 4 discusses the design of CodedStream in details. Sec. 5 illustrates our results of performance evaluation using simulations. Finally, Sec. 6 evaluates our proposal in the context of related work, and Sec. 7 concludes the paper.

2. NETWORK CODING

The information-theoretic aspect of network coding was first proposed and studied by Ahlswede *et al.*.[4] It is effective due to its departure from the traditional routing framework where nodes only store, replicate and route data. With network coding, nodes have the additional capability of encoding and decoding data at the packet level using efficient linear codes; the aim is to use bandwidth more efficiently and thereby increase network capacity.

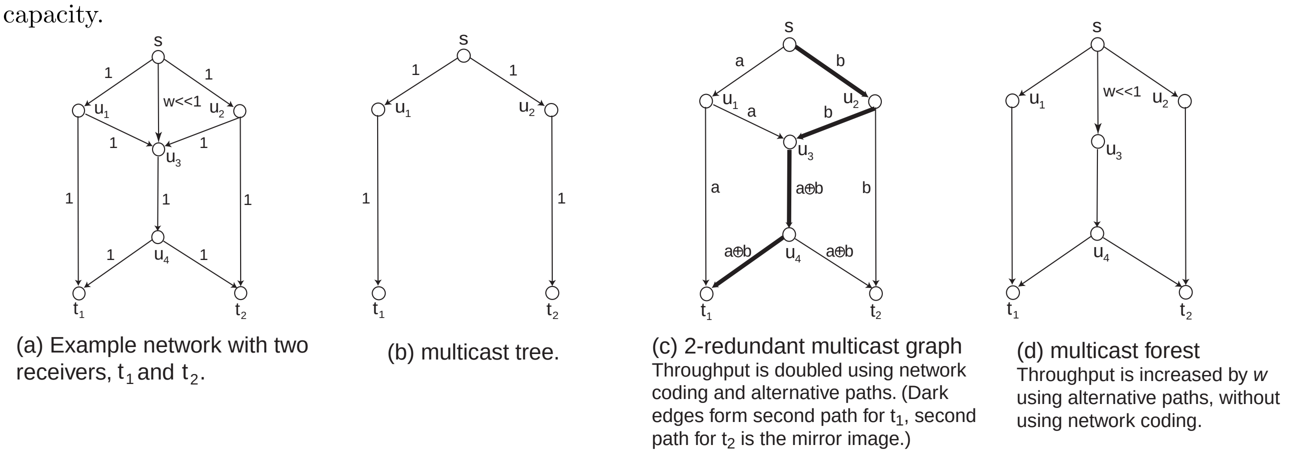

s
u1
u2
u3
u4
t1
t2
w<<1
1
a
b
a⊕b
(a) Example network with two receivers, t1 and t2.
(b) multicast tree.
(c) 2-redundant multicast graph
Throughput is doubled using network coding and alternative paths. (Dark edges form second path for t1, second path for t2 is the mirror image.)
(d) multicast forest
Throughput is increased by w using alternative paths, without using network coding.

Figure 1. The effects of network coding: an example.

We briefly review the concepts of network coding with an example shown in Fig. 1. The overlay network is represented by the graph in Fig. 1(a), in which t_1 and t_2 are the two receivers in the multicast group, and s is the source. Each edge has the same unit bandwidth of 1 except that the bandwidth available on edge (s, u_3) is $w \ll 1$. This is the case when, for example, s can not sustain an outgoing bandwidth of much more than 2. The usual all-widest-paths multicast tree is shown in Fig. 1(b); the widest alternative paths are added in Fig. 1(c), while the other choice of (narrower) alternative paths are shown in Fig. 1(d).

Without network coding, it is impossible to double throughput in Fig. 1(c), since the alternative paths to t_1 and t_2 interfere with each other's widest paths such that they cannot both double their throughput. It can be verified that the achievable throughput is only $3C/2$. The conflict can be eliminated by choosing the paths in Fig. 1(d), but the bandwidth of the alternative paths is much less (narrower) than that of the original tree, again making it not feasible to double the throughput.

With network coding, however, the graph in Fig. 1(c) can be safely used to double the multicast throughput to both receivers. The source transmits two units of information a and b to u_1 and u_2, respectively, and u_3 encodes a, b as $a \oplus b$, which is being transmitted to u_4. Receiver t_1 receives a and $a \oplus b$, and decodes to obtain both a and b; similarly for t_2.

Clearly, the topology in Fig. 1(d) represents the intuitive approach of constructing k distinct multicast trees, referred to as the *multicast forest*. The intuitive comparison in this example demonstrates the power of network coding in multicast acyclic graph topologies beyond multicast forests, with respect to improving session throughput. Such tremendous power of network coding lies in the fact that any conflicts resulting from interfering paths in the multicast graph can be surmounted to obtain the same throughput for each receiver, as if it were the only receiver.

However, network coding is not the panacea when it comes to increasing multicast session throughput. There exist many topologies — including all forms of multicast trees — where network coding fails to be more effective with respect to improving throughput. It helps to increase throughput only in network graphs that conform to special patterns. Overlay networks have exactly the properties that could be leveraged to employ network coding to achieve higher throughput in application-layer multicast, since (1) links, and therefore paths, can be constructed without too much complexity in overlays, in order to build a desired topology for application-layer coded multicast; and (2) all the nodes, being end systems, are capable of encoding and decoding.

3. CODEDSTREAM: PRELIMINARIES

The main concept of CodedStream is to combine source-based multiple description coding (MDC) with network-coded multicasting. A media stream is coded into k multiple complementary descriptions using MDC, and these descriptions are subsequently multicasted in a constructed multicast graph (rather than a tree) using network coding.

The quintessential strength of CodedStream lies in culling the unique advantages of both MDC and network-coded multicast. The benefit of media streaming with MDC is the resilience to lossy links, whereas the advantage of network-coded multicast is the higher end-to-end throughput. CodedStream achieves both loss-resilience and high throughput, as will be confirmed by the simulation results.

For clarity, we list the notations in Table 1.

Table 1. List of notations

Notation	*Definition*
s	the source node
u_i	each core node
t_i	each receiver node
k	the number of disjoint paths from s to t_i
n	the number of receiver nodes within the multicast group
α	the original live media signal
α'	the received media signal at t_i
v_i	the vector of linear codes assigned to each node i
V_{in}	the vector of incoming coded streams
V_{out}	the vector of outgoing coded streams
$\mathcal{T}$	the set of core nodes or receivers that have k disjoint paths from the source

3.1. k-redundant Multicast Graph

To apply network coding, it is required that (1) from the source s to each receivers t_i, there exist k ($k \geq 2$) disjoint paths; and (2) no directed cycles exist in the multicast graph. With Proposition 1 below, we can prove that, with only the source s and the set of receivers, it is impossible to arrive at a topological pattern that is amiable to network coding. We omit all proofs due to space constraints. Refer to[10] for detailed proofs.

Proposition 1: A multicast graph with only receivers, *i.e.,* every node besides s has k disjoint directed paths from s, contains a directed cycle.

In this paper, we propose to construct a *k-redundant multicast graph* which enables us to apply network coding. A k-redundant multicast graph for single-source multicast is a directed acyclic graph (DAG) that consists of three components: the source s, the *core* nodes that are not members of the multicast group, and the *receiver* nodes that are multicast group members. Formally, a k-redundant multicast graph has the following two properties:

1. The set of all nodes, A, is the union of three disjoint subsets $\{s\} \cup A_c \cup A_T$:
    - 1.1 $\{s\}$, the source, indegree$(s) = 0$, outdegree$(s) = k$;
    - 1.2 A_c, the *core* nodes (who are not members of the multicast group), denoted by $u_i, 1 \leq i \leq n_C$ (number of core nodes), $1 \leq$ indegree$(u_i) \leq k$ and outdegree$(u_i) > 0$;
    - 1.3 A_T, the *receiver* nodes (*i.e.,* multicast group members), denoted by $t_i, 1 \leq i \leq n_T$ (number of receivers), indegree$(t_i) = k$ and outdegree$(t_i) \geq 0$.
2. If each edge in the graph has unit bandwidth, then for any node v whose indegree is k, the individual maximum flow of v is k (since the minimum cut is k).

We proceed to establish the relation between individual maximum flow and the number of disjoint paths from the source s.

Proposition 2: Given a receiver node t with indegree k in a k-redundant multicast graph with source s and unit-bandwidth edges, t has k and no more than k disjoint paths from s if and only if t has individual maximum flow of k.

The most desirable property of k-redundant multicast graph is that, from the source s to each receiver t, there exist k disjoint paths. This enables us to apply network coding to ensure maximum throughput for each receiver.

3.2. Multiple Description Coding

Multiple description coding (MDC) is a method of encoding audio and/or video signal into $M > 1$ separate stripes, or *descriptions*, such that any subset of these descriptions that are received can be decoded into a signal with distortion. If any $m \leq M$ stripes are received, then the received streaming media quality is m/M with respect to the original signal. All M stripes are equally important; the more stripes that are received, the higher is the quality (*i.e.,* lower distortion). It is clear that MDC provides robustness since the probability for all stripes to concurrently fail to arrive is very low when M is sufficiently large. For instance, assuming the probability p for each stripe to fail is totally independent, let $p = 0.5$, when $M = 2$, the expected probability for at least one stripe to arrive is 0.75; it increases to 0.996 when $M = 8$. Furthermore, previous work[7, 8] has indicated that increasing M significantly decreases the distortion, and therefore results in much better reconstructed media quality at the receivers.

In CodedStream, MDC is seamlessly integrated with other components such as network coding. The live media stream α is encoded using MDC at the source node into k stripes and decoded at receiver nodes.

3.3. The Advantages of Live Multimedia Streaming

With respect to minimizing link stress, media streaming is a particularly attractive application of multicast in k-redundant graphs for $k \geq 2$. Unlike traditional data traffic that is usually elastic and greedy for bandwidth, media streaming requires a predictable range of bandwidth due to its constant frame rate (or, even better, constant bit rate for audio streams). Such characteristics of media streaming applications guarantee that for a physical link, the bandwidth stress it experiences is the same for any $k \geq 2$. Consider a typical CBR 128Kb/s audio stream and assume equal-sized stripes (descriptions), for $k = 2$, each of the 2-redundant paths (to a receiver) takes 64Kb/s, while for $k = 8$, each of the 8-redundant paths takes 16Kb/s. From the perspective of a physical link shown in Fig. 2, assume that it has a link stress of 2 for $k = 2$ and the total bandwidth it carries is 128Kb/s; however, with $k = 8$, even if the link has a stress of 8, the total bandwidth it carries remains the same (128Kb/s), since each of the 8 virtual paths mapped to it would carry 1/8 — instead of 1/2 — of the streaming bandwidth.

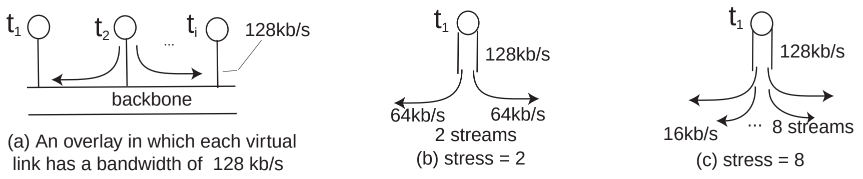


Figure 2. The relationship among bandwidth, stress and transmission rate of each flow

When k increases, the transmission rate of each stripe decreases (assuming that the media to be streamed is CBR at the source node), *i.e.*, each flow carries lower transmission rate. Therefore, each physical link can sustain additional concurrent flows, and each node in the multicast graph can have higher node degrees than in a graph with lower k. In other words, it is feasible to increase k in the context of live media streaming. As will be confirmed by our simulation results, increasing k means that more underlying physical links are exploited, and hence the received media quality improves. This, however, is at the cost of increasing system complexity in recruiting more core nodes and in calculating and assigning linear codes.

4. CODEDSTREAM: ALGORITHMS AND ANALYSIS

The primary objective of CodedStream is to build and maintain a k-redundant multicast graph at the application layer. Towards such an objective, there are several non-trivial challenges. Our algorithm addresses each of these challenges, and much of the complexity lies in tackling all of them in conjunction. (1) In order to subsequently apply network coding, we need to correctly construct a k-redundant acyclic multicast graph from the source to all members of the multicast group[*]. During the construction process, data delivery paths should be optimized in the multicast graph to the receivers. Each receiver essentially has k *disjoint* paths from the source; all paths should be carefully chosen to maximize the aggregated throughput to the receiver. (2) We need to minimize the number of core nodes with a given number of receivers while preserving good performance. (3) Minimizing stress is paramount since it directly determines how much actual bandwidth a virtual link has and high stress can severely diminish end-to-end throughput, *i.e.,* the number of descriptions received at each receiver.

The skeletons of components of the CodedStream algorithms are shown in Table 2. Details of these algorithms are further discussed and analyzed in subsequent sections.

Table 2. CodedStream Algorithms

```
On source node s
Consider a live media stream $\alpha$ on $s$:
    $ifStreamingEnd \leftarrow false$;
    Recruit a set of core nodes;
    Construct a $k$-redundant multicast graph;
    Assign a vector of linear codes $v_i$ to each node $i$;
    do
        Apply MDC encoding on $\alpha$ to generate $k$ stripes;
        Forward each stripe to downstream nodes;
    until ($ifStreamingEnd = true$);

On core nodes
Consider a core node $u$:
Upon receiving a vector of coded streams $V_{in}$:
    Apply network coding on $V_{in}$ using $v_u$ to encode;
    Forward generated vector $V_{out}$ to downstream nodes;

On receiver nodes
Consider a receiver node $t$:
Upon receiving a vector of coded streams $V_{in}$:
    Apply network coding on $V_{in}$ using $v_t$ to decode;
    Apply MDC decoding on generated vector $V_{out}$;
    Return decoded media stream $\alpha'$ to media player;
```

[*]Henceforth, the terms multicast group members and *receivers* will be used interchangeably.

4.1. Constructing the k-redundant Multicast Graph

The algorithms in CodedStream to construct the k-redundant multicast graph are fully distributed and consist of three steps. The objective of the first step is to build a connected graph of the set of *all* nodes in the group, referred to as the *rudimentary graph.* The second and third steps are carried out for data delivery. In the second step, an enhanced *core tree* (tree enhanced by adding some non-tree edges) is constructed from *only* the core nodes with source s as the root. Using the rudimentary graph and the enhanced core tree, the third step constructs the k-redundant multicast graph by carefully selecting k paths from the source to each receiver.

Step 1: Building the rudimentary graph

When a node joins the multicast group as a receiver or declares itself as a core node in the corresponding multicast group, it is given a set of nodes already in the group[†], these are its initial neighbors in the rudimentary graph. The new node contacts its neighbors so they are made aware of it. Every node maintains a set of neighbors with which it periodically exchanges group information in the following fashion: (1) Each node stores a list of addresses of all the nodes it knows about in the group; and (2) neighboring nodes periodically exchange and update local neighbor lists with each other. After a node joins, the information about the new node will eventually be propagated through the rudimentary graph. The information of whether a node is core or receiver is kept alongside its address in the list each node keeps (of all the nodes in the group to the best of its knowledge).

The process of building the rudimentary graph resembles that of constructing the Narada mesh.[2] The differences lie in the extra objectives of this process in addition to the high quality of the mesh that Narada seeks. The objective is to construct a rudimentary graph such that the following properties are satisfied. First, the core nodes (assisting the corresponding multicast group) are required to be connected, and form a subgraph referred to as *rudimentary core graph.* Second, each of the receivers maintains at least k neighbors. Third, several receivers have at least k neighbors that are core nodes.

Periodically, each node u randomly chooses another node v in the group that is not a neighbor and by sending a probing packet, estimates the bandwidth and latency of the direct overlay link, (u, v). If the direct link is better than most of its (direct) links to its current neighbors, then v is added as a neighbor of u and the edge (u, v) is added to the rudimentary graph. The goal is to have overlay links (edges) that have good performance in the rudimentary graph.

Let x be a current neighbor of u. If the number of neighbors of u is greater than k and (u, x) is much worse than the links u has to its other neighbors, and both u and x use this link rarely (*i.e.*, use it to reach very few nodes), then u drops x as its neighbor and (u, x) is removed from the rudimentary graph.

The dynamics of adding good edges and dropping poor edges is vital to the performance of the entire multicast scheme. This is because that, ultimately, the edges in the rudimentary graph are used to construct the data delivery paths, whose performance depends directly on the quality of these edges.

In order that some of the receivers will each have at least k core nodes as neighbors, we use a simple procedure: each core node contacts its receiver neighbor with which it has the best link, and if this receiver does not have k neighbors that are core nodes, then it probes all the other core nodes it knows about and selects the ones with the best links.

Step 2: Building the enhanced core tree

We first construct the *core tree* from the subgraph of the rudimentary graph consisting of s, as the root, and the core nodes only. We adopt the distributed algorithm proposed by Wang and Crowcroft[11] based on distance vectors that finds the *shortest widest* paths from s to each core node. With this algorithm, the widest path, or the path with the highest end-to-end bandwidth, is selected; and if there is more than one widest path, the shortest, one with the lowest end-to-end latency, is then selected.

The core tree is then *enhanced* by adding edges to the tree, in order to greatly facilitate the third step of building the k-redundant multicast graph. Each node chooses $k - 1$ non-tree edges to be part of the enhanced core tree so that the best possible k paths will be constructed in the third step. We use a labeling technique to efficiently find these non-tree edges with the result that each core node has k disjoint paths from s. The detailed algorithm is shown in Table 3.

[†]Any of the existing solutions in the literature to address this *bootstrapping* or *rendezvous* problem may be used.

Table 3. Building the enhanced core tree

```
*On core nodes*
Construct a spanning tree rooted at $s$ on core nodes based on the shortest-widest path algorithm;
Label the first path to $s$ on the spanning tree;
numOfPaths $\leftarrow$ 1;
**do**
  Find a disjoint path in the subgraph of $s$ and $A_c$;
  Label this path;
  Add edges in the path to the enhanced core tree;
  numOfPaths $\leftarrow$ numOfPaths + 1;
**until** numOfPaths = $k$;
```

We use a running example to illustrate the steps of our algorithm. In our example, $k = 2$, *i.e.,* we are constructing a 2-redundant multicast graph. The enhanced core tree is shown in Fig. 3(a). Nodes u_1, u_2, u_3, u_4 are all core nodes, the solid edges form the core tree and the dashed edges are the non-tree edges added for the enhanced core tree.

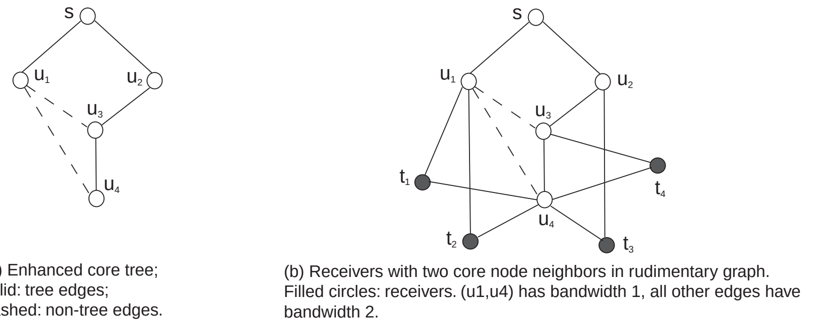

s
u1
u2
u3
u4
(a) Enhanced core tree;
solid: tree edges;
dashed: non-tree edges.
s
u1
u2
u3
u4
t1
t2
t3
t4
(b) Receivers with two core node neighbors in rudimentary graph.
Filled circles: receivers. (u1,u4) has bandwidth 1, all other edges have bandwidth 2.

Figure 3. Example enhanced core tree and receivers in rudimentary graph that have two core neighbors.

Step 3: Building the multicast graph

Based on the rudimentary graph and the enhanced core tree, a k-redundant multicast graph is constructed, by first finding data paths for receivers that have k or more core neighbors. Afterwards, the distributed shortest-widest-paths algorithm[11] is used to build an optimal spanning tree of all the receivers, called *receiver tree* starting from the receivers that already have data path from the first phase. Finally, the set of receivers with k incoming data paths is expanded to the set of all receivers by adding the best possible links to the receiver tree. All throughout, distributed labeling and relabeling procedures are run by the nodes to avoid creating directed cycles in the multicast graph.

Returning to our running example, there are four receivers in the rudimentary graph that have two core nodes as neighbours, they are shown in Fig. 3(b). All edges in our example rudimentary graph have bandwidth of 2, except the edge $u_1 - u_4$ which has bandwidth 1. For these receivers, they each find two disjoint paths from s using their two core neighbours. These are shown in Fig. 4(a),(b). For receivers t_1 and t_2, the best two paths for each of them are found in the core tree; while for t_3 and t_4, the paths found include non-tree edges in the enhanced core tree.

To complete the example, we also include some receivers, t_5 and t_6, that do not have two core nodes as neighbours, as shown in Fig. 4(c). The set of receivers with two incoming data paths is initially just $\{t_1, t_2, t_3, t_4\}$ before data paths are found for t_5 and t_6. They can add edges from any two nodes in $T = \{t_1, t_2, t_3, t_4\} \cup \{u_1, u_2, u_3, u_4\}$. Node t_5 chooses t_4 and u_2 as its parents, while t_6 chooses t_2 and t_3. Now, once they have found two optimal data paths, the set T is expanded to include them, $T = T \cup \{t_5, t_6\}$. This process proceeds until all receivers become members of the set T.

4.2. Limiting Node Degrees

Our main emphasis is on minimizing stress because it directly affects the actual end-to-end throughput. Placing a constraint on node degree, *i.e.*, limiting the number of neighbors that a node has, is an effective technique to

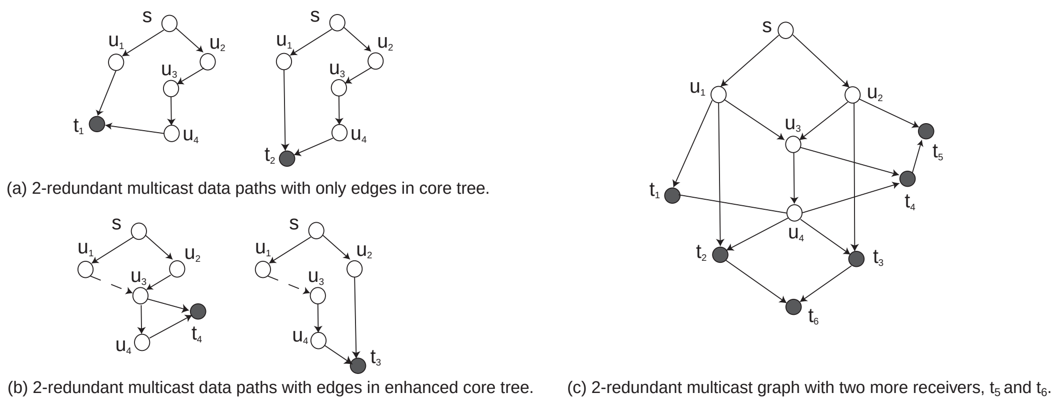

(a) 2-redundant multicast data paths with only edges in core tree.
(b) 2-redundant multicast data paths with edges in enhanced core tree.
(c) 2-redundant multicast graph with two more receivers, t_5 and t_6.

Figure 4. (a),(b) Data paths for the four receivers from our example. (c) The case of six receivers

minimize stress.

We need to impose the constraint of maximum node degree, Δ, during the construction of the rudimentary graph. However, the rigidity of a universal Δ imposed uniformly on all nodes is not suitable in overlay networks, where end hosts have vastly different available aggregate bandwidth (*i.e.*, the "last-mile" bandwidth). For example, end hosts that enjoy a high-bandwidth physical link should be able to afford carrying more virtual links in the overlay. We therefore define a new metric *node stress* to capture the distinction. Given the aggregate bandwidth B_u and the current number of neighbors δ of an overlay node u, the node stress of u is δ/B_u, which should be maintained at a stable level. When choosing neighbors to form the rudimentary graph, node stress of a candidate neighbor is considered along with link bandwidth and delay. As a result, nodes with higher aggregate bandwidth are chosen by more nodes to be their neighbors, and hence have higher degrees than nodes with lower bandwidth.

In order to incorporate node stress, a value of node stress is also associated with each node during the construction of the rudimentary graph. It is given in response to all probes searching for potential neighbors. When choosing neighbors, in addition to consideration of link bandwidth and delay, low node stress is explicitly favored.

4.3. The Dynamics of Node Joins and Departures

When a set of nodes join the multicast group, they are first given a list of group members, at least some of which are still members. The new nodes find neighbors with good links and low node stress, creating new edges in the rudimentary graph. If the best links that a newly joined member has are adjacent with core nodes, then it can run the procedure for finding data paths through core neighbors (the first phase of the construction of multicast graph). Otherwise, if this fails, the new receiver finds paths from choosing the best among all its neighbors which are either a core node or a receiver that is already k-redundant.

A node sends notification to its neighbors when it leaves the group. The periodic exchanges of information between neighbors ensure that eventually everyone will be informed. If a node fails or leaves without being able to notify others, then the neighbors will stop receiving the periodic information exchanges from this node. After a timeout period, they will assume the node is no longer in the group and send out notifications. Node failures are handled gracefully and efficiently. When a node v fails, only the children of v in the k-redundant multicast graph need to execute procedures to find another parent. This procedure is exactly the same one as if these children nodes are joining the group, except that they still keep their own descendants. Since node degree is constrained in the multicast graph, only a small number of nodes need to re-discover parents when a node fails.

4.4. Further Discussions

CodedStream has to recruit other nodes as core nodes to become functional. However, we argue that this is not a major problem. Although dedicated relaying nodes are rare and expensive, we can recruit other overlay nodes to assume such roles. For instance, CodedStream nodes might not be interested in receiving *all* the live media content. That means that not all CodedStream nodes join the same streaming session. Some of them might be idle and available to be core nodes and assist ongoing sessions, provided that specific incentives are given.

Furthermore, we might consider CodedStream as a self-contained application-layer protocol, which deals with group management, membership maintenance, etc. We expect that CodedStream can also run on top of a current existing peer-to-peer protocol as an add-on protocol and be able to take advantage of the underlying protocol to deal with such issues.

5. PERFORMANCE EVALUATION

We have conducted simulation-based experiments using a packet-level, event-based simulator to evaluate the performance and reveal the strengths of CodedStream, in the context of three benchmark protocols: (1) DVMRP, an implementation of IP multicast; (2) Narada, a tree-based application-layer multicast; and (3) multicast forest, a forest of k distinct multicast trees between source and all receivers.

We have chosen the INET topology generator from the University of Michigan,[12] which is fully capable of generating large-scale topologies that conform to the power-law characteristics. The underlying IP-layer topology consists of 6000 routers with end systems to each of which a router is randomly attached. IP-layer physical links conform to a power-law distribution pattern, with a range of $128 - 4433$ Kb/s. The average link bandwidth is about 410 Kb/s.

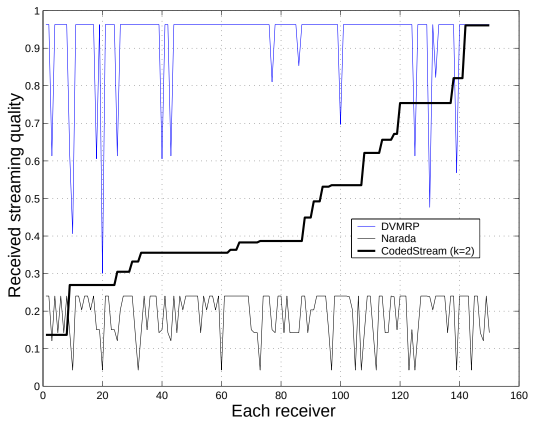


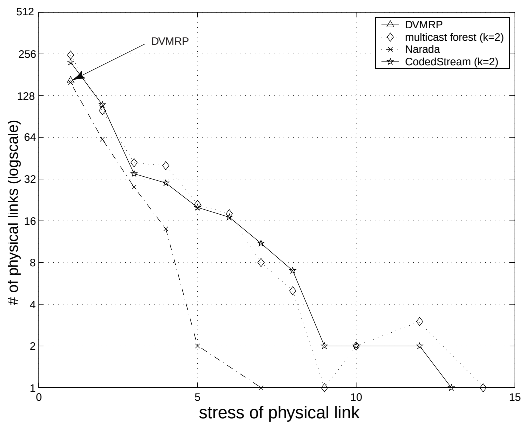
Figure 5. The distribution of per-receiver received media quality in (1) DVMRP; (2) Narada; (3) CodedStream ($k = 2$) (sorted by CodedStream).

A recent measurement study by Saroiu *et al.*[13] shows that in peer-to-peer file sharing systems the distribution of "last mile" bandwidth follows a power-law distribution spanning from 14.4 Kb/s to 100 Mb/s. In our simulation setup, end systems have higher last mile bandwidth. We argue that in the context of live media streaming, end systems generally have higher bandwidth. In fact, such higher last-mile bandwidth motivates us to introduce redundant paths to improve the performance of live media streaming.

As per our goal, we focus on the simulation results with respect to streaming quality as perceived by receivers, which is a performance measure that embodies both throughput performance and resilience to loss.

For DVMRP and Narada, MDC is not applied. We simply assume that a certain specific hop-by-hop congestion control protocol enables the graceful degradation of media streams in both systems. Therefore, for each receiver in DVMRP and Narada, the received streaming quality is the ratio of received throughput over the transmission rate at the source.

MDC is applied in multicast forest and CodedStream as follows. In multicast forest, each tree carries a description. In CodedStream, each disjoint path carries a description. Received streaming quality of each receiver in multicast forest and CodedStream is the number of received descriptions over k; however, due to the dynamics in the system, the received quality at each receiver is not discrete, rather, it is distributed over the range of $[0, 1]$. Due to the limitation of our simulator, we define the received media quality as the expected value of the ratio of the number of received stripes over k, which does not exactly match the term *media quality* with respect to distortion in the context of signal processing. However, we believe that such an estimate captures the system dynamics well.



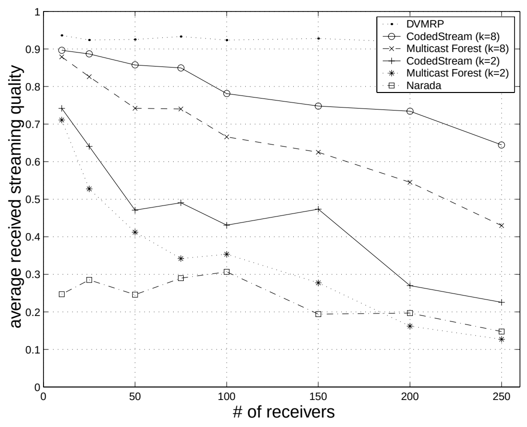


Figure 6. Left figure: Link stress distribution. Right figure: The quality of received streaming data averaged over all the receivers.

In our experimental setup, we randomly choose n overlay nodes as the core nodes. The simulation results would likely have been better had we selected overlay nodes with higher aggregate bandwidth during the process of recruiting core nodes. To provide fair comparison with multicast forest, we use the same scenario in simulating multicast forests.

Received media quality distribution over all receivers in a group size of 150 is plotted in Fig. 5. Quality for all but a few of the nodes in CodedStream ($k = 2$) are lower than DVMRP and higher than Narada. It can be seen that for CodedStream ($k = 2$), some nodes achieve quality as high as DVMRP. The worst quality of a node in CodedStream ($k = 2$) is higher than the worst in Narada.

It appears that Fig. 5 confirms the intuition that extra redundant paths do bring better streaming performance at the receiver end. However, in the application layer overlay, this might not be the case: introducing extra redundant paths may not necessarily improve the end-to-end performance. In the right graph in Fig. 6, in the case of $k = 2$, as n, the number of receivers, increases, the average received media quality of CodedStream and multicast forest both decreases. Moreover, multicast forest is outperformed by Narada when n increases beyond 200. The reason is not difficult to find: redundant paths may result in higher stress on physical IP links, and in some cases, the last mile bandwidth becomes the bottleneck, which might recoil the benefits brought by extra paths.

This observation is confirmed by the left graph in Fig. 6. In the figure, the horizontal axis is the link stress and the vertical axis is the number of physical links for a given stress. The number of virtual overlay links for CodedStream and multicast forest is higher, so stress on physical links in proximity of the end systems (overlay nodes) is bound to be higher. This explains the phenomenon that there are more physical links with higher stress for CodedStream than for Narada. Furthermore, there are more physical links with higher stress for multicast forest than for CodedStream. This is reasonable since CodedStream exploits network coding, which helps alleviate stress.

In the right graph in Fig. 6, it is clear that increasing k results in better receiver media quality for both CodedStream and multicast forest. However, increasing k from 2 to 8 does not yield an improvement of four times. The reason is the same as above: the improvement in received media quality is the result of introducing extra redundant paths; on the other hand, the extra paths put higher stress on some physical IP links to counteract such benefits.

On the bright side, CodedStream significantly outperforms multicast forest for both values of k; therefore, the simulation results confirm our observation. In the simulation, streaming has a constant transmission rate of 512Kb/s. When $k = 8$, it means that every complementary description has a rate of only 64Kb/s. This is quite small compared to most of the link bandwidths. An important advantage of CodedStream over multicast forest is the ability to find higher-bandwidth links for the multicast graph, because it is not constrained by the strict requirement of finding distinct trees, therefore it has more links to choose from during construction.

It might be of interest to the reader what the optimal value is for k. Unfortunately, the optimal k is variable in different topology setups and for different bandwidth limits of core nodes and receivers. The reason is two-fold: (1) As k increases, both the sustainable physical link stress leading to overlay nodes and the limited number of cores and receivers significantly decrease the probability of finding multiple good paths from the source to the receiver. (2) As k increases, the code assignment algorithm becomes more complex and adverse to the dynamics of node joins and departures.

6. RELATED WORK

This work was mainly inspired by previous work on the information theoretic aspects of network coding, which demonstrates the potential of achieving per-receiver maximum-flow throughput in a multicast graph by applying linear codes.[6] Although exciting insights have been provided, the existing studies in network coding have remained largely theoretical, and we are not aware of any published work that studies the feasibility of applying the theoretical insights in network coding to increase throughput in actual multicast streaming sessions over wide-area networks.

Many application-level multicast systems have been proposed recently, *e.g.*,.[1, 2, 14] All are based on a single multicast tree. Several systems exist that use end-system multicast for media distribution, notably Overcast[14] and Narada[2] for Internet video conferencing.

Inspired by previous two-step algorithms such as Narada,[2] our algorithm begins with the construction of the *rudimentary graph* that is similar to the Narada mesh. However, our proposal distinguishes from Narada in the following fundamental aspect: *the goal of our algorithm is to construct a* multicast graph*, rather than a tree*, in order to increase streaming quality at the receivers and to take advantage of the tremendous power of network coding.

Among all previous work, perhaps the work on CoopNet[15] and SplitStream[16] are most similar to CodedStream. Both papers have proposed to utilize multiple multicast trees to deliver striped data, using either multiple description coding or source erasure codes to split content to be multicasted. CoopNet proposes a centralized algorithm to facilitate using multiple multicast trees from different sources, and does not feature built-in support of optimizing link stress and stretch. SplitStream proposed a decentralized algorithm to construct a *forest* of multicast trees from a single source, with a focus on per-node load balancing. In both work, the inherent concerns of throughput limitations caused by conflicting paths have not been addressed. In comparison, CodedStream constructs an *acyclic multicast graph* from one multicast source, which, combined with coding, introduces a smaller degree of stress on overlay nodes compared with a forest. Further, CodedStream seeks a well-balanced trade-off between the constraints on link stress and the selection of good paths to achieve high throughput. Network coding, while essential to the performance of CodedStream, has not been incorporated in either CoopNet or SplitStream due to its infeasibility in a forest of multicast trees. In Sec. 5, we have demonstrated the best-case results using such multicast forest strategies, which are inferior to CodedStream.

Finally, Kostic *et al.*[17] and Byers *et al.*[18] have both proposed to construct an overlay mesh of concurrent data dissemination connections, each sending a (hopefully) disjoint set of data. As a node receives data from these connections and merges incoming data, throughput may be significantly improved due to the larger number of concurrent connections. Byers *et al.* have discussed the algorithmic details of merging differences from different downloading sources, while Kostic *et al.* have proposed an elaborate algorithm that allows nodes to send data to different points in the overlay, as well as to locate and recover missing data items. While both papers need to assume large or unlimited buffers at each overlay node in order to store elements of data to potentially serve others, they might not be appropriate in the context of live video streaming which is delay-sensitive.

7. CONCLUDING REMARKS

In this paper, we have proposed CodedStream to significantly improve the performance of end-to-end streaming of live multimedia. Such surprising results are achieved by the application of network coding when exploiting path diversity with k redundant paths to each multicast group receiver. We depart from the conventional wisdom that the multicast topology between the source and the receivers should be a tree, and that data may only be replicated and forwarded by intermediate overlay nodes. To the best of our knowledge, there do not exist similar proposals in previous literature. With respect to the effectiveness and performance of CodedStream, we have undertaken both analytical and simulation-based studies, which agree with our original claims. We are currently in the process of implementing CodedStream as an application-layer streaming protocol on wide-area overlay network testbeds, including the wide-area network within the infrastructure of *PlanetLab*.[19]

REFERENCES

1. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, August 2002.
2. Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications* , pp. 1456–1471, October 2002.
3. J. Liebeherr, M. Nahas, and W. Si, "Application-Layer Multicasting With Delaunay Triangulation Overlays," *IEEE Journal on Selected Areas in Communications* , pp. 1472–1488, October 2002.
4. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory* **IT-46**, pp. 1204–1216, 2000.
5. R. Koetter and M. Medard, "Beyond Routing: An Algebraic Approach to Network Coding," in *Proc. of IEEE INFOCOM*, 2002.
6. S.-Y. R. Li and R. W. Yeung, "Linear Network Coding," *IEEE Trans. on Information Theory* , 2002.
7. J. Apostolopoulos, T. Wong, W. Tan, and S. Wee, "On Multiple Description Streaming with Content Delivery Networks," in *Proc. of IEEE INFOCOM*, 2002.
8. V. K. Goyal, "Multiple Description Coding: Compression Meets the Network," *IEEE Signal Processing Magazine* , pp. 74–93, 2001.
9. V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao, "Cooperation in wireless ad hoc networks," in *Proceedings of IEEE INFOCOM*, 2003.
10. J. Guo, Y. Zhu, and B. Li, "CodedStream: Live Media Streaming with Overlay Coded Multicast," *Technical Report. Department of Electrical and Computer Engineering, University of Toronto* , June 2003.
11. Z. Wang and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications* **14**, pp. 1228–1234, September 1996.
12. J. Winick, C. Jin, Q. Chen, and S. Jamin, "INET: an Autonomous System (AS) level Internet Topology Generator, version 3.0," *available online at http://topology.eecs.umich.edu/inet/* , June 2002.
13. S. Saroiu, P. Gnummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proc. of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN 2002)*,
14. J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," in *Proc. 4th Symposium Operating System Design and Implementation (OSDI)*, pp. 197–212, October 2000.
15. V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. of NOSSDAV*, 2002.
16. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Content Distribution in a Cooperative Environment," in *SOSP*, 2003.
17. D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. of ACM SOSP*, 2003.
18. J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks," in *Proc. of ACM SIGCOMM*, 2002.
19. L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proc. of the First Workshop on Hot Topics in Networks (HotNets-I)*, (Princeton, New Jersey), October 2002.