A High-Throughput Overlay Multicast Infrastructure with Network Coding

Mea Wang, Zongpeng Li, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*mea, arcane, bli*}*@eecg.toronto.edu*

Abstract. Network coding has been recently proposed in information theory as a new dimension of the information multicast problem that helps achieve optimal transmission rate or cost. End hosts in overlay networks are natural candidates to perform network coding, due to its available computational capabilities. In this paper, we seek to bring theoretical advances in network coding to the practice of high-throughput multicast in overlay networks. We have completed the first real implementation of network coding in end hosts, as well as decentralized algorithms to construct the routing strategies and to perform random code assignment. Our experiences suggest that approaching maximum throughput with network coding is not only theoretically sound, but also practically promising. We also present a number of unique challenges in designing and realizing coded data dissemination, and corresponding solution techniques to address them.

1 Introduction

In recent years, application-layer overlay networks have emerged as important directions to evolve future network architectures, due to the *flexibility* of programming overlay nodes to execute any application-layer algorithm one has designed. This is in sharp contrast with the lack of flexibility at the IP layer. Regardless of the approach taken, most of the previous work in overlay or peer-to-peer networks focuses on accessing or disseminating information more efficiently over the current-generation Internet. We may attempt to find unicast paths with higher throughput or lower latency by passing through other overlay nodes, or to construct a high-quality overlay multicast tree from one source to multiple receivers [1, 2]. Depending on the applications, we may be disseminating bulk data, or streaming multimedia with stricter throughput and timing constraints [3].

Despite the contributions of existing work, we have still not answered one fundamental question: what is the maximum throughput one can achieve using overlay networks, given a single source with information to be disseminated, and a set of interested receivers? With the intuition of constructing an overlay multicast tree, it is easy to show that we still have residual idle network capacities after the tree is formed. In this paper, we consider the problem of distributing large volumes of data across overlay networks, and seeks to design and implement the best strategy to disseminate data from the source to the receivers with maximized throughput, even with the presence of the dynamic nature of overlays.

One naturally starts with constructing multiple multicast trees from the source to the destinations with the best possible performance [4]. The fundamental advantage of multicast over unicast is that multicast employs intermediate nodes to *replicate* data packets to achieve higher transmission performance. It's a unique property of information flows to be replicable. In fact, we can do better than transmitting along multiple

trees, by also taking advantage of another fundamental property of information flows: we can *code* multiple streams of information into one stream. In contrast, none of the normal commodity flows may be coded. *Network coding* extends the capabilities of network nodes in a communication session: from basic data forwarding (as in all unicast) and data replication (as in IP or overlay multicast), to *coding in finite fields*. It has been shown that, with linear codes, we may be able to achieve surprising results with respect to optimizing throughput for both delay-sensitive and delay-insensitive applications [5–7]. As overlay nodes can afford to code data flows computationally, they are ideal candidates to execute network coding based algorithms and protocols.

In this paper, we bring theoretical advances in network coding to realistic implementations, and present a complete set of network infrastructure and distributed protocols for coded data flow dissemination, which is ready to serve overlay applications that may benefit from high end-to-end transmission rate without high cost. Towards a realistic implementation of coded overlay flows, we present our algorithms for the construction of a transmission topology for network coding, and for randomized code matrix generation. Based on observations and experiences from our prototype system, we argue that network coding enables a more efficient way to compute the best transmission topologies to maximize session throughput and to utilize residual network capacities. Overlay multicast systems implemented in previous work generally employ a multicast tree or a multicast mesh (multi-tree) as the transmission topology, with encoding at source node only or no coding at all. To the best of our knowledge, our real-world implementation of multicast flows with network coding is the first in the research community. It is also the first real multicast system that targets mathematically provable near-optimal throughput, as contrasted to heuristically high throughput. We believe it is instrumental to develop additional insights of the practical implications of network coding and high-throughput data networking.

The remainder of this paper is organized as follows. In Sec. 2, we review related past research. In Sec. 3, we propose our decentralized algorithm to compute optimal routing strategies with network coding. We present our real-world implementation of both our algorithms and network coding itself in Sec. 4, followed by observations and experiences with such an implementation (Sec. 5). We conclude the paper in Sec. 6.

2 Related Work

Recent work on high-bandwidth data dissemination in overlay networks has focused on constructing multiple multicast trees or an overlay mesh, as exemplified by *SplitStream* [4], *Bullet* [8], as well as Digital Fountain [9]. In SplitStream, the original data is split into multiple stripes and is sent among interior-node-disjoint multicast trees to improve the throughput, such that all nodes share the burden of duplicating and forwarding data. Digital Fountain and Bullet uses source erasure codes and reconciles missing data among peers. This effectively leads to a topological overlay *mesh*. The designs depend on strong buffering capabilities to tradeoff end-to-end latency for achievable throughput. They work well on bulk data downloading or delay-insensitive streaming of media.

Another category of proposals have each overlay node establish k links to other overlay peers. Links established may be shortest (smallest latency), widest (highest bandwidth), randomly chosen, or a combination of the above. Previous experiences

show that, always selecting the k best links may result in poor connectivity and a large diameter in the resulting mesh. This problem may be resolved by selecting some best links mixed with a small number of random links [10]. Young *et al.* [11] proposed a distributed algorithm to compute k Minimum Spanning Trees (k-MST), where edge weights correspond to the latency or loss rate. The k-MST mesh ensures the existence of k edge disjoint overlay paths between any pair of nodes.

If we assume an overlay node may encode and decode data using linear codes in Galois fields, we may then take advantage of the recent theoretical advances in *network coding* [12, 13]. As opposed to source coding, where data is encoded and decoded only at the source and destinations, respectively, network coding allows *every* node in the network to encode and decode data streams as necessary. The coding process uses *linear codes* in the Galois field, and includes two basic operations: the $+$ and $\cdot$ operations in the Galois field GF(2^k). Since elements in a Galois field have a fixed-length representation, bytes in flows do not increase in length after being encoded.

While information flows differ from normal commodity flows in that they may be replicated and encoded, the transmission of information flows still exhibits an underlying network flow structure. Ahlswede *et al.* [12] and Koetter *et al.* [13] prove that, a multicast rate χ can be achieved for the entire multicast session if and only if it can be achieved from the sender to each of the multicast receivers independently. With this theorem, computing the routing strategy to maximize session throughput can be transformed into a sequence of maximum flow computations, which is not only polynomial-time solvable, but also allows fully distributed solutions. If flows to different receivers share some links in the network, the conflict may be resolved through network coding.

Recently, a number of multicast algorithms [6, 7, 14, 15] have been proposed to utilize the underlying network flow structure of coded multicast to efficiently achieve high transmission rate or low cost. Gkantsidis *et al.* [16] also propose to employ network coding in large-scale content distribution in peer-to-peer networks, to eliminate the need of strategic peer reconciliation. Our work in this paper focuses instead on high-throughput with controlled delay.

3 Computing the Optimal Routing Strategy

In this work, we achieve the objective of maximizing end-to-end session throughput in two phases: constructing the transmission topology, and designing the suitable coding strategy for data dissemination using a randomized code assignment algorithm. The network coding theorem reviewed in Sec. 2 establishes the underlying connection between multicast flow routing and network flows. Consequently, the computation of the multicast rate and the optimal multicast transmission topology is separable into a number of maximum flow computations. The maximum achievable throughput of a multicast session is the smallest throughput among all source-destination pairs. Given such a transmission topology, the final question is how data should be disseminated and coded. In this section, we present algorithms for all phases that may be realistically applied to compute the optimal transmission topology for coded overlay flows.

3.1 The Maximum Flow Problem

Maximum flow is a well studied problem in the theory of network flows. Given a directed network $G = (V, A)$ and nodes $u, v \in V$, the maximum flow from u to v is the maximum rate at which flows can be shipped from u to v along capacitied arcs in G. In

the *min-cost flow* problem, which is a more general version of the max-flow problem, a cost is associated with every unit flow shipped through an arc, and a given flow rate is to be achieved while introducing minimum link costs. A min-cost flow algorithm may be used to compute the maximum flow, by inserting a virtual arc (*a feedback link*) from receiver v to sender u with cost -1, while setting other arc costs to zero. We employ an ϵ-relaxation based algorithm [17] to compute the max-rate multicast topology with minimum bandwidth consumption. Our algorithm is amenable to fully distributed and fully asynchronous implementations.

In our notation, each link $(i, j) \in A$ is associated with bandwidth capacity b_{ij}. f_{ij} is the flow rate from node i to node j, c_{ij} is the cost of transmitting a unit flow via link (i, j), g_i is the flow excess on node i, and p_i is the dual variable acting as unit price charged for flow excess at node i.

3.2 Computing the Transmission topology

The first step towards maximum-rate multicast transmission is to compute a routing topology indicating the amount of bandwidth required on each link in the network. Given this topology, we assign flows on each link according to the allocated bandwidth and eventually transmit the data. In this section, we focus on computing the transmission topology and bandwidth allocation.

Unicast sessions In the case of unicast sessions, each link $(i, j) \in A$ is initialized with flow $f_{ij} = 0$ and cost $c_{ij} = 0$. We then add a feedback link with $b_{ds} = f_{ds} = \alpha$, and $c_{ds} = -|D|$, where α is a constant with any value known to be larger than the achievable maximum flow rate and $|D|$ is the maximum diameter of the network (in number of hops). For each node i, the flow excess g_i is calculated as $\sum_{(j,i)\in A} f_{ji} - \sum_{(i,j)\in A} f_{ij}$, and the price p_i is initialized to 0. After initialization, each node with a positive flow excess ($g_i > 0$) executes the algorithm in the Table 1. The algorithm terminates when every node has zero flow excess [17].

Multicast sessions We are now ready to generalize the algorithm to compute a transmission topology that seeks to achieve maximized throughput for any communication session. In a multicast session, data are sent from the source to a group of interested receivers at the same rate in the overlay. To achieve maximized multicast throughput, we first need to identify the maximum achievable throughput between each source-destination pair, using the previously described algorithm for unicast sessions. Given the maximum achievable throughput for each destination, the throughput of a multicast session corresponds to the smallest throughput achievable to all destinations [12, 13]. Since the maximum throughput for each destination may be different from each other, they need to be reduced to match the prevailing multicast flow rate.

We introduce a set of variables to maintain the status of each link with respect to each destination. The cost and bandwidth capacity are still denoted by c_{ij} and b_{ij} respectively on each directed link (i, j). We let f_{ij}^k be the flow rate on arc (i, j) serving destination k, g_i^k be the flow excess on node i in serving destination k, and p_i^k be the price on node i in serving destination k. The min-cost flow algorithm remains unchanged, except that we apply the algorithm independently for each source-destination pair.

When the min-cost flow algorithm terminates for all destinations, the maximum achievable throughput f_k of a source-destination pair is the flow rate on the feedback

Each node i maintains a price vector of its direct upstream and downstream nodes and a capacity vector of incident links, and execute:

1 **while** ($g_i > 0$)
2 Scan all links for an outgoing links (i, j) such that $p_i = p_j + c_{ij} + \epsilon$ and $f_{ij} < c_{ij}$, or an incoming link (j, i) such that $p_i = p_j - c_{ji} + \epsilon$ and $f_{ij} > 0$.
3 **if** (such outgoing link (i, j) is found)
Decrease excess by increasing f_{ij}
4 $\delta = \min(g_i, c_{ij} - f_{ij})$;
5 $f_{ij} = f_{ij} + \delta$;
6 $g_i = g_i - \delta$;
7 $g_j = g_j + \delta$;
8 **else if** (such incoming link (j, i) is found)
Decrease excess by reducing f_{ji}
9 $\delta = \min(g_i, f_{ji})$;
10 $f_{ji} = f_{ji} - \delta$;
11 $g_i = g_i - \delta$;
12 $g_j = g_j + \delta$;
13 **else**
Increase price of node i
14 $p_i = \min_{\xi \in R^+ \cup R^-} \xi$,
where,
$R^+ = \{p_j + c_{ij} + \epsilon\} | (i, j) \in A$ and $f_{ij} < b_{ij}\}$,
$R^- = \{p_j - c_{ji} + \epsilon\} | (j, i) \in A$ and $f_{ij} > 0\}$,

Table 1. The ϵ-relaxation based min-cost max-flow algorithm

link. The maximum achievable throughput is $f_{\max} = \min\{f_k\}$. To tune the transmission topology to conserve unnecessary bandwidth, we reduce the flow from the source to destination k by $\delta = f_k - f_{\max}$. We initiate the flow reduction process by reducing the flow on each feedback link by δ. The reduction algorithm is presented in Table 2.

3.3 Data Dissemination

Each transmission topology computed by the min-cost flow algorithm provides information not only on the maximum achievable throughput of a multicast session, but also on the amount of bandwidth to be reserved on each link. Notice that these topologies are computed independently based on the assumption that each topology has the privilege to utilize up to 100% of the currently available link capacity. Bandwidth contention problem occurs when two or more transmission topologies share links. For instance, we require $b_1 + b_2$ units of bandwidth on the link shared by two destinations, where b_1 and b_2 are the amount of bandwidth required by each destination. We are pushing the limit of the link capacity if $b_1 + b_2 > b_{ij}$. Fortunately, network coding resolves this issue by allowing coexistence of multiple flows. The coded flow requires no more than $\max\{b_1, b_2\}$ units of bandwidth on such link.

There are still a few remaining problems to be addressed. First, the transmission topology specifies only the amount of flows to be assigned on each link, but not the actual coded flows. We thus need an algorithm to perform flow assignment. Second, if each node simply forwards all flows that it has received, with high probability the

for (each destination k)

1 $I_i = \phi$

2 Scan all incoming links (j, i) such that link (j, i) serve flows for destination k

3 $I_i = I_i \cup (j, i)$

4 $I_{total} = \sum_{j \in I_i} f_{ji}$

5 $O_i = \phi$

6 Scan all outgoing links (i, j) such that link (i, j) serve flows for destination k

7 $O_i = O_i \cup (i, j)$

8 $O_{total} = \sum_{j \in O_i} f_{ij}$

9 $D = O_{total} - I_{total}$

10 **while** $(D > 0)$

11 Scan all links (j, i) in O_i such that $f_{ij} > 0$

12 $\delta = min\{f_{ij}, D\}$

13 $f_{ij} = f_{ij} - \delta$

14 $D = D - \delta$

Table 2. The flow reduction algorithm for multicast sessions

destinations are unable to successfully reconstruct the original data. Though there exist centralized polynomial time algorithms that guarantee optimal code assignment [18], it is not desirable in a realistic implementation of network coding due to their high complexity. We propose a randomized and distributed algorithm to assign codes on each overlay node, so that flows may be encoded and decoded appropriately, with significantly less complexity.

Without loss of generality, we explain the algorithm with an example shown in Fig. 1. Each link is labeled with both its capacity and the reserved bandwidth, as computed by algorithms proposed in Sec. 3.2. Before disseminating data, the source node A needs to determine how many *original flows* (the maximum achievable throughput) the transmission topology can handle. In our example, we have two original flows, labeled a and b, since the intended throughput of this transmission topology is 2.

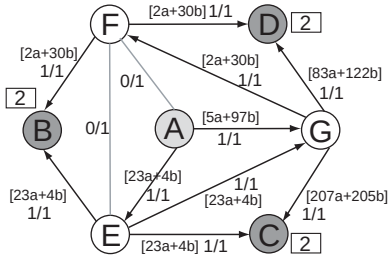

Fig. 1. Illustration of randomized code assignment.

First, each node generates a code matrix in a randomized fashion, which remains static unless the network environment changes. The number of rows and columns cor-

respond to the number of incoming and outgoing flows at this node, respectively. Each entry in the matrix is independently and uniformly taken from $GF(2^8)$. Next, the source node initiates the computation of flow content on each link, which is determined by both incoming flows and the code matrix at its tail node. Therefore at each node, the outgoing flows can be determined by taking a production of the incoming flow coefficient matrix and the code matrix. For example, at node G we have:

$$M_O = C_D \cdot M_I = \begin{pmatrix} 27 & 173 \\ 112 & 85 \\ 98 & 164 \end{pmatrix} \begin{pmatrix} 23 & 4 \\ 5 & 97 \end{pmatrix} = \begin{pmatrix} 207 & 205 \\ 2 & 30 \\ 83 & 122 \end{pmatrix}$$

Matrix operations are all computed over the Galois field GF(2^8). More detailed discussions on performing finite field operations can be found in [19]. Note that such matrix production needs to be performed only once upon session set-up, unless network dynamics occur. After successful execution of the algorithm, each destination should receive exactly n flows if there are n original flows. The coefficient matrix of incoming flows is then inverted at each receiver to serves as its decoding matrix. The product of the decoding matrix with each incoming flow yields the original flow.

3.4 Coding Challenges

So far, we have implicitly assumed links selected by the optimal routing strategy form a directed acyclic graph. However, a cycle in the routing topology may introduce a deadlock for code generation. An simple example is shown in Fig. 2(a), in which nodes B and C each expects a flow description from the other.

To address this problem, we label each link with a list of receivers it is serving, during the max-flow computation phase. Consequently, each node constructs a $0, 1$-matrix representing the input-output dependence relation among its incident flows. Only entries with value 1 will then be replaced by a uniform random symbol taken from $GF(2^8)$. The new solution applied to the previous example is shown in Fig. 2(b) and (c).

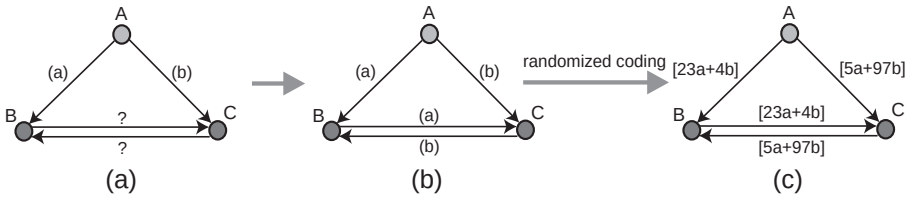

Fig. 2. An example of avoiding coding cycles.

3.5 Adapting to dynamic variations

Our algorithms are based on the knowledge of link capacity. Any realistically measured link capacities, however, may not reflect the actual link capacities between two nodes in the underlying physical network. For example, if two overlay links share the same physical link in the IP network, results from independent available bandwidth probes will not be accurate when both links are utilized by the overlay mesh. Furthermore, even if all overlay link capacities are independent, they may still fluctuate over time, due to cross traffic beyond the control of the overlay.

We highlight the fact that our algorithms naturally adapt to such uncertainty and network dynamics very well. If a certain link capacity turns out to be different than what was expected, the ϵ-relaxation algorithm may resume with cached states, including flow rates and node prices. Since the new optimal state is usually not far from the old one, convergence speed is much higher than re-computing the new multicast topology.

4 Implementation

we have experimentally implemented all the algorithms proposed in Sec. 3. To the best of our knowledge, this work represents the first work on a realistic implementation of network coding. In this section, we discuss our observations, experiences and challenges encountered during this implementation. We have implemented two main components, as illustrated in Fig. 3: (1) a generic application-layer message switch, with the multi-threaded capability of handling and switching multiple incoming and outgoing flows; and (2) an implementation of the architectural design supporting coded overlay flows.

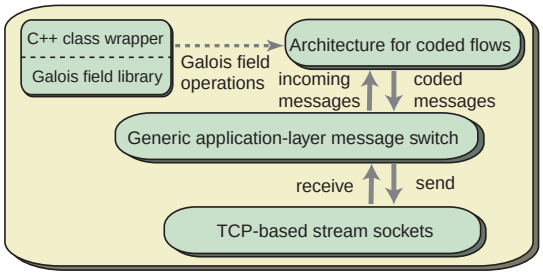

Fig. 3. The overlay network infrastructure.

4.1 Infrastructure

To simplify the implementation of both the algorithms for routing strategy and network coding, we have developed an infrastructure to reduce the mundane work. These include multi-threaded programming for message forwarding engines, failure detection and reaction, measurement of delay and throughput, as well as monitoring and deploying facilities.

To facilitate the switching of application-layer messages from multiple incoming connections to multiple outgoing connections, we have designed a high-performance application-layer message processing facility in UNIX, to support live data sessions from the source to the receivers.

The salient capabilities of the application-layer switch are three-fold: (1) *Message processing*. The application-layer message switch is able to efficiently switch data from upstream nodes to downstream nodes, and process each of them using algorithm-specific implementations. (2) *Measurements of performance metrics*. Important performance metrics such as per-link throughput and latency are measured by the switch. (3) *Emulation of bandwidth availability.* To verify correctness of the algorithm implementations, we sometimes prefer to perform preliminary tests of the algorithm under controlled environments, in which node and link characteristics are more predictable. The switch supports precise emulations of bandwidth availability on each overlay link. For detailed

discussions on the switch design and implementation, we refer the readers to our recent work on *ioverlay* [20].

We have further extended the message processing mechanism to support network coding, by allowing both 1-to-n and m-to-1 mappings between incoming and outgoing flows. Each incoming or outgoing flow is associated with a buffer managed by our customized FIFO queuing algorithm. Each flow consists of a continuous stream of messages. Messages belonging to the same flow reside in the same queue and are processed in their arrival order. An architectural illustration is shown in Fig. 4, in which the design for coded flows introduced in Sec. 3.3 is referred to as the *coding algorithm* for simplicity. We discuss how to identify flows in Sec. 4.3.

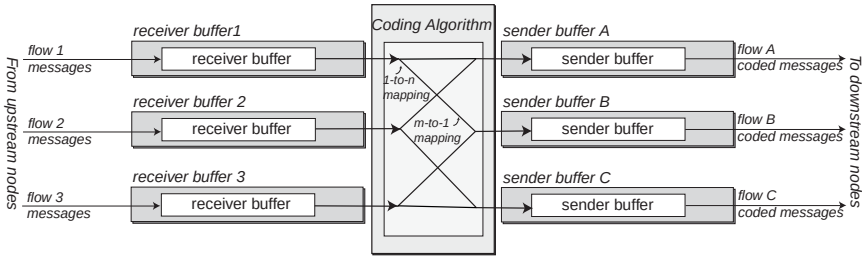

Fig. 4. Switch design: m-to-m mapping among input and output coded flows.

4.2 Routing Strategy

In computing the routing topology, the minimum assumption is that each node is aware of its one overlay-hop neighbors as well as the cost and capacity on its incident links. For any multicast session with m destinations, each node maintains m sets of local information. These information include node price p_i^k, flow excess g_i^k, and flow rate f_{ij}^k as defined in Sec. 3.

During the initialization phase of a multicast session, the source node s sends a `fInitiate` message to each destination d_k directly. On receipt of the `fInitiate` message, the node d_k adds to the source node s an outgoing link with $b_{ds} = f_{ds}^k = \alpha$, and $c_{ds} = \gamma$. In other words, the destination node d injects flows into the source to start the min-cost flow algorithm. For each neighbor, the source node then computes δ and sends the results in a `fPush` message to the corresponding neighbor. When a node receives the `fPush` message, it applies the min-cost flow algorithm to update all its local variables, and sends the value of δ in an `fPush` message to push flows on each link. The destination nodes never push any flow away. The flows on each link converge once the number of flows received by the destination and the number of flows sent by the source are the same. Links that are not selected by the algorithm will have *zero* flows on them. Eventually, the links with a positive flow rate form the transmission topology with the maximized throughput.

After determining the maximum flow between each source-destination pair, we need to perform flow reduction if max-flow rates toward different receivers do not agree. A few additional control messages are introduced into the implementation of our distributed algorithm. We explain the implementation with a representative example in Fig. 5(a), in which each link is labeled with the flow rates it's serving for destination B,

C, and D respectively. The source collects the optimal throughput from each destination using the `fReport` message, and compute the maximum multicast throughput as $\min(7, 10, 5) = 5$.

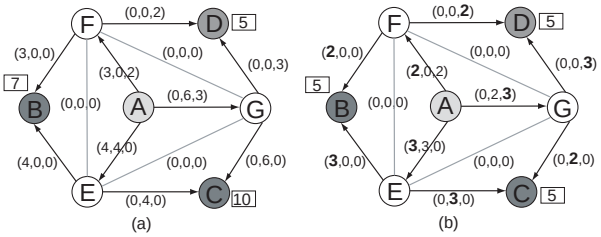

Fig. 5. Reducing flows.

The source node sends a `fReduce` message along each network flow, with the appropriate value for flow reduction (initially 2 in the flow to B and 5 in the flow to C). If a node i can not reduce the specified amount by reducing flow rate at a single outgoing link, it reduces rates on more than one outgoing links, and relay a `fReduce` message along each of them, with the total flow reduction amount sum up to the amount being reduced at i. Result of this reduction procedure is shown in Fig. 5(b). The numbers in bold face indicate the amount of flows each link must serve. Again, the computation of the optimal multicast transmission topology is based on the assumption that each node can support network coding. In Fig. 5(b), the optimal throughput 5 can be achieved for each of the destinations.

4.3 Network Coding

The randomized network coding algorithm presented in Sec. 3.3 is almost ready for direct implementation. Several challenges are still worth mentioning, though. We continue to use our network coding example, as shown in Fig. 1. The first problem is flow identification at each node. Recall that each flow in the transmission topology is a linear combination of the original flows, in the form of $\sum_i \lambda_i f_i$. Hence, each flow can be uniquely identified by the coefficient vector λ, referred to as the *flow description*. The flows are designed to be *self-explanatory*, in that the flow description of the flows are stored in the application-layer header of data messages, as an array of bytes, each byte representing one coefficient. In our example, the flow $23a + 4b$ can be represented by an array of two bytes $[23, 4]$.

Each data message may be coded with several other messages, from the same queue or other queues, to produce a message of an outgoing flow. Our second challenge is to keep the message in the queue long enough for each outgoing flow, while controlling the size of the queue as small as possible at all time. To this end, we modify the message buffer by attaching a reference counter, initialized to n, to each message as it is queued into the appropriate buffer. Every time a message is used to code a new outgoing message, its reference counter is decremented by 1. A message is deleted as soon as the its reference counter reaches zero. For the j^{th} incoming flow on a node, the value of n is the number of nonzero entries in the j^{th} column of the code matrix. More precisely, a positive value presented at the (i, j) cell in the code matrix means that the j^{th} incoming flow is required to produce the i^{th} outgoing flow.

Conventionally, only the message at the front of a FIFO queue is available to the algorithm. This raises the message blocking problem, which causes the algorithm to serve

either all outgoing flows or nothing. Consider two outgoing flows: flow Out_1 requires coding messages from all incoming flows A, B, and C; and flow Out_2 is just a duplication of flow A. Thus, the reference counters for messages of flow A are initialized to 2, whereas the reference counters for messages of flow B and C are initialized to 1. In the case where the buffer for flow A contains several messages, and the buffers for flow B and C are empty, this node is ready to serve flow Out_2, but not flow Out_1. Consequently, none of the messages, except the first one in A can be forwarded to Out_2 until messages become ready in B and C. The same problem occurs, but to a less extent, when the arrival rate varies among different flows. To overcome this problem, we allow algorithms to peek any message in the queue, but must process them in a sequential order.

The third challenge is to ensure the correctness of the decoded messages received at the destinations. At each node, the actual outgoing data messages are computed as linear combinations of the incoming messages, using the code matrix, and over GF(2^8). In Fig. 1, the outgoing message of node G $m^j_{out}(j = 1, 2, 3)$ is produced by taking a message $m^i_{in}(i = 1, 2)$ from each incoming flow i as input, and compute $m^1_{out} = 27 \cdot m^1_{in} + 173 \cdot m^2_{in}$, $m^2_{out} = 112 \cdot m^1_{in} + 85 \cdot m^2_{in}$, and $m^3_{out} = 98 \cdot m^1_{in} + 164 \cdot m^2_{in}$. Since TCP connections preserve message ordering as they are originally generated, the incoming messages are coded in the same order as they are received. For example, the i^{th} message of flow $23a + 4b$ is always coded with the i^{th} message of flow $5a + 97b$. Otherwise, the destination node will not be able to decode and restore the original messages correctly.

5 Evaluation

In this section, we perform an empirical study of various performance aspects of coded flows. we have completed a realistic implementation of our proposed algorithms, and conducted a series of experiments on a cluster of dual-CPU Pentium 4 Xeon 2.4GHz servers. The topology of the test networks are generated using the BRITE topology generator [21], with up to 100 overlay nodes.

The parameters we use include: (1) The number of original flows from the source to all receivers in the optimal routing strategy, henceforth referred to as *maxflow*. (2) The message size, which is the number of data bytes in typical messages of a particular flow, and (3) The session size, which is the number of sender and receivers in a session.

For the purpose of comparison, we implemented an alternative of the k-MST algorithm: rather than computing k minimum spanning trees, we devise and implement a distributed algorithm to compute k spanning trees with the maximum bandwidth, referred to as *k-MaxST*. Since we seek to maximize the throughput of data dissemination rather than minimizing latency, we naturally would like to employ the *widest* selection criterion, in order to achieve high throughput.

5.1 Performance of routing strategy computation

The message overhead introduced by our protocol for computing the optimal routing strategy is less than 12KB per node on average. Such low overhead is due to the reason that messages are passed between nodes only when the prices of flows and nodes are being updated. Fig. 6(a) illustrates the message overhead required to compute the optimal routing strategy is closely related to both the network size and session size. As the network size increases, the message overhead grows linearly. The optimal routing

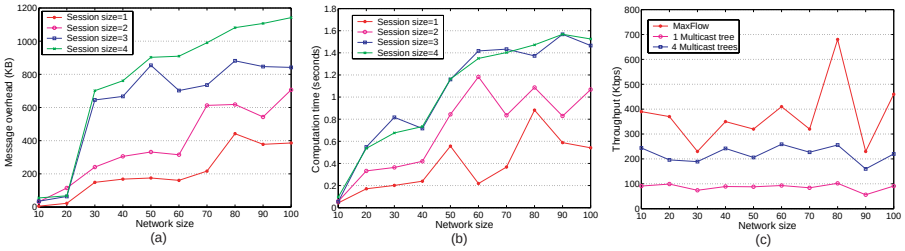

Fig. 6. (a) Messages required to compute the optimal strategy for different session sizes and network sizes; (b) computation time for the routing strategy over networks of different sizes; and (c) the throughput of the optimal strategy in comparison with k multicast trees for the session size of 3 in networks of various sizes.

strategy between each pair of source and destination nodes are computed separately. For every additional destination, the message overhead is increased.

The message overhead is also affected by the network topology. Depending on the number of nodes involved in the optimal transmission topology, the message overhead may vary. As shown in Fig. 6(a), the total message overhead is increased by 400KB in the network of size 30 when the session size is increased from 2 to 3. This is mainly because the optimal routing strategy between the source and the third destination introduces a number of new nodes to the final optimal routing strategy.

We count the computation time of the algorithm from the time a request is sent to the source node until the time the routing strategy is fully established, *i.e.*, when all nodes have no excess flows. The time plotted in Fig. 6(b) shows that the computation time again depends on both the network size and the session size. With more destinations, a larger number of messages are exchanged and processed. Note that the delay of less than 1.6 seconds is introduced only once upon session set up, and does not apply to the transmission of data messages.

The theoretical achievable throughput of the routing strategy is presented in Fig. 6(c) in comparison with single multicast tree and four multicast trees. We observe that our routing strategy always enjoys the highest throughput regardless of the network size.

5.2 Coding delay

In coded overlay flows, the end-to-end delay depends on not only the TCP delay, but also the coding delay introduced at each routing hop. Therefore, we measure the coding time on each node and the end-to-end delay. We present the average coding time per node and and the code-delay ratio under various parameter settings. The *code-delay* ratio is the ratio between the sum of the coding time on all nodes and the end-to-end delay. This sum is the upper bound of the end-to-end coding time since non-sequential nodes perform coding in parallel.

To quantitatively evaluate the average coding time per node, we vary the size of the application-layer data messages from 1KB to 35KB, and measure the time to code each message. The results are presented in Table 3. We observe that the computation time increases linearly as the data size increases, but they are all on the order of microseconds, which is insignificant compared to typical end-to-end latencies over wide-area networks.

To evaluate the code-delay ratio, we set the available bandwidth on each link in the overlay within the range (1KB, 10KB). In order to achieve the optimal throughput,

Size	1	5	10	15	20	25	30	35
Computation time (μsec)	224	1057	2194	3288	4447	5374	6256	7664

Table 3. Average computational overhead for coding one message at one node over different message sizes.

the message size is set to 1KB so that each link can accommodate as many flows as possible. We start the experiment by selecting a pair of source and destination node. One additional receiver node is added to the session for each repetition of the experiment. We set the maxflow of each session to be within the range $(11, 20)$. In each experiment, we collect the times that each node has spent on performing Galois field coding as well as the end-to-end delay.

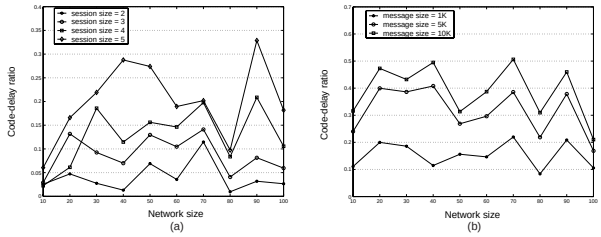

Fig. 7. The code-delay ratio of: (a) sessions with different sizes; and (b) sessions with different message sizes.

Nodes in larger networks spend less time on network coding since the flows are scattered in the network. In other words, network coding is performed in a more distributed fashion in larger networks. In Fig. 7(a), we have observed that the coding time becomes more significant as the session size increases. This is because each additional receiver requires at least k more flows in the network, where k is the maxflow from the source to all receivers.

We further investigate the effect of message size in various network settings. We vary message size from 1KB to 5KB and then to 10KB. The session size is set to 4, and the maxflow is within the (11-20) range throughout this experiment. We compare the coding and end-to-end delay of sessions with different message sizes. As shown in Fig. 7(b), the larger message sizes introduce higher coding time at each node, but the amount of increase is rather moderate, which is consistent with the results in Table 3.

5.3 Throughput

The key advantage of coded overlay flows is the improvement in end-to-end throughput. Theoretically, the throughput achieved by coded overlay flows approaches mathematically-provable optimality. In order to evaluate the achievable throughput of our algorithms, especially the optimal routing strategy, we conduct a set of experiments under various settings.

To see how the multicast throughput computed by our algorithm varies with more nodes joining the multicast session, we test the algorithm with increasing session sizes. For each network size, we select one node as the source node and add one new destination node to multicast group as the session size increases. The throughput performance over time (in increasing order of the sequence number) under different parameter settings is shown in Fig. 8(a). These throughput are collected from a network of size 80. The optimal throughput of the session of size 2 is 12Kbps. We observe that the throughput of a session quickly converges to the optimal throughput after starting up,

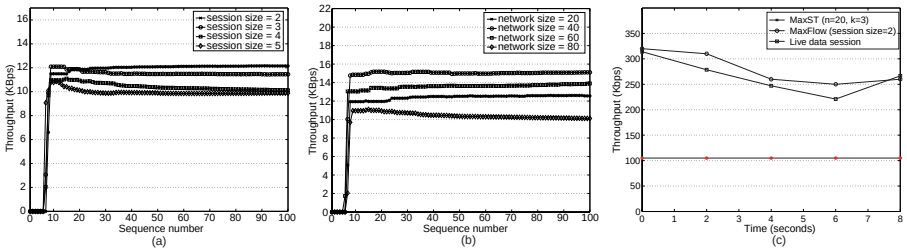

Fig. 8. The throughput of (a) sessions with different sizes; (b) sessions with size = 4, message size = 1KB, and maximum achievable throughput = $(11 - 20)$KBps in different networks; and (c) coded overlay flow in comparison with 1-MaxST and maxflow.

and remains stable throughout the session. The throughput decreases only when the newly added destination has lower end-to-end throughput with the source node than the throughput of the smaller session size. Such situation arises when a node with a small incident link capacity joins the multicast group. Otherwise, a high flow rate to each destination can be concurrently achieved, without interference with each other, thanks to network coding.

To further verify the scalability of coded overlay flows in terms of network sizes, we run experiments in various networks, and evaluate the throughput measurement of the first 100 messages of sessions with fixed size in different networks. For fair comparison, we control the maximum achievable throughput of the multicast session to be within the range 11Kbps and 20Kbps. Fig. 8(b) provides additional evidence for the optimality of the throughput achieved by coded overlay flows. Therefore, we claim that the coded overlay flows can achieve maximized throughput with minimum tradeoff of end-to-end delay.

Finally, we show the performance of all three algorithms working together over time. We start with a network of size 20, and select two links to decrease their bandwidth every 2 seconds. As shown in Fig. 8(c), the maximum throughput of the single multicast tree remains unchanged. It is because the bandwidth on the selected links are not decreased below the bottleneck link bandwidth of the corresponding tree. However, the throughput of a multicast session of size 2 is indeed affected by the bandwidth variation. The actual throughput for a *live data streaming session* with randomized network coding of this multicast session is also given in Fig. 8(c). We observe that the throughput of a *live coded data session* is rather close to the optimal throughput estimated by the optimal routing strategy. This justifies the case for coded overlay flows in realistic experiments.

6 Concluding Remarks

In this paper, we have established the case for coded overlay flows, which uses network coding to achieve maximized end-to-end throughput in multicast sessions. We propose distributed algorithms to construct the optimal transmission topology, and design corresponding coding strategies for data dissemination. Our main contribution is the implementation of coding in the Galois field and all our proposed algorithms in a realistic overlay testbed, which suggests that using coded overlay flows to maximize throughput is not only theoretically sound, but realistically feasible. To our knowledge, this is the first attempt towards implementing network coded flows in overlay networks.

References

1. Y. H. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," in *Proceedings of the ACM SIGMETRICS*, June 2000, pp. 1–12.
2. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proceedings of the ACM SIGCOMM*, August 2002, pp. 205–217.
3. S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-Time Applications," in *Proceedings of IEEE INFOCOM*, 2003.
4. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proceedings of ACM SOSP*, October 2003.
5. S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Transactions on Information Theory*, vol. 49, pp. 371, 2003.
6. Z. Li, B. Li, D. Jiang, and L. C. Lau, "On Achieving Optimal Throughput with Network Coding," in *Proceedings of IEEE INFOCOM*, March 2005.
7. Z. Li and B. Li, "Efficient and Distributed Computation of Maximum Multicast Rates," in *Proceedings of IEEE INFOCOM*, March 2005.
8. D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proceedings of ACM SOSP*, October 2003.
9. J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks," in *Proceedings of ACM SIGCOMM*, 2002.
10. K. Shen, "Structure Management for Scalable Overlay Service Construction," in *Proceedings of NSDI*, 2004.
11. A. Young, J. Chen, Z. Ma, L. Peterson A. Krishnamurthy, and R. Y. Wang, "Overlay Mesh Construction Using Interleaved Spanning Trees," in *Proceedings of the IEEE INFOCOM*, March 2004.
12. R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.
13. R. Koetter and M. Médard, "An Algebraic Approach to Network Coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, October 2003.
14. D. Lun, N. Ratnakar, R. Koetter, M. Médard, E. Ahmed, and H. Lee, "Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding," in *Proceedings of IEEE INFOCOM*, March 2005.
15. Y. Wu, P. A. Chou, Q. Zhang, K. Jain, W. Zhu, and S. Kung, "Network Planning in Wireless Ad Hoc Networks: A Cross-Layer Approach," *IEEE Journal on Selected Areas in Communication*, vol. 23, no. 1, January 2005.
16. C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proceeding of IEEE INFOCOM*, March 2005.
17. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, 1989.
18. P. Sanders, S. Egner, and L. Tolhuizen, "Polynomial Time Algorithm for Network Information Flow," in *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.
19. J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," in *Software - Practice & Experience*, September 1997, pp. 27(9):995–1012.
20. B. Li, J. Guo, and M. Wang, "iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implementations," in *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Conference (Middleware)*, October 2004.
21. A. Medina, A. Lakhina, I. Matta, and J. Byers, *BRITE: Boston University Representative Internet Topology Generator*, http://www.cs.bu.edu/brite.