CodedBulk: Inter-Datacenter Bulk Transfers using Network Coding

Shih-Hao Tseng, Saksham Agarwal, and Rachit Agarwal, *Cornell University;* Hitesh Ballani, *Microsoft Research;* Ao Tang, *Cornell University*

https://www.usenix.org/conference/nsdi21/presentation/tseng

This paper is included in the Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation.

April 12–14, 2021

978-1-939133-21-2

CodedBulk: Inter-Datacenter Bulk Transfers using Network Coding

Shih-Hao Tseng
Cornell University

Saksham Agarwal
Cornell University

Rachit Agarwal
Cornell University

Hitesh Ballani
Microsoft Research

Ao Tang
Cornell University

Abstract

This paper presents CodedBulk, a system for high-throughput inter-datacenter bulk transfers. At its core, CodedBulk uses network coding, a technique from the coding theory community, that guarantees optimal throughput for individual bulk transfers. Prior attempts to using network coding in wired networks have faced several pragmatic and fundamental barriers. CodedBulk resolves these barriers by exploiting the unique properties of inter-datacenter networks, and by using a custom-designed hop-by-hop flow control mechanism that enables efficient realization of network coding atop existing transport protocols. An end-to-end CodedBulk implementation running on a geo-distributed inter-datacenter network improves bulk transfer throughput by $1.2-2.5\times$ compared to state-of-the-art mechanisms that do not use network coding.

1 Introduction

Inter-datacenter wide-area network (WAN) traffic is estimated to quadruple over the next five years, growing $1.3\times$ faster than global WAN traffic [2]. In the past, such an increase in demand has been matched by physical-layer technological advancements that allowed more data to be pumped on top of WAN fibers, thus increasing the capacity of existing inter-datacenter links [14]. However, we are now approaching the fundamental non-linear Shannon limit on the number of bits/Hz that can be practically carried across fibers [14]. This leaves datacenter network providers with the expensive and painfully slow proposition of deploying new fibers in order to keep up with increasing demands.

Several studies have reported that inter-datacenter traffic is dominated by geo-replication of large files (*e.g.*, videos, databases, etc.) for fault tolerance, availability, and improved latency to the user [3, 6, 8, 23, 25, 26, 31, 39, 40, 47, 49]. We thus revisit the classical multicast question: given a fixed network topology, what is the most throughput-efficient mechanism for transferring data from a source to multiple destinations? The answer to this question is rooted in network coding, a technique from the coding theory community that generalizes the classical max-flow min-cut theorem to the case of (multicast) bulk transfers [9, 30, 33]. Network coding guarantees optimal throughput for individual bulk transfers by using in-network computations [9, 24, 30, 33]; in contrast, achieving optimal throughput using mechanisms that do not perform in-network computations is a long-standing open problem [34]. We provide a primer on network coding in §2.2. While network coding has been successful applied in wireless networks [28, 29], its applications to wired networks have faced several pragmatic and fundamental barriers.

On the practical front, there are three challenges. First, network coding requires network routers to buffer and to perform computations on data, which requires storage and computation resources. Second, computing "network codes" that define computations at routers not only requires a priori knowledge about the network topology and individual transfers, but also does not scale to networks with millions of routers and/or links. Finally, network coding requires a single entity controlling the end-hosts as well as the network.

In traditional ISP networks, these challenges proved to be insurmountable but the equation is quite different for inter-datacenter WANs. The structure of typical inter-datacenter WAN topologies means that, instead of coding at all routers in the network, coding can be done only at resource-rich datacenters—either at border routers or on servers inside the datacenter—without any reduction in coding gains (§2.1). Inter-datacenter WAN operators already deploy custom border routers, so increase in computation and storage resources at these routers to achieve higher throughput using the available WAN bandwidth (that is expensive and increasingly hard-to-scale) is a good trade-off to make. The second and third challenges are also alleviated by unique characteristics [23, 25] of inter-datacenter WANs: (1) network sizes limited to hundreds of routers and links enables efficient computation of network codes and implementation of network coding; (2) SDN-enabled routers combined with the fact that transfers are known a-priori [39, 40] allow for centralized code computations which can, in turn, be programmed into routers; and (3) a single entity controlling end-hosts as well as the network.

While inter-datacenter WAN features lower the pragmatic barriers, one fundamental challenge still needs to be resolved. Traditional network coding assumes that there is no other traffic in the network, either foreground (*e.g.*, interactive traffic) or background (*e.g.*, other bulk transfers). More generally, network coding assumes that all links have the same latency and bandwidth, and that link latencies and bandwidths remain static over time. This assumption does not hold in practice, *e.g.*, due to sporadic high-priority interactive traffic, and due to multiple concurrent bulk transfers atop inter-datacenter WANs. We refer to this as the *asymmetric link* problem.

CodedBulk is an end-to-end system for high-throughput inter-datacenter bulk transfers that resolves the asymmetric link problem using a simple Hop-by-hop Flow Control (HFC) mechanism. The core idea in HFC mechanisms is to partition available buffer space at routers among active flows, so as to avoid buffer overflow [41, 45]. HFC mechanisms have been explored for traditional non-coded traffic [37, 41, 42, 51]; however, using HFC mechanisms for network coding imposes an additional constraint: all flows that need to be coded at any router must converge to the *same* rate. Simultaneously, routers need to work with limited storage and compute resources in the data plane. We show that *any* buffer partitioning scheme that assigns non-zero buffers to each (coded) flow achieves the following desirable properties: (i) for each individual bulk transfer, all incoming flows that need to be coded at a router converge to the same rate; (ii) for all bulk transfers sharing a link, the rate for all their flows through the link converge to the max-min fair rate; and (iii) the network is deadlock-free.

The use of network coding, coupled with HFC, also means that flow control at a router is correlated across multiple flows. While this could be implemented via modifications in the network stack, we introduce a *virtual link* abstraction which enables CodedBulk without any modifications to existing flow control-enabled transport layer and multipath-enabled network layer implementations. For instance, CodedBulk currently runs on top of unmodified TCP and MPLS-enabled network layer mechanisms supported by existing inter-datacenter WANs. Our implementation requires no special traffic shaping mechanism, allows co-existence of high-priority interactive traffic, and handles failures transparently.

We envision two deployment scenarios for CodedBulk. First, an infrastructure provider can provide CodedBulk as a service to geo-distributed services (including its own). In the second scenario, a geo-distributed service renting compute, storage and inter-datacenter bandwidth resources from an infrastructure provider can use CodedBulk to improve bulk transfer throughput without any support from the provider. We have implemented CodedBulk for both scenarios — an overlay service, a software proxy and a hardware proxy. The first two implementations currently run on geo-distributed inter-datacenter WAN. All the three implementations, along with a CodedBulk simulator, are available at: `https://github.com/SynergyLab-Cornell/codedbulk`.

The benefits of network coding depend on the underlying network topology, the number of destinations in individual bulk transfers, the source and set of destinations in each bulk transfer, the number of concurrent transfers and interactive traffic load. To understand the envelope of settings where CodedBulk provides benefits, we evaluate CodedBulk over a testbed comprising 13 geo-distributed datacenters organized around the B4 [25] and Internet2 [5] inter-datacenter WAN topologies, and perform sensitivity analysis of CodedBulk performance against all of the above factors. Our evaluation demonstrates that CodedBulk achieves $1.2-2.5\times$ higher throughput for bulk transfers when compared to existing state-of-the-art mechanisms that do not perform network coding. All the results presented in this paper are for real implementations of CodedBulk; this paper uses no simulation results.

2 CodedBulk Overview

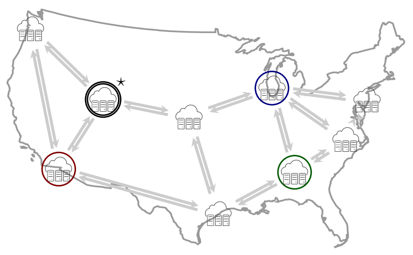
We begin by describing our model for inter-datacenter WANs (§2.1). We then provide a primer for network coding (§2.2). Next, we discuss several pragmatic challenges that have posed a barrier to adoption of network coding in wired networks and how unique characteristics of inter-datacenter WANs enable overcoming these barriers (§2.3). We close the section with a high-level description of the CodedBulk design (§2.4).

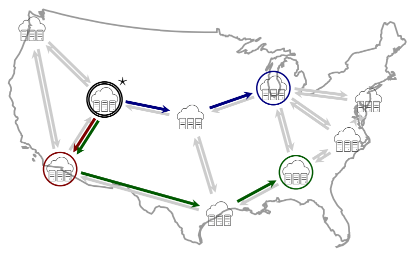
2.1 Preliminaries

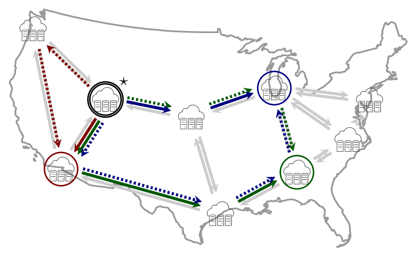
We follow the same terminology as in existing inter-datacenter WAN literature [26, 27, 31, 32, 39, 40, 49, 52]. Specifically, we model the inter-datacenter WAN as a directed graph $G = (V,E)$, where V is the set of nodes denoting datacenters and E is the set of links between pairs of datacenters. To account for the full-duplex nature of inter-datacenter WAN links, we create two links $u \rightarrow v$ and $v \rightarrow u$ for each pair of nodes $(u,v) \in V$ with a physical link between them. Each link has a capacity equal to its bandwidth available for bulk transfers.

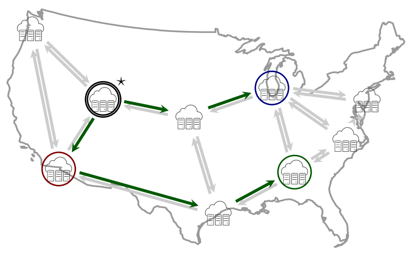
We discuss two important aspects of the network model. First, while links in wired networks are full-duplex, the graph in inter-datacenter literature is usually modeled as a directed graph since links in two directions can have different *available* bandwidths at different times, *e.g.*, due to high-priority interactive traffic using (a fraction of) the bandwidth in one direction. Second, in practice, geo-distributed datacenters are often connected via intermediate routers operating at layer-1 or layer-3; these routers either operate exactly like a relay (have degree two, with incoming bandwidth equal to outgoing bandwidth), or have their bandwidth statically partitioned across multiple outgoing links. Both these cases are equivalent to having a direct link with a specific capacity in each direction between each pair of datacenters with a physical link between them.

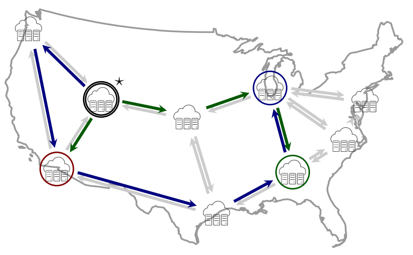
We define bulk transfers as in prior work [23, 25, 26, 27, 31, 32, 38, 39, 40, 49, 52]: transfers that are bandwidth-intensive. A bulk transfer is a source s sending a file to a subset of nodes $T \subseteq V - \{s\}$.

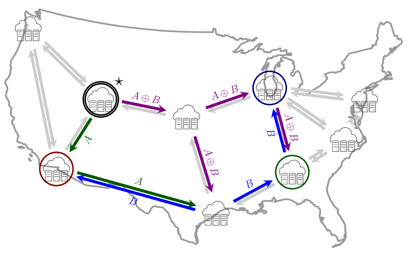

(a) Inter-datacenter bulk transfer example


(b) single-path solution (suboptimal)


(c) multi-path solution (suboptimal)


(d) Steiner-tree solution (suboptimal)


(e) Optimal non-coded solution (computed by hand; efficient algorithms to compute such an optimal solution are not known).



(f) CodedBulk solution (optimal).

Figure 1: Understanding benefits of network coding. (a) An instance of the inter-datacenter bulk transfer problem on the Internet2 topology [5], with one source (marked by a circle with a $\star$) and three destinations (marked by circles). The network model is as described in §2.1. (b, c, d) Existing solutions based on single-path, multi-path and Steiner arborescence packing can be suboptimal (detailed discussion in §2.2). (e) An optimal solution with Steiner arborescence packing (computed by hand); today, computing such a solution requires brute force search which is unlikely to scale to inter-datacenter deployment sizes (tens to hundreds of datacenters) [23, 25]. (f) CodedBulk, using network coding, not only achieves optimal throughput but also admits efficient algorithms to compute the corresponding network codes. More discussion in §2.2.

2.2 Network coding background

Suppose a source wants to send a large file to a single destination, and that it is allowed to use as many paths as possible. If there are no other flows in the network, the maximum achievable throughput (the amount of data received by the destination per unit time) is given by the well-known max-flow min-cut theorem—the achievable throughput is equal to the capacity of the min-cut between the source and the destination in the induced graph. The corresponding problem for a source sending a file to multiple destinations was an open problem for decades. In 2000, a now celebrated paper [9] established that, for a multicast transfer, the maximum achievable throughput is equal to the minimum of the min-cuts between the source and individual destinations. This is also optimal. For general directed graphs, achieving this throughput is not possible using solutions where intermediate nodes simply forward or mirror the incoming data—it necessarily requires intermediate nodes to perform certain computations over the incoming data before forwarding the data [9, 30, 33].

For our network model that captures full-duplex links, network coding achieves optimal throughput (since it subsumes solutions that do not perform coding); however, it is currently not known whether optimal throughput can be achieved without network coding [10, 34]. Figure 1 demonstrates the space of existing solutions, using a bulk transfer instance from our evaluation (§4) on the Internet2 topology. We present additional discussion and examples in [46]. Single-path (also referred to as multiple unicast) solutions, where the source transfers data along a single path to each individual destination, can be suboptimal because they neither utilize all the available network bandwidth, nor do they allow intermediate nodes to forward/mirror data to other destinations. Multi-path solutions, where the source transfers data along all edge-disjoint paths to each individual destinations (paths across destinations do not need to be edge disjoint), can be suboptimal because they do not allow intermediate nodes to forward/mirror data to other destinations.

The current state-of-the-art solutions for our network model are based on Steiner tree (or, more precisely, Steiner arborescence) packing [7, 18, 34]. These solutions use multiple paths, and allow intermediate nodes to mirror and forward the data; however, they can be suboptimal because the problem of computing optimal Steiner tree (or arborescence) packing is NP-hard, and approximation algorithms need to be used [12]. To demonstrate the limitations of existing Steiner packing solutions, consider the example shown in Figure 1(d): here, once the shown Steiner tree is constructed, no additional Steiner trees can be packed in a manner that higher throughput can be achieved. Figure 1(e) demonstrates the complexity of computing an optimal solution (that we constructed by hand)—to achieve the optimal solution shown in the figure, one must explore intermediate solutions that use a suboptimal Steiner tree (shown in blue color). Today, computing such an optimal solution requires a brute force approach, which is unlikely to scale to inter-datacenter network sizes. Thus, we must use approximate suboptimal solutions; to the best of our knowledge, the state-of-the-art algorithms for computing approximate Steiner packing solutions for our network model do not even admit polylogarithmic approximation factors [11, 21].

Network coding avoids the aforementioned limitations of existing solutions by allowing intermediate nodes to perform certain computations, which subsume forwarding and mirroring, on data (as shown in the Figure 1(f) example)—it utilizes multiple paths, guarantees optimal throughput, and admits efficient computation of network codes that achieve optimal throughput [24]. Thus, while designing optimal non-coded solutions for bulk transfers remains an open problem, we can efficiently achieve throughput optimality for inter-datacenter bulk transfers *today* using network coding.

2.3 Resolving pragmatic barriers

While network coding is a powerful technique, its applications to wired networks have been limited in the past due to several pragmatic challenges. In this subsection, we use the example in Figure 1(f) to discuss these challenges, and how inter-datacenter WANs allow overcoming these challenges.

Buffering and computation at intermediate nodes. Network coding requires intermediate nodes to buffer data and perform computations. For instance, the bottom-center node in Figure 1(f) needs to perform XOR operations on packets from two flows A and $A \oplus B$. This requires the node to have storage (to buffer packets from A and $A \oplus B$), and computation resources (to compute $A \oplus (A \oplus B) = B$) in the data plane. While this was challenging in traditional ISP networks, inter-datacenter WANs allow overcoming this barrier easily: as noted in prior studies [31], each node in an inter-datacenter WAN is a datacenter with compute and storage resources that are significantly cheaper, more scalable and faster to deploy than inter-datacenter bandwidth.

Computing and installing network codes. Nodes in Figure 1(f) perform specific actions (forward, mirror, code-and-forward and code-and-mirror). These actions are specified using network codes, computing which requires a priori knowledge of the network topology, and the source and the set of destinations for each bulk transfer. This information was hard to get in ISP networks; however, inter-datacenter WANs already collect and use this information [23, 25, 31, 49]. Network coding also requires transmissions from end-hosts to be coordinated by the controller. In inter-datacenter WAN scenarios, this is feasible as a single entity controls the end-hosts as well as the network. Existing SDN infrastructure [23, 25] is also useful for this purpose—a centralized controller can compute the code, and can populate the forwarding tables of intermediate nodes (using existing support for MPLS tags and multi-path routing) before the bulk transfer is initiated.

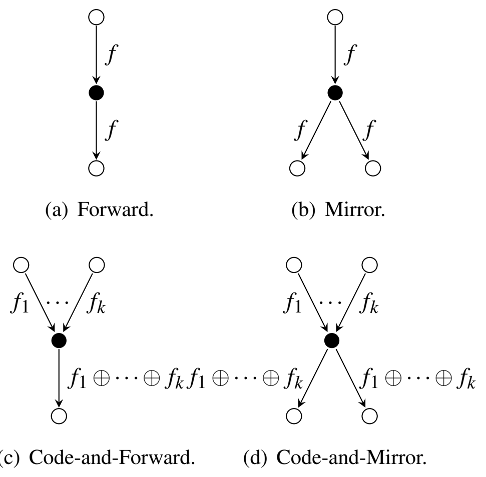


Figure 2: Four basic coding functions available at each intermediate node to implement the network code generated by CodedBulk.

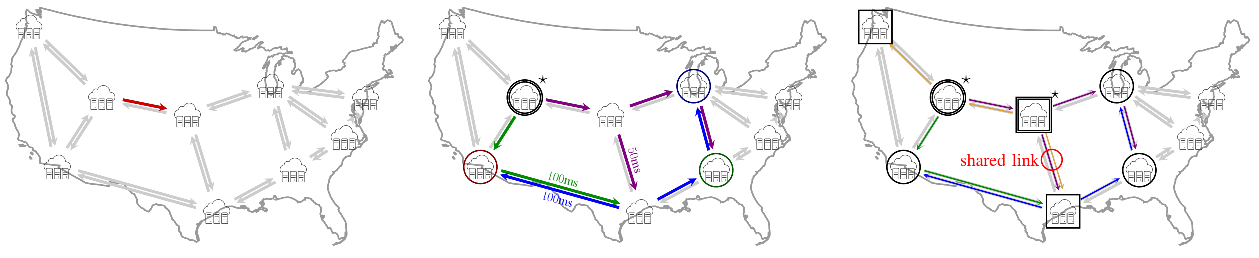
Existing algorithms [24] for computing network codes run in polynomial time, but may not scale to networks with millions of nodes and edges; however, this is not a concern for CodedBulk since inter-datacenter WANs comprise of only hundreds of nodes and links. Computation and installation of network codes, and buffering of data at intermediate nodes may also lead to increased latency for bulk transfers. However, since bulk transfers are not latency-sensitive [23, 25], a slight increase in latency to achieve significantly higher throughput for bulk transfers is a favorable tradeoff [31].

2.4 CodedBulk design overview

The high-level CodedBulk design for a single bulk transfer case can be described using the following five steps:

1. The source node, upon receiving a bulk transfer request, notifies the controller of the bulk transfer. The notification contains the source identifier, the identifiers for each destination, and an optional policy on the set of links or intermediate nodes not to be used (*e.g.*, for security and isolation purposes).
2. The controller maintains a static topology of the inter-datacenter network graph. While optimizations are possible to exploit real-time traffic information, the current CodedBulk implementation does not use such optimizations. The controller computes, for each destination, the set of edge-disjoint paths between the source and the destination, along with the bandwidth for each path. Using these paths, the controller computes the network code for the bulk transfer using the network coding algorithm from [24][1]. The network code comprises of the routing and forwarding

[1]The network coding algorithm in [24] requires as input a directed acyclic graph. However, the multipath set in our construction may lead to a cyclic graph. We use an extension similar to the original network coding paper [9] to generate network codes for cyclic graphs. Please see [46] for details.



(a) Reason: Interactive traffic. (b) Reason: Non-uniform delay. (c) Reason: Multiple bulk transfers.

Figure 3: Understanding asymmetric link problem. (a) Due to sporadic high-priority interactive traffic (*e.g.*, the one shown in red), different links may have different (time-varying) bandwidths; (b) If network links have significantly different round trip times, naïvely implementing traditional network coding would require large amount of fast data plane storage to buffer data that arrives early at nodes; (c) multiple concurrent bulk transfers, especially those that end up sharing links, make it hard to efficiently realize traditional network coding solutions that assume a single bulk transfer at all times. Detailed discussion in §3.1.

information for each flow, and the computations done at each intermediate node for the flows arriving at that node. These codes can be expressed as a combination of four basic functions shown in Figure 2.

3. Once the network code is computed, the controller installs the network code on each node that participates in the bulk transfer. We discuss, in §3.3, a mechanism to implement the forwarding and routing functions that requires no changes in existing inter-datacenter WAN infrastructure.
4. Once the code is installed, the controller notifies the source. The source partitions the bulk transfer file into multiple subfiles (defined by the code) and then initiates the bulk transfer using CodedBulk, as described in the remainder of the paper. For instance, for the example of Figure 1(f), the source divides the file into two subfiles (A and B) of equal sizes and transmits them using the code shown in the figure. Each intermediate node *independently* performs CodedBulk's hop-by-hop flow control mechanism. Importantly, a "hop" here refers to a datacenter on the network topology graph. CodedBulk assumes that interactive traffic is always sent with the highest priority, and needs two additional priority levels.
5. Once the bulk transfer is complete, the source notifies the controller. The controller periodically uninstalls the inactive codes from all network nodes.

The core of CodedBulk's mechanisms are to efficiently enable the fourth step. We describe these in the next section.

3 CodedBulk Design

We describe the core techniques in CodedBulk design and implementation. We start by building an in-depth understanding of the asymmetric link problem (§3.1). We then describe how CodedBulk resolves the asymmetric link problem using a custom-designed hop-by-hop flow control mechanism (§3.2). Finally, we discuss the virtual link abstraction that enables implementation of CodedBulk without any modifications in underlying transport- and network-layer protocols (§3.3).

3.1 Understanding fundamental barriers

We start by building an in-depth understanding of the asymmetric link bandwidth problem, and how it renders techniques in network coding literature infeasible in practice. We use Figure 3 for the discussion in this subsection.

Asymmetric links due to sporadic interactive traffic. Inter-datacenter WANs transfer both latency-sensitive interactive traffic (*e.g.*, user commits, like emails and documents) and bandwidth-intensive bulk traffic [23, 25]. While interactive traffic is low-volume, it is unpredictable and is assigned higher priority. This leads to two main challenges. First, links may have different bandwidths available at different times for bulk transfers (as shown in Figure 3(a)). Second, the changes in available bandwidth may be at much finer-grained timescales than the round trip times between geo-distributed datacenters.

Traditional network coding literature does not consider the case of interactive traffic. An obvious way to use traditional network coding solutions for non-uniform link bandwidths is to use traffic shaping to perform network coding on the minimum of the available bandwidth across all links. For instance, in the example of Figure 3(a), if the average load induced by interactive traffic is $0.1\times$ link bandwidth, then one can use network coded transfers only on $0.9\times$ bandwidth. However, the two challenges discussed above make this solution hard, if not infeasible: bandwidths are time-varying, making static rate allocation hard; and, bandwidth changing at much fine-grained timescales than geographic round trip times makes it hard to do dynamic rate allocation.

Asymmetric links due to non-uniform delay. Traditional network coding solutions, at least the practically feasible ones [24], require computations on data arriving from multiple flows in a deterministic manner: packets that need to be coded are pre-defined (during code construction) so as to allow the destinations to decode the original data correctly. To achieve this, existing network coding solutions make one of the two assumptions: either the latency from the source to each individual node is uniform; or, unbounded storage

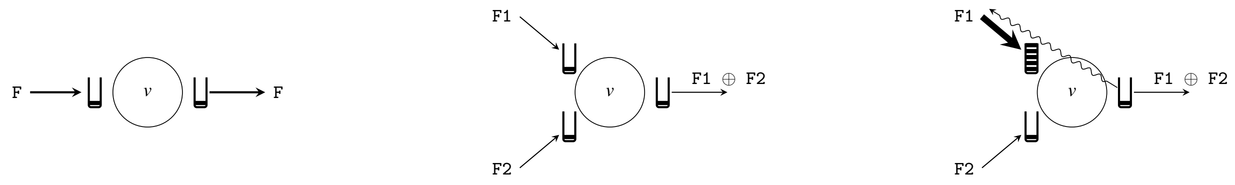


Figure 4: Understanding hop-by-hop flow control for a single bulk transfer. (left) if the outgoing link has enough bandwidth to sustain the rate of incoming traffic (flow `F` in this example), then all buffers will remain unfilled and flow control will not be instantiated; (center) the same scenario as the left figure holds as long as the two conditions hold: (1) both flows that need to be coded at some node v send at the same rate; and (2) the outgoing link has enough bandwidth to sustain the rate of incoming traffic; (right) If two flows need to be coded at some node v, and one of the flows `F1` is sending at higher rate, then the `Rx` buffer for `F1` will fill up faster than it can be drained (due to v waiting for packets of `F2`) and flow control to the downstream node of `F1` will be triggered, resulting in rate reduction for flow `F1`. Detailed discussion in §3.2.

at intermediate nodes to buffer packets from multiple flows. Neither of these assumptions may hold in practice. The delay from the source to individual intermediate nodes can vary by hundreds of milliseconds in a geo-distributed setting (Figure 3(b)). Keeping packets buffered during such delays would require an impractical amount of high-speed storage for high-bandwidth inter-datacenter WAN links: if links are operating at terabits per second of bandwidth, each intermediate node would require hundreds of gigabits or more of storage.

Asymmetric links due to simultaneous bulk transfers. Traditional network coding literature considers only the case of a single bulk transfer. Designing throughput-optimal network codes for multiple concurrent bulk transfers is a long-standing open problem. We do not solve this problem; instead, we focus on optimizing throughput for individual bulk transfers while ensuring that the network runs at high utilization.

Achieving the above two goals simultaneously turns out to be hard, due to each individual bulk transfer observing different delays (between respective source to intermediate nodes) and available link bandwidths due to interactive traffic. Essentially, as shown in Figure 3(c), supporting multiple simultaneous bulk transfers requires additional mechanisms for achieving high network utilization.

3.2 CodedBulk's hop-by-hop flow control

Network coding, by its very nature, breaks the end-to-end semantics of traffic between a source-destination pair, thus necessitating treating the traffic as a set of flows between the intermediate nodes or hops. Recall that a "hop" here refers to a (resource-rich) datacenter on the network graph. To ensure that we do not lose packets at intermediate nodes in spite of the fact that they have limited storage, we rely on a hop-by-hop flow control mechanism—a hop pushes back on the previous hop when its buffers are full. This pushback can be implicit (*e.g.*, TCP flow control) or explicit.

Hop-by-hop flow control is an old idea, dating back to the origins of congestion control [41, 45]. However, our problem is different: traditional hop-by-hop flow control mechanisms operate on individual flows—each downstream flow depends on precisely one upstream flow; in contrast, CodedBulk operates on "coded flows" that may require multiple upstream flows to be encoded at intermediate nodes. Thus, a flow being transmitted at a low rate can affect the overall performance of the transfer (since other flows that need to be encoded with this flow will need to lower their rate as well). This leads to a correlated rate control problem. For instance, in Figure 2(c) and Figure 2(d), flows f_1 to f_k must converge to the same rate so that the intermediate node can perform coding operations correctly without buffering large number of packets. To that end, CodedBulk's hop-by-hop flow control mechanism maintains three invariants:

- All flows within the same bulk transfer that need to be encoded at any node must converge to the same rate;
- All flows from different bulk transfers competing on the congested link bandwidth must converge to a max-min fair bandwidth allocation;
- The network is deadlock-free.

CodedBulk maintains these invariants using a simple idea: careful partitioning of buffer space to flows within and across bulk transfers. The key insight here, that follows from early work on buffer sharing [45], is that for large enough buffers, two flows congested on a downstream link will converge to a rate that corresponds to the fair share of the downstream link bandwidth. We describe the idea of CodedBulk's hop-by-hop flow control mechanism using two scenarios: single isolated bulk transfer and multiple concurrent bulk transfers.

Single bulk transfer. First consider the two simpler cases of forward (Figure 2(a)) and mirror (Figure 2(b)). These cases are exactly similar to traditional congestion control protocols, and hence do not require any special mechanism for buffer sharing. The main challenge comes from Code-and-Forward (Figure 2(c)) and Code-and-Mirror (Figure 2(d)). For these cases, the invariant we require is that the flows being used to compute the outgoing data converge to the same rate since otherwise packets belonging to the flows sending at a higher rate will need to be buffered at the node, requiring high storage. This is demonstrated in Figure 4, center and right figures.

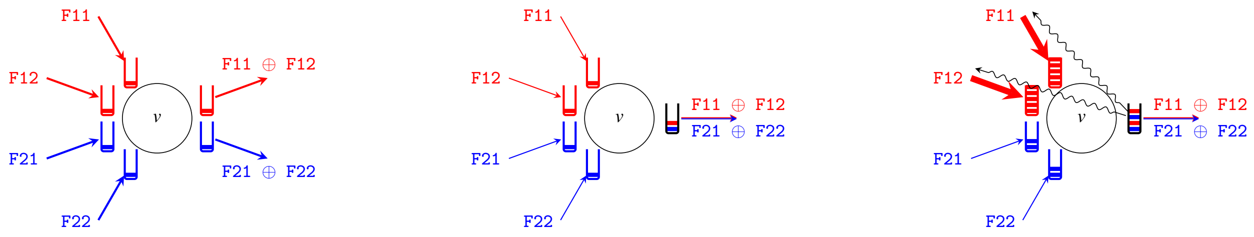


Figure 5: If concurrent bulk transfers use completely different outgoing links (left) or use the same outgoing link but with enough bandwidth (center), the hop-by-hop flow control mechanism does not get triggered. However, if the outgoing link is bandwidth-bottlenecked, and one of the bulk transfers is sending at higher rate (say the red one), then the buffers for the red flows will fill up faster than the buffers for blue flows; at this point, hop-by-hop flow control mechanism will send a pushback to the downstream nodes of the red flows, resulting in reduced rate for the red flows. Detailed discussion in §3.2.

Our insight is that a buffer partitioning mechanism that assigns *non-zero* buffers to each incoming flow maintains the second and the third invariants. It is known that non-zero buffer allocation to each flow at each link leads to deadlock-freedom [45]. It is easy to see that the second invariant also holds—if one of the flows sends at a rate higher than the other (Figure 4(right)), the buffer for this flow will fill up faster than the buffer for the other flow, the flow control mechanism will be triggered, eventually reducing the rate of the flow.

Multiple simultaneous bulk transfers. CodedBulk handles each bulk transfer independently using its hop-by-hop flow control mechanism. Again, we provide intuition using an example. Consider two simultaneous bulk transfers at some intermediate node. If the two bulk transfers use different incoming and outgoing links, these transfers remain essentially independent. So, consider the case when the two bulk transfers compete on one of the incoming or outgoing links. We first discuss when they compete on one of the outgoing links (see Figure 5). If the sum of "coded rates" for individual bulk transfers is less than the outgoing link bandwidth, no flow control is triggered and hence max-min fairness is achieved.

The situation becomes more interesting when the sum of coded rates for individual bulk transfers is greater than the outgoing link bandwidth. In this case, suppose the coded rate of the first bulk transfer is greater than the second one. Then, since outgoing link is shared equally across the two bulk transfers, the buffers for the flows in the first bulk transfer will fill more quickly, leading to triggering the flow control. Thus, flows in the second bulk transfer will reduce the transmission rate finally converging to outgoing link being shared equally across the two coded bulk transfers.

Multi-priority transfers to fill unfilled pipes. Asymmetric link problem, despite our hop-by-hop flow control mechanism, can lead to "unfilled pipes" (Figure 6). Essentially, due to different bulk transfers bottlenecked at different links, no more coded traffic can be pushed into the network despite some links having available bandwidth. CodedBulk fills such unfilled pipes by sending uncoded data; however, to ensure minimal impact on the coded traffic, CodedBulk uses a lower

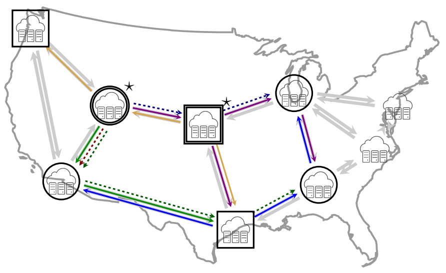


Figure 6: By sending non-coded flows at lower priority (the gray traffic), CodedBulk exploits the "unfilled pipes" left by coded traffic.

priority level for the uncoded data. Thus, CodedBulk uses three priority levels—the highest priority is for interactive traffic, the medium priority for coded traffic, and a lower priority level for uncoded traffic.

3.3 Virtual links

CodedBulk's hop-by-hop flow control mechanism from the previous section addresses the asymmetric link problem, at a design level. In this subsection, we first discuss a challenge introduced by network coding in terms of efficiently implementing the hop-by-hop flow control mechanism. We then introduce the abstraction of virtual links, that enables an efficient realization of CodedBulk's flow control mechanism without any changes in the underlying transport protocol. For this subsection, we use TCP as the underlying congestion control mechanism; however, the idea generalizes to any transport protocol that supports flow control.

The challenge. In traditional store-and-forward networks, implementing hop-by-hop flow control is simple: as data for a flow is received in the Rx buffer, it can be directly copied to the Tx buffer of the next hop, either using blocking or non-blocking system calls. When implementing network coding, this becomes non-trivial—since data from multiple flows needs to be coded together, neither blocking nor non-blocking calls can be used since these calls fundamentally operate on individual flows. For instance, consider the case of Figure 1(f), where a node needs to compute $(A \oplus B) \oplus A$ using packets

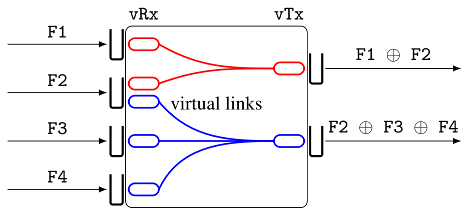


Figure 7: The figure demonstrates the virtual link abstraction used by CodedBulk to implement its hop-by-hop flow control mechanism without any modifications in the underlying network stack.

from the two flows. Blocking calls require expensive coordination between two buffers since the node requires data from *both* flows to be available before it can make progress. Non-blocking calls cannot be used either—the call will return the data from one of the flows, but this data cannot be operated upon until the data from the other flow(s) is also available. The fundamental challenge here is that we need efficient ways to block on multiple flows, and return the call only when data is available in all flows that need to be coded.

It may be tempting to have a shared buffer across different flows that need to be coded together. The problem, however, is that shared buffers will lead to deadlocks [41]—if one of the flows is sending data at much higher rate than the other flows, it will end up saturating the buffer space, the other flows will starve, and consequently the flow that filled up the buffer will also not make progress since it waits to receive data from other flows to be coded with. As discussed in §3.2, non-zero buffer allocation to each individual flow is a necessary condition for avoiding deadlocks in hop-by-hop flow control mechanisms.

Virtual links (see Figure 7). CodedBulk assigns each individual bulk transfer a virtual link per outgoing physical link; each virtual link has a single virtual transmit buffer `vTx` and as many virtual receive buffers `vRx` as the number of flows to be coded together for that outgoing link. For instance, consider four incoming flows in a bulk transfer `F1, F2, F3, F4` such that `F1 ⊕ F2` is forwarded on one of outgoing physical links, and `F2 ⊕ F3 ⊕ F4` is forwarded on another outgoing physical link. Then, CodedBulk creates two virtual links each having one `vTx`; the first virtual link has two `vRx` (one for `F1` packets and another for `F2` packets) and the second virtual link has three `vRx` (one for each of `F2, F3` and `F4` packets). Virtual links are created when the controller installs the network codes, since the knowledge of the precise network code to be used for the bulk transfer is necessary to create virtual links. As new codes are installed, CodedBulk reallocates the space to each `vTx` and `vRx`, within and across virtual links, to ensure that all virtual buffers have non-zero size.

Using these virtual links resolves the aforementioned challenge with blocking and non-blocking calls. Indeed, either of the calls can now be used since the “correlation” between the flows is now captured at the virtual link rather than at the flow control layer. Data from the incoming socket buffers for individual flow is now copied to their respective `vRx` buffers, either using blocking or non-blocking calls. A separate thread asynchronously checks when the size of all the `vRx` buffers is non-zero (each buffer has at least one packet); and when this happens, performs the coding operations and copies the resulting packet to the corresponding `vTx`.

4 Evaluation

We implement CodedBulk in C++ and use TCP Cubic as the underlying transport protocol. We use default TCP socket buffers, with interactive traffic sent at higher priority than bulk transfers (using TCP differentiated services field) set using standard Linux socket API. To enforce priority scheduling, we use Linux tc at each network interface.

We now evaluate CodedBulk implementation over two real geo-distributed cloud testbeds. We start by describing the experiment setup (§4.1). We then discuss the results for CodedBulk implementation over a variety of workloads with varying choice of source and destination nodes for individual bulk transfers, interactive traffic load, number of concurrent bulk transfers, and number of destinations in individual bulk transfers (§4.2). Finally, we present scalability of our CodedBulk prototype implementation in software and hardware (§4.3).

4.1 Setup

Testbed details. To run our experiments, we use two testbeds that are built as an overlay on geo-distributed datacenters from Amazon AWS. Our testbeds use 13 and 9 geo-distributed datacenters organized around B4 [25] and Internet2 [5] topologies, respectively. The datacenter locations are chosen to closely emulate the two topologies and the corresponding geographical distances and latencies. Within each datacenter, we take a high-end server; for every link in the corresponding topology, we establish a connection between the servers across various datacenter using the inter-datacenter connectivity provided by Amazon AWS. To reduce cost of experimentation, we throttle the bandwidth between each pair of servers to 200 Mbps for our experiments. The precise details on the inter-datacenter connectivity provided by Amazon AWS, whether they use public Internet or dedicated inter-datacenter links, is not publicly known. We run all the experiments for each individual figure within a short period of time; while the inter-datacenter links provided by Amazon AWS may be shared and may cause interference, we observe fairly consistent inter-datacenter bandwidth during our experiments. We use a server in one of the datacenters to act as the centralized controller (to compute and install network codes on all servers across our testbed).

Workloads. As mentioned earlier, the benefits of network coding depend on the underlying network topology, the number of destinations in individual bulk transfers, the location of the source and the set of destinations in each bulk transfer, the number of concurrent transfers and interactive traffic load.

While there are no publicly available datasets or workloads for inter-datacenter bulk transfers, several details are known. For instance, Facebook [43], Netflix [1], Azure SQL database [3] and CloudBasic SQL server [4] perform replication to (dynamically) locate their datasets closer to the customers; for such applications, the destinations for each replica are selected based on the diurnal traffic patterns and customer access patterns. Many other applications [13, 19, 23, 25, 31, 50] perform replication levels based on user needs, usually for fault tolerance; for such applications, the choice of destinations may be under the control of the service provider.

We perform experiments to understand the envelope of workloads where CodedBulk provides benefits. Our evaluation performs sensitivity analysis against all parameters—we use two inter-datacenter network topologies, interactive traffic load varying from $0.05-0.2\times$ of the link bandwidth, the number of destinations/replicas in individual bulk transfers varying from 2 to maximum possible (depending on the topology), and the number of concurrent bulk transfers varying from 1 to the maximum possible (depending on the topology). For each setting, we run five experiments; for individual bulk transfers within each experiment, we choose a source uniform randomly across all nodes, and choose the destinations from the remaining nodes. Each node can be the source of only a single bulk transfer but may serve as a destination for other bulk transfers; furthermore, each node may serve as a destination for multiple bulk transfers. We present the average throughput across all experiments, as well as the variance (due to different choices of the source and set of destination across different experiments).

We generate interactive traffic between every pair of datacenters, with arrival times such that the overall load induced by the interactive traffic varies between $0.05-0.2\times$ of the link bandwidth; while 0.2 load is on the higher end in real-world scenarios [23, 25], it allows us to evaluate extreme workloads. Interactive traffic is always assigned the highest priority and hence, all our evaluated schemes will get the same interactive traffic throughput. Our results, thus, focus on bulk traffic throughput.

As mentioned above, there are no publicly available datasets or workloads for inter-datacenter bulk transfers. We make what we believe are sensible choices, state these choices explicitly, and to whatever extent possible, evaluate the sensitivity of these choices on our results. Nonetheless, our results are dependent on these choices, and more experience is needed to confirm whether our results generalize to workloads observed in large-scale deployments.

Evaluated schemes. We compare CodedBulk with three mechanisms for bulk data transfers discussed earlier in Figure 1—single-path, multi-path, and Steiner arborescence packing—each of which take the graph described in §2.1 as an input. For the single-path mechanism, the bulk traffic is transferred along the shortest path between the source and each destination; when multiple choices are available, the mechanism selectively picks paths that minimize total bandwidth usage (*e.g.*, to send bulk traffic to two destinations d_1, d_2, the mechanism prefers the path $s \rightarrow d_1 \rightarrow d_2$, where d_1 can simply forward the data to d_2, over two different paths $s \rightarrow d_1$ and $s \rightarrow v \rightarrow d_2$ for some other node v). The multi-path mechanism selects edge-disjoint paths from the source to each destination so as to greedily minimize the sum of the path lengths. Our third baseline is a state-of-the-art Steiner arborescence based multicast mechanism that allows each node in the network (including the destinations) to forward (Figure 2(a)) and mirror (Figure 2(b)) incoming data. To compute the Steiner arborescence, we use the algorithm in [48] that is also used in other Steiner arborescence based inter-datacenter multicast proposals [38, 39, 40]. We take the arborescence computed by the algorithm, and integrate it with a store-and-forward model, along with TCP for transfers between every pair of nodes in the Steiner arborescence. For concurrent bulk transfers, paths and Steiner arborescence are computed independently for each individual bulk transfer.

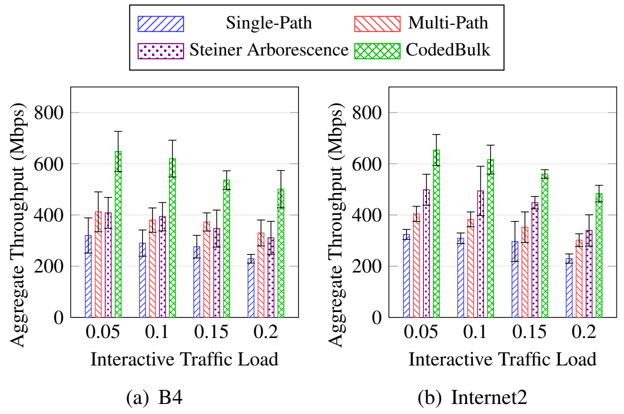


Figure 8: Performance of various bulk transfer mechanisms for varying interactive traffic load. For the B4 topology, CodedBulk improves bulk transfer throughput by $1.9-2.2\times$, $1.4-1.6\times$ and $1.5-1.6\times$ compared to single-path, multi-path, and Steiner arborescence based mechanisms, respectively. For the Internet2 topology, corresponding numbers are $1.9-2.1\times$, $1.6\times$, and $1.2-1.4\times$ (discussion in §4.2).

For CodedBulk, we use a finite field size of 2^8, that is all finite field operations are performed on individual bytes; this finite field size is sufficient for inter-datacenter networks with as many as 128 datacenters. We could have used a smaller finite field size since our topologies are much smaller than real inter-datacenter network topologies; however, this allow us to keep the operations byte aligned, which simplifies CodedBulk software and hardware implementation.

Performance metric. Our primary metric is the aggregate throughput for bulk transfers. For each individual bulk transfer, the throughput is computed as the maximum throughput at which the source can send to *all* destinations. We then calculate the aggregate throughput by summing up the throughput of all bulk transfers.

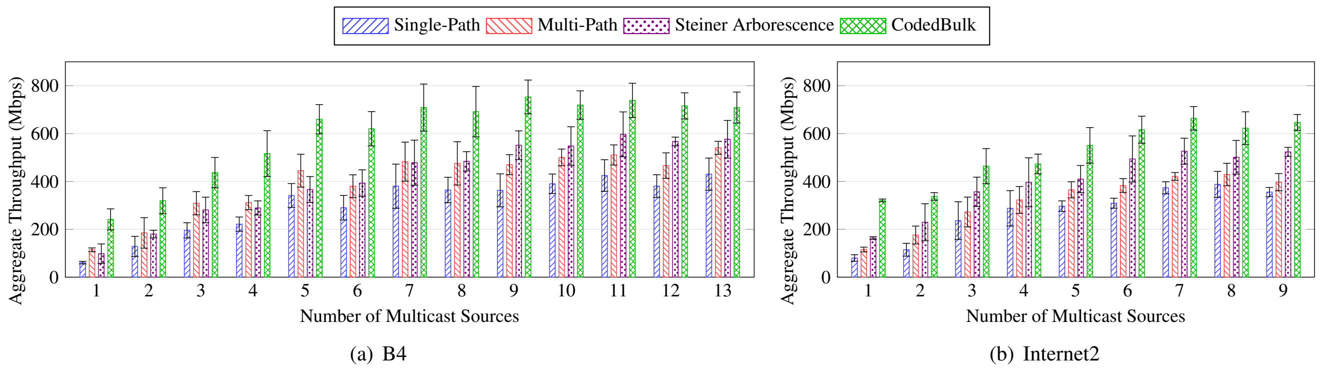


Figure 9: Performance of various bulk transfer mechanisms for varying number of concurrent bulk transfers. CodedBulk improves the bulk transfer throughput by $1.6-4\times$, $1.3-2.8\times$ and $1.2-2.5\times$ when compared to single-path, multi-path, and Steiner arborescence based mechanisms, respectively (discussion in §4.2).

4.2 Geo-distributed Testbed Experiments

We compare CodedBulk with the three baselines for varying interactive traffic loads, varying number of concurrent bulk transfers and varying number of replicas per bulk transfer.

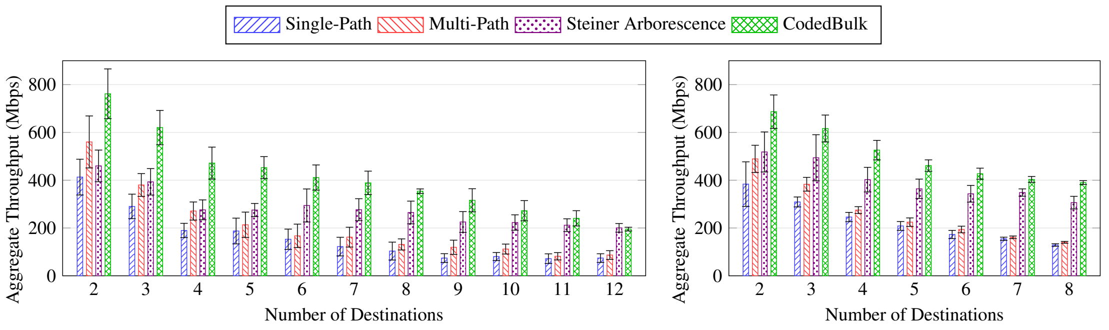
Varying interactive traffic load. Figure 8 presents the achievable throughput for each scheme with varying interactive traffic load. For this experiment, we use 3-way replication and 6 concurrent transfers (to capture the case of Facebook, Netflix, Azure SQL server and CloudBasic SQL server as discussed above), and vary the interactive traffic load from $0.05-0.2\times$ of the link bandwidth.

As expected, the throughput for all mechanisms decreases as interactive traffic load increases. Note that, in corner-case scenarios, the multi-path mechanism can perform slightly worse than single-path mechanism for multiple concurrent bulk transfers due to increased interference across multiple flows sharing a link, which in turn results in increased convergence time for TCP (see [46] for a concrete example). Overall, CodedBulk improves the bulk traffic throughput over single-path, multi-path and Steiner arborescence mechanisms by $1.9-2.2\times$, $1.4-1.6\times$ and $1.2-1.6\times$, respectively, depending on the interactive traffic load and the network topology. Single-path mechanisms perform poorly because they do not exploit all the available bandwidth in the network. Both multi-path and Steiner arborescence based mechanisms exploit the available bandwidth as much as possible. However, multi-path mechanisms suffer since they do not allow intermediate nodes to mirror and forward to the destinations. Steiner arborescence further improves upon multi-path mechanisms by allowing intermediate nodes to mirror and forward data, but they suffer due to approximation algorithm often leading to suboptimal solutions. CodedBulk's gains over multi-path and Steiner arborescence mechanisms are, thus, primarily due to CodedBulk's efficient realization of network coding—it not only uses all the available links, but also computes the optimal coding strategy (unlike Steiner arborescence mechanism that uses an approximation algorithm). The Steiner arborescence mechanism performs better on Internet2 topology because of its sparsity—fewer links in the network means a Steiner arborescence solution is more likely to be the same as network coding solution due to fewer opportunities to perform coding. Nevertheless, CodedBulk outperforms Steiner arborescence based mechanism by $1.4\times$.

Varying number of concurrent bulk transfers. Figure 9 shows the performance of the four mechanisms with varying number of concurrent transfers. For this evaluation, we use the same setup as earlier—3-way replication, multiple runs with each run selecting different sources and set of destinations, etc.—with the only difference being that we fix the interactive traffic load to 0.1 and vary the number of concurrent bulk transfers. With larger number of concurrent bulk transfers, Steiner arborescence mechanisms slightly outperform multi-path due to improved arborescence construction. Nevertheless, CodedBulk provides benefits across all sets of experiments, achieving $1.2-2.5\times$ improvements over Steiner arborescence based mechanisms. The gains are more prominent for B4 topology and for fewer number of concurrent transfers, since CodedBulk gets more opportunities to perform network coding at intermediate nodes in these scenarios.

Varying number of destinations/replicas per bulk transfer. Figure 10 shows the performance of the four mechanisms with varying number of destinations/replicas for individual bulk transfers. For this evaluation, we use the same setup as Figure 8—6 concurrent bulk transfers, multiple runs with each run selecting different sources and set of destinations, etc.—with the only difference being that we fix the interactive traffic load to 0.1 and vary the number of destinations/replicas per bulk transfer from 2 to the maximum allowable replicas for individual topologies. Notice the results show the aggregate throughput per destination.

As the number of destinations per bulk transfer increases, the per-destination throughput decreases for all schemes (although, as expected, the sum of throughput of all destinations increases). Note that multi-path outperforming single-path



(a) B4 (b) Internet2

Figure 10: Performance of various bulk transfer mechanisms for varying number of destinations/replicas per bulk transfer. CodedBulk improves the bulk transfer throughput over single-path and multi-path mechanisms by $1.8-4.3\times$ and $1.4-2.9\times$, respectively, depending on the number of destinations in each bulk transfer and depending on the topology. CodedBulk outperforms Steiner arborescence mechanisms by up to $1.7\times$ when the number of destinations is not too large. When each bulk transfer creates as many replicas as the number of datacenters in the network, CodedBulk performs comparably with Steiner arborescence. Note that the aggregate bulk throughput reduction is merely because each source is transmitting to increasingly many destinations, but the metric only captures the average throughput per destination. Discussion in §4.2.

and Steiner arborescence based mechanism in Figure 10(a) is primarily due to B4 topology being dense, thus providing enough path diversity to offset the benefits of approximate Steiner arborescence construction. Figure 10(a) and 10(b) show that CodedBulk outperforms single-path and multi-path mechanisms by $1.8-4.3\times$ and $1.4-2.9\times$, depending on the number of destinations and on the topology; moreover, the relative gains of CodedBulk improve as number of destinations increases. The comparison with Steiner arborescence based mechanism is more nuanced. CodedBulk achieves improved performance when compared to Steiner arborescence based mechanism when number of destinations is less than 10 for B4 topology, and less than 6 for Internet2 topology. The performance difference is minimal for larger number of destination. The reason is that for larger number of replicas/destinations, each source is multicasting to almost all other nodes in the network; in such cases, the benefits of coding reduce when compared to forwarding and mirroring of data at intermediate nodes and at the destination nodes as in Steiner arborescence based mechanism. Thus, the benefits of CodedBulk may be more prominent when the number of replicas is a bit smaller than the total number of datacenters in the network.

4.3 Microbenchmarks

We now evaluate CodedBulk performance in terms of scalability of its software and hardware implementations. Our goal here is to demonstrate the feasibility of CodedBulk implementation; deployment of CodedBulk in large-scale systems would typically require much more optimized implementation since the traffic volume is expected to be much higher.

Software implementation. CodedBulk software implementation runs on commodity datacenter servers, performing network coding as discussed in §3. Figure 11 shows the scalability of CodedBulk software implementation. We observe that CodedBulk implementation scales well with number of cores,

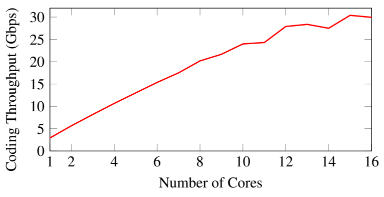


Figure 11: CodedBulk implementation performs network coding for as much as 31Gbps worth of traffic using a commodity 16 core server, achieving roughly linearly coding throughput scalability with number of cores.

Element	Used	Available	Utilization
LUT	69052	433200	15.94%
BRAM	1365	1470	92.86%

Table 1: Resource utilization of CodedBulk implementation on Xilinx Virtex-7 XC7VX690T FPGA (250 MHz clock). Our implementation provides up to 31.25 Gbps throughput with 15.94% LUTs and 92.86% BRAMs. No DSP is needed in our design.

with a single 16-core server being able to perform network coding at line rate for as much as 31Gbps worth of traffic.

Hardware implementation. We have synthesized an end-to-end CodedBulk implementation on an FPGA. For our CodedBulk hardware implementation, we had two choices. First, we could implement a finite field engine that performs finite field operations during the network coding process; or second, we could precompute and store finite field operation results, and use a simple look up table while performing network coding operations. The first approach requires multiple clock cycles to encode two bytes from two different packets; the second approach trades off BRAM to save cycles during coding operations. Since CodedBulk uses a small finite field size (2^8), the second approach offers a better tradeoff — it requires just 256×256 byte look up table per 16 bytes for individual operations to complete in one cycle. We replicate the lookup table accordingly to perform network coding for all bytes in an MTU-sized packet within a single clock cycle. Table 1 shows the results for CodedBulk hardware implementation on Xilinx Virtex-7 XC7VX690T FPGA, which offers 100, 200, and 250 MHz fabric clocks. With respect to the clocks, our FPGA-based codec can achieve throughput 12.5, 25, and 31.25 Gbps. Without needing any DSP, our hardware design consumes 92.86% BRAMs and only 15.94% LUTs.

We believe that trading off compute and storage resources to improve inter-datacenter bulk transfer throughput is a favorable tradeoff to make. However, more experience from industry is needed to do a thorough cost/benefit analysis.

5 Related Work

We have already discussed the differences between CodedBulk's goals and the traditional multicast problem in ISP networks; it would be futile to attempt to summarize the vast amount of literature from ISP multicast problem. We compare and contrast CodedBulk with two more closely related key areas of research: inter-datacenter WAN transfers, and network coding applications in computer networks.

Inter-datacenter bulk transfers. There has been significant amount of recent work on optimizing inter-datacenter bulk transfers [26, 27, 31, 32, 38, 39, 40, 49, 52]. These works optimize inter-datacenter bulk transfers along a multitude of performance metrics, including improving flow completion time [27, 38, 39, 40, 49, 52], and throughput for bulk transfers [26, 31, 32]. CodedBulk's goals are aligned more closely with the latter, and are complementary to the former—CodedBulk improves the throughput for bulk transfers; any bulk transfer scheduling mechanism can be used on top of CodedBulk to meet the needs for timely transfers.

As discussed earlier, the state-of-the-art approach for high-throughput inter-datacenter bulk transfers are based on packing of Steiner arborescence: here, each intermediate node as well as destination nodes are allowed to forward and mirror data toward other destination nodes. Several recent inter-datacenter bulk transfer proposals [38, 39, 40] are based on this approach. Our evaluation in §4 shows that CodedBulk achieves throughput improvements over state-of-the-art Steiner arborescence based mechanisms in a wide variety of scenarios. This is because all prior techniques are limited by network capacity, and by limitations of existing non-coded techniques to achieve this capacity.

CodedBulk, by using network coding, achieves improvement in throughput for bulk transfers by trading off a small amount of compute and storage resources.

Network coding in computer networks. Network coding has successfully been applied to achieve higher throughput in wireless networks [20, 29], in TCP-based networks [44], in content distribution [17, 35, 36], in peer-to-peer communication [16], to name a few; please see [15] for additional applications of network coding. Our goals are complementary—enabling network coding for high-throughput inter-datacenter WAN bulk transfers by exploiting the unique characteristics of these networks. Throughout the paper, we have outlined the unique challenges introduced by applications of network coding in wired networks, and how CodedBulk overcomes these challenges. Our design can be applied to any of the applications where network coding is useful.

Network code construction algorithms. Early incarnations of network coding solutions used a technique referred to as random linear network coding [9, 22]. These random linear network codes have the benefit of being independent of the network topology. However, they have high implementation cost: they require complex operations at intermediate nodes (due to computations over large finite field sizes and due to requiring additional packet header processing). In addition, realizing random linear network codes in practice also requires changes in packet header format. Follow-up research has led to efficient construction of network codes [24]—for a bulk transfer to T destinations, it suffices for intermediate nodes to perform computations over a finite field of size at most $2|T|$; if the min-cut is h, the complexity of computations at the source and at the destination are $O(h)$ and $O(h^2)$, respectively. In §4.1, we discussed how at the inter-datacenter WAN scale, these computations entail simple and efficient byte-level XOR operations. Furthermore, these codes can be realized without any changes in the packet header format. CodedBulk, thus, uses the network code construction algorithm of [24].

6 Conclusion

We have presented the design, implementation and evaluation of CodedBulk, an end-to-end system for high-throughput inter-datacenter bulk transfers. CodedBulk uses network coding, a technique that guarantees optimal throughput for individual bulk transfers. To achieve this, CodedBulk resolves the many pragmatic and fundamental barriers faced in the past in realizing the benefits of network coding in wired networks. Using an end-to-end implementation of CodedBulk over a geo-distributed inter-datacenter network testbed, we have shown that CodedBulk improves throughput for inter-datacenter bulk transfers by $1.2-2.5\times$ when compared to state-of-the-art mechanisms that do not perform coding.

Acknowledgments

We would like to thank our shepherd Keith Winstein, anonymous NSDI reviewers, Rashmi Vinayak and Robert Kleinberg for their insightful feedback. We would also like to thank the students in Cornell graduate computer networking class (CS6450) for many helpful discussions during the early parts of this project. This work was supported in part by a Google PhD fellowship, NSF 1704742, ONR N00014-17-1-2419, and gifts from Snowflake and Cornell-Princeton Network Programming Initiative.

References

[1] [ARC 305] How Netflix leverages multiple regions to increase availability. https://tinyurl.com/4mac28jy.

[2] Cisco annual Internet report (2018–2023) white paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3] Creating and using active geo-replication – Azure SQL database. https://docs.microsoft.com/en-us/azure/azure-sql/database/active-geo-replication-overview.

[4] Geo-replication/multi-AR. http://cloudbasic.net/documentation/geo-replication-active/.

[5] The Internet2 network. https://internet2.edu/.

[6] Mapping Netflix: Content delivery network spans 233 sites. http://datacenterfrontier.com/mapping-netflix-content-delivery-network/.

[7] Steiner tree problem. https://en.wikipedia.org/wiki/Steiner_tree_problem.

[8] Using replication across multiple data centers. https://docs.oracle.com/cd/E19528-01/819-0992/6n3cn7p3l/index.html.

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

[10] M. Braverman, S. Garg, and A. Schvartzman. Coding in undirected graphs is either very helpful or not helpful at all. In *ITCS*, 2017.

[11] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

[12] J. Cheriyan and M. R. Salavatipour. Hardness and approximation results for packing steiner trees. *Algorithmica*, 45(1):21–43, 2006.

[13] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3):8, 2013.

[14] R.-J. Essiambre and R. W. Tkach. Capacity trends and limits of optical communication networks. *Proc. IEEE*, 100(5):1035–1055, 2012.

[15] C. Fragouli, J.-Y. Le Boudec, and J. Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68, 2006.

[16] C. Gkantsidis, J. Miller, and P. Rodriguez. Comprehensive view of a live network coding p2p system. In *IMC*, 2006.

[17] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. In *INFOCOM*, 2005.

[18] M. X. Goemans and Y.-S. Myung. A catalog of steiner tree formulations. *Networks*, 23(1):19–28, 1993.

[19] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. G. Dhoot, A. R. Kumar, A. Agiwal, S. Bansali, M. Hong, J. Cameron, M. Siddiqi, D. Jones, J. Shute, A. Gubarev, S. Venkataraman, and D. Agrawal. Mesa: Geo-replicated, near real-time, scalable data warehousing. *VLDB*, 2014.

[20] J. Hansen, D. E. Lucani, J. Krigslund, M. Médard, and F. H. Fitzek. Network coded software defined networking: Enabling 5G transmission and storage networks. *IEEE Communications Magazine*, 53(9):100–107, 2015.

[21] M. Hauptmann and M. Karpiński. *A compendium on Steiner tree problems*. Inst. für Informatik, 2013.

[22] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[23] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[24] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. M. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6):1973–1982, 2005.

[25] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[26] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford. Optimizing bulk transfers with software-defined optical WAN. In *SIGCOMM*, 2016.

[27] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula. Calendaring for wide area networks. In *SIGCOMM*, 2014.

[28] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: Analog network coding. In *SIGCOMM*, 2012.

[29] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: Practical wireless network coding. In *SIGCOMM*, 2006.

[30] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Information Theory*, 11(5):782–795, 2003.

[31] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with NetStitcher. In *SIGCOMM*, 2011.

[32] N. Laoutaris, G. Smaragdakis, R. Stanojevic, P. Rodriguez, and R. Sundaram. Delay tolerant bulk data transfers on the Internet. *IEEE/ACM Transactions on Networking*, 21(6):1852–1865, 2013.

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

[34] Z. Li, B. Li, and L. C. Lau. A constant bound on throughput improvement of multicast network coding in undirected networks. *IEEE Transactions on Information Theory*, 55(3):1016–1026, 2009.

[35] Z. Liu, C. Wu, B. Li, and S. Zhao. UUSee: Large-scale operational on-demand streaming with random network coding. In *INFOCOM*, 2010.

[36] E. Magli, M. Wang, P. Frossard, and A. Markopoulou. Network coding meets multimedia: A review. *IEEE Transactions on Multimedia*, 15(5):1195–1212, 2013.

[37] P. P. Mishra and H. Kanakia. A hop by hop rate-based congestion control scheme. In *SIGCOMM*, 1992.

[38] M. Noormohammadpour, S. Kandula, C. S. Raghavendra, and S. Rao. Efficient inter-datacenter bulk transfers with mixed completion time objectives. *Computer Networks*, 164:106903, 2019.

[39] M. Noormohammadpour, C. S. Raghavendra, S. Kandula, and S. Rao. QuickCast: Fast and efficient interdatacenter transfers using forwarding tree cohorts. In *INFOCOM*, 2018.

[40] M. Noormohammadpour, C. S. Raghavendra, S. Rao, and S. Kandula. DCCast: Efficient point to multipoint transfers across datacenters. In *HotCloud*, 2017.

[41] C. Özveren, R. Simcoe, and G. Varghese. Reliable and efficient hop-by-hop flow control. In *SIGCOMM*, 1994.

[42] G. Ramamurthy and B. Sengupta. A predictive hop-by-hop congestion control policy for high speed networks. In *INFOCOM*, 1993.

[43] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM*, 2015.

[44] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros. Network coding meets TCP. In *INFOCOM*, 2009.

[45] A. S. Tanenbaum and D. Wetherall. *Computer Networks, 5th Edition*. Pearson, 2011.

[46] S.-H. Tseng, S. Agarwal, R. Agarwal, H. Ballani, and A. Tang. Codedbulk: Inter-datacenter bulk transfers using network coding. Technical report, https://github.com/SynergyLab-Cornell/codedbulk.

[47] M. Vuppalapati, J. Miron, R. Agarwal, D. Truong, A. Motivala, and T. Cruanes. Building an elastic query engine on disaggregated storage. In *NSDI*, 2020.

[48] D. Watel and M.-A. Weisser. A practical greedy approximation for the directed Steiner tree problem. *Journal of Combinatorial Optimization*, 32(4):1327–1370, 2016.

[49] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. C. Lau. Orchestrating bulk data transfers across geo-distributed datacenters. *IEEE Transactions on Cloud Computing*, 5(1):112–125, 2017.

[50] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. SPANStore: Cost-effective geo-replicated storage spanning multiple cloud services. In *SOSP*, 2013.

[51] Y. Yi and S. Shakkottai. Hop-by-hop congestion control over a wireless multi-hop network. *IEEE/ACM Transactions on Networking*, 15(1):133–144, 2007.

[52] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang. Guaranteeing deadlines for inter-data center transfers. *IEEE/ACM Transactions on Networking*, 25(1):579–595, 2017.