Airlift: Video Conferencing as a Cloud Service using Inter-Datacenter Networks

Yuan Feng, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

Bo Li
Department of Computer Science
Hong Kong University of Science and Technology

***Abstract*—It is typical for enterprises to rely on services from cloud providers in order to build a scalable platform with abundant available resources to satisfy user demand, and for cloud providers to deploy a number of datacenters inter-connected with high-capacity links, across different geographical regions. In this paper, we propose that video conferencing, even with its stringent delay constraints, should also be provided as a *cloud service*, taking full advantage of the inter-datacenter network in the cloud. We design *Airlift*, a new protocol designed for the inter-datacenter network, tailored to the needs of a cloud-based video conferencing service. *Airlift* delivers packets in live video conferences to their respective destination datacenters, with the objective of maximizing the total throughput across all conferences, yet without violating end-to-end delay constraints. In order to simplify our protocol design in *Airlift*, we use intra-session network coding and the concept of conceptual flows, such that the optimization problem that can be conveniently formulated as a linear program. Our real-world implementation of *Airlift* has been deployed over the Amazon EC2 cloud. We show that *Airlift* delivers a substantial performance advantage over state-of-the-art peer-to-peer solutions.**

I. Introduction

The gist of cloud computing is to maximize the sharing of resources with statistical multiplexing, while keeping users of the cloud satisfied. To provide cloud services with a higher quality, it is customary for cloud providers to deploy a number of datacenters across different geographical regions, inter-connected with high-capacity links. Enterprises, such as Netflix, are moving their entire platform to the cloud [1] to take advantage of its abundant resources that are available on demand.

From the perspective of both bandwidth demand and end-to-end delay constraints, multi-party video conferencing may be one of the most demanding multimedia applications. Existing conferencing solutions in the literature have traditionally focused on the use of peer-to-peer (P2P) [2], [3] or client-server architectures (*e.g.*, Microsoft Lync). With abundant bandwidth between datacenters, one would naturally wonder if it is feasible to take full advantage of inter-datacenter networks in the cloud to support higher bit rates in video conferences, yet still maintaining acceptable delays.

In this paper, we promote the use of inter-datacenter networks to support live multi-party video conferencing as a cloud service. Our protocol and real-world implementation, collectively referred to as *Airlift*, is designed from the ground up to support multiple live conferences with an inter-datacenter network operated by a cloud provider. As its name suggests, the unique advantage of *Airlift* is to provide low-latency end-to-end paths among participants in multiple conferences, yet without the "hustle and bustle" of the public Internet. With *Airlift*, packets in conferences can be routed through a high-capacity inter-datacenter network, as if they are traveling around the world in chartered private flights with minimal congestion, rather than cruise ships with long lines waiting for embarkation.

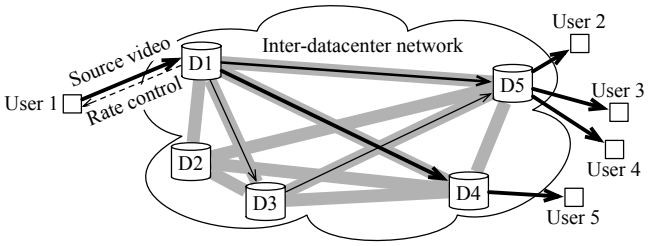


Fig. 1. Packets from User 1 in a 5-party conference are being transmitted in an inter-datacenter network with 5 datacenters, via a combination of direct paths (*e.g.*, $D1 \rightarrow D4$) and relay paths (*e.g.*, $D1 \rightarrow D3 \rightarrow D5$).

A highlight of this paper is our design of a new application-layer protocol for inter-datacenter networks. Its original features are two-fold: *First*, to be more scalable, it aggregates user-initiated conferences to a smaller number of multicast sessions among datacenters. *Second*, it is designed to *maximize the total throughput* across all the sessions, while maintaining basic fairness across different conferences, and making sure that stringent delay constraints are not violated.

Due to the multicast nature of aggregated sessions, traditional wisdom resorts to *Steiner tree packing* [3] in order to maximize the video flow rate from a single source to the remaining participants in a video conference. Since the problem is NP-Complete, existing works [3], [4] pack only depth-1 and depth-2 trees. With a large number of conferences served concurrently in an inter-datacenter network, packing Steiner trees for each source and in each conference is computationally prohibitive, even with trees of limited depth. To solve this problem, we use *intra-session network coding* as an integral part in both our protocol design and our real-world implementation. Thinking from the perspective of *conceptual flows* [5], the upshot of network coding is its power of resolving conflicts competing for bandwidth resources in

This research was supported in part by the SAVI NSERC Strategic Networks Grant, by a grant from NSFC/RGC under the contract N_HKUST610/11, by grants from HKUST under the contract RPC11EG29, SRFI11EG17-C and SBI09/10.EG01-C, by a grant from ChinaCache Int. Corp. under the contract CCNT12EG01.

bottleneck links. With the help of network coding, we are now able to formulate the problem of maximizing the total throughput across all aggregated sessions as a *linear program*, easily solvable using a standard LP solver. Its optimal solution serves as the foundation of the *Airlift* protocol.

Finally, using *Airlift* as a video conferencing cloud service is *simple*. In our design, the cloud service can be treated as a full-service broker: a participating user with a video source in a conference only needs to transmit its packets to one of the datacenters in the cloud, and to process acknowledgments from the cloud service. Fig. 1 shows an illustrative example of a 5-party video conference, supported by *Airlift*.

We evaluate the validity and performance of *Airlift* as a cloud service with our real-world implementation, with 17,000 lines of code in C++. Our implementation has been developed with performance in mind: to maximize packet processing rates, asynchronous networking has been used; to minimize the computational overhead of network coding, our network coding implementation is accelerated with the Intel/AMD SSE2 instruction set.

Our real-world experimental results over PlanetLab and the Amazon EC2 cloud have shown substantially (3 to 24 times) higher throughput as compared to *Celerity* [3], a state-of-the-art P2P solution, yet without any disadvantage on end-to-end delays that can be perceptible to end users. To the best of our knowledge, this paper presents the first design and implementation of a cloud-based solution for video conferencing.

The remainder of this paper is organized as follows. In Sec. II, we motivate the *Airlift* cloud service and discuss our design objectives and choices. In Sec. III, we present our analytical study on maximizing the total throughput in an inter-datacenter network, which serves as the foundation of our protocol design. In Sec. IV, we present details of our protocol, designed based on results from our theoretical analysis. In Sec. V, we present our real-world implementation of *Airlift*, and evaluate its validity and performance in the Amazon EC2 cloud. We discuss related work and conclude the paper in Sec. VI and Sec. VII, respectively.

II. Airlift: Motivation and Design Objectives

A. Conferencing via the Cloud: Motivation

The Achilles' heel of peer-to-peer video conferencing solutions, such as Celerity [3], is the challenge of computing the flow rate on each overlay link between users who participate in the same conference. Such a challenge comes from the fact that overlay links may compete for the same physical link in the layer-3 Internet topology; yet due to the lack of complete knowledge about the underlying layer-3 topology, it would be infeasible to determine how overlay links share common physical links.

Such a challenge is present even in a simple "dumbbell" topology, illustrated in Fig. 2. Without the knowledge that a physical bottleneck exists between the two user pairs, the number of overlay links competing for the bottleneck may not be optimal, if incorrect trees are formed to route packets. To address such a challenge, Celerity resorts to a complex mix of algorithms, including decentralized optimization to converge to optimal overlay link rates based on loss rates and queueing delays, as well as spanning tree packing at each source to compute its overlay trees.

In contrast, if we take advantage of the high-capacity inter-datacenter network in the cloud, the pairs of users on both sides of the dumbbell topology can each transmit to their respective datacenters, as Fig. 3 shows, with user 1 as an example. Each datacenter is responsible for aggregating incoming video flows from both users, and forwarding them to the other datacenter. With such aggregation, the number of video flows sharing the bottleneck, which resides between the two datacenters in the cloud, is naturally minimized, without the complexity of Celerity. If the inter-datacenter link has a higher capacity (*e.g.,* due to private peering relationships), the gain on the total throughput in the conference is even more substantial.

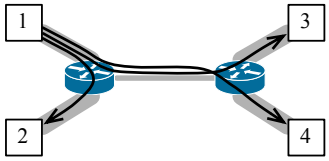


Fig. 2. With P2P conferencing, more than the minimum number of overlay links may compete for the same physical bottleneck.

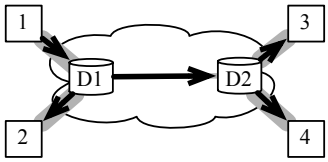


Fig. 3. Conferencing via the cloud: flow rates over the physical bottleneck are aggregated and minimized.

But are end-to-end delays sacrificed by routing packets through the cloud? We answer this question with results from real-world experiments, using PlanetLab nodes as video sources. Table I shows the measured throughput values and end-to-end delays between three pairs of conference participants with diverse geographic locations, comparing the performance of a P2P overlay link with that of routing through the Amazon EC2 cloud. In the latter case, each participating user connects to its closest datacenter in the EC2 cloud, forming a three-hop path. For example, users in Beijing and Seoul will connect to the datacenter located in Japan, and users in Cambridge, UK and Moscow will connect to the datacenter in Ireland. As is self-explanatory in the table, when routed through the cloud, all three pairs have enjoyed higher throughput values (substantially higher in two of the three pairs), yet this is achieved with similar or even shorter end-to-end delays, as compared to the overlay link in a P2P solution.

TABLE I
Conferencing with P2P overlays or via the cloud? A comparison of throughput and end-to-end delay.

Cloud/P2P	Throughput (Mbps)	Delay (msec)
Toronto-Beijing	2.202/0.179	171.6/148.8
Cambridge-Sao Paulo	1.687/1.432	103.4/204.6
Seoul-Moscow	7.189/1.103	201.7/436.9

We will revisit such a comparison between *Airlift* and Celerity with a more elaborate set of experiments in Sec. V. Suffice it to say, there exists a clear performance advantage to provide video conferencing as a cloud service.

B. Airlift: Design Objectives and Choices

Towards the design of a new application-layer protocol in the inter-datacenter network, we target a number of important objectives.

Performance. The best possible incentive that can be touted to attract users to establish video conferences using *Airlift* is its superior performance, with respect to higher video flow rates from each of the participants in a live conference, while still maintaining an acceptable end-to-end delay. *Airlift* should, first and foremost, be designed with performance in mind.

Simplicity. As a cloud service, *Airlift* should be conceptually *simple* to use, and work as a *full-service broker*. A participating user in a conference should only need to connect to the "cloud," and to start transmitting packets from its video source after a connection is established. The "cloud" should provide informative feedback to the user as packets arrive, so that the user can adequately increase or throttle its video flow rate by varying parameters of its video codec. In this sense, as long as a packet is acknowledged by the "cloud," the user will have complete "peace of mind" that the packet will be delivered *intact* and *on time* to other participants in the conference, subject to a typical end-to-end delay constraint.

But what is the "cloud" that a participating user should connect to? Our design in *Airlift* has intentionally left the decision *open* with respect to which datacenter that a user should connect to, as existing work has already covered this complementary problem quite well. It is typical to select an appropriate datacenter by taking advantage of the customized IP address returned by DNS servers. Alternatively, users can outsource datacenter selection to third parties [6], with customizable mapping policies. Since video conferencing is sensitive to end-to-end delays, the recommended mapping policy is to choose the "closest" datacenter with respect to delay, using any of the existing selection protocols that can be tailored to consider client proximity (*e.g.,* [6]).

Scalability. Datacenters operated by a cloud provider are often inter-connected with high-capacity links. As such, each inter-datacenter link may be able to carry thousands of video flows from different sources and in different conferences simultaneously. This brings the challenge of *scalability* to the spotlight, in that any online algorithm in the *Airlift* protocol needs to complete its computation in real-time, so that a large number of conferences can be routed through the inter-datacenter network efficiently and without much fanfare.

To be more scalable, we believe that all the video flows from different participants — in their respective conferences — need to be *aggregated*, provided that these participants connect to the same *source* datacenter, and are destined to the same *subset* of *destination* datacenters, which, in turn, are responsible for delivering them to all other participants in each of the conferences. To put it simply, we wish to aggregate all the video flows routed through the same source datacenter and transmitted to the same subset of destination datacenters, regardless of which conference they belong to. Each of these *aggregated sessions* is inherently a *multicast session* in the inter-datacenter network.

Considering only aggregated sessions, rather than individual conferences that use the cloud as a service, makes *Airlift* much more scalable. For example, in order to maximize the total throughput of all conferences routed through the inter-datacenter network, we only need to maximize the total throughput across all aggregated sessions in our problem formulation, with a significantly reduced number of variables that need to be determined. To be more precise, in an inter-datacenter network with N datacenters, the maximum number of aggregated sessions is $\sum_{i=2}^{N}[i \cdot \binom{N}{i}]$. With 7 datacenters in the Amazon EC2 cloud, the maximum number of simultaneous aggregated sessions is only 441, which may be an order of magnitude smaller than the total number of participants in all the concurrent conferences routed through the *Airlift* cloud service.

III. Maximizing Total Throughput in the Cloud

In a nutshell, a key idea in the design of *Airlift* is to take full advantage of the available inter-datacenter capacity in the cloud, so that the total throughput across all conferences is maximized, subject to delay and fairness constraints. We precede our protocol design with a theoretical formulation of this problem.

A. Feasible Paths Satisfying a Delay Bound

Let us consider an inter-datacenter network with multiple datacenters that are geographically distributed around the world, operated by the same cloud provider. These datacenters form a complete directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ indicates the set of datacenters, and $\mathcal{E}$ indicates the set of directed edges inter-connecting them. For each directed edge $e \in \mathcal{E}$, we use a positive real-valued function $C(e)$ to denote its available capacity, which is the maximum available rate of packet transmission on e.

We use S^i to denote the source datacenter in an *aggregated session* i, and $R^i_j, j = 1, 2, \ldots, k_i$ to denote the set of k_i destination datacenters in session i. If we overlook fairness concerns for a moment, our objective is to maximize the total throughput of all the aggregated sessions in $\mathcal{G}$, as long as the end-to-end delays from S^i to each of $R^i_j, j = 1, \ldots, k_i$ are acceptable, *i.e.,* they do not violate a certain delay bound, $D_{\max}$.

Let us now examine such a delay constraint with a microscope. Each directed edge e in $\mathcal{E}$ has a corresponding propagation delay, $d(e)$, which is readily measurable in practice. Assuming that queueing delays on a relaying datacenter are minimal with the use of small buffers, the end-to-end delay from S^i to each R^i_j can be estimated as the sum of all propagation delays, on each of the edges along the acyclic *path* that packets follow. Considering our objective of maximizing the total throughput of all aggregated sessions as a variant of the maximum flow problem, it is conceivable that packets from S^i to each R^i_j may need to follow *multiple* acyclic paths, rather than a single one. We need to make sure that the end-to-end delay on *any* of these acyclic paths does not violate

the delay bound that we impose; in other words, we need to *exclude* paths that violate such a bound, and only consider the set of *feasible paths* — denoted by $\mathcal{P}_j^i$ — that do not. More formally:

$$\mathcal{P}_j^i = \{p \mid p \text{ is an acyclic path from } S^i \text{ to } R_j^i \text{ s.t. } \sum_{e \in p} d(e) \leq D_{\max}\}.$$

Given the inter-datacenter graph $\mathcal{G}$ and the delay bound $D_{\max}$, one can easily find the set of all feasible paths $\mathcal{P}_j^i$ from S^i to R_j^i using a simple variant of the depth-first search algorithm, where the search only continues if, with the path obtained so far, there are no cycles and the delay bound $D_{\max}$ has not yet been violated. In our subsequent formulation of the problem, we have the convenience of only considering the set of feasible paths $\mathcal{P}_j^i$.

B. The Problem of Maximizing Total Throughput

On the surface, it appears that the problem of maximizing the total throughput of all aggregated sessions in $\mathcal{G}$ corresponds to the traditional multi-commodity maximum flow problem. Unfortunately, this is not the case, simply because aggregated sessions[1] are aggregated *multicast* sessions by nature, rather than *unicast* sessions between source-destination pairs as in the multi-commodity maximum flow problem. In essence, packet transmission in a multicast session is more efficient than in multiple unicast sessions, due to the ability for a datacenter to replicate and forward packets to its downstream datacenters in a multicast tree.

To maximize the throughput of a multicast session in $\mathcal{G}$, traditional wisdom resorts to *Steiner tree packing* [3]. As an NP-Complete problem, Steiner tree packing seeks to find the maximum number of pairwise edge-disjoint Steiner trees, in each of which the datacenters involved in the session remain connected. To reduce its complexity, existing work on P2P video conferencing [3] packs only depth-1 and depth-2 trees. However, packing Steiner trees within each session is still computationally prohibitive, due to the large number of concurrent sessions.

Fortunately, the concept of *network coding* provides us with a way out of the woods. Having been studied extensively in the past decade, *network coding* [7] extends the capabilities of nodes in a network session: from basic forwarding (as in the maximum flow problem) and replication (as in multicast), to coding in Galois fields. For a multicast session in any directed acyclic graph, if a rate x can be achieved from the source to each of the destinations independently, it can also be achieved for the entire multicast session [7]. In other words, network coding has the power of resolving the competition among different source-destination pairs for edge capacities. To take advantage of such power, Li *et al.* [5] introduced the concept of *conceptual flows*, defined as network flows that co-exist in the network without contending for edge capacities if they are destined to different destinations, each of which is from a source to a destination, transmitted in a coded form.

To our surprise, inspired by [5], the idea of conceptual flows allows us to formulate the problem of maximizing total throughput as the following *linear program*, which can be solved by a standard LP solver:

$$\begin{aligned}
\max \quad & \mathcal{X} \\
\text{s.t.} \quad & \mathcal{X} \leq \sum_{p \in \mathcal{P}_j^i} x_j^i(p)/w_i, \, \forall i, \, j = 1, \ldots, k_i && (1) \\
& \sum_{p \in \mathcal{P}_j^i(e)} x_j^i(p) \leq x^i(e), \, \forall i, \, j = 1, \ldots, k_i && (2) \\
& \sum_i x^i(e) \leq C(e), \, \forall e \in \mathcal{E} && (3) \\
& x_j^i(p) \geq 0, \, x^i(e) \geq 0, \, \mathcal{X} \geq 0, \\
& \forall p \in \mathcal{P}_j^i, \, \forall i, \, j = 1, \ldots, k_i, \, \forall e \in \mathcal{E}. && (4)
\end{aligned}$$

The objective of this linear program is to maximize the total throughput, which is the sum of flow rates in all the multicast sessions, $\mathcal{X}$. In each session, its flow rate is the minimum of the flow rates that can be *independently* achieved from the source to each of the destinations in the session. In constraint (1), w_i is used to provide weighted proportional fairness across different sessions, and $x_j^i(p)$ represents the conceptual flow rate from S^i to R_j^i, along an acyclic path p in the set of feasible paths $\mathcal{P}_j^i$. Since the flow rate is specified along a particular path p, the flow conservation constraint for a conceptual flow is implicitly satisfied.

Since conceptual flows destined to different destinations within the same session do not compete with one another for edge capacity, the effective flow rate within a session i on edge e is $x^i(e) = \max_j \sum_{p \in \mathcal{P}_j^i(e)} x_j^i(p)$, where $\mathcal{P}_j^i(e)$ represents the set of paths in $\mathcal{P}_j^i$ that uses edge e. Since the $\max$ function is not linear, this constraint is relaxed to constraint (2). Finally, constraint (3) reflects the fact that the summation of the effective flow rates of different sessions should not exceed the capacity of an edge, as they contend with one another for edge capacities.

A feasible solution to our linear program provides the conceptual flow rates $x_j^i(p)$ along all feasible paths for each destination, within every session. The effective flow routing scheme $x^i(e)$ for each session, as well as the feasible total throughput $\mathcal{X}$, are all guaranteed to be non-negative, with constraint (4). Since only feasible paths are considered in our linear program, the delay constraint is naturally satisfied.

As an example using the inter-datacenter network that we have shown previously in Fig. 1, Fig. 4 shows the optimal solution obtained by solving our linear program using a standard LP solver. To keep such a conceptual example simple, we assume that all the edge capacities are 10 Mbps. The value labeled on each edge e in the figure indicates its propagation delay $d(e)$, which is the same in both directions. Let us now consider two sessions, $S1$ and $S2$. In $S1$, video flows are transmitted from $D1$ to $D4$ and $D5$; and in $S2$, they are

[1]When it is clear from the context of our discussions, *aggregated sessions* in the inter-datacenter network is simply referred to as *sessions* from this point onwards.

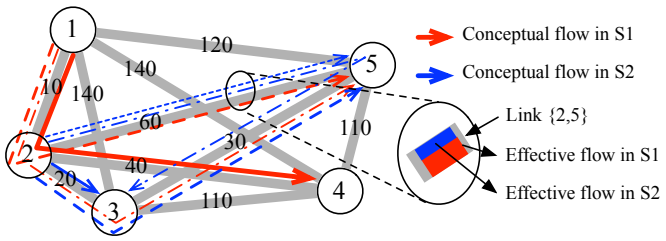


Fig. 4. Maximizing the total throughput in an inter-datacenter network: an example of the optimal solution obtained by solving the linear program.

transmitted from $D2$ to $D3$ and $D5$. If the delay constraint $D_{\max}$ is set to be 100 milliseconds, some of the paths need to be excluded from the set of feasible paths. The conceptual flows along all the feasible paths in each session are shown in the figure, where their widths indicate the corresponding conceptual flow rates.

As we can see, the feasible path in $S1$ to $D4$ is $D1 \rightarrow D2 \rightarrow D4$, with a flow rate of 10 Mbps for the corresponding conceptual flow; the feasible paths in $S1$ to $D5$ are $D1 \rightarrow D2 \rightarrow D3 \rightarrow D5$ and $D1 \rightarrow D2 \rightarrow D5$, each with 3.9 Mbps and 6.1 Mbps respectively. Similarly, feasible paths in $S2$ to $D3$ are $D2 \rightarrow D3$ and $D2 \rightarrow D5 \rightarrow D3$, with 6.1 Mbps and 3.9 Mbps, respectively; and feasible paths in $S2$ to $D5$ are $D2 \rightarrow D5$ and $D2 \rightarrow D3 \rightarrow D5$, with 3.9 Mbps and 6.1 Mbps, respectively. In the optimal solution, the total throughput in both sessions is 20 Mbps in this example, and edge capacities along the feasible paths have been saturated.

To better illustrate the concept of conceptual flows, we examine the edge from $D2$ to $D5$. Though two conceptual flows in $S2$ — each with a rate of 3.9 Mbps — pass through this edge, the effective flow rate in $S2$ on this edge remains to be 3.9 Mbps, since the power of network coding guarantees that conceptual flows destined to different destinations in the same session do not compete for edge capacities. On the other hand, the effective flow rate in $S1$ on this edge is 6.1 Mbps, competing with the effective flow in $S2$ for the edge capacity.

With the use of conceptual flows, the optimal solution of our linear program is quite expressive. It may impose that an incoming flow be *replicated*, be *split*, or that multiple incoming flows be *merged* using network coding, and then forwarded along outgoing edges. In our example, consider datacenter $D2$. Its incoming flow of 10 Mbps in session $S1$ is not only forwarded to $D4$ directly, but also *split* and forwarded *at the same time* to $D3$ and $D5$, with an outgoing flow rate of 3.9 Mbps and 6.1 Mbps, respectively. The flexibility and power of the optimal solution expressing a wide variety of forwarding strategies at each datacenter have provided a solid foundation for *Airlift*, yet they also pose a challenge to our protocol design, to make sure that the optimal solution can be realized faithfully in practice.

IV. Airlift: Protocol Design

With the available capacity and propagation delay on each inter-datacenter edge as input, the optimal solution along the set of feasible paths provides the *complete plan* to start actual packet transmission: In each *conceptual flow*, the optimal solution computes its flow rate $x_j^i(p)$, along the path p in a session i from the source S^i to the destination R_j^i that packets will follow.

The design objective of the *Airlift* protocol is to faithfully realize the *complete plan* that the optimal solution provides in a real-world implementation, with as little gap between theory and practice as possible. As we shall soon observe, such a goal is challenging to achieve; and subsequent experimental evaluations of our *Airlift* implementation will focus on how tradeoffs in our design will contribute to the gap between theory and reality.

A. Transport with Network Coding

Before we discuss challenges in our design, we first present a primer on how network coding can be implemented in practice. With random linear network coding [8], [9], a *generation* of live video is divided into n packets (called the *generation size*) $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$, where each packet has a fixed number of bytes, k. To code a new coded packet x_j, the source first independently and randomly chooses a set of coding coefficients $[c_{j1}, c_{j2}, \cdots, c_{jn}]$ in $GF(2^8)$, one for each original or coded packet it has buffered. It then produces one coded packet $x_j = \sum_{i=1}^{n} c_{ji} \cdot b_i$. The destination decodes as soon as it has received n linearly independent coded packets $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$. It first forms an $n \times n$ coefficient matrix $\mathbf{C}$, using the coefficients of each packet b_i, which are embedded in the packet. Each row in $\mathbf{C}$ corresponds to the coefficients of one coded packet. It then recovers the original packets $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$ as $\mathbf{b} = \mathbf{C}^{-1}\mathbf{x}$. Gauss-Jordan elimination is used in such a decoding process, performed progressively as coded packets are being received. The inversion of $\mathbf{C}$ is only possible when its rows are linearly independent, *i.e.,* $\mathbf{C}$ is full rank.

Airlift uses UDP as its transport protocol on each inter-datacenter edge, the rate of which is controlled by an implementation of TCP-Friendly Rate Control (TFRC) [10] at the application layer. In such a context, network coding has been applied extensively in the *Airlift* protocol design. This is not only because our problem formulation hinges upon the concept of conceptual flows made possible by network coding, but also since random linear codes are rateless erasure codes, and coded packets — each with its own vector of randomly chosen coefficients — can be generated *ad infinitum*. As long as n linearly independent packets are received, they are sufficient to recover the original generation. This is a perfect match to UDP as a transport protocol: losing some coded packets is no longer a concern, as more from the source will be arriving soon, provided that the source receives some form of feedback.

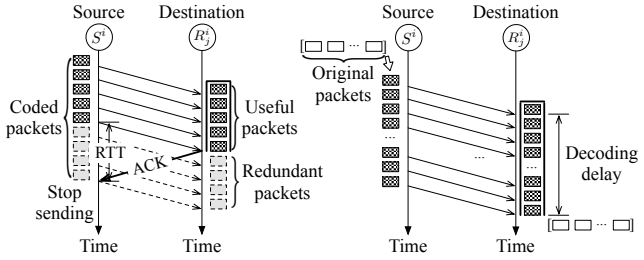
Unfortunately, one seemingly trivial question — on an implementation detail when network coding is used at the source — puts the very idea of using network coding at risk.

B. Bandwidth Overhead vs. Delay: A Dilemma

It is the question of what an appropriate *generation size*, n, is, which the source datacenter should use when it applies

network coding. In other words, shall we use a smaller number of packets in each generation, or a larger number of them?

Let us consider the outcome of using a smaller number of (say, 5) packets. In the example illustrated in Fig. 5(a), we can observe that a small generation size will lead to significant bandwidth overhead. At the time when the source finishes sending all 5 packets, the acknowledgement, to be sent when the entire generation is completely received and decoded at the destination, has not yet arrived at the source. Such an acknowledgement may only be received by the source after a round-trip time since the last coded packet in the generation has been sent. During such a period of time, the source will have no choice but to either *stop sending*, in which case its instantaneous flow rate is throttled to zero and outgoing bandwidth is idled; or to *keep sending more* coded packets, in which case these packets are redundant and useless when received by the destination, leading to significant bandwidth overhead. As the number of packets in a generation becomes smaller, the overhead of such redundant packets, as a percentage, will be even more significant.



(a) Using a small generation size: more bandwidth overhead

(b) Using a large generation size: longer decoding delay

Fig. 5. Bandwidth overhead vs. decoding delay: a dilemma over the choice of the generation size.

Using a much larger generation size (say, a few hundred packets) would certainly mitigate such overhead, but as the example in Fig. 5(b) illustrates, it leads to a substantially longer *decoding delay* at the destination. Consider the analogy that the destination holds a "bucket" — the capacity of which is the generation size — waiting for coded packets to arrive and fill the bucket. Due to the nature of random linear codes, the destination will have to *wait* till the bucket is almost full in order to recover the first original packet in the generation, even if Gauss-Jordan elimination is used. Such a waiting period adds an additional *decoding delay*, which becomes longer as the generation size becomes larger. With a generation size of 128 packets, a packet size of 1 KB, and a source flow rate of 64 KB/sec, the decoding delay can be as long as 2 seconds, which is excessive considering typical end-to-end delay constraints in a video conference.

An additional piece of bad news is that the *sliding window* approach — designed by Sundararajan *et al.* [11] for incorporating network coding into TCP as a transport protocol — does *not* solve this dilemma. In [11], a TCP source transmits random linear combinations of packets in its (sliding) congestion window, and advances the window as it receives an acknowledgment from the destination. These acknowledgments are in the form of the *degree of freedom* of the "bucket" that the destination holds; *i.e.,* an original packet is acknowledged as it is received in a coded form, even before decoding is complete. The size of a generation in this approach is the size of the sliding congestion window, which reflects the bandwidth-delay product on the end-to-end path, and can lead to excessively large decoding delays at the destination.

To add insult to injury, the high probability of receiving *linearly dependent* packets at the destination is a grave concern if a small generation size is used. It has been shown in [12] that the overhead due to linearly dependent packets can be up to 18% if each generation has 8 packets. Fortunately, as proposed by [12], the source can use systematic Reed-Solomon codes rather than random linear codes to mitigate such linear dependence, and such a change does not affect any theoretical benefits of network coding.

C. The Protocol at Source and Destination Datacenters

Our design of the *Airlift* protocol demonstrates that a blend of valuable elements in existing protocols can go a long way towards solving the dilemma.

Decoupling the notion of a generation from a sliding window. First, we must choose to use a small generation size to reduce decoding delays, as we are not in a position to compromise on the end-to-end delay constraint in a live video conference. In order to mitigate the bandwidth overhead, our design in *Airlift* decouples the notion of a *generation* from a *sliding window*: coded packets are generated within a generation, while a sliding window represents all the packets that have been sent but not yet acknowledged by the destination. These two windows do not have to be the same, and the sliding window can contain a large number of generations. Our design makes it possible to use a sliding window size that corresponds to the bandwidth-delay product between the source and the destination, while keeping the generation size small.

With a generation size of n, the source datacenter transmits $n \cdot (1+\delta)$ coded packets from the current generation by using systematic Reed-Solomon codes, where δ can be adapted on-the-fly to reflect the amount of redundancy we wish to add to combat packet loss[2]. Afterwards, the source moves onward and starts to transmit $n \cdot (1+\delta)$ coded packets from the next generation in the sliding window. The rate at which the source transmits in session i is $\min_j \sum_{p \in \mathcal{P}_j^i} x_j^i(p)$, which is part of the *complete plan* stipulated by the optimal solution.

Since packets from multiple generations are in the pipeline, the destination datacenter will need to hold multiple active buckets accordingly, each containing packets received so far within the same generation. Upon receiving each coded packet, it is placed into its corresponding bucket after performing Gauss-Jordan elimination, so that the corresponding coding matrix in each bucket is guaranteed to be in reduced row-echelon form (RREF). As n linearly independent packets

[2]In our real-world implementation, δ is adapted by the source periodically, based on the packet loss rate measured at the source in its TFRC implementation.

arrive in a bucket, all the original packets will be recovered and the bucket will no longer be active.

Acknowledging the degrees of freedom in all active buckets. Upon receiving each coded packet, the destination immediately sends an acknowledgment to the source. The acknowledgment contains the *degrees of freedom* in all active buckets, corresponding to the number of linearly independent coded packets received in each generation that has not yet been fully decoded. As it receives and examines each acknowledgment, the source transmits a sufficient number of additional coded packets from each of the generations contained in the acknowledgment, starting from the oldest generation, but not including the current generation, from which the source is still in the process of transmitting coded packets.

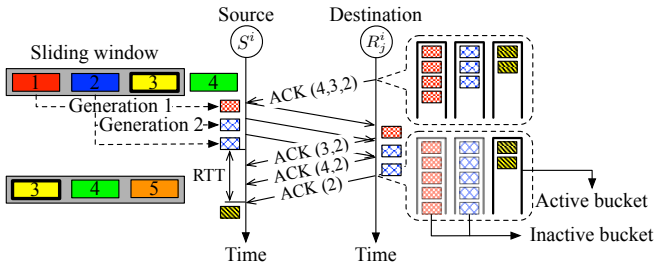


Fig. 6. The *Airlift* protocol at the source and each destination: an example.

For example, in Fig. 6 with a generation size of 5, once the source observes that the destination has received 4 packets from generation 1, 3 packets from generation 2, and 2 packets from generation 3 (the current generation at the source), it will immediately transmit one new coded packet from generation 1 and two new packets from generation 2, before it resumes the process of transmitting packets from the current generation.

Since an aggregated session is a multicast session from a source to multiple destinations, the source will only remove the oldest existing generation and advance its sliding window once all the destinations in the session have indicated that they have successfully decoded the generation (in that it is not included in their acknowledgments).

D. Realizing Conceptual Flows with Source Routing

The optimal solution to our linear program contains a number of conceptual flows in each session, each consisting of a flow rate and a path from the source to one of the destinations. From the perspective of how packets in an actual flow are processed in reality, there are three distinct cases when multiple conceptual flows pass through an intermediate datacenter: (1) Packets in an actual flow are to be replicated and forwarded to multiple outgoing edges; (2) an actual flow is to be split and forwarded, with different portions of its packets destined to different outgoing edges; and (3) packets in multiple actual flows — from the same source and destined to different destinations — are to be *merged* with random network coding. These cases are illustrated in Fig. 7. An intermediate datacenter may be responsible for handling multiple cases concurrently, as in the example of datacenter $D2$ in Fig. 4.

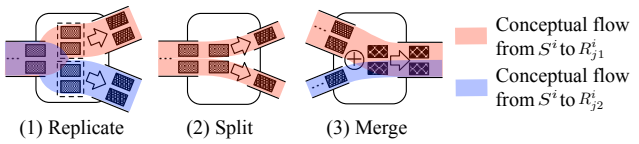


Fig. 7. Three cases of realizing conceptual flows passing through an intermediate datacenter.

Since the first two cases do not require network coding on the intermediate datacenter, they are realized in *Airlift* with the use of *source routing*. The source has full knowledge of all its conceptual flows in the optimal solution, and by allocating outgoing packets to each of the conceptual flows based on their flow rates, it is able to compute the complete *tree* that an outgoing packet should follow to reach its destination(s), and include the tree in the packet header[3]. A packet then becomes *self-routing*, in that an intermediate datacenter only needs to examine its header, extract its next-hop datacenters in the tree, and then forward it to its next hop, making copies as needed.

To realize the third case, all datacenters in *Airlift* finds path *overlaps* between different conceptual flows if they are destined to different destinations in the same session, by examining the complete plan given by the optimal solution. If such an overlap exists, as shown in Fig. 7(3), the corresponding datacenter will produce random linear combinations of packets from actual incoming flows, and transmit a *merged* outgoing flow according to the rate on the outgoing edge, given by the optimal solution.

All such random linear combinations are only performed on packets from the same generation at the source. If packets from the same generation are not readily received from all the incoming flows, the merging process will wait for a timeout period with a buffer of recently received packets. After the timeout expires, it simply merges packets that it was able to receive so far. Since such a timeout period adds to the end-to-end delay, it is not feasible to use a long timeout value.

E. A Full-Service Broker to Conference Participants

With simplicity in mind, *Airlift* is designed to serve as a full-service broker: a participant in a conference would simply select a datacenter to connect to, and start transmitting to this datacenter using an adaptive video source rate, coupled with an adaptive video codec. The datacenter adds this new video source to one of its aggregated sessions that share the same subset of destination datacenters. As a full-service broker, any datacenter in *Airlift* maintains full knowledge about all active conferences, including the list of participants and the datacenter each of them connects to. Updates are simply broadcast to all other datacenters in the cloud.

Original video packets from each participant are transmitted to the datacenter over UDP and TFRC, and with the same *Airlift* protocol in Sec. IV. The only revision in the protocol is an additional form of periodic acknowledgment: the datacenter

[3]It is a *tree* — rather than a *path* — that a packet should follow when being routed, since conceptual flows to different destinations may share the same outgoing edge from the source.

suggests a new *video source rate* to the participant, who then adapts its sending rate to be the minimum of the suggested rate and what TFRC imposes.

How does *Airlift* compute the video source rate that a datacenter suggests to each connected participant? Since the source rate of the aggregated session, $\min_j \sum_{p \in \mathcal{P}_j^i} x_j^i(p)$, is specified by the optimal solution of our linear program, the datacenter simply allocates such a source rate to all participants in the aggregated session. The allocation is to be max-min fair, in that if a participant transmits at a lower rate than its fair allocation due to its last-mile bottleneck, it will be allocated what it needs, plus a certain margin to allow upward allocation adjustments if the last-mile bottleneck bandwidth improves; otherwise, it will be allocated its fair share. With our design, the source rate of the session may not be fully allocated to current participants — in the case that the bottleneck for all of them resides in the last-mile link. Such a residual source rate makes it straightforward to accommodate newly arriving participants, without the need to re-optimize globally by solving a revised linear program.

One remaining challenge is to determine the weight, w_i, of aggregated session i, which is used as input in our linear program. Again, to keep the design simple, w_i will be proportional to the number of participants in session i, and $\sum_i w_i = 1$. This guarantees basic fairness across participants in all the conferences, when their video sources share the inter-datacenter capacity.

V. Real-World Implementation and Evaluation

Airlift is the name for not only our application-layer protocol, but also its real-world implementation. The basic unit for our *Airlift* implementation is a *broker*, which runs in a VM in one of the datacenters. Beyond its role as a full-service broker to conference participants (Sec. IV-E), a broker is also responsible for two core features in the protocol.

Multi-generation sliding window with network coding. A broker is responsible for implementing the *Airlift* protocol at the source and destination datacenters, which involves sending and acknowledging packets in multiple generations within the source sliding window, coded with random linear codes. Since random network coding lies at the core of the *Airlift* protocol, our implementation of network coding is optimized with Intel SSE2 acceleration, which offers a five-fold performance gain on average, as compared to a vanilla implementation.

Packet processing and forwarding. As an intermediate datacenter in a multi-hop path, a broker is capable of replicating, splitting, and merging incoming flows, and of forwarding packets based on source routing information embedded in their headers. Using asynchronous event-driven networking, our implementation is highly efficient, designed with performance in mind. It supports major UNIX variants and Windows (both are typical in cloud VMs), and incurs less memory and CPU overhead compared to the traditional thread pool concurrency model, especially at high packet processing rates.

A centralized *optimizer* in our implementation receives periodic reports from all the brokers with respect to measured capacities and propagation delays on inter-datacenter edges. It then generates the set of feasible paths, and solves the linear program that maximizes the total throughput across all the aggregated sessions. The optimal solution is then transmitted back to all the brokers for them to route outgoing packets using source routing.

We choose to use the Amazon EC2 inter-datacenter network in our experiments, which is one of the predominant Infrastructure-as-a-Service (IaaS) cloud providers. We have launched 7 standard on-demand small VM instances on all 7 EC2 datacenters[4], with a broker in each. We used a large VM instance located in the Virginia EC2 datacenter to host our centralized optimizer.

A. Airlift vs. Celerity: A Performance Comparison

With our new implementation of *Airlift*, our first set of experiments is to compare its performance with *Celerity* [3], in a comparable real-world conference. Since *Celerity* is designed to maximize the total throughput within a single conference only, we initiated only one conference with *Airlift* for a fair comparison. The challenging dumbbell topology, shown in Fig. 2 and used extensively in [3], was adopted in our comparison studies. As an example, in the Toronto-Beijing topology, we used two PlanetLab nodes in Toronto and two more in Beijing as conference participants, forming 4 *sessions*, each corresponding to a video source at one of the participants. In *Airlift*, users in Toronto were connected to their nearest datacenter in Virginia, and users in Beijing were connected to their nearest datacenter in Tokyo, leading to two aggregated sessions in the EC2 inter-datacenter network.

Our experimental results with both Celerity and *Airlift* have been summarized in Table II, over three different pairs of geographic locations: Toronto – Beijing, Vancouver – Berlin, and Seoul – Rio de Janeiro. As we can easily observe, with respect to the total throughput of all the sessions in the conference, *Airlift* was able to offer a substantial performance advantage, on the order of 325% – 2387% (*i.e.*, 24 times better than Celerity). Yet, *Airlift* did not suffer from a noticeable disadvantage with respect to end-to-end delays: it typically incurred a similar end-to-end delay as compared to Celerity, which uses overlay links. The substantial throughput advantage with similar delays have made *Airlift* a clear winner with respect to performance in our experiments.

Airlift's throughput advantage can be explained by abundant inter-datacenter network capacities, and the excellent performance on end-to-end delays can be attributed to design and implementation choices in our protocol, which have collectively minimized queueing delays as packets were forwarded through the two intermediate datacenters. Our measurements, not presented in the table, have shown that the queues on these datacenters have remained *empty* at most times throughout our experiments.

During the startup phase of a conference, another important disadvantage of Celerity is the longer period of time for it

[4]Amazon EC2 datacenters are located at Oregon, Northern California, Virginia, Ireland, Tokyo, Singapore, and San Paulo.

TABLE II
AIRLIFT VS. CELERITY: A COMPARISON WITH RESPECT TO THE TOTAL THROUGHPUT AND THE MAXIMUM END-TO-END DELAY ACROSS ALL 4 SESSIONS IN THE SAME CONFERENCE.

Airlift/Celerity	Total throughput (Mbps)	End-to-end delay (msec)
Toronto-Beijing	14.11/4.04	169.8/142.3
Vancouver-Berlin	34.32/1.38	137.2/104.9
Seoul-Rio	20.32/2.32	228.6/203.5

to ramp up its throughput in each session to the steady state. Fig. 8 and 9 have shown the per-session throughput over time for Celerity and *Airlift*, respectively, as a conference started up. we note that it took Celerity three times longer to ramp up to a steady-state throughput of three times lower. Such subpar performance is due to the nature of the Celerity protocol design: its rate control algorithm that governs overlay link rates does not have any knowledge of the underlying physical topology, and would have to take some time to converge to optimality based on online measurements. Such a challenge is nonexistent in *Airlift*, as it simply routes packets through the nearest datacenter over the cloud.

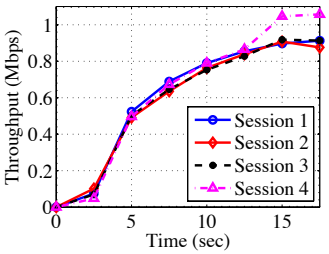


Fig. 8. *Celerity* takes 15 seconds to ramp up its per-session throughput to the steady state of 0.9 Mbps on average.

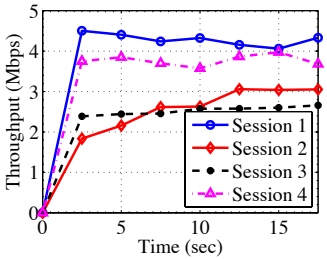


Fig. 9. *Airlift* takes less than 5 seconds to ramp up its per-session throughput to the steady state of 3.5 Mbps on average, a three-fold improvement.

B. Airlift: Performance with Multiple Aggregated Sessions

We now proceed to evaluate *Airlift*'s performance with multiple aggregated sessions in the EC2 inter-datacenter network. For this purpose, we launched 10 actual conferences, with participants connecting to their nearest datacenters. In order to simulate the geographic diversity of conference participants, the number of datacenters involved in each conference is generated with a discrete uniform distribution in the range of $[2, 7]$, resulting in a total of 35 sessions.

As measured by the brokers every 3 minutes on their respective datacenters and reported to the centralized optimizer, the edge capacities ranged from 20.9 to 130.8 Mbps, while their propagation delays ranged from 11.3 to 441.7 msec. These capacity and delay values were used as input to our linear program, which the optimizer used to obtain the optimal solution. With 10 conferences and 35 aggregated sessions, we imposed an end-to-end delay constraint, $D_{\max}$, of 300 milliseconds. As a result, the optimal solution, as computed by the optimizer, involved a total of 1869 feasible paths, with 54 feasible paths per session on average, and a maximum of 5 hops in a feasible path. With a reasonable $D_{\max}$, the large number of feasible paths is indeed a piece of good news, in that the optimizer would have the freedom of using all of these paths to deliver packets, saturating edge capacities as much as possible to improve the total throughput.

Again, as a conferencing cloud service, we are most concerned with the achievable throughput and end-to-end delays in our aggregated sessions, both to be measured in actual experiments. Fig. 10 shows the throughput observed in each of the aggregated sessions, while Fig. 11 shows the maximum end-to-end delays observed from the source to the destinations in 10 sessions in EC2. With stable edge capacities in EC2 over the short term, the observed throughput values in Fig. 10 were exactly the same as what the optimizer has computed, thanks to *Airlift* capability of transmitting, coding, forwarding, and decoding packets at the designated flow rates computed by the optimizer. The end-to-end delays are well controlled, again thanks to the ability of our implementation to keep queueing delays to the minimum while packets traverse through intermediate datacenters on their paths.

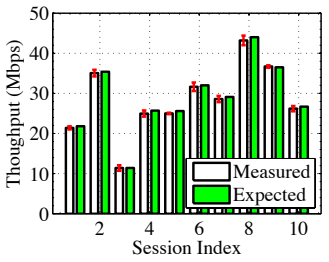


Fig. 10. Observed throughput values in 10 of the aggregated sessions.

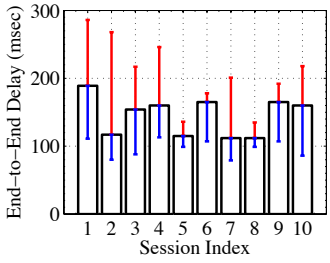


Fig. 11. Observed maximum end-to-end delays from the source to the destinations in 10 of the aggregated sessions.

C. A Packet's Life: A Microscopic View

Our readers may be left wondering: What had contributed to *Airlift*'s substantial performance advantage against Celerity, as well as its superior inter-datacenter performance with multiple aggregated sessions? To best answer this question, we decided to delineate the process of sending, processing, and acknowledging packets, showing fine-granularity details within a single aggregated session involving two destination datacenters. Our experiments ran for a period of 3 minutes.

To saturate inter-datacenter edge capacities on the order of 100 Mbps, let us first examine the transport protocol with multi-generation network coding from the source to a destination. In our experiments, we used a generation size of 10 packets and a packet size of 4 KB. Thanks to our accelerated network coding implementation, we could comfortably reach a session throughput of 59.1 Mbps with a CPU load of just 78% at the source, within a small EC2 VM instance. On the other hand, since the size of the sliding window is decoupled from the generation size, we were able to achieve such a throughput with an average of 36 outstanding generations at the source that are not yet acknowledged by both destinations. Thanks to our application-layer TFRC implementation (based on RFC

3448) and the stability of the EC2 cloud, a negligible packet loss event rate of 4.5×10^{-5} was observed. As a result, there were no more than 4 active buckets used concurrently for decoding at both destinations. This minimized the bandwidth overhead of acknowledgments from both destinations back to the source: an average of 1.65 Mbps was used for acknowledgments (with 3905 packets per second and 54 bytes per packet on average), reflecting an overhead of only 2.8%.

The negligible packet loss event rates that we observed have also had a positive effect when it comes to minimizing the bandwidth overhead of using network coding. In our experiments, the number of *outdated* packets — defined as redundant packets who arrived after their generations have already been decoded — remained in the range of $[1\%, 4\%]$, as a percentage of all coded packets transmitted. With a low percentage of outdated packets transmitted, the number of linearly dependent packets due to a small generation size was also negligibly small, accounting for only 0.00004% of all the coded packets in the session.

Under our control in the *Airlift* protocol, there are two factors that affect one-way end-to-end delays in our experiments: (1) the *queue lengths* on intermediate datacenters along a path that packets traversed; and (2) the *decoding delays* at a destination. It turns out that, in terms of the number of packets, the queue lengths we observed on each of the outgoing edges involved in the session were *zero* most of the time, and were no more than 18 packets in the worse case, as shown in Fig. 12. Further, Fig. 13 shows the maximum decoding delays observed at runtime every 15 seconds, which represented the time from when a new active bucket was created at a destination to when it was completed decoded. As the figure shows, the decoding delays remained in the range of $[10, 27]$ milliseconds. On average, our measurements have indicated that the queueing and decoding delays combined had accounted for only 23% of one-way end-to-end delays, the bulk of which was attributed to propagation delays on inter-datacenter edges in EC2.

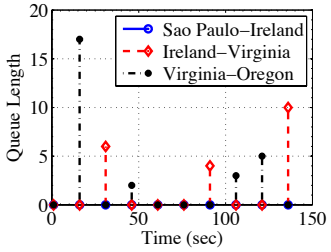


Fig. 12. Observed queue lengths on each hop along a path (Sao Paulo → Ireland → Virginia → Oregon).

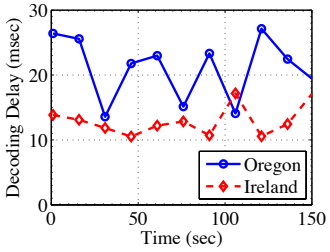


Fig. 13. Observed decoding delays at both destinations (Oregon and Ireland) over time, measured every 15 seconds.

Finally, Fig. 14 shows a microscopic view of a packet's lifetime in our experiment with two destination datacenters. The figure has been annotated by propagation, queueing, and decoding delays that we measured, from the moment the packet was sent by the source datacenter (and acknowledged to the conference participant), to the time it was received by all its destination datacenters (and relayed to the rest of the participants).

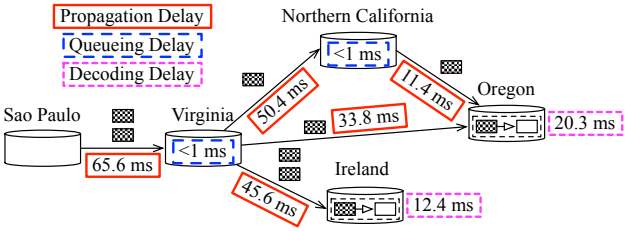


Fig. 14. A packet's life as measured from the source datacenter (Sao Paulo) to both destination datacenters (Oregon and Ireland).

VI. Related Work

The most recent P2P conferencing solution is Celerity [3], which held the view that bandwidth bottlenecks may not be limited to the last-mile uplink bandwidth, and instead may reside in the core of the Internet. In this spirit, Celerity represented an advance as compared to other existing works on P2P conferencing protocols in the literature [2], [4], [13], [14], which sought to optimize the aggregated utility of conference participants, as well as the utilization of peer uplink bandwidth.

The essence of *Airlift* is to provide video conferencing as a cloud service that routes video flows through the inter-datacenter network in the cloud. In contrast to Celerity that focuses on one conference, *Airlift* supports multiple conferences by aggregating video flows to *aggregated sessions* (each with its own subset of datacenters). Liang *et al.* [4] did consider multiple conferences (called *swarms* in the paper) in a P2P system; and focused on optimal bandwidth sharing of resources at both peers and helpers, using a utility maximization framework. Since *Airlift* handles aggregated video traffic, it is designed to maintain basic fairness among all the video sources with respect to their flow rates, without the use of utility functions.

Network coding was used as a rateless erasure code in Celerity [3], and detailed implementation hurdles have not been addressed. In *Airlift*, network coding serves as the foundation throughout our problem formulation and our protocol design. Only with network coding can we design a new protocol that achieves throughput optimality in theory, yet lends itself to a feasible real-world implementation.

VII. Concluding Remarks

With Google+ Hangouts implemented as a cloud-based client-server video conferencing solution [15], we believe that designing a high-performance cloud service to serve enterprise video conferencing needs is an industry trend that should not be overlooked. In this paper, we part with the traditional wisdom of a peer-to-peer design; rather, we motivate and present *Airlift*, a new protocol and real-world implementation that provide video conferencing as a *cloud service*, by routing video flows through the inter-datacenter network in the cloud, with higher capacities than the public Internet. The design of *Airlift* is driven by our objective of maximizing the total

throughput of all conferences, yet without compromising on end-to-end delays. Intra-session network coding lies at the heart of the *Airlift* protocol, which is designed with attention to detail and with no stone left unturned. With real-world experiments using PlanetLab nodes and the Amazon EC2 cloud, we show that *Airlift* is able to deliver on its promises: it is capable of supporting substantially higher video bit rates, yet with similar end-to-end delays as compared to P2P solutions.

REFERENCES

[1] A. Cockcroft, "Netflix in the Cloud," 2011. [Online]. Available: http://velocityconf.com/velocity2011/public/schedule/detail/17785

[2] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility Maximization in Peer-to-Peer Systems," in *Proc. ACM SIGMETRICS*, 2008, pp. 169–180.

[3] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: A Low-Delay Multi-Party Conferencing Solution," in *Proc. ACM Multimedia*, 2011, pp. 493–502.

[4] C. Liang, M. Zhao, and Y. Liu, "Optimal Bandwidth Sharing in Multiswarm Multiparty P2P Video-Conferencing Systems," *IEEE/ACM Trans. Networking*, vol. 19, no. 6, pp. 1704–1716, 2011.

[5] Z. Li, B. Li, and L. C. Lau, "On Achieving Maximum Multicast Throughput in Undirected Networks," *IEEE Trans. Info. Theory*, vol. 52, no. 6, pp. 2467–2485, 2006.

[6] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "DONAR: Decentralized Server Selection for Cloud Services," in *Proc. ACM SIGCOMM*, 2010.

[7] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans. Info. Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[8] P. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. Allerton Conference on Communications, Control and Computing*, 2003, pp. 40–49.

[9] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger, "A Random Linear Network Coding Approach to Multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in *Proc. ACM SIGCOMM*, 2000.

[11] J. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP: Theory and Implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011.

[12] Y. Feng, Z. Liu, and B. Li, "GestureFlow: Streaming Gestures to an Audience," in *Proc. IEEE INFOCOM*, 2011.

[13] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: an Efficient Mechanism for Content Distribution in a P2P Network," in *Proc. ACM SIGCOMM Asia Workshop*, 2005.

[14] M. Ponec, S. Sengupta, M. Chen, J. Li, and P. Chou, "Multi-rate Peer-to-Peer Video Conferencing: A Distributed Approach using Scalable Coding," in *IEEE International Conference on Multimedia and Expo*, 2009.

[15] J. Uberti, "Announcing Google+ Hangouts," 2011. [Online]. Available: http://juberti.blogspot.ca/2011/06/announcing-google-hangouts.html