Practical, Real-time Centralized Control for CDN-based Live Video Delivery

Matthew K. Mukerjee* mukerjee@cs.cmu.edu
David Naylor* dnaylor@cs.cmu.edu
Junchen Jiang* junchenj@cs.cmu.edu
Dongsu Han† dongsuh@ee.kaist.ac.kr
Srinivasan Seshan* srini@cs.cmu.edu
Hui Zhang*‡ hzhang@cs.cmu.edu

*Carnegie Mellon University †KAIST ‡Conviva Inc.

Abstract

Live video delivery is expected to reach a peak of 50 Tbps this year [7]. This surging popularity is fundamentally changing the Internet video delivery landscape. CDNs must meet users' demands for fast join times, high bitrates, and low buffering ratios, while minimizing their own cost of delivery and responding to issues in real-time. Wide-area latency, loss, and failures, as well as varied workloads ("mega-events" to long-tail), make meeting these demands challenging.

An analysis of video sessions [32] concluded that a centralized controller could improve user experience, but CDN systems have shied away from such designs due to the difficulty of quickly handling failures [29], a requirement of both operators and users. We introduce VDN, a practical approach to a video delivery network that uses a centralized algorithm for live video optimization. VDN provides CDN operators with real-time, fine-grained control. It does this in spite of challenges resulting from the wide-area (e.g., state inconsistency, partitions, failures) by using a hybrid centralized+distributed control plane, increasing average bitrate by 1.7× and decreasing cost by 2× in different scenarios.

CCS Concepts

•Networks → Traffic engineering algorithms; Overlay and other logical network structures;

Keywords

live video; CDNs; central optimization; hybrid control

SIGCOMM '15, August 17 - 21, 2015, London, United Kingdom
© 2015 ACM. ISBN 978-1-4503-3542-3/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2785956.2787475

1 Introduction

Demand for live video is increasing by 4–5× every three years and the live video peak streaming rate is expected to reach 50 Tbps this year [7]. This demand spans widely different types of videos (professionally-produced and user-generated) and workloads ("mega-events" to long-tail). The 2014 World Cup, a recent live video mega-event, used traditional CDNs to deliver live streams totaling several terabits per second [38], which is estimated to be 40% of all Internet traffic during that time [37]. At the other extreme, 55 million Twitch users [4] watch more than 150 billion minutes of live video each month, generated by over 1 million users, making it the fourth largest Internet traffic producer in the US [5, 41].

The diversity and volume of live video delivery makes it a complex challenge for modern content delivery infrastructures. However, huge demand isn't the only challenge; users, CDNs, and the network environment impose additional requirements. Users demand high quality, instant start-up (join) times, and low buffering ratios [10]. CDNs want to meet client demands while minimizing their delivery costs and responding to issues in real-time [35]. Finally, operating over the wide-area network environment requires designs that handle common latency variations and communication failures.

The traditional solution to these problems is traffic engineering. However, even state-of-the-art systems [23, 25] work on traffic aggregates at coarse timescales. Users' demands for high per-stream quality and CDNs' demands for fast failure recovery require control over individual streams at fine timescales. Overlay multicast systems [12, 14, 26, 30], while focusing on individual stream optimization, overlook issues that arise with the many concurrent, independent, high-bandwidth streams in today's environment. Internet-scale, video-specific systems like Conviva's C3 [19] use client-side analytics to pick the best CDN for a given client at a given time but ignore the actual data delivery. Although current CDNs provide good video quality, a previous analysis of a large collection of video sessions [32] concluded that

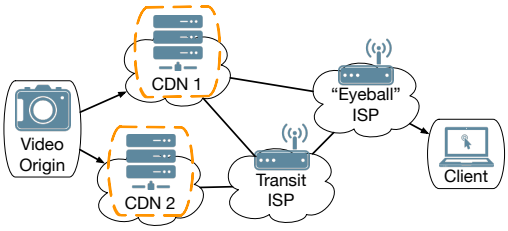


Figure 1: Entities involved in live video distribution. Unlike Conviva's C3 [19], which focuses on clients, we focus on optimizing CDNs.

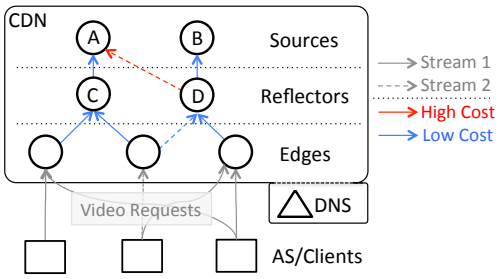


Figure 2: CDN live content distribution [35].

a centralized video controller algorithm could greatly improve end-user experience. Even so, traditional CDNs have shied away from centralized designs due to the difficulty of providing good performance while quickly handling failures in the wide area [29].

Unfortunately, past work on live video delivery does not meet the practical and real-time requirements of both CDNs and users. In summary, we need a live video control plane that: 1) enables proactive control over **cost** and **quality** at fine-grained timescales, 2) **scales** to today's largest CDNs and their workloads, 3) achieves **real-time responsiveness** to minimize join time and respond to failures, and 4) meets these goals despite wide-area network delays and failures.

In order to address these challenges, we propose a new system, called *video delivery network* (VDN), that allows CDN operators to dynamically control both stream placement and restrict bitrates automatically, at a very fine timescale in a WAN environment. Traditionally, clients adapt bitrates independently; VDN gives CDN operators a say, as they have the best view of current resources and delivery costs. At its core, VDN uses a centralized algorithm that performs end-to-end optimization of live stream routing. The centralized algorithm is a piece of the larger VDN framework, which mitigates WAN challenges with a *hybrid* approach that balances the benefits of centralized control with the resilience and responsiveness of distributed control.

We evaluate VDN using traces of real video sessions from multiple live content providers as well as a WAN testbed. We show that, in a variety of scenarios such as heavy-head (e.g., a sports game) and heavy-tail (e.g., user-generated streams), VDN provides a 1.7× improvement in average bitrate and reduces delivery costs by 2× compared to current CDNs. We scale VDN to 10,000 videos and show it can react at a timescale of 200 ms.

In summary, our contributions are:

- A **centralized algorithm** based on integer programming that coordinates delivery to provide high-quality live video streaming at scale while giving control "knobs" to operators to balance cost and quality.
- A responsive **live video delivery framework** that minimizes join time and mitigates WAN challenges using a *hybrid* centralized+distributed control plane.

2 Motivation

2.1 Setting

CDN background: We focus on optimizing CDNs for HTTP-based *live video delivery*. Each entity on the video delivery path (see Figure 1) can be independently optimized (e.g., clients in Conviva's C3 [19]), however the focus of this work is CDN optimization.

Live video: Live video is particularly challenging due to *lack of caching and buffering* within the delivery network. In HTTP-based live streaming, a video is encoded at multiple pre-determined bitrates. At each bitrate, each stream is broken into multiple 2–10 second *chunks*, which clients fetch independently via standard HTTP GETs. Clients typically adapt to network issues by fetching different bitrates [6].

CDN structure: Figure 2 presents the high-level structure of a CDN's video delivery system [29, 35, 40]. Each node represents a cluster of co-located servers. A CDN's internal network consists of three logical pieces: video *sources* that import videos into the system, *reflectors* that forward content internally, and *edge clusters* that directly serve end-users (individual clients or aggregate ASes). Each link has a delivery cost associated with it. These link costs are a result of private business deals, but they tend to be more expensive for source/reflector links (typically long-haul WAN links) and less expensive (some entirely free) for edge/AS links [2]. In Figure 2, the link between A and D is a high cost link.

CDNs and DNS: When clients resolve the name of a video stream, the CDN's DNS-based client mapping service maps them to a nearby edge cluster, based on a number of factors (e.g., load, latency, etc.) [35]. When a client's request for a particular channel arrives at an edge cluster, the edge cluster forwards it to a reflector (found via DNS), which in turn forwards it to a source (found via DNS); the content is returned via the reverse path. When multiple requests for the same content arrive at the same node (e.g., C in the figure), only one request is forwarded upwards. The end result is a *distribution tree* for each video from sources to edge clusters.

This has been the design used by Akamai for live streaming since 2004 [29], externally observed in 2008 [40] and referenced by Akamai in 2010 [35]. We confirm this holds today [2].

Problems with modern CDNs: Using DNS to map requests to the appropriate upstream cluster is very natural as CDN workloads have shifted from web-oriented to live streaming. Mapping clients to edge clusters with DNS makes sense, since most live video content is embedded in websites, which already use DNS. However, using DNS to map internal clusters to upstream clusters causes issues: 1) CDNs can't "push" updates to clusters and must instead wait for clusters to "pull" from DNS after a timeout period (the DNS TTL); and 2) To reduce load on DNS, CDNs group different streams together, reducing granularity [29, 40]. Furthermore, CDNs today update DNS mappings using heuristics [2, 29, 35, 40], impacting performance. We explore these issues in more detail.

DNS TTLs: DNS relies on DNS *clients* (i.e., clusters) to ask for updates when cached DNS results expire (every ~30 seconds) [40], preventing a central controller from sending updates as they are ready. This reduces the efficacy of the controller, thus lowering end-user quality and responsiveness to failures. Furthermore, CDN clusters can spot local network failures long before a TTL-bound DNS update could arrive and thus could react quicker. Lowering the TTL would help approximate a "push"-based system but at the cost of a large increase in the number of DNS queries.

Heuristic-based mapping algorithm: A monitoring system collects performance and load information and, based on this knowledge, updates the DNS system every minute [35]. Generally, CDNs map end-users to edge clusters based on geography, load, whether or not a cluster is already subscribed to the video, and performance relative to the requester [2, 29, 35, 40]. It is implied that the mapping of edge clusters to reflectors is done similarly [35], but the specific algorithm is not publicly known. A measurement study points out that geographically close edge clusters all map to the same reflector for the same groups of videos, providing further evidence [40]. Additionally, an analysis of video traces shows that mapping requests based on a global view of the network [32] could provide major benefits for end-users, implying that there are opportunities to improve this mapping.

Goal: VDN's job is twofold: 1) coordinate the selection of distribution trees for each channel and 2) assign groups of clients within a given geographic region to a good edge server. It must perform these tasks while meeting the goals listed below.

2.2 Design goals

Video-specific quality and low cost (quality/cost tradeoff): CDN operators must satisfy user's expectation for high video quality, while minimizing their delivery cost. Thus, VDN must optimize for video quality directly, while considering its cost.

Internet-scale video delivery (scalability): Many different types of workloads exist for live video: (a) *"mega-events"* (e.g., World Cup) serving 1M+ users [38], (b) *TV-style channels* serving 100K users [3], and (c) *"long tail" user channels* (e.g., Twitch, Ustream) serving 1-10,000 users [11]. (a) tends to be easier as one tree can serve everyone, whereas workload (c) is the toughest, as it requires coordinating across many videos. VDN must support these workloads, out to a target scale of 10,000 channels [40] and 2000 edge clusters [17], beyond the scale of today's largest CDNs. Such scale is challenging as finding the optimal placement is NP-hard.

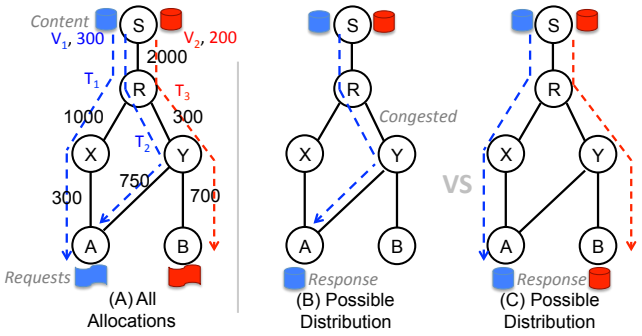


Figure 3: Motivating central coordination.

Fine timescale (responsiveness): VDN must provide fast join time (less than a second) and fast failure recovery, despite challenges in the wide area (e.g., inconsistent state, partitions, loops).

2.3 Case for centralized optimization

Despite the lack of public information on how the CDN internal mapping is done, prior work has shown that a control plane designed around centralized optimization can provide great benefit [32]. In this section we focus on the reasons for these benefits.

Coordination: Throughout this paper, we use coordination to mean the ability to consider all individual streams simultaneously. As mentioned, modern CDNs have difficulty with this as they aggregate videos and get "locked in" to decisions due to DNS TTLs [40]. Figure 3 illustrates why stream coordination can lead to better resource allocation. Two video channels (V_1 and V_2) originate from a single source, S. The goal is to deliver V_1 to AS A and V_2 to B. Three possible distribution trees exist: T_1, T_2 and T_3 (Figure 3a). We present two feasible distribution strategies in Figure 3b and c. In Figure 3b only client A is served, whereas in Figure 3c both clients are served. The issue is that using distribution tree T_2 would congest the RY link. However, knowing this in advance is difficult; it would require not only knowledge of network resources, but also the placement of other streams. A natural solution would be centralization, which affords both a global view of the network and the ability to coordinate streams.

This observation generalizes to large-scale networks. Figure 4 compares a system with a global view that places streams independently without coordination (OM in §7) to one that has a global view *and* coordinates

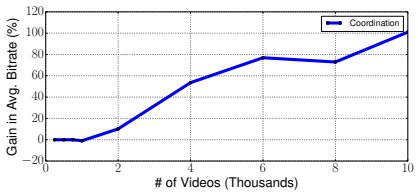


Figure 4: The importance of coordinating streams generalizes to larger systems. This graph shows the gain of our system compared to a multicast-style approach as we'll see in §7.

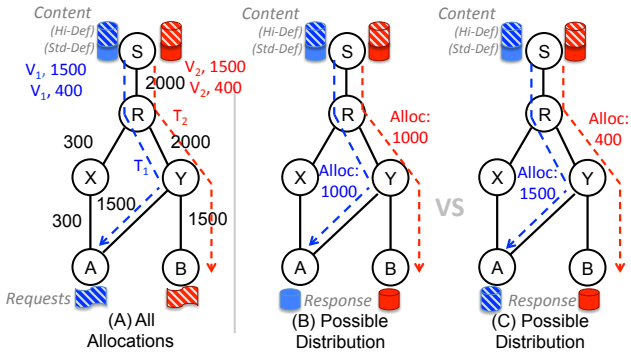


Figure 5: Motivating app-specific optimization.

streams (VDN in §7) for a 100 node topology. With 10K videos, we observe up to a 100% improvement in average bitrate.

Application-specific optimization: Generic traffic engineering at a centralized controller is not enough; we must perform app-specific optimization. For example, in Figure 5, two videos are encoded as low quality (400 Kbps) and high quality (1500 Kbps) versions. Due to bandwidth constraints (Figure 5a), we must deliver both over link *RY*. We present two ways to allocate bandwidth in Figure 5b and c. Despite fairly allocating bandwidth between the streams, Figure 5b does worse overall, as both clients only receive the low quality stream. Figure 5c is "unfair", but is a better strategy as one client is able to get the higher quality stream. Thus, careful consideration of bitrate at the granularity of streams is needed to provide the best quality.

From the two examples, we conclude that we can improve quality with: 1) a global view of network resources; 2) coordination across streams; and 3) consideration of the streaming bitrates. This argues for a video-specific control plane that is logically centralized.

2.4 Case for hybrid control

Live video-specific, centralized optimization alone is not sufficient. A fully centralized system would require new video requests to reach the controller across WAN latencies before the video could be viewed, yielding terrible join times. Additionally, centralized optimization using an integer program can take quite long (e.g., 10s of seconds), further impacting join time. Yet, a distributed scheme would be highly responsive as clusters could react to requests immediately, yielding low join times and fast failure recovery. However, we argue that a distributed scheme is challenged to provide the high quality demanded by users at reasonable cost, due to the lack of coordination (§2.3).

A combination of the two schemes, with the quality of a centralized system and the responsiveness of a distributed system would be best suited. We refer to this combination as **hybrid control**. We avoid poor interactions between the two schemes by exploiting properties of our highly structured topology (§2.1) and by keeping track of alternate paths with enough capacity for each video channel (§4).

3 VDN system overview

VDN reuses existing CDN internal infrastructure (source clusters, reflector clusters, edge clusters, and DNS) but employs a new control plane based on hybrid control—a centralized controller makes optimal decisions slowly while clusters simultaneously make decisions quickly based on distributed state. VDN treats each cluster as an atomic unit (as in Figure 6) and controls the distribution of live video from sources to clients; traffic management within a cluster is outside the scope of this paper.

When video v is requested at bitrate b by a client in an AS a, a request is sent to VDN's DNS server; the response directs the client to send video chunk requests to a nearby edge cluster. If this edge cluster knows about v (i.e., has a entry for (v, b) in its forwarding table), then it forwards the request upstream accordingly. If not, it runs the *distributed control algorithm* (§4). Reflectors pick source clusters similarly. The video chunk follows this path in reverse. Eventually, centralized control updates the clusters' forwarding tables (§5).

As a control plane, VDN (1) populates application-layer forwarding tables at each cluster with centrally computed entries, (2) creates forwarding table entries on-the-fly when necessary using distributed control, and (3) updates the client to edge server mapping accordingly in the DNS infrastructure.

3.1 Design

Physical view: VDN introduces two physical pieces to the traditional CDN infrastructure: a logically centralized *central controller* and a *local agent* in each server cluster. The central controller and local agents are each decomposed into two pieces: (1) a *control* subsystem that computes path assignments based on network state and (2) a *discovery* subsystem that tracks incoming requests and topology information.

Logical view: VDN's control plane is driven by two control loops, which update clusters' forwarding tables at different timescales. A *central control loop* computes optimal distribution trees (as well as client/server mappings) using the algorithm described in §5. This loop runs continuously and operates on the timescale of tens

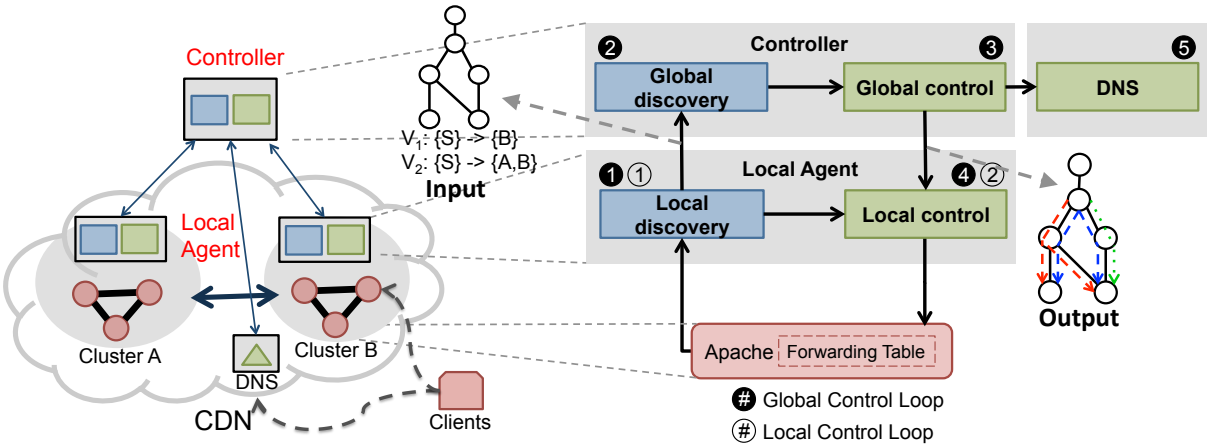


Figure 6: VDN system overview.

Routing Information Base

Central/ Dist.	Channel	Next Hop	Version	Evidence	
				Network Stats	Viewership Stats
C	V_0/800/*	R2	15:20	Link_1: 10Mbps Link_2: 15Mbps	V_0:{800}Kbps, 3000 requests
D	V_0/*/*	R1	15:23	Link_1: 10Mbps Link_2: failed	V_0:{800}Kbps, 3007 requests

Forwarding Information Base

Channel	Version	Next Hop
V_0/*/*	15:23	R1

Figure 7: Sample RIB and FIB entries. The local agent uses network and viewer state as "evidence" to decide when to override potentially stale decisions from the central controller.

of seconds to minutes. Meanwhile, the local agent runs a *distributed control loop* that amends its cluster's forwarding table to quickly (i.e., sub-second) respond to local changes (e.g., link failures) using distributed state.

Central control loop:

❶ Local discovery measures link information and tracks AS- and cluster-level channel viewership.
❷ Global discovery collects measurements from each cluster and builds a global network view.
❸ Global control computes optimal distribution trees.
❹ Local control merges local state with the global decision and updates the forwarding table.
❺ Global control updates DNS.

Distributed control loop:

① Local discovery measures link information and tracks AS- and cluster-level channel viewership.
② Local control merges local state with the global decision and updates the forwarding table.

The two loops have different views of the system, and use their information for different purposes. The central loop sees all clusters, the status of their links, and channel viewership information so that it can assign optimal distribution trees. The distributed loop sees local link conditions and video requests at one cluster as well as a small amount of distributed state. The local agent merges the controller's decision with this information and installs appropriate forwarding rules. Our hybrid control plane strikes a balance between the high quality of a centralized system and the responsiveness of a distributed system.

4 Hybrid control

Running two control loops in tandem can lead to challenges that destroy any benefit that either control loops would have had individually, resulting in a "worst of both worlds" scenario, as hinted in §2.4. When distributed decision-making is used, hybrid control handles this by only considering end-to-end paths that provide enough bandwidth. In this section we examine the interactions of our central and distributed control loops in detail and how we balance them, as well as how hybrid control mitigates issues in the wide-area.

4.1 Central control

Central control takes in a global view of the network (video requests and topology information) as input and uses the algorithm described in §5 to calculate the optimal configuration of distribution trees as output. To avoid having a single point of failure, VDN uses multiple geo-replicated controllers, synchronized with Paxos [31]. After making a decision, VDN's central controller distributes it to individual clusters. To do this, the central controller sends each cluster's local agent a routing information base (RIB) specific to that cluster, as shown in Figure 7. VDN's RIB contains information to support hybrid decision-making in addition to the typical routing information (e.g., a prefix and a next hop). In particular the RIB maintains where the information came from (centralized or distributed control), a version number (timestamp), and a set of "evidence" providing the context when this particular RIB entry was computed (link and viewership information sent by this cluster to the

central controller when this decision was computed). Evidence helps distributed control decide if it should override the global decision.

The RIB gets merged with distributed control's own decision to become the *Forwarding Information Base* (FIB), used by the data plane. If distributed control decides nothing is amiss, the global RIB entry's (channel prefix, version number, next hop) tuple is used directly as the FIB entry.

Discovery: In order for central control to generate good distribution trees, it needs to have up-to-date information on the state of the network and requests. Keeping track of new requests is relatively simple at the edge clusters. Estimating changes in link capacity available to applications in overlay networks (e.g., due routing changes, background traffic, or failures) is a well studied topic [33, 36, 39], and is thus considered out of scope in this work.

4.2 Distributed control

Distributed control keeps track of viewership and path information of upstream neighbors to make quick local decisions in response to changes in network performance, viewership, and failures. The objective is to improve responsiveness by avoiding the costly latency of centralized optimization. Thus, distributed control overrides the central decision in response to dramatic changes.

Initial requests (DNS): VDN's DNS takes into account the user's geographic location and AS in order to map them to the proper edge cluster as computed by the central controller. If this particular AS has not previously been assigned to an edge cluster, simple heuristics are used to provide a reasonable starting assignment (e.g., an edge cluster that already is subscribed to this video, an edge cluster that's typically used by this location/AS, etc.). This provides an initial instant mapping of clients to edge clusters.

Distributing state: Clusters distribute video subscription and link information to other nodes via a distance vector-like algorithm to aide in reacting to large changes. Each cluster periodically (e.g., every second) sends all connected clusters at the next lower layer (see Figure 2) its "distance" from each channel+bitrate (v, b), denoted $d(v, b)$, representing how many hops away it is from a cluster that is subscribed to v at bitrate b; if a cluster is already subscribed to v at bitrate b, then $d(v, b)$ at that cluster is 0. Recall that we focus on live video, thus caching is largely unhelpful; clusters only advertise videos they are currently receiving.

When a cluster receives these distance values, it stores them in a table (see Figure 8) along with the available capacity of the bottleneck link on the path to that cluster $c(v, b)$. The cluster propagates the distance to the closest subscribed cluster with enough path capacity for this bitrate downwards, similar to a distance vector protocol.

Distance & Capacity Table

For Node A	Via X	Via Y	Via Z
To v_0,b_1	1, 5000	1, 1500	2, 4500
To v_1,b_1	2, 2000	1, 1500	2, 4000
To v_2,b_1	2, 5000	1, 1500	1, 3000

Figure 8: Example of the distributed state table used in Algorithm 1.

Reacting to large changes: If local discovery has detected significant changes in the local network state or viewership used to calculate the most recent central decision (i.e., the "evidence" in the RIB entry), it concludes that its current central forwarding strategy is out of date. Specifically, a cluster considers a RIB entry stale if one or more of the following conditions are met:

- A link referenced in the evidence changes capacity by some percentage (e.g., 20%) set by the operator.
- A link, node, or controller fails, as detected by a timeout.
- It receives a request it doesn't have a FIB entry for.

Input: request for channel v, bitrate b
Output: next-hop cluster for channel v, bitrate b

```
/* randomly pick a parent that has a
   min-hop path to (v,b) with enough
   capacity to support delivery          */
useful := ∅
for parent in parents do
    if d(v,b)_via_parent == min(d(v,b)) and
      c(v,b)_via_parent > b then
        useful = useful ∪ {parent}
    end
end
return pick_at_random(useful)
```

Algorithm 1: Distributed control algorithm.

If the global "evidence" is deemed invalid, a forwarding strategy is computed by Algorithm 1, using local request and link information as well as the distributed state from upper nodes (Figure 8).

For example, when a cluster receives a request for a video it's not subscribed to, it uses its table to forward the request to the parent "closest" (based on "distance" $d()$ values) to the video that has enough spare path capacity ($c()$). If there are no paths available the request is denied, to be serviced by a different edge cluster. It breaks ties randomly to avoid groups of clusters potentially overloading a high capacity cluster after failure. If the parent is not already subscribed to the video, the process repeats until a subscribed cluster is found. The algorithm produces a forwarding strategy that VDN places in the RIB and FIB of the cluster for future use (Figure 7). Large-scale link variations, link failures, and node failures, can all be handled by treating the existing videos as new requests.

Discussion: The algorithm ensures that video streams that deviate from global control only traverse paths with enough spare capacity to support them. This is critical because it means that (1) if the parent of a cluster is already subscribed to the requested video (and has ample bandwidth to the requesting cluster), requests to this cluster will not propagate past the parent (i.e., 1 hop), (2) more generally, in an n-level CDN (where n is typically 3 today), only $n-1$ clusters are affected by network / viewership changes as clusters only forward to parents on a path with enough capacity, always reaching source nodes after $n-1$ hops, and (3) clusters that are involved in this algorithm will not be forced to degrade the quality of an existing stream, as we know there is enough capacity on the path to support the incoming request. Thus, the distributed algorithm will not interfere with central control's already implemented decisions.

Note, through the use of local/global discovery, the central controller will eventually become aware of new requests and failures. By checking evidence in the RIB, clusters will know when central control has "caught up" to the current network state at which point they make the past local decisions obsolete.

4.3 Issues in the wide area

Handling state transitions: When requests are sent up the distribution tree for a given channel, they are tagged with the version number from the RIB. VDN keeps previous versions of the FIB ("shadow FIBs") to allow clusters to forward requests with old version numbers (e.g., during global state transitions), similar to previous work [20, 28, 34]. When an unknown version is encountered, VDN resorts to using distributed control.

Partitions: Versioning helps with network partitions, where some clusters no longer receive updates from the central controller. Clusters that are partitioned ("invisible" clusters from the controller's perspective) can still interact with "visible" clusters by using these old version numbers. Eventually the partitioned clusters will switch to exclusively distributed control after they detect that they're partitioned (e.g., after a controller timeout). As distributed control and central control interact in beneficial ways, partitions are also not a problem.

Loops: Our system cannot have loops as requests only travel "upwards" towards sources, and responses "downwards" towards ASes in our hierarchy.

5 Centralized optimization

This section describes our optimization algorithm that maximizes the overall service VDN delivers to each video channel while minimizing cost. Our algorithm takes in video requests and topology information and outputs the best way to distribute those videos. While the formulation is relatively straightforward, the easiest way to achieve scalability is to eschew finding the true optimal solution in favor of finding a good approximately optimal solution that can be computed relatively fast.

$$\max\ w_s * \sum_{l \in L_{AS}, o \in O} \text{Priority}_o * \text{Request}_{l,o} * \text{Serves}_{l,o} - w_c * \sum_{l \in L, o \in O} \text{Cost}(l) * \text{Bitrate}(o) * \text{Serves}_{l,o}$$

subject to:

$$\forall l \in L, o \in O : \text{Serves}_{l,o} \in \{0,1\}$$

$$\forall l \in L : \sum_o \text{Bitrate}(o) * \text{Serves}_{l,o} \leq \text{Capacity}(l)$$

$$\forall l \in L, o \in O : \sum_{l' \in \text{InLinks}(l)} \text{Serves}_{l',o} \geq \text{Serves}_{l,o}$$

Figure 9: Integer program at the controller.

The optimization is called iteratively (around once a minute) allowing parameters (e.g., measured capacities, link costs, new requests) to be modified each iteration.

Input: *Videos:* We denote a set of live video channels as $V = \{v_1, \ldots, v_k\}$. Each video channel v has its own set of bitrate, B_v. Our system treats each item in $V \times B$ as a distinct video object. We denote the set of video objects as $O = \{o_1, \ldots, o_m\}$. Bitrate(o) is the bitrate of the video object o in Kbps. Every video object o has a priority weight associated with it, $\text{Priority}_o > 0$, set by operators indicating how important it is to serve o.

Topology: Our network topology (see Figure 10a) is a directed graph made of server clusters (sources, reflectors, and edges as explained in §2) and ASes, connected by links $\{l_1, \ldots, l_n\} \subset L$ in a four-tier topology. We assume each video object is available at each source cluster (not unreasonable [35], but not fundamental). We add additional *dummy* links out of every AS node in the graph [1]. We refer to this set of dummy links as $L_{AS} \subset L$. For some link $l = (s, s')$, InLinks(l) is the set of incoming links to s.

Link capacities: Each link $l \in L$ has a capacity defined by Capacity(l), in Kbps. This capacity is the measured amount of capacity of the overlay link available to video delivery (i.e., the overall path capacity minus background traffic), which is updated by information from local discovery.

Link costs: Additionally, each link $l \in L$ has a cost defined by Cost(l) indicating the relative price for delivering bits over that link. This cost can vary over time (i.e., updated between iterations of the ILP) as updated by management (e.g., after business negotiations a link is perhaps free: $Cost(l) = 0$; perhaps cost varies based on usage, such as "95-percent-rule" billing; or even more complicated policies such as a cap on total externally-bound traffic, etc.).

Requests are associated with a link in L_{AS} (i.e., a requesting AS) and a video object. For some link $l \in L_{AS}$ associated with an AS a, if a request for video o originates from a then $\text{Request}_{l,o} = 1$, else $\text{Request}_{l,o} = 0$.

Weights: The system operator provides a global weight for cost $w_c \geq 0$ and a global weight for service $w_s \geq 0$ to strike a balance between service and cost.

Formulation: Figure 9 presents our problem formulation. The optimization takes the following as input (and treats them as constants): w_s, w_c, Priority, Request,

[1] This is a common technique in optimization to make the formulation easier.

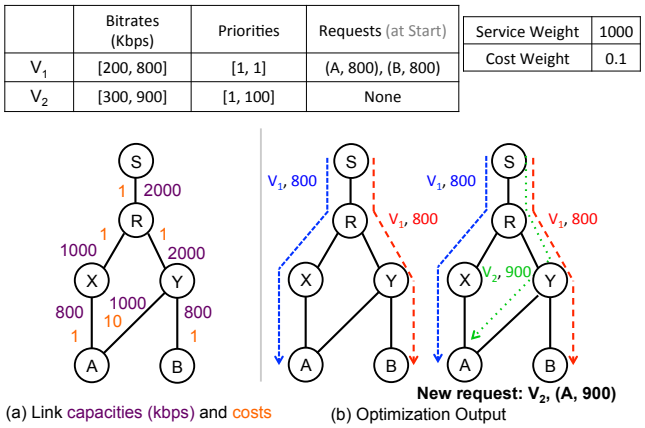


Figure 10: Example input and output of the centralized optimization.

Cost, Bitrate, Capacity, and InLinks. It outputs variables $\text{Serves}_{l \in L, o \in O} \in \{0, 1\}$, which indicates whether video object o should be distributed over link l.

Our objective function directly maximizes service, while simultaneously minimizing its cost (i.e., max: service − cost). We model service as $\sum_{l \in L_{AS}, o \in O} \text{Priority}_o \cdot \text{Request}_{l,o} \cdot \text{Serves}_{l,o}$. Thus we only serve videos objects to ASes that requested them, with the biggest wins coming from higher priority video objects. Service is only improved if a requested video reaches its destined AS. As for priority, we explore various schemes (exploring the quality/quantity tradeoff) in §7. We model cost as $\sum_{l \in L, o \in O} \text{Cost}(l) \cdot \text{Bitrate}(o) \cdot \text{Serves}_{l,o}$, the amount of data being transferred around (and out of) the CDN times the link costs.

Our constraints encode the following:

1. A link either does or doesn't send a video.
2. Obey the link capacity constraint.
3. Only send videos you've received.

Output: $\text{Serves}_{l,o}$, determines a set of distribution trees for every requested video. This can be easily translated into forwarding tables for incoming requests within the CDN internal network, and DNS records for mapping clients to edge clusters.

Example: Figure 10 gives an example input with two channels V_1 and V_2, with bitrate streams of [200, 800] and [300, 900] kbps respectively. We see that the operator has decided that video object (V_2, 900) has a very high priority (100)—this may be a stream viewers pay to watch (e.g., a pay-per-view sports event). Figure 10a shows the topology, link capacities, and costs. Link YA has a relatively high cost of 10. Figure 10b shows the optimization result in which two requests for V_1 are satisfied. Note, the optimization avoids using the high cost YA link, even though it would have cut down the total number of links used, reducing redundant data transmissions. Once a third request is added (for the high priority stream V_2), we observe that YA is used, as the video's priority outweighs the high link cost.

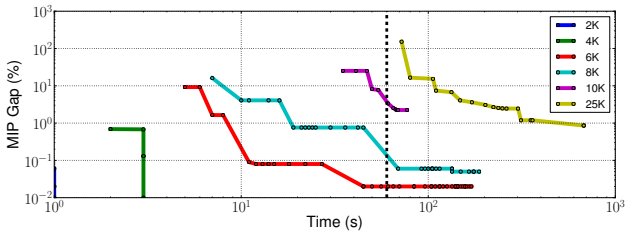


Figure 11: The MIP gap of the centralized optimization shows rapid improvement in a short time-frame, even for large numbers of videos.

Approximating optimality: An integer program can take a very long time to find an optimal solution. We employ two techniques (initial solutions and early termination) for fast approximation.

Often a good initial solution can dramatically reduce the integer program runtime. Although it's tempting to reuse the previous central decision as the initial solution for the next iteration, our formulation changes enough (e.g., new link capacities, video requests, etc.) per iteration that our previous decision may no longer be valid. Thus, we instead we calculate an initial solution greedily.

Another important parameter of integer programs is the termination criteria. Often integer programs will find a feasible solution very quickly that is only slightly worse than optimal, then spend many minutes working towards the optimal solution. This time/quality tradeoff guides our decision to use a timeout to terminate our optimization. In Figure 11 we plot the *MIP gap*[2] as a function of computation time for differing numbers of videos. We see that for all series up to our target scale of 10,000 videos (see §2.2), a 60 second timeout can provide an almost optimal solution (e.g., ~1%)). Although 60 seconds may seem like a long timescale for optimization with respect to view duration, live video viewers watch on average 30 minutes per session [1], making this a reasonable target.

6 Prototype implementation

Control plane: We build a prototype central controller that uses Gurobi [21] to solve the integer program. For trace-driven experiments, we run the controller on an r3.8xlarge EC2 instance [8]. For end-to-end experiments, since our testbed is smaller, we run the controller on a machine with a 2.5GHz quad-core Intel Core i5 processor with 4GB of RAM. For these experiments, our controller communicates with data plane nodes over the public Internet, with ~10ms latency. We believe this to be representative of a real-world deployment.

Data plane: We also build a prototype data plane using Apache [16] running on t2.small EC2 instances. Our data plane uses standard Apache modules: *mod_proxy* configures nodes as reverse HTTP proxies and *mod_cache*

[2]The distance between the current upper and lower bounds expressed as a percentage of the current upper bound

gives us multicast-like semantics. The use of Apache is representative of a real-world deployment as modern live video streaming is HTTP-based. Since these nodes communicate with the controller across the WAN, we see realistic cross-traffic, loss, and delays representative of a real-world deployment.

7 Evaluation

We evaluate VDN in two ways: a trace-driven evaluation of the central optimization focusing on the quality/cost tradeoff and scalability; and an end-to-end wide-area evaluation to test the responsiveness and performance of hybrid control in the presence of variation and failures in real-world environments.

7.1 Trace-driven evaluation

We answer three questions:

1. *Does VDN improve video quality and reduce cost?* VDN improves the average bitrate at clients by 1.7× in heavy-tail scenarios and can reduce cost by 2× in large-event scenarios over traditional CDNs.
2. *How does VDN scale? How sensitive is VDN to the network topology?* We scale VDN's control plane to 10K videos and 2K edge clusters and see it performs well even with low topological connectivity.
3. *How much control do operators have over VDN?* The knobs offered by VDN are sensitive enough for operators to fine-tune the quality/cost tradeoff and distribution of service over bitrates and videos.

Traces: We evaluate the efficacy of our controller with three traces representative of common workloads:

- Average Day: A one-hour trace from a service provider with detailed client-side analytics for a large number of live video providers. It is comprised of 55,000 requests for 4,144 videos from 2,587 cities (18,837 clients) and an average request bitrate of 2725 Kbps. This trace has a long tail: 7% of the videos account for 50% of the requests. This represents an average day for a low-demand live video service.
- Large Event: A partially synthetic trace made by adding four concurrent sports games with 1 million simultaneous viewers each to Average Day. It is comprised of 48M+ requests for 4,148 videos from 2,587 cities (4M clients) and an average request bitrate of 2725 Kbps. This trace has a very heavy head: 99.89% of requests are for one of the sports games. This represents a heavy (but easily coordinated) load. Although the requests are synthesized, the request bitrate and arrival times maintain the same distribution as the raw trace.
- Heavy-Tail: A synthetic trace generated from Average Day imposing a heavy tail distribution with narrower bitrate variety. It is comprised of 240,000 requests for 10,000 videos from 2,587 cities (82,000 clients) and an average request bitrate of 6850 Kbps. This trace has a heavy tail: the lowest 99% of videos (the tail) account for 60% of requests. Bitrates are drawn from the recommended 240p, 480p, 1080p (400, 1000, 4500 Kbps) guidelines from YouTube live [44], with an additional bitrate of 30 Mbps representing future 4K streams. This represents a heavy load that is hard to coordinate, akin to Twitch or Ustream. Although this trace is synthesized, the mapping of clients to cities and the request arrival times maintain the same distributions as the raw trace.

Topology: The traces contain no information about the internal CDN topology, so we generate a three-tiered topology similar to current CDNs [35] consisting of 4 source clusters, 10 reflectors, and 100 edge clusters. Akamai has roughly 1,800 clusters (1,000 networks) located worldwide [17], so this is roughly the right scale for *US-wide* distribution. We push the scale of the topology up to 2,000 clusters in some experiments. We use a "hose model" to determine link capacities in our overlay network. Each source is given 1 Gbps to split between 100 Mbps overlay links to each of the 10 reflectors. Each reflector has 3 Gbps to split into 100 Mbps overlay links to 30 of the 100 edge clusters. Each edge cluster is given 9 Gbps to connect to clients. We chose these capacities based on the high cost of long-haul WAN links (see §2.1).

Our prototype considers requests at the granularity of *client groups*, which we define to be *(city, AS)* pairs; we assume caching and/or multicast with a client group can efficiently distribute videos to individual users. Edge clusters are randomly placed in the 100 largest cities (determined by number of requests from that city) and each client group is connected to the 3 nearest edge clusters with 150 Mbps overlay links. (Cities in our traces are anonymized, so each city ID is randomly assigned a coordinate on a 2D grid to estimate latency.)

In addition, we assign each link a *cost*, loosely modeling a CDN's cost for transferring data on that link. Source-reflector and reflector-edge link costs vary from 10 to 50 units over a normal distribution. Links from edge clusters to client groups are handled differently: half have a cost of 0, since CDNs often strike deals with edge ISPs [2]; the remaining half vary from 1 to 5.

Methodology: We break each trace into one minute windows and compute distribution trees for each window using VDN and three additional strategies:

- Everything Everywhere (EE)—This strawman naively tries to stream all videos to all edge clusters so clients can simply connect to the nearest cluster.
- Overlay Multicast (OM)—This strawman represents an "optimal" overlay multicast-like scheme. Each video channel individually computes the distribution tree with the highest quality (found using our integer program). This is effectively VDN without coordination across channels.
- CDN—We model a DNS-based CDN that extensively monitors links and servers [2, 29, 35, 40]. As there is not public information on the specific algorithm used to produce these DNS mappings, we use the following model (based on measurement studies [32, 40] and

	EE	OM	CDN	VDN
Avg. Bitrate	624	**2,725**	**2,725**	**2,725**
Cost / Sat. Req.	174	**1.1**	1.54	**1**
Clients at BR	12%	**100%**	**100%**	**100%**

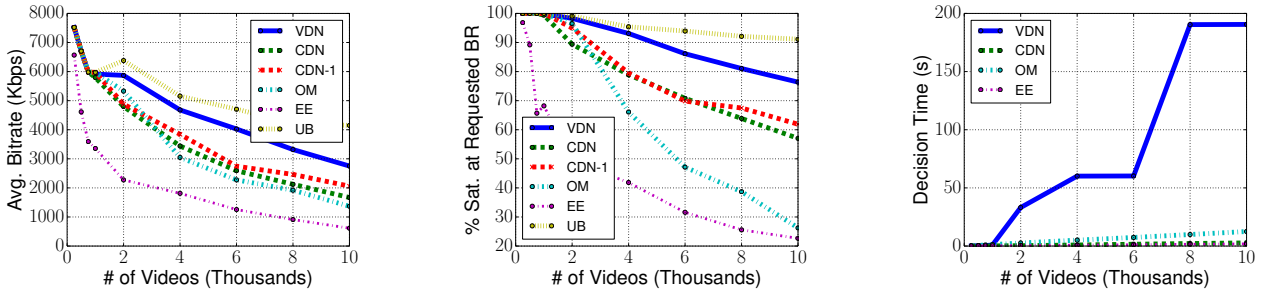
Table 1: Average Day trace.

	EE	OM	CDN	VDN
Avg. Bitrate	0.08	**2,725**	**2,725**	**2,725**
Cost / Sat. Req.	167K	**1.1**	2.0	**1**
Clients at BR	0%	**100%**	**100%**	**100%**

Table 2: Large Event trace.

	EE	OM	CDN	VDN
Avg. Bitrate	812	1,641	2068	**3,454**
Cost / Sat. Req.	7.7	4.1	**1.2**	**1**
Clients at BR	25%	34%	54%	**78%**

Table 3: Heavy-Tail trace.



(a) Avg. client bitrate. **(b) % at requested bitrate.** **(c) Processing time.**

Figure 12: Scaling load: increasing the number of videos.

a high-level description [2, 35]): upon receiving a request for a new video, a cluster picks the parent with the highest path capacity that is already subscribed to that video. If no parents are subscribed, it picks the parent with the highest path capacity. ASes are mapped to edge clusters that are geographically close (based on their city ID) and lightly loaded. Unlike OM, CDN does *not* focus on optimal end-to-end paths, just individual overlay links. This model is more fine-grained than an actual CDN as it considers each video independently [40]. CDN assumes that server selection is stored in a DNS cache with a TTL of 30 seconds [40]. We also test a variant, CDN-1, with a 1 second TTL. Note that CDN-1 would cause a large number of DNS requests, especially if combined with VDN's per-video control.

Metrics: We use three performance metrics:

- **Average Client Bitrate**—The average bitrate delivered to each client in the trace.
- *Cost / Satisfied Request (***Cost / Sat. Req.***)*—The cost of data transfer per client who receives the bitrate they request, i.e., the sum over all links of (*link cost* × *usage*) / *number of satisfied requests*.
- *% of Clients Satisfied at Requested Bitrate (***Clients at BR***)*—What percentage of the client requests were served at the bitrate they requested? (Clients not served will re-request at lower bitrates.)

7.1.1 Trace results

Tables 1, 2, and 3 summarize the results across our workloads. Each number is the average across one minute time windows in the trace. In Average Day and Large Event, VDN, CDN, and OM serve all videos at their requested quality (thus achieving the best possible average bitrate for the trace). Additionally, VDN reduces the delivery cost by 1.5-2× compared to CDN. As CDN and VDN both satisfy all clients, this decrease in cost must come from VDN finding *lower cost distribution trees* than CDN. The Large Event workload is easy to satisfy as almost all edge clusters should receive the four sports games. OM is as effective as VDN in both workloads.

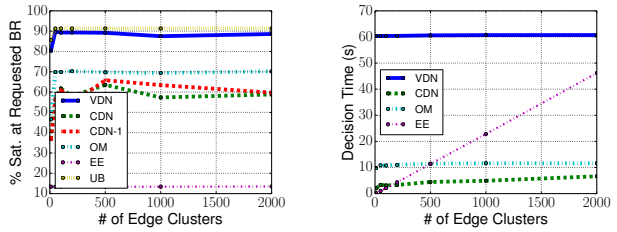
Heavy-Tail is the toughest workload to coordinate. VDN provides a 1.7× improvement in quality, while serving 24% more clients at their requested bitrate. This is because other schemes react to requests individually; DNS-based schemes like CDN get "locked in" to decisions until DNS records time-out, making it hard to coordinate streams, whereas VDN performs optimization across all requests simultaneously. With OM, the lack of coordination causes a 44% degradation in satisfied requests and a 4× increase in cost.

7.1.2 Exploring the parameter space

Next we use our traces to evaluate the control plane scalability and the topology sensitivity of VDN. Throughout, we compute naive upper bounds (UB) on "average bitrate" and "% satisfied at requested bitrate" by comparing the demand placed on each level in the topology to the aggregate capacity at that level.

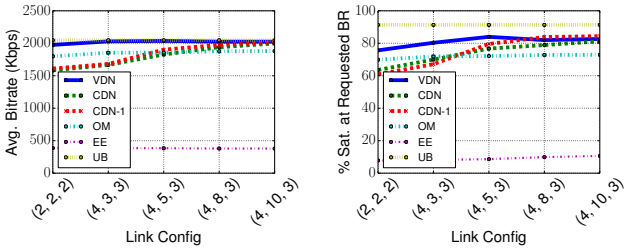
Control plane scalability: As we increase the number of videos and the size of the topology, we are interested in (1) the quality of the assignments VDN makes and (2) the time it takes to compute those assignments.

Number of videos: In Figure 12, we augment Heavy-Tail with increasing numbers of videos and requests, keeping the video/request ratio, topology, and capacity constant. As we stress the system, it becomes more difficult to place videos. Thus, coordination becomes more important with less spare capacity in the network. Since VDN considers all streams simultaneously, unlike CDN and OM, as load increases the gap between them grows in terms of both quality (up to 1.6×; Figure 12a) and the number of clients satisfied at the requested bitrate (Figure 12b). CDN-1 does marginally better



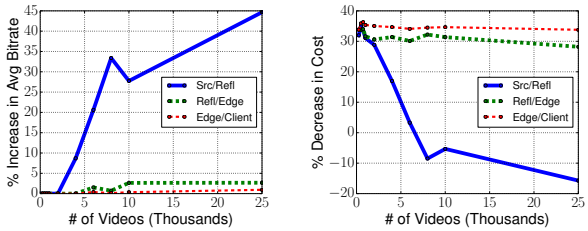
(a) % at requested bitrate. **(b) Processing time.**

Figure 13: Scaling network size: increasing the number of edge clusters.



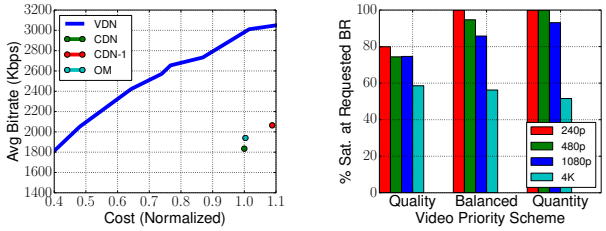
(a) Avg. client bitrate. **(b) % at requested bitrate.**

Figure 14: Topology sensitivity: (x, y, z) indicates reflectors are connected to x sources, edge clusters to y reflectors, and client groups to z edge clusters.



(a) Avg. client bitrate. **(b) Cost.**

Figure 15: Bottleneck location: improvement over CDN with the bottleneck between source/reflector links, reflector/edge cluster links, and edge cluster/client links.



(a) Quality vs. cost: the weight of the cost term is varied from 1 to 0.

(b) Quantity vs. quality: impact of video prioritization strategies.

Figure 16: VDN gives operators fine-grained control.

than CDN, but a system with asynchronous control plane updates (e.g., VDN) does substantially better, as expected. Interestingly, OM satisfies fewer clients than CDN, most likely due to OM grabbing key resources early, starving later clients.

As expected, VDN's improved assignments come at the cost of longer decision times (Figure 12c). However, in this experiment, we intentionally pushed the system outside the bounds of reality; in realistic scenarios for this topology and workload (up to 6,000 videos), decision time remains under 60 seconds (in line with §5). In the real world, if a CDN expects to serve upwards of 6,000 videos in a heavy-tail workload, we imagine the network capacity would be upgraded as well.

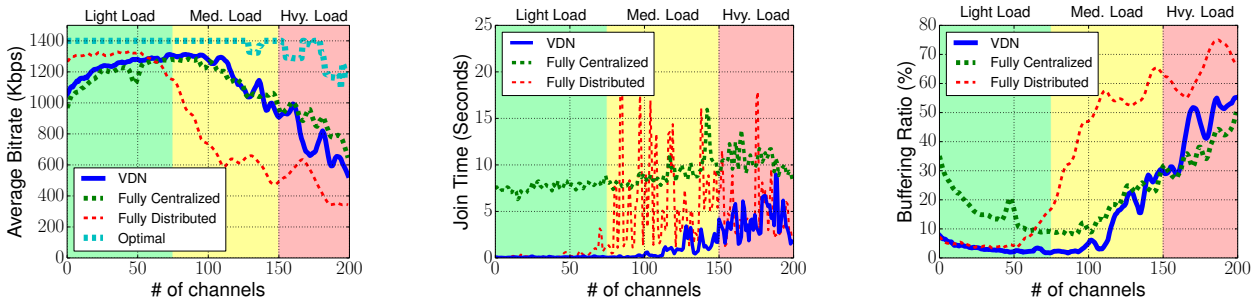
Network size: We expand Average Day to 10K videos (to increase demand) and vary the number of edge clusters (Figure 13). We see that VDN maintains the ability to satisfy roughly 90% of clients at their requested bitrate (effectively the naive upper bound) in under 60 seconds for this workload (as opposed to Heavy-Tail, which required 190 seconds; Figure 12c).

Topology sensitivity: Next, we explore the impact of the network topology on VDN, CDN, CDN-1, OM, and EE. We vary two aspects of the topology: (1) the degree of connectivity between tiers of the CDN and (2) the aggregate network capacity between tiers.

Network connectivity: Figure 14 shows the impact of network connectivity. As we increase the number of links between tiers, we decrease their individual capacities so the aggregate capacity between those tiers remains constant. In general, a *less* connected topology is going to be *easier* to manage as fewer links potentially means fewer failures. CDN performs better in highly connected topologies, likely because it has more opportunity to find upstream neighbors that already have the video they're looking for. VDN, on the other hand, is not significantly affected; it is able to effectively use a small number of large links. OM does not benefit from more connectivity as it focuses on path quality rather than link quality.

Bottleneck placement: We evaluate the impact of the location of capacity bottleneck. We begin with the topology described in §7.1; the aggregate capacity from sources to reflectors is 4 Gbps, from reflectors to edge clusters is 30 Gbps, and from edge clusters to client groups is 900 Gbps (denoted $(4, 30, 900)$ and named Source Constrained). We now construct two additional topologies: Reflector Constrained $(400, 30, 900)$ and Edge Constrained $(4000, 3000, 900)$.

Figure 15 shows VDN's percentage improvement over CDN as a function of number of videos (generated from Average Day). We see the largest gains in Source Constrained; we expect this scenario to be the most realistic since their long-haul links are more expensive than links at the edges (as pointed to by Akamai [2, 7]). In all three cases, VDN improves average bitrate (Figure 15a). It also reduces cost up through 6,000 videos (Figure 15b), at which point (in Source Constrained) it slightly increases cost in favor of 28%-45% quality improvements—next, we discuss how to explicitly control this tradeoff.



(a) Avg. client bitrate. **(b) Join time.** **(c) Buffering ratio.**

Figure 17: Client-side quality in testbed: increasing the number of videos.

7.1.3 *Customizing VDN*

Quality vs. cost: By adjusting the weight of the global cost term in the objective function, operators can tune the quality/cost tradeoff. Figure 16a shows an ROC-like curve depicting the average bitrate and data transfer cost in Heavy-Tail as the weight of the cost term varies from 1 to 0. For comparison, we plot CDN, CDN-1, and OM's performance on the same trace. VDN achieves about a 1.7× increase in performance over CDN for the same cost (1.5× over OM and CDN-1), or can reduce cost by 60% at similar quality.

Quality vs. quantity: VDN allows operators to assign each $(video, bitrate)$ pair a priority. We test three priority assignment strategies: Quality ($priority = bitrate$), Balance ($priority = 1$), and Quantity ($priority = 1/bitrate$). Quality favors serving high bitrate streams; Quantity favors serving as many streams as possible. Figure 16b shows the percentage of satisfied requests for each strategy broken down per bitrate for Heavy-Tail. Quality favors the 4K streams and Quantity favors sending more streams overall, as expected. This allows operators to not only control how things are delivered, but what is delivered (e.g., ensuring premium customers *always* receive HD streams even when many free customers request SD streams).

7.2 End-to-end experiments

We answer two questions:

1. *Is VDN highly responsive?* VDN reacts to events at a timescale of 200 milliseconds while staying within 17% of the optimal decision.
2. *Does VDN cope well with the issues of a wide-area environment?* Hybrid control allows VDN to function well despite losing controller updates and performs similarly to other schemes during high link fluctuations (traffic dynamics).

Setup and topology: We use 10 co-located nodes on EC2 [8], each representing a cluster, configured in a three-tiered CDN topology (described in §2). Two nodes are sources, another two are reflectors, and the remaining six are edge clusters. Each tier is fully connected to the next one, with measured link capacities of 75 Mbps. The controller is located outside of EC2 in the eastern US, communicating with EC2 via the public Internet.

Methodology and traffic: To demonstrate the benefits of hybrid control, we compare VDN to two other designs. Fully Distributed relies entirely on the distributed control algorithm in §4 and Fully Centralized uses only the centralized controller in §5. For each experiment we generate 200 videos, each requested by one client to a random edge cluster. Each channel has multiple bitrates: 200 Kbps, 600 Kbps, and 1.4 Mbps. We add a new channel to the system once a second. With 10 nodes and 75Mbps links, 100 videos can easily place great load on the system. 100 videos * 1.4Mbps is ~150 Mbps, filling two of the four source/reflector links. 200 videos would fill all four links, overloading the system. The video client is a simple HTTP chunk requester that always fetches a new chunk 2 seconds (the chunk duration) after the previous chunk was received.

7.2.1 *Quality of experience*

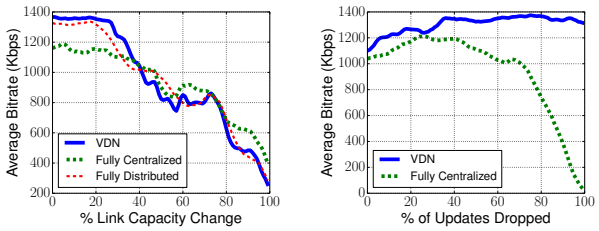
Figure 17a shows the average client bitrate as requests are added. Similar to the trace-driven evaluation, in a system with medium load, VDN gives up to a 2x performance gain over Fully Distributed. As the system becomes more loaded, Fully Distributed sharply drops while VDN and Fully Centralized decay gradually. Even when the system is under medium loaded, VDN stays close to the controller's original decision (Optimal). Once the system reaches heavy load (~150 videos), other problems emerge (e.g., connection establishment overhead, request incast) causing performance to decay.

In Figure 17b, we see that VDN is highly responsive. Although Fully Centralized provides good average bitrate during load, its join time (time from request to first byte of video) suffers (~7 seconds, compared to VDN's ~200 milliseconds). Fully Distributed also provides sub-second join times, but as the system gets loaded, it sees massive spikes in latency as the lack of coordination overloads interior clusters.

Figure 17c shows buffering ratio. Despite having good quality overall, Fully Centralized has a much worse buffering ratio due to its lack of responsiveness.

7.2.2 *Coping with network events*

Figure 18a shows the effects of link fluctuations. We select 25% of links at random, degrade their capacity by increasing amounts (using `tc`), and measure the perfor-



(a) Link fluctuation. **(b) Updates dropped.**

Figure 18: VDN handles network issues without much degradation.

mance 10 seconds after adding 10 channels. We see that all three systems perform similarly.

Figure 18b shows the effects of loss. We drop updates from the controller and measure the performance 10 seconds after adding 10 channels. As expected, Fully Centralized performs much worse as updates are dropped. VDN performs well even when it starts to lose all update messages by falling back to distributed control.

8 Discussion

Complexity versus improvement: Despite the inherent complexity of hybrid control, VDN manages to provide a significant monetary benefit (2×) to CDN operators as well as increased flexibility (see Figure 16a and 16b). Additionally, VDN provides a centralized point of management to adjust link costs and video priorities. Furthermore, as seen in §7, simple tweaks on current CDNs, like shorter TTLs, don't provide these benefits.

Alternate topologies: We assume an *n*-tiered topology as we feel this is representative of modern CDNs [2, 29, 35, 40]. Additional work would be needed to fit our scheme to arbitrary topologies.

Client-side bitrate adaptation: Although not explicitly included in our system, we assume clients independently do bitrate adaptation through some black-box assessment of delivery quality. Distributed control allows VDN to quickly respond to bitrate switching, but we assume that the rate of switching is fairly low [9].

9 Related work

Content delivery networks: Large- (e.g., [29, 35]) and medium-scale (e.g., [18, 42]) CDN systems have explored various design choices, including peer-to-peer, hybrid [22, 43], centralized, or hierarchical architectures [24] as well as their tradeoffs [45]. None of these papers provides the key combination of global coordination, video-specific optimization, cost-minimization, attention to live-video specific issues, and practical end-to-end system design.

Overlay multicast: Prior work on providing the sustained high-throughput connections needed for live video [12, 14, 26, 30] focuses on how to best organize individual streams. However, they do not perform extensive coordination across video streams. P2P-based approaches [30] can potentially benefit VDN, but may cause additional issues with hybrid control (e.g., loops) as they complicate the topology.

Traffic engineering: Recent work [13, 15, 23, 25] shows the benefits of centralized traffic engineering in ensuring high utilization and fairness in both intra- and inter-datacenter settings. Unlike VDN, they work on flow aggregates at coarse timescales, making it hard for them to provide the fine-grained dynamic control required for live video.

Video optimization: There is much prior work on understanding and improving video delivery, including client-side bitrate adaptation [27], metrics [9, 10], cross-CDN optimization (e.g., [32]), and CDN-selection strategies (e.g., [19]). Our work focuses on end-to-end delivery and provides a practical system design.

10 Conclusion

VDN is a platform for live video delivery that helps balance the concerns of both users and CDN operators by providing real-time control over individual streams from the CDN side. VDN employs centralized quality optimization and hybrid control for responsiveness. We show that centralized optimization can greatly improve video quality while minimizing cost. Our hybrid control plane mitigates WAN challenges, providing quick join times and responsiveness to failures. Using a live video trace, we show that VDN provides a 1.7× improvement in average bitrate and a 2× reduction in delivery cost in different scenarios. Using Amazon EC2, we show that our design is responsive at a timescale of 200 ms.

Acknowledgments

The authors would like to thank Nicolas Feltman for help with the ILP, Eric Anderson and Raja Sambasivan for help with distributed control, JungAh Hong for help with the initial evaluation, Dave Oran for shepherding this paper, and the anonymous reviewers for their feedback. This work is supported in part by the NSF under award #CNS-1345305, NDSEG Fellowship 32 CFR 168a, the National Research Foundation of Korea (NRF-2013R1A1A1076024), and the IITP under grant No. B0126-15-1078 funded by the Korean Government (MSIP).

11 References

[1] Ooyala global video index q3 2013. http://go.ooyala.com/rs/OOYALA/images/Ooyala-Global-Video-Index-Q3-2013.pdf.

[2] Private conversation with Bruce Maggs, vice president, research at Akamai.

[3] Private conversation with Hui Zhang, chief executive officer, at Conviva.

[4] Twitch. http://twitch.tv.

[5] Twitch is 4th in peak us internet traffic. http://blog.twitch.tv/2014/02/twitch-community-4th-in-peak-us-internet-traffic/.

[6] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia* (2011).

[7] Akamai. Akamai investor summit: 2013. http://www.akamai.com/dl/investors/2013_ir_summit_presentation.pdf.

[8] Amazon. Amazon Elastic Compute Cloude (Amazon EC2). http://aws.amazon.com/ec2/.

[9] Balachandran, A., Sekar, V., Akella, A., Seshan, S., Stoica, I., and Zhang, H. A quest for an internet video quality-of-experience metric. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2012), HotNets-XI, ACM, pp. 97–102.

[10] Balachandran, A., Sekar, V., Akella, A., Seshan, S., Stoica, I., and Zhang, H. Developing a predictive model of quality of experience for internet video. In *Proc. ACM SIGCOMM* (2013), ACM, pp. 339–350.

[11] Bashore, A. Twitch stats. http://stats.twitchapps.com/.

[12] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., and Singh, A. Splitstream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 298–313.

[13] Chowdhury, M., Zaharia, M., Ma, J., Jordan, M. I., and Stoica, I. Managing data transfers in computer clusters with orchestra. *SIGCOMM CCR 41*, 4 (2011), 98.

[14] Chu, Y., Rao, S., Seshan, S., and Zhang, H. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM computer communication review 31*, 4 (2001), 55–67.

[15] Fortz, B., Rexford, J., and Thorup, M. Traffic engineering with traditional ip routing protocols. *Communications Magazine, IEEE 40*, 10 (2002), 118–124.

[16] Foundation, A. Apache HTTP Server Project. http://httpd.apache.org/.

[17] Frank, B., Poese, I., Lin, Y., Smaragdakis, G., Feldmann, A., Maggs, B., Rake, J., Uhlig, S., and Weber, R. Pushing cdn-isp collaboration to the limit. *ACM SIGCOMM CCR 43*, 3 (2013).

[18] Freedman, M. J. Experiences with coralcdn: A five-year operational view. In *Proc. USENIX NSDI* (2010).

[19] Ganjam, A., Siddiqui, F., Zhan, J., Liu, X., Stoica, I., Jiang, J., Sekar, V., and Zhang, H. C3: Internet-scale control plane for video quality optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 131–144.

[20] Ghorbani, S., and Caesar, M. Walk the line: consistent network updates with bandwidth guarantees. In *Proc. HotSDN* (2012), ACM, pp. 67–72.

[21] Gurobi. Gurobi optimization. http://www.gurobi.com/.

[22] Han, D., Andersen, D., Kaminsky, M., Papagiannaki, D., and Seshan, S. Hulu in the neighborhood. In *Proc. COMSNETS* (Jan. 2011), pp. 1 –10.

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

[24] Huang, C., Wang, A., Li, J., and Ross, K. W. Measuring and evaluating large-scale cdns. In *Proc. ACM IMC* (2008).

[25] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al. B4: Experience with a globally-deployed software defined wan. In *Proc. ACM SIGCOMM* (2013).

[26] Jannotti, J., Gifford, D. K., Johnson, K. L., Kaashoek, M. F., et al. Overcast: reliable multicasting with on overlay network. In *Proc. 4th conference on Symposium on Operating System Design & Implementation* (2000).

[27] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. ACM CoNEXT* (2012).

[28] Katta, N. P., Rexford, J., and Walker, D. Incremental consistent updates. In *Proc. HotSDN* (2013), ACM.

[29] Kontothanassis, L., Sitaraman, R., Wein, J., Hong, D., Kleinberg, R., Mancuso, B., Shaw, D., and Stodolsky, D. A transport layer for live streaming in a content delivery network. *Proceedings of the IEEE 92*, 9 (2004), 1408–1419.

[30] Kostić, D., Rodriguez, A., Albrecht, J., and Vahdat, A. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 282–297.

[31] Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (May 1998), 133–169.

[32] Liu, X., Dobrian, F., Milner, H., Jiang, J., Sekar, V., Stoica, I., and Zhang, H. A case for a coordinated internet video control plane. In *Proc. ACM SIGCOMM* (2012), pp. 359–370.

[33] Liu, Y., Zhang, H., Gong, W., and Towsley, D. On the interaction between overlay routing and underlay routing. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE* (2005), vol. 4, IEEE, pp. 2543–2553.

[34] McGeer, R. A safe, efficient update protocol for openflow networks. In *Proc. HotSDN* (2012), ACM, pp. 61–66.

[35] Nygren, E., Sitaraman, R. K., and Sun, J. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[36] Prasad, R., Dovrolis, C., Murray, M., and Claffy, K. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE 17*, 6 (2003), 27–35.

[37] Sandvine. Global internet phenomena report: 1h 2014. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf.

[38] Spangler, T. World cup sets new internet-video streaming records for espn, univision, and akamai. http://variety.com/2014/digital/news/world-cup-sets-new-internet-video-streaming-record-1201221997/.

[39] Strauss, J., Katabi, D., and Kaashoek, F. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2003), IMC '03, ACM, pp. 39–44.

[40] Su, A.-J., and Kuzmanovic, A. Thinning akamai. In *Proc. ACM IMC* (2008).

[41] Team, T. How twitch fits in amazon's strategy. http://www.forbes.com/sites/greatspeculations/2014/08/28/how-twitch-fits-in-amazons-strategy/.

[42] Wang, L., Park, K., Pang, R., Pai, V. S., and Peterson, L. L. Reliability and security in the codeen content distribution network. In *Proc. USENIX ATC, General Track* (2004).

[43] Xu, D., Kulkarni, S. S., Rosenberg, C., and keung Chai, H. A cdn-p2p hybrid architecture for cost-effective streaming media distribution. *Computer Networks 44* (2004), 353–382.

[44] YouTube. Live encoder settings, bitrates and resolutions. https://support.google.com/youtube/answer/2853702?hl=en.

[45] Yu, M., Jiang, W., Li, H., and Stoica, I. Tradeoffs in cdn designs for throughput oriented traffic. In *Proc. ACM CoNEXT* (2012), ACM, pp. 145–156.