C3: Internet-Scale Control Plane for Video Quality Optimization

Aditya Ganjam[†]*, Junchen Jiang*[⋆]*, Xi Liu*[†]*, Vyas Sekar*[⋆]*,*
Faisal Siddiqi[†]*, Ion Stoica*[+†◦]*, Jibin Zhan*[†]*, Hui Zhang*[⋆†]
[†]*Conviva,* [⋆]*CMU,* [+]*UC Berkeley,* [◦] *Databricks*

Abstract

As Internet video goes mainstream, we see increasing user expectations for higher video quality and new global policy requirements for content providers. Inspired by the case for centralizing network-layer control, we present C3, a control system for optimizing Internet video delivery. The design of C3 addresses key challenges in ensuring scalability and tackling data plane heterogeneity. First, to ensure scalability and responsiveness, C3 introduces a novel split control plane architecture that can tolerate a small increase in model staleness for a dramatic increase in scalability. Second, C3 supports diverse client-side platforms via a minimal client-side sensing/actuation layer and offloads complex monitoring and control logic to the control plane. C3 has been operational for eight years, and today handles more than 100M sessions per day from 244 countries for 100+ content providers and has improved the video quality significantly. In doing so, C3 serves as a proof point of the viability of fine-grained centralized control for Internet-scale applications. Our experiences reinforce the case for centralizing control with the continued emergence of new use case pulls (e.g., client diversity) and technology pushes (e.g., big data platforms).

1 Introduction

Internet video is a significant and growing segment of Internet traffic today [2]. In conjunction with these growing traffic volumes, users' expectations of high quality of experience (e.g., high resolution video, low rebuffering, low startup delays) are continuously increasing [35, 3, 14]. Given the ad- and subscription-driven revenue model of the Internet video ecosystem, content providers strive to deliver high quality of experience while meeting diverse policy and cost objectives [4, 38].

In this respect, several previous efforts have shown that the observed video quality delivered by individual CDNs can vary substantially across clients (e.g., across different ISPs or content providers) and also across time (e.g., flash crowds) [39, 37]. Similarly, because the video player has only a few seconds worth of buffering and the bandwidth could fluctuate significantly, we need to make quick decisions (e.g., future bitrates) based on the current client buffer level and bandwidth so that the buffer does not drain out [27].

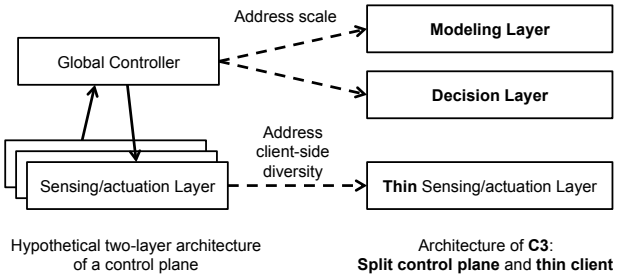


Figure 1: Conceptual two-layer architecture and the ideas of split control plane and thin client

These observations have made the case for a logically centralized control plane for Internet video that uses a *global* and *real time* view of network conditions to choose the best CDN and bitrate for a given client.[1] Furthermore, content providers have complex system-wide policy and optimization objectives such as balancing costs across CDNs or servicing premium customers, which are also difficult to achieve without a global real-time view.

Conceptually, one can consider a hypothetical two-layer architecture (the left part of Figure 1) of a control plane consisting of a global controller and a client-side sensing/actuation layer. The controller uses real-time measurements of client performance observed by the sensing/actuation layer to create a *model* of expected performance, uses this model to *decide* suitable parameters (e.g., CDN and bitrate) for the clients and sends them to sensing/actuation layer to execute. Unfortunately, realizing such an Internet-scale control plane is easier said than done! For instance, at peak load we have had to handle over 3M concurrent users and we expect this to grow by up to two orders of magnitude. This *scale* makes it challenging to maintain up-to-date global views while simultaneously being responsive to client-side events.

[1] Conceptually, this can be viewed as a management layer overlaid on top of multiple CDNs and this optimization is orthogonal to the server allocation optimizations done by the individual CDNs.

To address this fundamental scaling challenge, our solution, called C3, introduces a *split control plane* architecture that logically decouples the modeling and decision functions of the controller as shown in the right part of Figure 1. The key insight underlying this unique split control plane architecture is an observation that we can tolerate a small increase in amount of staleness in the global modeling for a dramatic increase in scalability; i.e., our decisions will be close to optimal even if the global view used for decision making is out of date by a few minutes (§3).

Building on this insight, the **modeling layer** operating at a *coarse time granularity* is updated every tens of seconds or minutes to build a global model based on global view of client quality information. The **decision layer** makes real-time decisions using the global model from the modeling layer at a sub-second timescale in response to client-side events; e.g., arrivals or bandwidth drops or quality changes. Note that in contrast to traditional web serving architectures, the decision layer is not a dumb replicated caching layer but is actively making real-time decisions merging global (but stale) models with local (but up-to-date) data. An immediate consequence of the decoupling is that the decision layer interfacing with the clients is *horizontally scalable*.

Now, for any such control architecture to be effective, we need a sensing/actuation layer to (a) accurately measure video quality from video players and (b) execute the control decisions. Here, we observe a practical challenge due to *client-side diversity*. For instance, we see close to 100 distinct application combinations of framework (i.e., providing libraries to support video players) and streamer (i.e., the module responsible for downloading and rendering video). The diversity of video players coupled with practical challenges in players' long software update cycles makes it difficult to implement new measurement techniques or control algorithms. To address this client-side diversity, we make an explicit choice to make the sensing/actuation layer functionality as minimal as possible. Thus, we eschew complex control and data summarization logic in the video players in favor of a very *thin* sensing/actuation layer that exports raw quality-related events using a common data model (§4). These designs simplify adding support for new content providers, accelerate testing and integration, and also enable independent evolution of client-side platforms and C3's control logic.

Over the 8 years of operation, C3 has optimized over 100M sessions each day from over 100 name brand content providers. Our operational experience and microbenchmarks confirm that: (1) C3 controller is horizontally scalable; (2) our client-side sensing/actuation layer imposes small bandwidth overhead on the clients; and (3) C3 can dramatically improve the video quality of C3's customers within the bounds of global policies.

While C3 has evolved in response to video-specific technology trends and use-cases (§2), we believe that our lessons and design decisions are more broadly applicable to other aspects of network control (§7). First, we observe more drivers for centralized control due to greater client-side heterogeneity and more complex policy demands. Second, we see more enablers for centralizing control with the advent of big-data solutions and the ability to elastically scale service instances via cloud providers. Third, our journey reinforces the belief that separation of control and data and moving more functionality to the controller is a powerful architectural choice that enables rapid and independent evolution for different stakeholders.

2 Evolutionary Perspective

Operational systems such as C3 do not exist in vacuum—they have to constantly evolve in response to use case pulls (e.g., more complex provider policies and multi-CDN deployments) and technology trends (e.g., move from P2P to CDNs or RTMP to HTTP). In this section, we provide an evolutionary perspective of the 8-year operation of C3. This retrospective is useful because it gives us the context to understand both how the requirements (e.g., scale, diversity) have evolved and how our design decisions have adapted accordingly. We conclude with major trends that reinforce our decision to centralize the control plane.

2.1 Overview of evolution

We identify three high-level phases in the evolution of C3 (shown in Figure 2).

Phase I: The origins of C3 can be traced back to a very different operational context. The original C3 architecture was motivated by the problem of optimizing P2P live streaming. This was around 2006, when video streaming via CDNs was quite expensive with an effective cost of $\approx$ 40 cents/GB. At the time, P2P was widely perceived as an alternative low-cost solution. Unfortunately, existing overlay schemes were unreliable and unable to deliver high-quality streams equivalent to CDN-based performance. Inspired by concurrent work on the 4D architecture for network control [45], C3 was a centralized solution to manage the overlay tree in order to deliver high (CDN equivalent) quality streaming over P2P. This centralized view also enabled to implement simple per-stream global policies; e.g., limiting total bandwidth or number of viewers on a specific live channel.

During this early stage, most video streaming was based on Flash/RTMP and clients were largely homogeneous. They were largely desktop clients that needed to explicitly download/install our P2P client software, similar to other P2P systems at the time. This software would

Phases (Time)	Environments				Design overview	
	Video delivery	User scale	Platform	Policy	Key decisions	Major considerations
I (2006-2009)	P2P Live	100s-10Ks	Single	Per-stream	Centralized overlay-tree construction in the controller. Frequent update and control.	Modest size of users. Existing protocols not sufficient for high-quality streams.
II (2009-2011)	CDN, Live/VoD	1M-10Ms	Single	Global	Joint control: Controller changes the logic and bitrate/CDN list. Clients run real-time control.	Controller unable to support real-time control at scale. Flash supports dynamical loading plugin.
III (2011-now)	CDN, Live/VoD	10Ms-100Ms	Diverse	Global Complex	Minimal clients: push all decision making and quality summary to the controller.	Diverse client platforms, long software update cycle. Advent of big-data technology.

Figure 2: Overview of C3 evolution.

work in close coordination with the C3 controller to construct a robust overlay tree that gracefully handled user churn.

Moreover, Internet video was still in its infancy, and many premium providers were yet to step in to the market. Thus, the scale of the client demand was also relatively small. As such, C3's controller was deployed using custom server software running in dedicated datacenters. This was sufficient to provide the desired sub-second responsiveness to handle tens of thousands of clients.

Phase II: Around 2009, we saw an inflection point with several key technology and industry shifts. First, the cost of streaming using CDNs dropped significantly to ≈ 5 cents/GB. Second, many mainstream providers (e.g., iTunes, Hulu) started warming up to the potential of Internet video and started discovering monetization strategies for online video, for both live and VoD content. While Flash/RTMP still dominated as the de-facto platform for video streaming, we saw the emergence of alternatives (e.g., due to Apple refusing to support Flash). On a practical note, given that content was now being monetized, as opposed to the free video over P2P, there was some understandable reluctance from the providers to force clients to install a new client software.

These transitions had significant effects on the design of C3. First, the entry of mainstream providers meant that the workload grew several orders of magnitude from Phase I to 100s of thousands to millions users. Second, the transition to CDN-based delivery for both live and VoD meant that the C3 logic had to evolve. Specifically, the emergence of HTTP- and chunk-based video streaming protocols (e.g., [13, 8]) meant that a video client could seamlessly choose a suitable bitrate and CDN (server) at the beginning as well as in the middle of a video session with little overhead. Thus, C3 was now targeted toward the goal of better CDN and bitrate selection instead of the earlier goal of computing optimal overlay trees for live streaming.

However, our control platform was not yet mature enough to provide sub-second responsiveness at such scale. Our response in this phase was a pragmatic solution that had to sacrifice both the global view and real-time requirement to ensure the required scalability. Specifically, our solution relied on a combination of exploiting application-level resilience and clever engineering. We made a decision to coarsen the control functionality of C3. Instead of the client software, we designed a player plugin that clients would download from C3 when the video session started. This gave us coarse control wherein we could modify the player logic at the beginning of the session (e.g., choosing a CDN intelligently). For subsequent adaptation (e.g., dynamic bitrate adaptation), however, we had to rely on the local decision making and capabilities. To deal with the moderate amount of client heterogeneity, we developed custom cross-compiler techniques that allowed us to integrate our development across platforms. While this cross-compiler served us well as an interim solution, the approach soon started showing cracks as more diverse client platforms given the idiosyncrasies of different technologies.

Phase III: The current phase of C3's operation, starting in 2011, can be truly described as the coming of age of Internet video. With the ad- and subscription-driven revenue models, and the availability of rich content, many more providers and users now rely on Internet video. In fact, some industry analysts report that Internet video consumption might even exceed traditional TV.

Consequently, C3 had to evolve to once again handle 2-3 orders of magnitude increase in the client population—tens of millions clients, with 10s-100s of thousands new client arrivals per minute at peak hours. In addition, C3 now faced a more serious challenge due to client heterogeneity as we now observed very diverse client-side platforms of streaming protocols (proprietary protocols and HTTP chunking, etc), application frameworks (e.g., OSMF, Ooyala, Akamai) and devices (e.g., set-top boxes, connected TVs, tablets).

There was an independent technology shift that was synergistically aligned with these trends—the emergence of big data platforms to enable real-time processing of very large volumes of data. We embraced this technology and exploited it to enable novel solutions to handle the client-side heterogeneity. Specifically, it enabled us to make the client implementation very minimal; e.g., moving the data summarization logic originally located in the client in Phase II to the controller. This allowed us greater flexibility in adapting to new client platforms and also simplified the development cycle.

However, big-data platforms by themselves do not address the scalability challenge of providing sub-second responsiveness to client-side events for millions of clients. This required us to significantly rearchitect the control plane and motivated the split control plane architecture that we describe in the next section. Specifically, we split the controller to a geo-distributed decision layer with sub-second responsiveness exploiting the reach of cloud providers, and a consolidated modeling layer that provides minute-level freshness w.r.t. global visibility.

2.2 Major trends

The above evolution highlights two key trends that reinforce the case (in terms of both drivers and enablers) for centralizing network control:

- **Drivers:** First, we see ever increasing demands of user experience and growing complexity of the video delivery system. This naturally motivates us to move more control logic to the controller in order to use the global visibility and satisfy global policies. Second, the proliferation of diverse client platforms makes it difficult from an engineering standpoint to integrate and test the client-side logic.
- **Enablers:** With the recent advances in big data technology, we can build a backend system with unprecedented capacity to support scalable data processing in real-time with low cost. Furthermore, it is now possible to deploy the centralized control plane on-demand using cloud services with global presence. The emergence of big data platforms and cloud services can enable even greater centralized control.

In the rest of this paper, we focus on the design and implementation of the split control plane architecture and the sensing/actuation layer during this most recent phase of C3's operation.

3 C3 Split Control Plane

The goal of the C3 controller is to optimize the video quality and enforce global policies given by content providers. Note that C3 does not control the CDN servers or distribution logic. Rather it acts an additional management layer to enable content providers to achieve their quality and policy objectives on top of their existing delivery ecosystem.

There are three (arguably conflicting) goals that the C3 controller needs to meet. First, given the variability in video quality across time and space (e.g., ISP-CDN combinations) the C3 controller needs an *up-to-date global view* of network conditions to be effective in choosing a suitable CDN and bitrate for clients. Second, it needs to be *responsive* at sub-second timescales to handle new client arrivals (e.g., to minimize video startup delay) and quality-related events during video playback (e.g., drop in bandwidth or CDN congestion). Finally, and most importantly, it must be *scalable* to handle 10s-100s millions of concurrent clients.

Unfortunately, simultaneously achieving all three requirements of freshness, responsiveness and global view is hard. To see why, let us consider two strawman solutions. The first option is a single controller handling all clients. However, even state-of-the-art big data processing platforms cannot provide sub-second responsiveness with new samples arriving at the rate of 50-100 GB per minute. Even if such a system exists, there is an inherent delay to collect enough data for making decisions with high confidence; e.g., it may take minutes to infer with high confidence that a particular CDN is overloaded. A second option is to deploy replicated servers with each replica responsible for a subset of clients. Though this parallelism ensures scalability and responsiveness, the quality of the decisions will degrade as each replica will make decisions only on the partial view from its clients rather than on the global view.

Next, we present the split control plane architecture of C3 and discuss how it is crucial to simultaneously meet three key requirements—freshness, responsiveness, and global views.

3.1 Logical view

The key insight behind the split control plane is a domain-specific observation that we can slightly relax the *freshness* requirement to simultaneously achieve scalability, responsiveness, and a global view. Specifically, we observe that some global characteristics (e.g., relative rankings of CDN based on quality) are relatively stable on the order of minutes [29].

Figure 3 shows one representative result showing the *persistence* of the best CDN for clients in a given AS, which we define as the number of contiguous minute-level epochs in which this CDN has the lowest buffering ratioa cross all available CDNs. Figure 3 shows the distribution of this persistence metric across three content providers (A,B,C) that use multiple CDNs.[2] We see that the 80%ile of the persistence is 3 minutes across all three content providers.

However, such persistence does not hold for states of individual clients. For instance, when the current CDN is not available, CDN must be switched immediately to prevent the buffer from draining out (e.g.,buffer length for live videos is no more than several seconds). In this case decisions must rely on the freshest information (e.g., buffer length) to prevent quality from suffering.

The above observations on global state persistence coupled with local per-client variability make a case for a *split control plane* scheme that consists of two loosely

[2]To avoid any potential bias due to C3's control decisions, this result explicitly focuses on content providers who have not opted-in for C3's optimized control but use only the quality monitoring services.

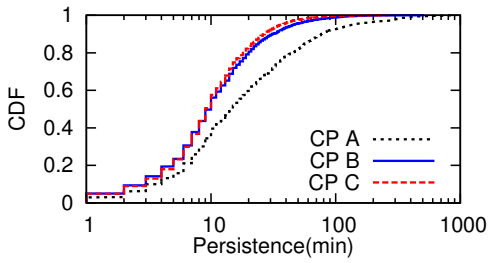


Figure 3: Distribution of the persistence of the best CDN for different content providers.

coupled stages:

- A **coarse-grained global model layer** that operates at the timescale of few tens of seconds (or minutes) and uses a global view of client quality measurements to build a data-driven prediction global model of video quality (see below).
- A **fine-grained per-client decision layer** that operates at the millisecond timescale and makes actual decisions upon a client request. This is based on the latest (but possibly stale) pre-computed global model and up-to-date per-client state.

Figure 4(a) shows how the split control plane is mapped into the two logical layers of the C3 controller. Specifically, the modeling layer implements the coarse-grained control loop (i.e., blue arrows) in a coarse timescale and trains a global model based on the measurements of each client session it receives from the decision layer. The decision layer implements the fine-grained control loop (i.e., red arrows) for each client, and makes CDN and bitrate decisions based on the global model trained by the modeling layer and latest heartbeats received from the sensing/actuation layer (see §4).

We make one important observation to distinguish the functionality of the decision layer that presents a significant departure from traditional replicated web services. Unlike traditional web services where the serving layer is a "dumb" distributed caching layer, the decision layer makes *real-time* control decisions by combining freshest quality measurements of the client under control and the global model.

This above split control plane design has two key characteristics that are critical for a *scale-out* realization of the decision layer without synchronization bottlenecks. First, note that there is a loose coupling between the fine-grained per-client and coarse-grained global control loops. Thus, we do not need the decision layer to be perfectly synchronized with the modeling layer. Second, the decision layer is operating on a per-client basis, which eliminates the need for coordination across decision layer instances. Taken together, this means that we can effectively partition the workload across clients by having a *replicated* decision layer where instances are deployed close to the clients and independently execute

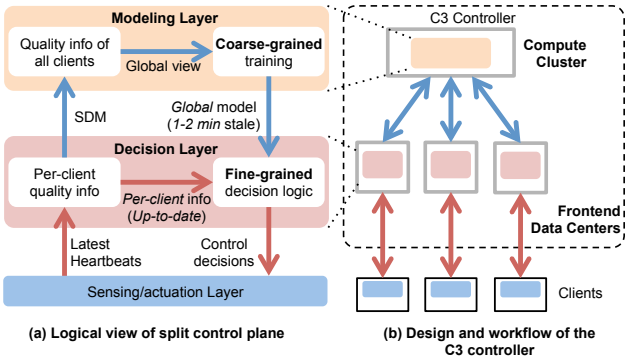


Figure 4: Overview of the C3 controller. The SDM and Heartbeats are discussed in the next section.

the logic for the subset of clients assigned to it.

3.2 End-to-end workflow

Having discussed the core ideas in the previous section, next we discuss the concrete *physical* realization of the C3 controller (shown in Figure 4(b)) and describe the end-to-end workflow.

3.2.1 Modeling layer workflow

The modeling layer is a compute cluster running a big data processing stack. The modeling layer periodically uses the information (its specific format will be introduced later in §4.2.1) collected from all clients to learn a global model that encodes actionable information useful for decision making.

Our focus in this paper is primarily on the control architecture and the design of the specific algorithms in the modeling layer is outside the scope of the paper. For completeness, we provide a high-level sketch of the algorithm. The model is similar to the nearest neighbor like prediction model suggested in prior work [39]. In particular, we leverage the insight that similar video sessions should have similar quality. Therefore, quality measurements of sessions sharing certain spatial (e.g., CDN, ISP, content provider) and temporal (e.g., time of day) features are grouped together, and intuitively, the quality of a new session can be predicted based on the quality of the most similar sessions.

To enforce global policies (e.g., traffic caps for certain CDNs), the modeling layer also includes the relevant global states as part of the global model (e.g., amount of traffic currently assigned to each CDN), so that the decision logic can take into account the global information it needs. There is a large space of potential decision logics that can take the global model, individual client state, and global policies to make optimal per-client decisions. The design of policies and algorithms to meet policy objectives is outside the scope of this paper.

The remaining question is disseminating the global model to the decision instances. Instead of a pull model

like traditional web caching, the modeling layer *pushes* the global model to each frontend data center where decision instances run. The reasons for a push rather than pull approach is to ensure that each decision instance has an up-to-date model as soon as the modeling layer recomputes the global model. The overhead of the push step is quite low since the size of the global model is 100s of MBs, which can be disseminated to all data centers in several seconds without any additional optimizations and can easily fit in the memory of a modern server. (In contrast, web caches cannot know the set of requests and have to use a pull model because they cannot store the entire content catalog.)

3.2.2 Decision layer workflow

In order to minimize the response latency between clients and their corresponding decision instances, the decision instances are hosted in geographically distributed frontend datacenters as close to the clients as possible. When C3 clients arrive, they are assigned to specific decision instances via standard load balancing mechanisms, which ensure that subsequent requests (both control requests and heartbeats) of the same client are consistently mapped to the same instances. These mechanisms operate across data centers and across decision instances, and handle geographic locality, load balancing, and fault tolerance. We use industry-standard mechanisms such as DNS-based consistent mapping of clients to instances based on latency measurements. These mechanisms use standard failure detection mechanisms to detect if a specific instance has failed and reassigns clients as needed. (As shown in §4.3, the measurement collection from clients can be easily re-synced when the decision instances are reassigned.)

The clients send periodic heartbeats (described in §4.2.1) to decision instances. Based on the heartbeats of a client, a decision instance maintains a state-machine, which provides an accurate and up-to-date view of the current video quality experienced by the client. Upon receiving a control request from a client , the decision instance runs a proprietary decision algorithm to choose a suitable CDN and bitrate.

This decision logic combines both the up-to-date per-client information and the (slightly stale) global model the decision logic, and tries to optimize video quality while operating within the bounds of global polices (e.g., load per CDN or cost). In a simple example, consider two CDNs; CDN1 provides poor quality to viewers and CDN2 provides good quality in a certain city. The decision logic is able to detect that CDN1 has worse quality than CDN2 based on the quality feedback from clients using both CDNs from that particular city, and it will then instruct new clients to CDN2.

3.3 Summary of key design decisions

In summary, we make the following key decisions.

1. To balance global visibility and data freshness, we use a split control plane mechanism with a coarse timescale modeling layer loosely coupled with a fine timescale decision layer.
2. The modeling layer trains on a minute-level timescale and pushes the global model to decision layer.
3. The decision layer is horizontally scalable and can operate on a millisecond-level timescale. It combines the up-to-date per-client information, the global model, and other policies to makes optimal CDN/bitrate decisions for clients.

4 Sensing/Actuation Layer

This section presents the design of the C3 client side modules, which provide three functions. First, it reports video quality from the players to the C3 controller. Second, it receives and implements control decisions from the controller. Third, it has built-in fault tolerance when it loses connectivity to the C3 controller. There are two practical challenges in implementing these functions: (1) diversity of client-side platforms and (2) slow software update cycles of client-side platforms. We first elaborate the challenges and then describe how C3 addresses them.

4.1 Challenges

To understand the causes of the practical challenges, we need some background on the structure of the client-side platform. Each client-side platform consists of four key components: client operating system, application framework, streamer, and player application. The application framework runs on top of the operating system and provides the libraries to support the development of video player applications. Many application frameworks can run atop the same operating system and device hardware. The streamer is responsible for downloading and rendering the video. Finally, the player application is the software developed by a content publisher based on specific application framework, to implement the user interface, access to content library, and player navigation.

Diversity: We observe client diversity along several dimensions; e.g., programming language (e.g., C, Javascript, Lua), system support for code execution (e.g., support for multi-threading), application framework, and streamer. The diversity of application frameworks and streamers is especially critical as it defines the interfaces used to monitor and control the video quality, and specifies the programming environment. Table 1 shows three examples of operating systems and devices and a subset of application frameworks. In total, we encounter 95 distinct application framework and streamer combinations. Each such combination requires special attention to mon-

OS	Devices	Application Frameworks
Android OS	Android phones/tablets	MediaPlayer, Irdeto, NexStreaming, Video View, VisualOn, PrimeTime, Akamai
PlayStation OS	PS3, PS4, PS Vita	Trilithium, LibJScript, WebMAF, Touchfactor
Mac OS & Windows with Flash	PCs	OSMF, Kaltura, Ooyala, PrimeTime, Brightcove, FlowPlayer, ThePlatform, JWPlayer
iOS	iPhone, iPad, iPod Touch	AVFoundation, Ooyala, Brightcove, PrimeTime, MediaPlayer, Irdeto

Table 1: Examples of OS and devices with the corresponding application frameworks.

itoring and control interfaces. Such diversity further reinforces the need to minimize the client-side functionality.

Long software update cycles: The second major challenge is long software update cycles for client-side platforms. There are a host of contributing factors here; e.g., device firmware update cycle (3-12 months), publisher app updates (1-6 months) and app store ratifications (1-4 weeks), and user delays in applying updates (weeks to months). Unlike in Phase II where Flash/RTMP platforms support a player module to be downloaded dynamically, the integration code in most Phase-III platforms is embedded in the player binary and cannot be changed arbitrarily. This long update cycle fundamentally constrains the pace of evolution of the C3 platform with respect to any functions that depend on sensing/actuation layer. Although the decision algorithms are not constrained by the update cycle as it is already on the controller, it does impact the information available to the control logic. For instance, if some quality metric is currently not collected or cannot be extrapolated from the collected information, the control logic will not be able to use it until the next release, which as we have see can take months or even a year (e.g., set-top boxes).

4.2 Thin Client-Side Design

Next, we discuss how we address the challenge of client diversity and long update cycles by making the sensing/actuation layer functionality as *minimal* as possible. We do so via two key design decisions. First, we introduce a general and abstract representation of video player actions. This allows us to handle client diversity. Second, we make an explicit decision to export raw events rather than summary statistics and push this computation to the backend. This delayed binding enables us to tackle the uncertain software upgrade cycles in the wild.

4.2.1 Abstracting player state and control

To minimize the amount of engineering effort required to support the client-side heterogeneity, we identify a logical narrow waist that we call the *ConvivaStreamingProxy (CSP)* (Figure 5). CSP abstracts away the idiosyncrasies of different players, and implements high-level monitoring APIs for collecting player performance information, and control APIs for switching bitrate and CDN. For each unique pair of streamer and application framework that we want to integrate, we implement an *adaptor* using the CSP API. The adaptor translates between the framework/streamer-specific APIs and the CSP APIs.

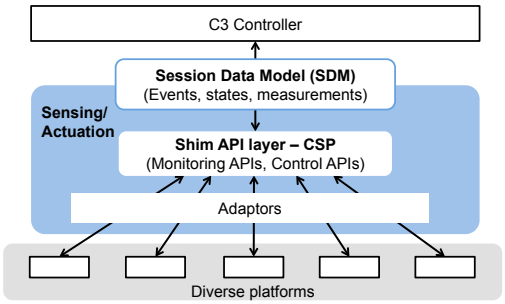


Figure 5: Overview of the sensing/actuation layer.

While the adaptor is specific to each framework and streamer, the common logic above the CSP is reusable across different platforms. To reduce the engineering efforts and support diverse programming languages used by the application, we developed a custom language translator that can take the source code from one language (in this case C#) and generate the equivalent source code for other languages. The design of this translator is outside the scope of this paper.

The CSP uses a unified monitoring interface, called *Session Data Model (SDM)* between clients and the C3 controller (Figure 5). SDM is a conceptual model for Internet video sessions and is agnostic to device, OS, application framework, or streamer features. Consequently new platforms can be integrated with little change on the controller. The SDM defines events, states and measurements as following:

- Events encode one-time actions and may change a state variable. Examples are bitrate switch start/end, application error.
- States encode persistent state variables, such as player state (buffering, playing, etc), bitrate and CDN.
- Measurements are continuous variables that show the health of the player, such as frame rate, available bandwidth, and buffer length.

CSP also provides the control APIs between clients and the controller. The clients send *poll* requests to get control decisions of bitrate and CDN at well-defined intervals (e.g., either at periodic intervals or at video chunk boundaries).

4.2.2 Exposing raw data

While the SDM abstraction minimizes the effort in integrating new platforms, it does not address the other practical challenges arising from long software update cycles. Specifically, this means that some logic (e.g., quality metric computation) becomes inflexibly hardcoded in client side. In order to reduce the need to make changes

to clients, we build on top of the SDM abstraction and instrument the client to *report the raw events and player states*. For example, we could calculate the average bitrate of a session on the client and send this to the controller. In contrast, our approach reports all bitrate switch events to the backend and allow it calculate the average bitrate (or any other bitrate-related metric). This *delayed binding* in postponing summarization of quality metrics to the controller further embodies the high-level decision to make the client-side as minimal as possible.

The frequency at which the clients report the controller naturally induces a tradeoff between overhead and information freshness. On one hand, the clients should report quality frequently so that the controller can detect client-side events (e.g., session exit, buffer draining out) in time and make decisions accordingly. On the other hand, updating too frequently may overload the controller and/or consume too much client-side resources. To address this problem, we take the following practical approach. The sensing/actuation layer periodically batches the collected information into *heartbeats* before sending them to the controller. In practice, we choose a sweet spot between 5 seconds and 20 seconds; intervals $\leq$ 5 seconds introduce undesirable interactions especially on mobile devices (e.g., draining battery by increasing CPU and radio use) and intervals $\geq$ 20 seconds significantly reduce freshness of data used in decision making. Fortunately, most playback buffers are on the order of 30 seconds, so the controller is always able to react before the buffer drains out. Additionally, the controller can dynamically tune the reporting frequency; e.g., decreasing the frequency during flash crowds to reduce the overhead and increasing the frequency for a client with a low buffer.

4.3 Fault tolerance

The main failure mode is when the client can no longer contact one of the C3 servers implementing the decision layer functions; e.g., the server failed or the network link is unreliable. There are two potential concerns we need to address: (1) loss in quality (because the client cannot receive control decisions) and (2) information loss (because the client cannot send quality measurements). Fortunately, we can leverage application-level resilience in conjunction with the SDM to address both issues.

First, we ensure that client-perceived quality will degrade gracefully when the C3 controller is unreachable. Because there is no tight coupling between the client and the decision layer, we can handle decision layer failures by simply resending requests and reports, and allow the load balancer to reassign the client to a new server. If the client is unable to contact the controller, it will fall back to the native bitrate adaptation algorithms [30, 9], which most platforms support today. The native algorithms are able to select bitrate with local logic (e.g., using throughput or buffer occupancy), so the player can still provide descent quality of experience.

Second, to mitigate the impact of information loss, we use built-in resilience provided by the SDM semantics because it explicitly includes current player states. To see why, consider an alternative solution that only reports the events without reporting current player states. The problem is that even a single lost event can mislead the controller; e.g., if we miss an event where the player transitioned from playing to buffering state, the controller will incorrectly assume that it is currently in playing state. However, we can mitigate the impact of lost heartbeats by including a snapshot of current states in each heartbeat. When a heartbeat is lost or a C3 server is down (when all history events are lost), even though the controller cannot recover lost events, the new C3 server can infer the current state using the next heartbeat.

4.4 Summary of key design decisions

In summary, the C3 client-side component has the following key aspects:

1. A common data/control abstraction via the SDM interface to tackle client-side diversity.
2. Exposing raw data to address slow client release cycle.
3. Providing a configurable reporting frequency to reduce the overhead.
4. Enabling graceful degradation by falling back to the native adaptation algorithm to handle transient failures and using stateful SDM features to re-establish context when raw events are lost.

5 Evaluation

In this section, we evaluate the performance of C3 and the benefits it offers. We divide our evaluation into the following parts;

- We evaluate the C3 controller in terms of (a) scalability and responsiveness of the decision layer (§5.1) and (b) the ability of the modeling layer to handle various workloads within the deadlines (§5.2).
- We show the sensing/actuation layer is lightweight in terms of bandwidth consumed and can gracefully degrade user experience under failures (§5.3).
- We analyze the quality improvement that C3 offers in the wild and discuss anecdotal experiences in handling high impact events (e.g., FIFA World Cup) (§5.4).

5.1 Decision layer

Scalability: By design, the decision layer is horizontally scalable (§3), with no synchronization needed between its instances to handle client requests.

Here, we focus on evaluating the *requests per second* (RPS) that a single decision instance can process within a given response time threshold. The instance under test is

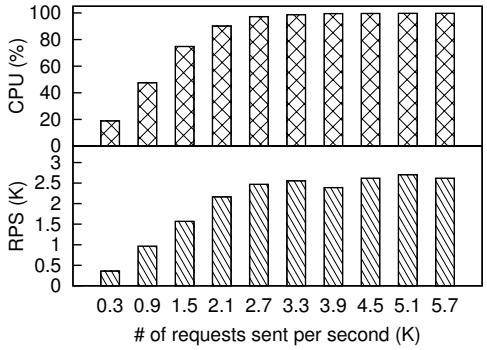


Figure 6: Scalability of single decision instance.

based on an Intel Xeon L5520 2.27Gz with 4 GB RAM, and only one core is used. Note that the requests include heartbeats and control requests, both of which are processed via the same procedure.

Figure 6 shows the RPS and CPU utilization when receiving different number of requests. It shows a linear growth in RPS and CPU utilization before the instance is saturated at the point of 2500 requests.

Load balancing: Next, we use real world measurements to show that the requests assigned by the load balancer to each decision instance is evenly distributed, even under high load. Figure 7 shows the distribution of RPS and CPU utilization of decision instances in different decision data centers, under a high load during the one of the most popular games in World Cup 2014. The means and standard deviations are based on all instances in each data center. It shows that the load balancer can assign the requests almost evenly across all decision instances within each data center with little variance and no request being dropped. The differences across the data centers is because the wide-area load balancing used geographical proximity and the workload was unevenly distributed around the world. Finally, the low CPU utilization suggests this load is well below the instance capacity.

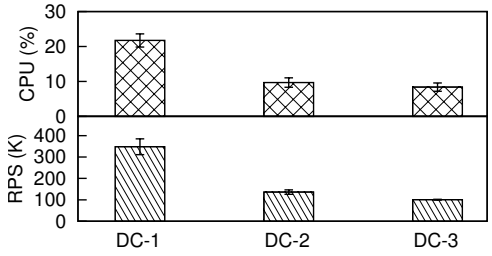


Figure 7: Number of requests and CPU utilization (mean and standard error) of decision instances under a high load.

End-to-end response time: The key metric of interest for the decision layer is the end-to-end response time – the time between client sends a request and it receives the response (typically a control decision). It includes internal processing latency as well as network latency. We measured the response time of requests in the real production system. Figure 8 presents the response time of requests observed from 10 countries representing different continents. It does show variance among different countries, but overall, we observe the 50%ile (or 80%ile) is always below 400ms (or 800ms)

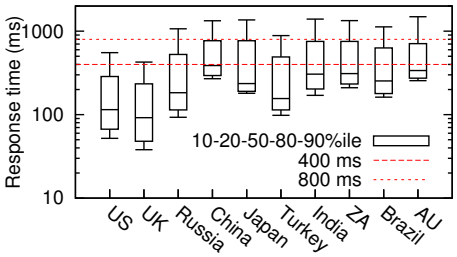


Figure 8: Response time is consistently low across countries from different continent.

5.2 Modeling layer

The modeling layer has to process various processing jobs with different characteristics. Figure 9 shows the processing latency of three most typical types of jobs and compares them with their deadlines, i.e., maximum expected processing time. The global model ("GlobalTraining") needs to be refreshed every minute and needs to run complex machine learning algorithms over the recent (few minutes) of global measurements. Customer interactive queries ("InteractiveQuery") run on-demand, and have to be completed in sub-second response time to prevent the customers from waiting too long. Finally, live update of quality metrics ("LiveUpdate") is the metric computation and aggregation process and has to be sub-second. As shown in Figure 9, the processing latency can easily satisfy the deadlines of different jobs.

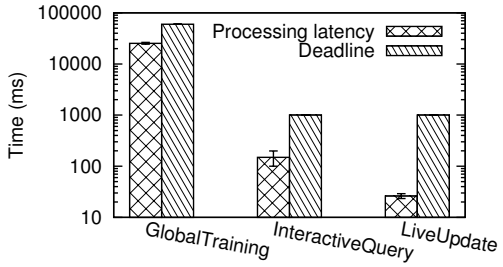
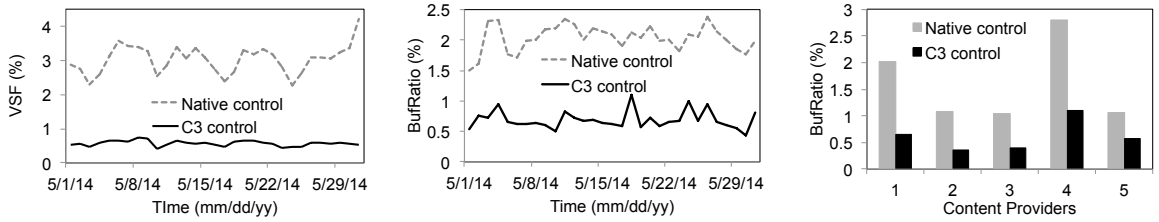


Figure 9: Processing latency (mean and standard deviation) of different types of jobs in modeling layer compared to their deadlines.

5.3 Sensing/actuation layer

Next, we show that sensing/actuation layer has relatively light overhead compared to player execution and it provides local failover under decision layer failures.

Overhead of sensing/actuation layer: We evaluate the sensing/actuation layer overhead in terms of the additional bandwidth used by comparing a C3-enabled



(a) Video start failure rate (VSF). (b) Buffering ratio (BufRatio). (c) Buffering ratio of 5 content providers.

Figure 11: Quality improvement of using C3.

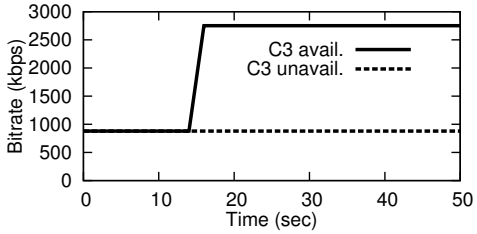


Figure 10: Local application-level resilience when C3 controller is unavailable.

player and a base video player. The base player we use is a fully functional player with OSMF as its streamer, and we let both the base player and C3-enabled version play a video encoded in 1.5Mbps bitrate on a laptop running Windows OS. We find that the additional bandwidth used by C3 is are typically very low and $\leq 1\%$ of the bandwidth used to download the video (not shown).

Local failover under decision layer failures: Next, we stress test the client under the scenario when it loses communication with the C3 controller. By design (§4.3), the player should fall back to the player's native adaptive bitrate logic and play smoothly; i.e., as long as the available bandwidth can sustain the lowest bitrate, the player should play smoothly, though with a low quality. We set up a real player to play a video encoded in multiple bitrates (from 880 to 2750 kbps), and the content is available from two CDNs. Figure 10 shows the time-series of bitrate downloaded by the player over two runs. In the first run, the C3 controller is not available and we throttle the bandwidth to the default CDN, so that the player backoffs to the native control logic and sticks to the low bitrate but not crash. In the second run, the C3 controller is available, and we again throttle the bandwidth to the same CDN. This time, because the C3 controller identifies the performance difference of two CDNs, it instructs the player to start with the unthrottled CDN, and thus it is able to switch to higher bitrate after a few chunks.[3]

5.4 C3 real-world deployment

Finally, we present the quality improvement C3 offers in the wild and our experiences in managing some high-impact events.

[3]The bitrates of the initial chunks are intentionally chosen to be low to minimize join time and avoid early buffering.

5.4.1 Quality improvement

We begin with the real-world quality improvements brought by centralizing decision logic detailed in §3.2.

Benefits of using C3: To estimate the quality improvement, we did a randomized trial where each session was randomly assigned to use either C3 or the native control logic. Figure 11 shows the improvement during May 2014, in terms of two quality metrics and across five content providers by comparing the median quality of sessions using C3 vs. native clients.

First, Figure 11(a) shows that C3 can significantly reduce the video start failure rate (percent of video sessions that failed to start) of a content provider. The reason is that using the global view allows us to predict CDN performance and choose an initial CDN that is not overloaded, unlike the native logic, which is typically statically configured (or chosen at random). Second, Figure 11(b) compares the buffering ratio of a content provider, and it shows a consistent reduction by 50% in buffering ratio by C3. The reason is that we can adapt the midstream selection of bitrate and CDN by leveraging the quality information of other sessions to achieve higher bitrate and a lower buffering ratio. Finally, Figure 11(c) shows that the quality improvement is consistent across five different content providers that use C3.

Impact of data staleness: Next, we present a trace-driven what-if analysis to quantify how much the decrease in freshness can impact the optimality of such quality improvement. Recall that allowing the global view to be stale on the order of few minutes was a key enabler for the scale-out design in §3. To avoid any biases introduced by our own control logic, we use the trace of sessions of Feb 10 from a content provider whose decisions are not controlled by C3 (as in §3.1). We simulate the effect of selecting the best observed CDN t minutes ago for each AS, and vary the degree of staleness by modifying this observation window t. Ideally, the decisions are made using the most recent view, i.e., $t = 1$. We evaluate multiple levels of staleness with $t = \{2, 10, 20, 40, 80, 160\}$ minutes. For each t, we compare *FreshestDecision* (i.e., times of picking the actual best CDN based on the freshest data) and *StaleDecision* (i.e., times of picking the actual best CDN based on the

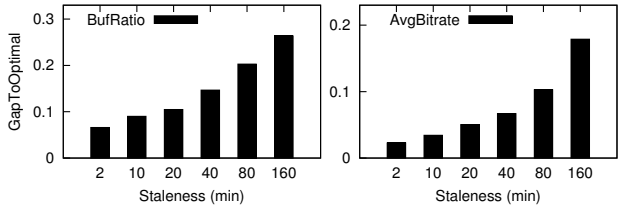


Figure 12: Impact of using stale data as input.

t-minute stale data), and compute the $GapToOptimal = 1 - \frac{StaleDecision}{FreshestDecision}$. The impact of staleness is smaller when *GapToOptimal* is closer to zero.

Figure 12 shows the $GapToOptimal$ of buffering ratio and average bitrate under different staleness t. First of all, it shows that using 2-minute stale data has very little impact, increasing buffering ratio by less than 7% and average bitrate by less than 2.5%. This suggests that minute-level stale data is still useful to make near-optimal decisions. Second, in both metrics, $GapToOptimal$ increases slowly when t increases from 2 to 20 minutes, and begins to increase much faster from $t = 40$. This suggests that near-optimal decisions can be made with slightly stale global model.

5.4.2 Case studies with high-impact events

In this section, we focuses on case studies on high-impact events (e.g., popular sports events) to showcase that the design of C3 is flexible and scales out horizontally.

Scale-out capability: The first case study highlights the flexibility of the scale-out design that enabled us to invoke public resources on demand. During World Cup 2014, we provisioned additional decision instances capability, including instances from a major public cloud service provider, in order to handle the scale of clients, which was expected to be 6M concurrent viewers. As a result, we were able to successfully handle the peak of 3.2M concurrent viewers during the US-Germany game. The reason for this scale-out capability is that the client-facing decision layer is decoupled from the modeling layer, which is the only centralized function (§4.2).

Dynamic reconfiguration: The second case is an example of flexibility that enables C3 to drop certain functionality under unexpected flash crowd. During a very popular soccer game of one European soccer league we saw about 2M concurrent viewers. Specifically, we saw a large flash crowd when the content provider switched all of its viewers to another channel, causing a high join rate which exceeded the capacity of our hardware load balancer for doing SSL offload. Since we could not add capacity to the hardware load balancer, we needed to devise quick workarounds to reduce this load. Our solution was to reduce the heartbeat frequency and disable HTTPS for that particular content provider. This effectively reduced the overhead of per-session operation (e.g., SSL handshakes) and ensured the availability of C3 controller. This type of fast reaction to unexpected flash crowds was possible because we had a thin client and had moved most of the functions to the C3 controller—it would have been virtually impossible to reconfigure client behaviors with hardcoded client-side player logics.

6 Related Work

C3 is related to a rich body of work in network control, application-layer systems, and big data platforms. While C3 borrows and extends ideas from these relevant communities, the core contribution of our work is in synthesizing these ideas to demonstrate the feasibility of an Internet-scale control plane architecture for Internet video. We briefly describe key similarities and differences between C3 and related work.

Network control plane: As discussed earlier in §2, the origins of C3 were inspired by the precursors to SDN (e.g., [45, 18, 43, 33, 19]) and in many ways C3's evolution has paralleled the corresponding rise of SDN. As such, there are natural analogies between C3 and SDN, in terms of the sets of challenges that both have to address: interface between control and data plane (e.g., [41, 17], distributed and global state management (e.g., [33, 15, 36]), consistency semantics (e.g., [40, 44]), centralized optimization algorithms (e.g., [26, 28]). The key differences are that C3 focuses on a specific video application ecosystem, which entails different domain-specific challenges (e.g., larger scale of clients and more data plane heterogeneity) and domain-specific opportunities (e.g., weaker consistency).

Video quality optimization: Previous work confirms that video quality impacts user engagement [23, 14]. It also identifies that many of the quality issues today are a result of sub-optimal client-side control logic (e.g., [27, 30]), and spatial and temporal diversity of performance across different CDNs and content providers [39, 38, 29], which suggests a centralized controller or a federated architecture [16] that provides global state and enables better informed decision making. However, these prior studies made a case for centralized control but fell short of actually demonstrating the viability of that control plane or the real benefits in the wild. In contrast, C3 is a concrete production design and implementation that achieves the benefits identified by these efforts. In doing so, it addresses many challenges (e.g., scalability, fault tolerance) that these prior works did not try to address.

Application resilience: The idea of exploiting domain- and workload-specific insights for improving system scalability and resilience is far from new and has been repeatedly identified; e.g., both in Internet services [24, 31, 5] and distributed file systems [25, 20]. For instance GFS exploits a unique workload pattern [25] while Spanner exploits tolerance for weaker consistency [20]. Our specific contribution is in reinforcing this insight in the

context of an Internet-scale control plane architecture.

Real-time data processing systems: In some sense, C3 can also be viewed as an instance of a scale out analytics and control system in the spirit of other big-data solutions (e.g., [21, 6, 22, 12, 11, 34, 10, 1]). Indeed, the C3 implementation builds on (and has actively contributed to) a subset of these existing technologies. While the C3 implementation relies on tools such as Spark [11] and Kafka [34], the core ideas are quite general and can be ported to other platforms. At a conceptual level, C3 also shares some similarity with the broad purview of the recent Lambda Architecture [7], with a combination of batch, serving, and speed layers. More recent work on Velox [21] also recognized the separation of global modeling and per-client prediction as a powerful system design choice for other data analytics applications. While C3 follows a similar high-level multi-layer architecture, the key is the specific division of functions between layer for video quality optimization. For instance, unlike the front-end layer in most scale-out systems, C3's front-end decision layer is an active layer that runs decision-making functions as well. Moreover, C3 justifies the separation of modeling and decision making by the domain-specific observation that slightly stale global models can still achieve near-optimal decisions.

7 Lessons

We conclude with key lessons we have learned from building and operating C3. Even though C3's design was driven by video-centric challenges and opportunities, these lessons have broader implications to concurrent and future efforts for centralized network control.

Feasibility of Internet-scale control: The one obvious lesson from our journey is that it is indeed possible to implement an Internet-scale control platform that achieves policy goals with global visibility. While there are parallel SDN success stories demonstrating the viability of centralized control, these have been in different (and arguably more scoped) domains; e.g., low-latency datacenters [32, 42], wide-area networks with a few PoPs [28] or coarse-grained inter-domain route control [18]. With C3, we provide another proof point in what we believe to be a much more global deployment, with larger scale and more heterogeneous clients, more stringent policy, and greater expectations in quality of experience.

Decisions that worked: Among the many decision choices that have made C3 a proof point of Internet-scale control, we highlight three that were particularly useful:

- **Exploiting application-level resilience:** A key enabler for our scale-out control architecture is that we were able to appreciate and exploit domain-specific properties that allows us to weaken some requirements (e.g., consistency and model freshness). While this idea may not be new and has been re-observed in many contexts, we believe that it is especially useful for network control applications. For instance, there has been considerable research effort developing consistent update schemes for SDN [40]. Rather than building a general-purpose solution, it might be possible to leverage application-specific resilience and engineer simpler schemes with weaker consistency properties.
- **Minimal client functionality:** We cannot stress enough the advantages that we have derived from minimizing the client functionality. This has (i) dramatically simplified our product development, integration cycles to support the increasingly heterogeneous client-side platforms, and (ii) made the C3 controller flexible to enforce global policy and evolve. The minimal client design is also made possible by the increasing compute capabilities of the backend.
- **Exposing lower-level APIs:** While minimal client functionality is useful, it has also been proved surprisingly useful to expose as many lower-level APIs from clients as possible, since they maximize control logic extensibility with a (relatively) slowly evolving data plane. We see immediate implications of this in SDN. While SDN started off with a minimal API (e.g., early OpenFlow versions), it soon devolved into the same complexity pitfalls that it sought to avoid (e.g., OpenFlow 1.3 spec is 106 pages long [17]). We believe it might be worthwhile for the SDN community to revisit minimality in light of the benefits we have derived and we already see early efforts to this end [17].

New research opportunities: C3's design and deployment opens up new possibilities for video quality optimization. The current C3 design is only a step in our journey and we acknowledge that there are several directions for future work that we do not cover in this paper. For example, one interesting question is analyzing the interaction of C3 with CDN control loops. In terms of quality improvement, we need a better understanding of clients that show little improvement and techniques to leverage network and CDN information for better quality diagnosis. Similarly, there are interesting modeling and algorithmic questions in the design of the prediction modeling and decision algorithms that have significant room for improvement.

Acknowledgments

This paper would not be possible without the contribution of the Conviva staff. Junchen Jiang was supported in part by NSF award CNS-1345305. The authors thank Sachin Katti for shepherding the paper and the NSDI reviewers for their feedback.

References

[1] Apache flume. incubator.apache.org/flume.

[2] Cisco forecast. http://blogs.cisco.com/sp/comments/cisco_visual_networking_index_forecast_annual_update/.

[3] Cisco study. http://www.cisco.com/web/about/ac79/docs/sp/Online-Video-Consumption_Consumers.pdf.

[4] Driving Engagement for Online Video. http://events.digitallyspeaking.com/akamai/mddec10/post.html?hash=ZDlBSGhsMXBidnJ3RXNWSW5mSE1HZz09.

[5] Facebook scribe. github.com/facebook/scribe.

[6] Hadoop. http://hadoop.apache.org/.

[7] Lambda archictecture. lambda-architecture.net.

[8] Microsoft Smooth Streaming. http://www.microsoft.com/silverlight/smoothstreaming.

[9] Netflix. www.netflix.com/.

[10] S4 distributed stream computing platform. incubator.apache.org/s4.

[11] Spark. http://spark.incubator.apache.org/.

[12] Storm. storm-project.net.

[13] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia*, 2011.

[14] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM '13*.

[15] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. OConnor, P. Radoslavov, W. Snow, et al. Onos: towards an open, distributed sdn os. In *ACM HotSDN 2014*.

[16] A. Biliris, C. Cranor, F. Douglis, M. Rabinovich, S. Sibal, O. Spatscheck, and W. Sturm. Cdn brokering. *Computer Communications*, 2002.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR*, 2014.

[18] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *NSDI 2005*.

[19] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM CCR 2007*.

[20] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Googles globally distributed database. *ACM Transactions on Computer Systems (TOCS) 2013*.

[21] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *Conference on Innovative Data Systems Research (CIDR)*, 2015.

[22] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI 2004*.

[23] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. A. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. SIGCOMM*, 2011.

[24] M. J. Freedman. Experiences with coralcdn: A five-year operational view. In *NSDI 2010*.

[25] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review 2003*.

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

[27] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: evidence from a large video streaming service. In *ACM SIGCOMM 2014*.

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

[29] J. Jiang, V. Sekar, I. Stoica, and H. Zhang. Shedding light on the structure of internet video quality problems in the wild. In *ACM CoNEXT 2013*.

[30] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Streaming with Festive . In *ACM CoNEXT 2012*.

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

[32] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, et al. Network virtualization in multi-tenant datacenters. In *NSDI 2014*.

[33] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI 2010*.

[34] J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, 2011.

[35] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In *ACM IMC*, 2012.

[36] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *ACM HotSDN 2012*.

[37] H. Liu, Y. Wang, Y. R. Yang, A. Tian, and H. Wang. Optimizing Cost and Performance for Content Multihoming. In *Proc. SIGCOMM*, 2012.

[38] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian. Optimizing cost and performance for content multihoming. *ACM SIGCOMM CCR*, pages 371–382, 2012.

[39] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. In *SIGCOMM*, 2012.

[40] R. Mahajan and R. Wattenhofer. On consistent updates in software defined networks. In *ACM HotNets 2013*.

[41] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2):69–74, 2008.

[42] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: a centralized zero-queue datacenter network. In *ACM SIGCOMM 2014*.

[43] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *ACM HotNets '12*.

[44] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin. A network-state management service. In *ACM SIGCOMM 2014*.

[45] H. Yan, D. A. Maltz, T. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: A 4d network control plane. In *NSDI*, volume 7, pages 27–27, 2007.