Eliminating Adverse Control Plane Interactions in Independent Network Systems

Matthew K. Mukerjee

May 2018
CMU-CS-18-106

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Srinivasan Seshan, Chair
Vyas Sekar
Peter Steenkiste
Bruce Maggs *(Duke; Akamai)*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was supported by the National Science Foundation under grants numbered CNS-1040801, CNS-1314721, CNS-1345305, and CNS-1565343.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government, or any other entity.

Keywords: networks, control plane, split control, control competition, information sharing, interfaces, reconfigurable datacenters, datacenter optics, content brokering, CDNs, live video delivery

To Sophie
for putting up with
this long, crazy ride.

Abstract

Network system operation is typically divided into control and data planes— while the data plane is responsible for processing individual messages or packets, the control plane computes the configuration of devices and optimizes system-wide performance. Unfortunately, the control plane of each network system or protocol layer typically operates independently (e.g., CDNs selecting servers without coordinating with ISPs), resulting in poor interactions between control planes across systems. We can categorize existing systems into one of four general control plane coordination mechanisms that overcome these problems, based on how much information can be shared between control planes. If no information can be shared, control planes simply **react** to data plane changes as a rudimentary form of coordination (e.g., CDN server selection + ISP traffic engineering). If all information can be shared, **transparency** in decision making can remove most poor interactions (e.g., Coflow datacenter flow scheduling). In many scenarios, however, only *some* information can be shared (e.g., between control planes running in different companies). Coordination in these scenarios is more specialized; control planes with separate data plane resources can use **priority ranking** (i.e., providing a list of preferences for resources without needing to show how these preferences were decided; e.g., BGP routing between ISPs), and control planes with shared data plane resources can use **hierarchical partitioning** (i.e., making coarse-grained decisions globally, and fine-grained decisions locally; e.g., internet-wide BGP + OSPF routing). While systems utilizing control plane coordination exist today, they have been designed ad hoc. We propose a set of recipes that show when it's appropriate to use different coordination mechanisms, based on key properties (information sharing and shared resources) in varied scenarios (layering, administrative separation, and internet-scale systems).

We use these recipes to guide system design in a variety of contexts, as a case study in control plane coordination. First, many systems use *layer separation* for modularity, but in doing so trade performance for generality. As layered systems have no information sharing constraints, we argue that transparency (i.e., cross-layer optimization; allowing layers to run specialized code to better use other layers) is the correct technique for regaining performance. We explore this with our emulator Etalon [116], in the context of reconfigurable datacenters. Second, some systems are *administratively separate* (i.e., are split across different companies), limiting information sharing for business reasons. These systems may overcome adverse interactions using priority ranking. We explore this with VDX [114, 115], in the context of content brokering. Finally, systems needing *internet scalability* are increasingly combining a slow centralized control plane with a fast distributed control plane. Complete information sharing (and thus, transparency) would appear possible, but timescale separation makes this fundamentally impractical. Instead, hierarchical partitioning can overcome the challenges present in this scenario. We explore this with VDN [117], in the context of live video delivery. Through this case study we find that these coordination mechanism not only solve a variety of problems, but can be efficiently implemented.

Acknowledgments

First and foremost I would like to thank my girlfriend Sophie for always encouraging me to push forward on what I want to work on and providing infinite emotional support. I would also like to thank my family (Mom, Dad, Amanda, and Austin) for their love and emotional support.

I would like to thank my advisor, Srini Seshan, for seven years of fun tackling a large variety of interesting problems, as well as giving me the needed encouragement to keep submitting papers until they get in. I would also like to thank the faculty on my committee (Peter Steenkiste, Vyas Sekar, and Bruce Maggs) for all the help and feedback that went into this dissertation, as well as fun collaborations throughout the years.

I'd like to thank my undergrad and masters advisors, Andrew Campbell, Tanzeem Choudhury, and Daniel Freedman, for pushing me towards research and eventually a PhD.

I'd like to thank my mentors at Google, Ben Greenstein, Mike Piatek, and Matt Welsh, for a fun Summer internship.

I'd like to thank the research and support staff in CSD and XIA, Deb Cavlovich, Angie Miller, Kathy McNiff, Jenn Landefeld, Angy Malloy, Dan Barrett, and Nitin Gupta, for making sure everything just works and minimizing a lot of the pain of getting a PhD.

I would like to thank all my friends, collaborators, and mentors: David Naylor, Nico Feltman, Lili Ehrlich, Susu and Ray Kanemoto, Evan Shimizu, Alex Lloyd and Dana Daugherty, Lauren and Logan Plath, Ravi Choudhuri, Elizabeth, Matt, and Argus Griffin, Kris Salada, Mitch Cairns, and Michaela Nachtigall, Yu Zhao, Lindsey and Vince Slaugh, Brian Russman, Richard Wang and Yvonne Yu, Yuchen Wu, Alex Poms, John Wright, Hyeontaek Lim, David Witmer, Junchen Jiang, Bruno Vavala, Ying Jung Chen, Eunjin Lee, Nadi Bozkurt, Conglong Li, Devdeep Ray, Ranysha Ware, Athula Balachandran, George Nychis, Wolf Richter, Raja Sambasivan, Dongsu Han, Justine Sherry, Hui Zhang, Dave Andersen, Michael Kaminsky, George Porter, Alex Snoeren, other people working under XIA, and the CMU systems seminar folks. I apologize if I forgot anyone.

Finally, I'd like to thank the anonymous reviewers for SIGCOMM, NSDI, CoNEXT, and HotNets for their feedback on drafts of the included work.

The work presented in this dissertation is based on work in submission to *SIGCOMM 2018* [116], appearing in *CoNEXT 2017* [115], *HotNets 2016* [114], and *SIGCOMM 2015* [117], and builds on work appearing in *CoNEXT 2015* [98] and *ANCS 2017* [95].

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1 Introduction

Control of networked systems has a long and storied past, from initial telephone system control "Signaling System No. 7" (SS7) [72], to internet routing [101, 112], to modern software-defined networks [44]. Network control is not just about routing packets from some source to some destination within a network, but also includes a myriad of decision making problems, such as: selecting the best server to deliver content to users around the globe, deciding when is the right time to migrate a virtual machine to a new physical machine in a datacenter, or more classic problems like internet congestion control.

The core focus of this thesis is that no matter how clever a system is at solving one of these problems, it doesn't exist in isolation; many other systems external to it make decisions that directly impact it in complex ways, in a variety of metrics (e.g., performance, correctness, cost). The crux of this issue is that other systems may make decisions without understanding what's important to our system. Succinctly, these systems need to communicate over interfaces that allows them to make decisions that are mutually beneficial. We call this explicit communication for decision making **coordination**.

To understand the benefits of coordination, we first need to understand the problems caused by the lack of coordination. We take routing as an illustrative example (e.g., OSPF [112], BGP [101]). Routing (and more generally, networked systems) can often be logically broken into two pieces: a *control plane* and a *data plane*. The data plane's responsibility is the forwarding of packets or messages, while the control plane is responsible for optimizing system performance by configuring different devices in the data plane (e.g., routers, switches, servers).

In internet routing, the control plane (called "routing") periodically determines the best paths from sources to destinations (in terms of a variety of metrics, e.g., path length, cost, bandwidth/latency), while the data plane (called "forwarding") handles the actual per-packet data transmission. Separation of routing and forwarding allows forwarding to be very fast (line-rate) while easing the timescale requirements for routing. This provides them both flexibility. While forwarding is simply a mapping of packets to router output ports, what data is used during forwarding (e.g., destination address, destination port, other headers) is left open to implementation.

Similarly, control planes for routing have much variety. They can be a *distributed* computation across routers within one ISP (e.g., OSPF [112]) or across ISPs (e.g., BGP [101]), they can be a *centralized* computation for all routers within an ISP (e.g., SDN [44]), etc. Each has different pros and cons. Distributed approaches tend to have quick failure response, but may have trouble providing optimal performance (e.g., path lengths, cost, bandwidth), as they have a limited *local* view of current network conditions. Centralized approaches have been employed more recently

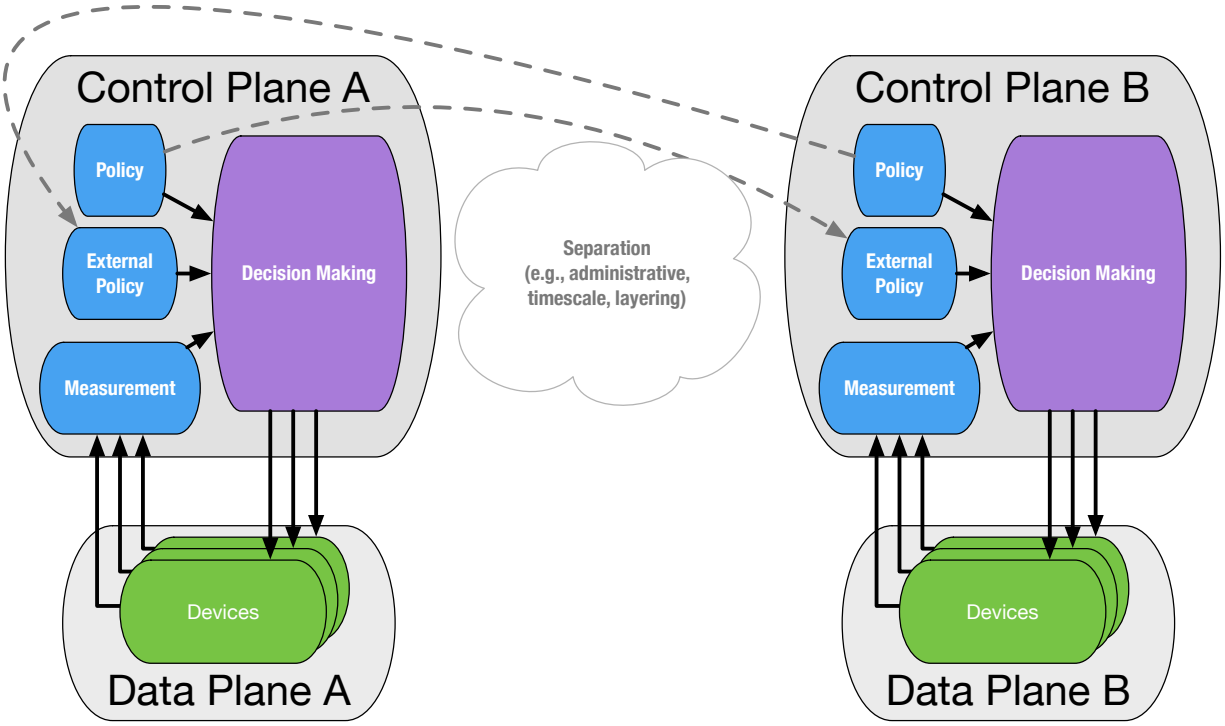


Figure 1.1: Control planes decide how to configure devices within the data plane. These decisions are reached by measuring the data plane, using internal policy, and potentially through external policy learned from other control planes (coordination). Two control planes might be separated by administrative, timescale, or layering boundaries.

due to their ability to provide optimal performance from their *global* view of the network, but tend to have more complex failures to contend with (e.g., partitions between the controller and routers).

Network control plane design is not as simple as choosing a distributed approach or a centralized one; even for a relatively well understood problem like routing, other systems (e.g., CDN server selection) may make decisions that make a routing control plane's job of determining the best paths unnecessarily difficult. Figure 1.1 shows an abstract view of the systems we consider. Each control plane is responsible for deciding how to configure a set of data plane devices. These decisions are reached through *measurements* of certain aspects of the data plane (e.g., current load, link capacities, latencies, etc.), through pre-configured *policies* within the control plane (e.g., move traffic to a new server after current load exceeds 60%), and potentially through *external policy* learned from other control planes (e.g., traffic can be sent to devices in other data planes but will cost $0.05 per Gbps). We refer to decisions made using external policy as a decision achieved through coordination.

We refer to systems comprised of two or more control planes as **split control plane systems** (e.g., CDN server selection + ISP traffic engineering, or BGP [101] running in two different ISPs). Control planes may be split due to administrative separation, timescale separation, layering, etc. Decisions made by split control planes may be made for different data plane devices (Figure 1.1), or the same set of data plane devices (Figure 1.2). If decisions are made by multiple control planes for the same set of devices, we say these control planes have **shared resources**.

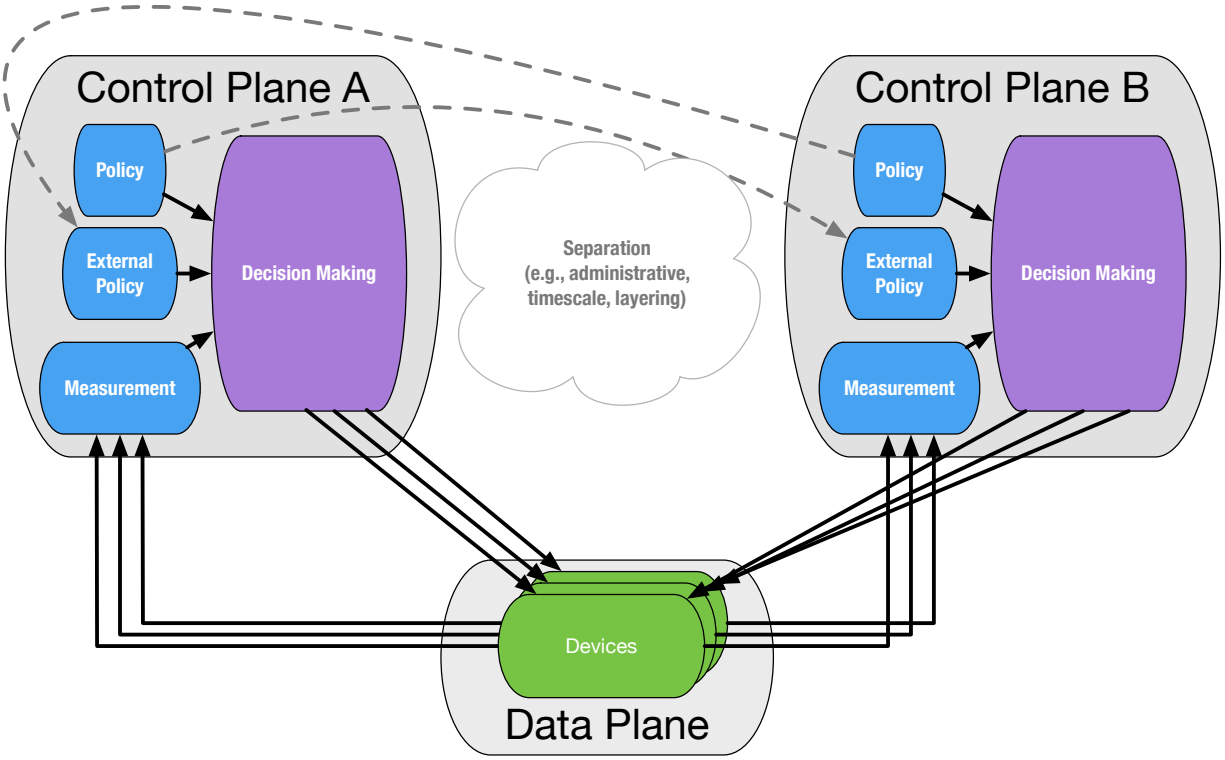


Figure 1.2: Separate control planes may make decisions on how to configure shared data plane devices.

Returning to our internet routing example, a single OSPF domain is not a split control plane system, but BGP running across two neighboring ISPs is. BGP instances only make decisions for their own independent set of data plane routers, so they do not have shared resources. Now that we have some understanding about split control planes and routing, we can understand the problems caused by a lack of coordination between split control planes.

For example, imagine a content delivery network (CDN) can choose to deliver content from one of two servers within an ISP. If one server is in an expensive part of the ISPs network (in terms of energy cost, capacity limitations, etc.), and another server is in a cheap location, the ISP would prefer the CDN uses the cheaper alternative. The CDN, however, likely does not know about these internal costs, and therefore may consistently use the expensive server, (unknowingly) putting pressure on the ISP.

A second example of issues arsing from a lack of coordination between control planes is BGP route flapping, where an ISP observes a change in the network and responds by reconfiguring its network. Another ISP responds to this reconfiguration by doing its own reconfiguration. The first ISP sees this reconfiguration in response to its own reconfiguration, and decides to respond by doing an additional reconfiguration, ad infinitum. Building a proper interface to explain *why* a reconfiguration happened would help ISPs come to a proper coordinated decision.

The key problem is that, in terms of coordination, many split control plane systems have been designed in an ad hoc fashion, leading to the problems we've seen. The focus of this thesis is a design methodology for eliminating adverse control plane interactions. While adverse interactions may arise from an overwhelming variety of reasons, we show in this thesis that for a core set of

very common scenarios (layering, administrative separation, and internet-scale systems), there are simple design recipes for control plane coordination that help eliminate issues.

While we see coordination mechanisms used in an ad hoc fashion in prior work, in this thesis we present a case study of principled uses of these recipes, building systems in a variety of contexts (reconfigurable datacenters, Ch. 2; content brokering, Ch. 3; and live video delivery, Ch. 4). This case study shows that these coordination mechanisms arise naturally from key properties of each scenarios (information sharing constraints and shared resources), and shows why these mechanism perform well in practice.
In summary:

THESIS STATEMENT

While past control plane coordination systems have been designed in an ad hoc fashion, we show that control plane coordination designs that operate efficiently and avoid adverse interactions can be designed and implemented using a framework built upon simple design recipes for a variety of very distinct, yet common scenarios.

In the rest of this chapter, we first define a core set of control coordination mechanisms (§1.1), before categorizing prior work based on these mechanisms (§1.2). We then explore key scenarios that naturally lead to split control plane systems (§1.3). We map these scenarios to proper mechanisms by exploring the design space of control plane coordination, ultimately providing a set of recipes guiding coordination design (§1.4). We conclude with the scope, goals, and contributions of the thesis (§1.5–1.7).

1.1 Coordination Mechanisms

Coordination between control planes is done through two parts: 1) information is shared using some *interface* between the two (or more) control planes, and 2) the control planes may need to *reconcile their decisions* in some manner (e.g., one has priority over the other if the decisions are for shared resources [51, 81, 117, 149], or if decisions are for separate resources reconciliation may be unnecessary [101, 103], etc.).

In this section, we identify four general coordination mechanism designs seen across a large variety of prior work, providing examples of systems that make use of them: **Reaction** is a rudimentary form of coordination without a coordination interface; decisions by other control planes are just observed by measuring the data plane (example: CDN server selection and its interactions with ISP traffic engineering). **Transparency** (cross-layer optimization) is the other end of this spectrum; all information that is needed for coordination is shared through an interface (example: Coflow [29, 31]. **Priority ranking** has control planes share a ranking of how they would like their resources used, without explaining how these rankings were computed (example: BGP across two ISPs [101]). **Hierarchical partitioning** has a global control plane make coarse-grained or long-timescale decisions while a local control plane makes fine-grained or short-timescale decisions (example: internet-wide BGP + OSPF routing [101, 112]).

Reaction

Control planes that don't have an explicit interface between them have difficulties coordinating with each other, effectively being limited to reacting to each other's decisions as they measure them in the data plane. There are likely few scenarios that outright forbid an interface between control planes. Most systems that use this design were built to work autonomously for simplicity, rather than for any hard information sharing constraint. Building a proper coordination interface (e.g., between content brokers and CDNs, like in VDX [114, 115]; Chapter 3)), may help systems improve performance, cost, and/or responsiveness issues. If a system does have strong requirements forbidding any information sharing, reaction is likely the *only* coordination mechanism that can be employed.

Example

CDN server selection and its effects on **ISP traffic engineering (TE)** is a common issue reported in content deliver. CDNs try to find the optimal server to deliver content to clients, but this may be a server that is costly for transit ISPs to deliver from. To avoid costly paths, some ISPs may route traffic through slower links than the CDN expects, leading to poor performance. As both parties don't communicate their decisions to each other, they can only react to each other's changes as they observe them.

Transparency

Transparency builds an interface where all information that needs to be shared for coordination is shared. Thus, two control planes effectively have perfect insight into how one another will make decisions, given the state of the network. Issues caused by the lack of coordination between control planes (e.g., poor performance, high cost, lack of responsiveness) can mostly be avoided, as assumptions made about the other control plane should almost always hold true. This doesn't imply that both control planes become one; depending on their locations in the network, both control planes may have slightly different views of current network state. If a system can share all information necessary for coordination between control planes, it likely should use transparency.

Example

Coflow scheduling [29, 31] is the idea that the network should schedule *bundles of flows* together that represent an application workload (coflows). Coflow scheduling presents a by-the-book layering problem: while a flow abstraction is great for modularity, it hides the fact that many flows are dependent upon one another, and that scheduling them without knowing this can greatly impact performance and responsiveness. For example, an application may require a set of flows to all complete before it can progress. Naive network scheduling will not know this, leading to slow applications. As the separation between applications and network scheduling is simply layering, Coflow can easily share as much information is needed for coordination purposes. Thus, Coflow uses transparency to have applications inform the network scheduler of their demands.

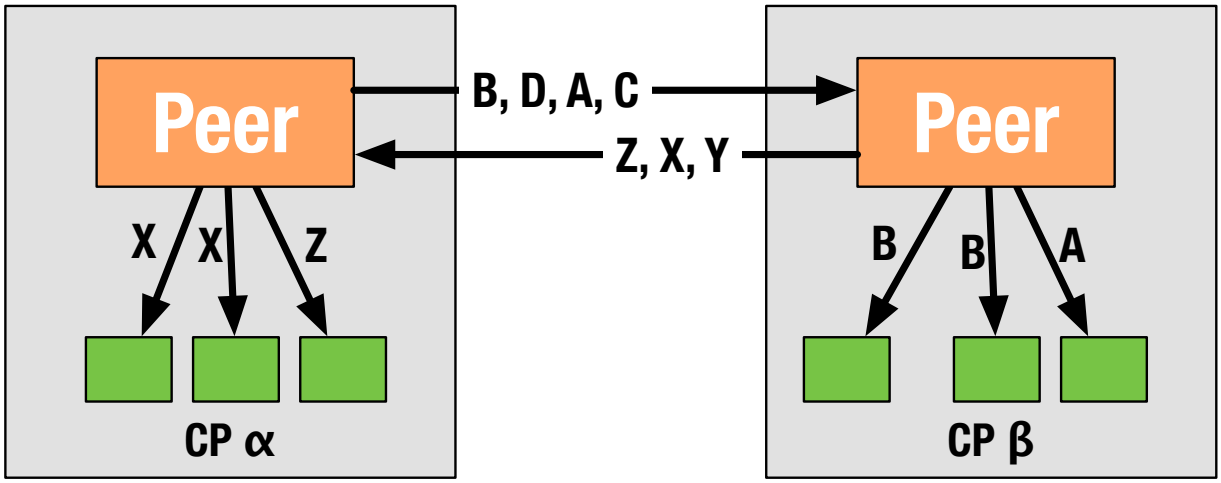


Figure 1.3: Priority ranking is a coordination mechanism where control planes communicate a list of data plane device configurations with priorities, to tell each other how they would like their devices configured. Priority ranking allows control planes to share their preferences, without needing to explain how those preferences were decided. As these are just preferences, peers need not honor them precisely.

Priority Ranking

Priority ranking is a coordination design that shares a ranked list of how to use resources between control planes, without explaining how this list was computed. Figure 1.3 shows an abstract view of priority ranking. Two control planes (α and β) give each other an ordered list of configurations ([B, D, A, C] and [Z, X, Y], respectively) they would like the other to use in configuring their data plane devices (shown in green). Control plane β would like α to use configuration Z on all of its devices, but this may cause issues for α, e.g., in terms of cost, performance, responsiveness, etc. Thus α uses decision X (β's second choice) for two data plane devices, and decision Z for another device. β configures devices similarly, based on its concerns and α's priority rankings.

The key point is that priority ranking allows control planes to share configurations with each other in a way that provides options, without needing to explicitly show how/why those options were picked. As shown, the rankings are simply preferences. Which configuration to use is ultimately up to the receiver. Systems that use priority ranking generally incentivize honoring priorities though symmetry; if α consistently ignores β's priority values, then β will likely ignore α's, negatively impacting α's performance, cost, responsiveness, etc. In this example, some decisions were mixed (e.g., X and Z), which may cause issues in certain scenarios. System designers need to decide whether this mixing should be allowed.

Example

Across ISPs, two instances of **BGP** [101] communicate to try to find high performance / low cost routes through each other's networks. As both instances are in separate administrative domains, not all information can be shared. Naively, this lack of transparency may result in ISP A sending a large volume of traffic to ISP B through a path that is very costly for ISP B. Multi-Exit Discriminators (MEDs) are designed to solved this. ISPs can tell each other a priority ranking of different entry

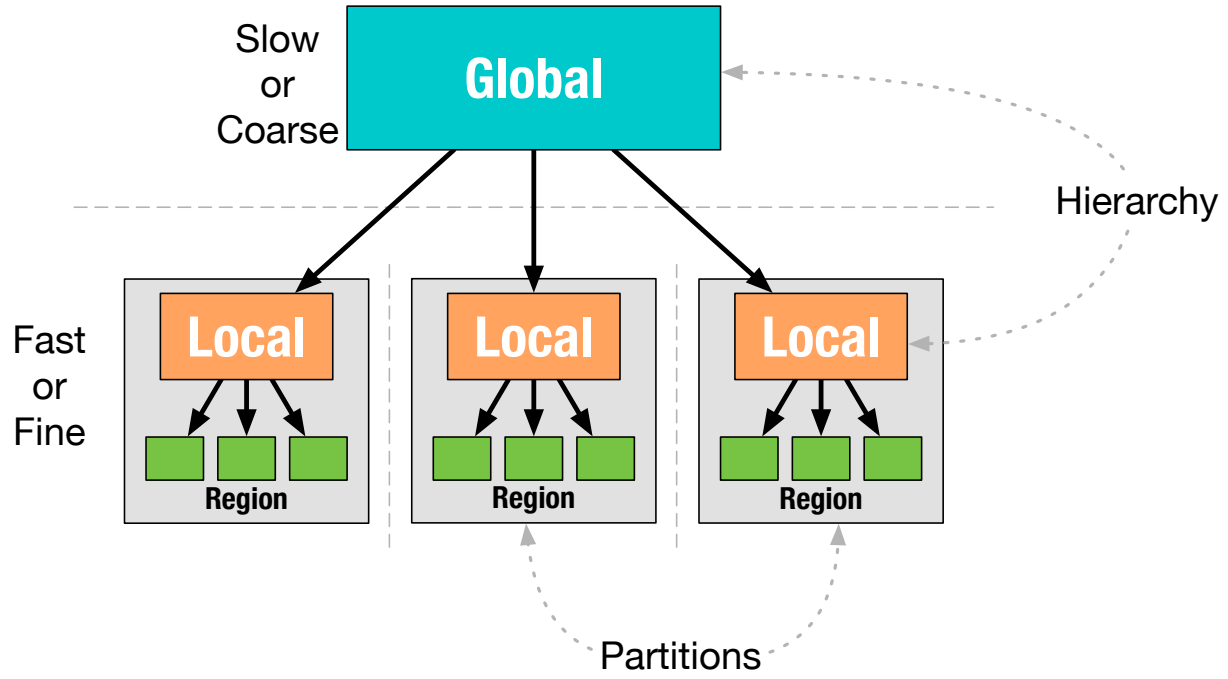


Figure 1.4: Hierarchical partitioning is a coordination mechanism where slow-timescale or coarse-grained global control decisions are augmented by fast-timescale or fine-grained local control decisions for an individual regions. For the timescale variant, global control generally optimizes for performance and local control provides responsiveness.

points into their network, without needing to tell the other ISP why those values were chosen (e.g., for cost reasons, capacity, performance, etc.).

Hierarchical Partitioning

Hierarchical partitioning is a coordination design that combines a slow (or coarse-grained) global controller with multiple fast (or fine-grained) local controllers. Figure 1.4 shows an example of hierarchical partitioning. Data plane devices are partitioned into "regions" (e.g., geographically) supported by a "local controller". The local controllers and a "global controller" form a hierarchy of controllers.

Typically, the global controller computes a complex performance optimization for all data plane devices, internet-wide, using its global view. If the optimization is fine-grained, this may take minutes, depending on the application (e.g., internet-scale live video delivery, Chapter 4). The most recent decision is disseminated to all local controllers in all regions. A local controller uses this (potentially stale) decision until a newer decision is propagated to it. During network events (e.g., link failures, client churn, etc.), a local controller can decide (typically based on the severity of the network event) to deviate from the global decision. These deviations are typically tolerated by maintaining "slack" in the global decision (e.g., reserved network capacity). The key point is that global control gets priority over local control in steady-state, and that its global view affords it a unilaterally better decision.

If the optimization is coarse-grained (e.g., internet-wide routing with BGP [101] + OSPF [112]), the global controller can quickly make a very coarse configuration decision for data plane devices, and then the local controllers can refine that decision for their local region (again by using slack in the decision). Well slightly different, this alternate version is functionally similar, and solves the same set of coordination issues. Deciding which version to use is likely tied to the specifics of the scenario.

Example

In **internet-wide routing**, we can consider internet-wide BGP as one global control plane, and look at its interactions with many "local" instances of OSPF. With this view, it clearly uses hierarchical partitioning. Effectively, BGP forms a coarse-grained "big picture" as to how routing should work across the whole internet at the ISP-level, leaving each of the local OSPF instances to refine that picture within their ISP.

1.2 Understanding designs for split control plane systems

In this section, we provide intuition as to how split control systems may fundamentally differ from one another by examining salient features in related work. We use this intuition to build a design space of split control plane systems.

1.2.1 Salient features

Peer-Peer / Master-Slave: Do the control planes have equal say in decisions (peer-peer; e.g., BGP [101]) or does one have clear priority in decision making (master-slave; e.g., Coflow [29, 31])?

Separation of Control Planes: How are the control planes separated from one another? e.g., by layering (e.g., Coflow [29, 31]), granularity (e.g., Bohatei [43]), timescale (e.g., C3 [51]), soft administration boundaries (within a company; e.g., OSPF areas [113]), hard administration boundaries (between companies; e.g., BGP [101]), etc.

Information Sharing: How much information can / is shared between the two control planes? Typically this depends on the nature of the control plane separation. e.g., it is simple to share information between layers (e.g., Coflow [29, 31]) but difficult to share information between companies (e.g., BGP [101]).

Shared Resources: Are the data plane resources controlled by each control plane completely disjoint (e.g., BGP [101]), are there partially shared resources (e.g., BGP + OSPF route redistribution [147]), or fully shared resources (e.g., C3 [51])? If a control plane is distributed, we consider all devices participating in the control plane to be "part" of it.

Information Flow: Is information shared in both directions between control planes? Do the control planes avoid sharing information (None; e.g., CDN server selection + ISP traffic engineering), does one control plane simply inform the second of its decision (One-way; e.g., Coflow [29, 31]),

or do both control planes tell each other their decisions (Two-way; e.g., BGP [101]). We do not consider measurements a part of information flow, just explicit communication between control planes.

Incorrect Assumptions: What are possible incorrect assumptions that one control plane might make about the other? These tend to be very context-specific (e.g., applications make assumptions about what types of traffic are easy for the network to deliver, which are broken in reconfigurable datacenters [116]; Chapter 2).

Decision Making: Is the decision itself a combination of the decisions made by the control planes (Joint; e.g., BGP [101]), is it a choice between alternatives (Choice; e.g., C3 [51]), is it always just the decision proposed by one side (Master; e.g., Coflow [29, 31]), or does each side make independent decisions over their resources (Separate; e.g., OSPF areas [113]). Some joint decisions are made in a two-way exchange of decisions, and some are made by having one side make a coarse decision, with the other refining it (Joint (Refinement); e.g., internet-wide BGP + OSPF routing [101, 112]).

Problems from Bad Decisions: What are possible metrics impacted by poor decisions? We consider Performance (P), Responsiveness/Fault Tolerance (R), Cost/Profits (C), and Correctness (Cor).

Distilling down to the most important features

While there are many salient features, we find that two of them are fundamental: **information sharing** (None, Incomplete, Complete) and **shared resources** (Disjoint, Partial, Full). Namely, information sharing effectively defines the control plane coordination interface, while shared resources limits the styles of decision making that can be used, as we shall show.

We find the rest of the features have strong overlaps with these features. Peer-peer systems tend to have disjoint resources, while master-slave systems tend to have fully shared resources. Layer separated systems tend to allow for complete information sharing, while other separations tend to require incomplete information sharing. Systems will fully shared resources tend to have one-way information flow, while systems with disjoint resources vary in their information flow. Systems with fully shared resource tend to use refinement or choice as a decision making process, while systems with disjoint resources tend to use master or joint as a decision making process.

System	Context	Control Planes	PP/MS	Separation	**Info. Sharing**	**Shared Resources**	Info. Flow	Assumptions	Decision Making	Problems
Solution: **Reaction**										
CDN Server Selection + ISP TE	Content Delivery	CDN + ISP TE	MS	Admin (Hard)	None	Disjoint	None	CDN server selection / ISP paths optimal	Master	P / C
OSPF Areas [113]	Intra-Domain Routing	OSPF + OSPF	PP	Admin (Soft)	None	Disjoint	None	Splits are optimal	Separate	P / Cor
Competing CC Algos.	Congestion Control	CC + CC	PP	Admin (Soft/Hard)	None	Disjoint / Full	None	CC tries maximizing link util.	Separate	P
CC + AQM	Congestion Control	CC + AQM	MS	Admin (Soft/Hard)	None	Disjoint	None	CC backs off on drops	Master	P
DASH [1, 140]	Video Streaming	DASH + HTTP + TCP	MS	Layering	None	Full	None	HTTP/TCP quickly retrieves chunks	Master	P
Solution: **Transparency**										
Coflow [29, 31]	Datacenter Network Scheduling	Apps + Net Scheduling	MS	Layering	Complete	Disjoint	One-way	Complex app/flow relationships	Master	P / R
Route Redistribution [147]	Inter-/Intra-Domain Routing	BGP + OSPF	MS	Layering	Complete	Partial	One- or Two-way	Redistributed routes are up-to-date	Joint	P / C
Reconfigurable DCs [116] (Ch. 2)	Reconfigurable Datacenters	Apps + Net Scheduling	MS	Layering	Complete	Disjoint	One-way	What's easy for the network to deliver	Master	P / R
Solution: **Priority Ranking**										
BGP (Across ISPs) [101]	Inter-Domain Routing	BGP + BGP	PP	Admin (Hard)	Incomplete	Disjoint	Two-way	Routers expose best paths	Joint	P / C
Wiser [103]	Inter-Domain Routing	BGP + BGP	PP	Admin (Hard)	Incomplete	Disjoint	Two-way	Routers expose best paths	Joint	P / C
P4P [155]	P2P/ISP TE	P2P Apps + ISPs TE	MS	Admin (Hard)	Incomplete	Disjoint	One-way	Apps honor priority, ISP cost honesty	Master	P / C
VDX [114, 115] (Ch. 3)	Content Brokering	Broker + CDNs	PP	Admin (Hard)	Incomplete	Disjoint	Two-way	CDN / client assignments optimal	Joint	P / C
Solution: **Hierarchical Partitioning**										
Internet-wide Routing [101, 112]	Routing	BGP + OSPF	MS	Granularity	Incomplete	Full	One-way	OSPF provides good paths	Joint (Refinement)	P / C
Klein [132]	Cellular Core	Global + Local Control	MS	Granularity	Incomplete	Full	One-way	Global decision is correct	Joint (Refinement)	P
Bohatei [43]	DDoS Defense	Global + Local Control	MS	Granularity	Incomplete	Full	One-way	Global decision is correct	Joint (Refinement)	P / Cor
OSPF Fibbing [149]	Intra-Domain Routing	SDN + OSPF	MS	Timescale	Incomplete	Full	One-way	Global decision is correct	Choice	P / Cor
Pytheas [81]	Video Delivery	Global + Local Control	MS	Timescale	Incomplete	Full	One-way	Global decision is correct	Choice	P / R
C3 [51]	Content Brokering	Global + Local Control	MS	Timescale	Incomplete	Full	One-way	Global decision is correct	Choice	P / R
VDN [117] (Ch. 4)	Live Video Delivery	Global + Local Control	MS	Timescale	Incomplete	Full	One-way	Global decision is correct	Choice	P / R

Table 1.1: Examining prior work with split control planes to understand the nature of their separation, possible problems, and how they overcome them.

1.2.2 Related Work

We examine systems with split control planes to better understand why they naturally arise, what sorts of problems may be faced due to the separation, and what solutions are encountered. This examination is summarized in Table 1.1. We walk through this table in detail below, sorting the work based on which coordination mechanism they use (defined in §1.1):

Coordination through reaction

CDN server selection and its effects on **ISP traffic engineering (TE)** is explained in §1.1.

OSPF areas [113] are a means of dividing large networks run by one ISP into smaller regions ("areas"), helping minimize the cost of recomputing shortest paths when link state changes. While this does provide scalability, care must be given to how regions are divided, or there can be performance (i.e., stretched paths) or correctness (i.e., partitions) issues. Other OSPF areas are considered opaque black-boxes. Coordination is mainly just through reacting to changes.

Competing congestion control (CC) algorithms may run on the same machine or on different machines, but share some bottleneck link in the network. The assumption is that both CC algorithms are trying to maximize their link bandwidth, but neither communicate their strategy with each other. This can lead to poor performance (i.e., overshooting bottleneck link bandwidth leading to increased delay, or fairness issues, depending on the CC algorithms). This is, perhaps, the most classic example of two control planes that make independent decisions, but coordinate through reaction. If both congestion control algorithms are running on the same machine (or even within the same administrative domain) smarter coordination solutions could be applied (e.g., Congestion Manager [17]).

Congestion control (CC) and active queue management (AQM) interact in a manner similar to two CC algorithms. Namely, when an AQM algorithm makes a decision that a flow is sending too much, it sends the CC algorithm associated with this flow an implicit signal (e.g., a packet drop), effectively making a unilateral decision that the CC algorithm should slow this flow down. An AQM algorithm can limit CC performance by sending these signals at different times than the CC algorithm expects.

Dynamic Adaptive Streaming over HTTP (DASH) [1, 140] is a protocol for streaming video that is widely used today. DASH builds on the already widely used HTTP and TCP protocols to enable easy deployment of video delivery servers throughout the web. DASH uses HTTP and TCP as-is, treating them as black-boxes for reliable video chunk delivery. We argue that while this does simplify things, transparency (cross-layer optimization) could potentially provide opportunities to improve user quality-of-experience.

Coordination through transparency

Coflow is explained in §1.1.

Within the realm of a single ISP, the interactions between BGP and OSPF come down to **route redistribution** [147]. Typically OSPF routes are redistributed into BGP and can be used in their entirety, as there is no hard administrative or timescale separation. Thus, transparency can be used. As long as the OSPF routes are up-to-date, BGP should be able to use OSPF route information to communicate lower cost / lower stretch paths to other ISPs. Interestingly, this is the only system we've identified that has partially shared data plane resource, as all routers generally run OSPF, but only some routers run BGP.

Our work on end-to-end problems in **reconfigurable datacenters** [116] (Ch. 2) focuses on issues caused by application-/transport-layer assumptions about the network that become invalidated in future reconfigurable datacenter networks. We show three key end-to-end problems that must be solved to provide good performance and responsiveness in these kinds of networks. The key idea is coordination between endhost and network control planes using transparency (cross-layer optimization). This trades some of the modularity obtained from layering for massively improved performance.

Coordination through priority ranking

BGP interactions between ISPs are explained in §1.1.

Wiser [103] presents a modification to BGP, allowing two ISPs to better expose cost / performance / capacity concerns with each other without exposing information directly, through priority ranking. Wiser extends MEDs with a more complex scheme that provides incentives for ISPs to act truthfully, leading to globally shorter paths.

P4P [155] is a system for coordinating peer-to-peer (P2P) application performance concerns and ISP delivery cost concerns. The key observation is that P2P applications often have multiple locations they can choose to request content from. ISPs provide P2P applications with a priority ranking for different destinations, based on their internal concerns (e.g., delivery cost, path capacity, etc.). As P2P applications ultimate make the decision where to request content from, it is unclear if there are incentives for P2P applications to honor ISP concerns. This problem is very similar to CDN server selection + ISP TE.

Our work on content brokering, **VDX** [114, 115] (Ch. 3), focuses on building a system that allows content brokers and CDNs to interact with each other in a way that is fair to both. Effectively, as content providers have started using content brokers to send users to the best CDN at a given time, CDNs have seen erratic request patterns that are potentially very costly to respond to. Conversely, brokers have noticed that treating CDNs as black-boxes limits the extent of the performance / cost optimization they can do for content providers. We argue that exposing more information between brokers and CDNs, using priority ranking (building a "marketplace"), solves both problems simultaneously.

Coordination through hierarchical partitioning

Internet-wide routing is explained in §1.1.

Klein [132] is a system focused on building an elastic cellular core network that can run various network functions (NFs) at the right point in the network, at the right time. The key idea is to compute a coarse global decisions of NF placement, and then make a refined decision for a local region, i.e., hierarchical partitioning.

Bohatei [43] is a distributed denial of service (DDoS) prevention system that uses network function virtualization (NFV) to elastically add individual DDoS prevention NFs at the right place in the network, at the right time. Once again, it computes a coarse-grained decision globally, and then refines it locally (hierarchical partitioning).

OSPF Fibbing [149] is a system that seeks to provide SDN-like capabilities to legacy networks by injecting fake links/hosts into OSPF computations. Fibbing uses the SDN decision except during failures, when it reverts back to vanilla OSPF. Fibbing provides the ability to use SDN-like policy constraints on routes, but during failures the correctness of these policies can be violated.

Pytheas [81] is a control plane for internet-scale video delivery. It uses local clusters to communicate with clients and a global controller to build the correct mapping of clients to local clusters. The global controller maintains a slightly stale view of the network, requiring local clusters to make small adjustments to the global decision during network events, a clear instance of hierarchical partitioning.

C3 [51] is an internet-scale content brokering system. C3 maps clients to the best CDN for them, at the current time, based on client meta-data (e.g., location, device type, ISP). C3 uses hierarchical partitioning, computing a fine-grained global decision periodically, using it directly in data plane devices. During significant changes (e.g., failures), local control deviates from the global decision.

Our work, **VDN** [117] (Ch. 4), focuses on building a system that increases performance and lowers cost for CDN-based live video streaming. The core contribution of the system is applying hierarchical partitioning ("hybrid control") to live video delivery. VDN seeks to provide the high quality of centralized optimization, with the responsiveness of a distributed control plane. A fine-grained global decision is computed periodically, and is used directly by data plane devices. If there are significant changes (e.g., link failure), local control makes a quick decision how to deviate from the global decision, through the use of slack network capacity.

1.2.3 Design Space

Now that we have a way to characterize the space of split control plane systems, we look at how to categorize different systems ("design space"), as well as the mechanisms these systems use to coordinate ("coordination mechanisms").

Figure 1.5 organizes the systems from §1.2.2 in terms of the two key features we previously identified (§1.2.1): information sharing and shared resources. A few interesting insights come from

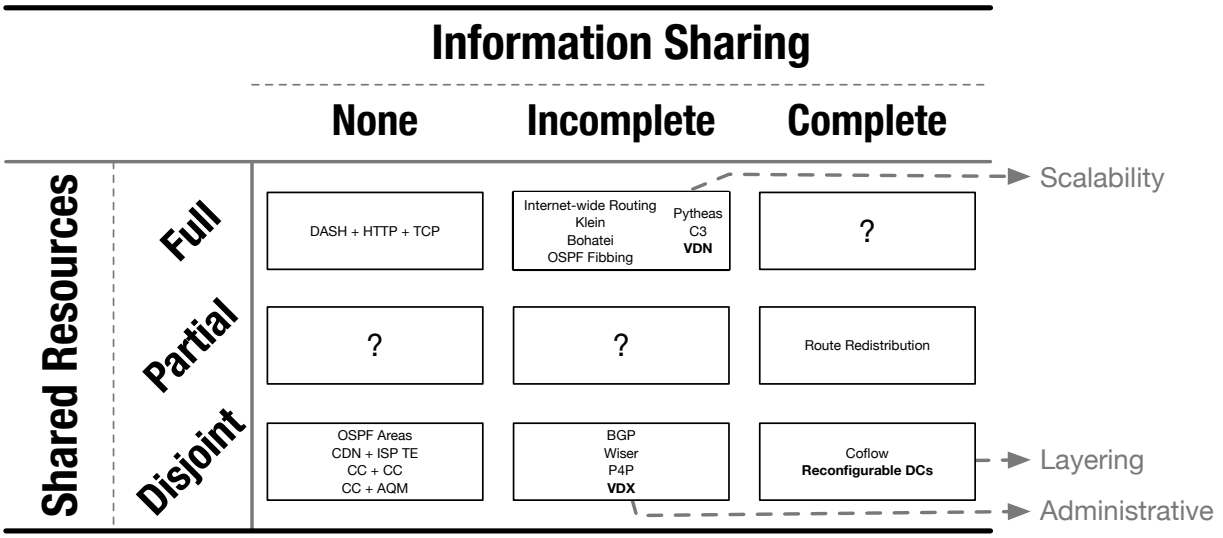


Figure 1.5: Design space. Systems in Table 1.1 can be classified based on their information sharing constraints and if they make decisions for shared data plane resources.

this categorization: older systems tend to not share information and focus on splitting problems into smaller independent pieces. Incomplete information sharing systems with fully shared resources tend to be almost entirely modern systems. A few areas do not have any prior work, mainly partially shared resources. These areas represent unlikely scenarios.

Control plane coordination mechanisms fall out nicely from our categorization of the design space, as illustrated in Figure 1.6. If no information can be shared in a control plane coordination interface, very little can be done; reacting to changes seen in the data plane is likely the only rudimentary coordination solution. If all information can be completely shared, transparency is a solution that can side-step virtually all adverse control plane interactions.

Incomplete information sharing represents the interesting middleground; if control planes make decisions for the same set of data plane resources, then hierarchical partitioning should be used. If decisions are made for a disjoint set of resources, priority ranking should be used. If other coordination mechanisms exist, they likely fall under incomplete information sharing, but may differ in terms of granularity (e.g., *how* incomplete is their information sharing or *how many* shared resources are there), or require an additional (albeit less important) axis (e.g., information flow).

1.3 Common Scenarios

From the design space presented in Figure 1.5, we notice many systems have similar scenarios: layering, administrative separation, and internet scalability. While each of these three scenarios naturally leads to split control plane systems, they cover wildly different aspects of system design (i.e., modularity, business concerns, and scale). For the rest of this thesis, we will use these scenarios to gain insight into why certain control plane coordination mechanisms naturally suit certain design constraints. While we find these scenarios both intuitive and illustrative, they are not the only scenarios that lead to split control planes (as touched on in §5.2).

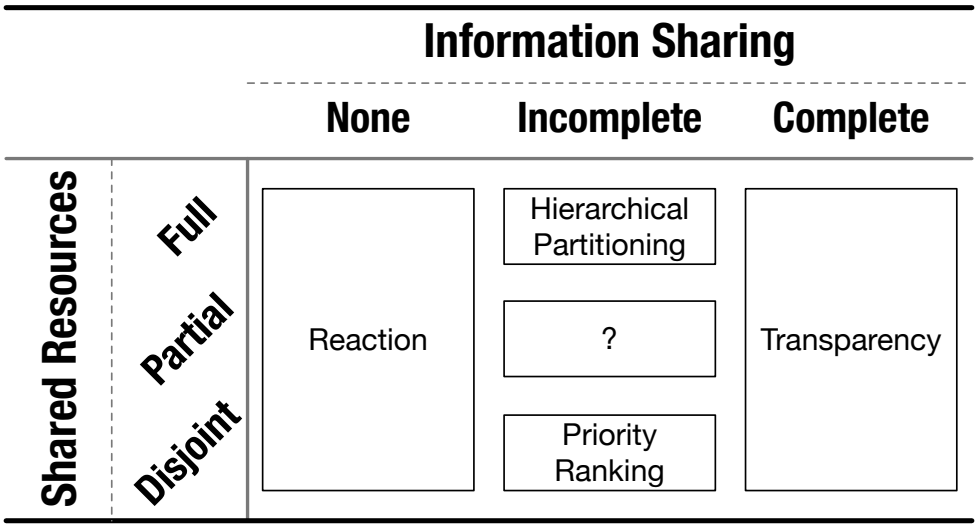


Figure 1.6: Coordination mechanisms. If no information can be shared, the only way to coordinate decisions is through reaction. If the system allows for complete information sharing, then adverse interactions are effectively avoided using transparency. With incomplete sharing, priority ranking or hierarchical partitioning is used depending on if data plane resources are shared.

Layering

Systems with layers generally trade *performance* for *modularity*. This can often cause issue if high performance is needed. We argue that such systems can be easily augmented with a coordination mechanism to improve performance, as there are no hard boundaries (e.g., administrative, timescale). Nothing stops additional code specialization in each layer with respect to other layers (cross-layer optimization), or a wider interface that explicitly covers special cases. Either can greatly recover performance. Coflow [29, 31] and application / network scheduling in reconfigurable datacenters [116] (Ch. 2) provide examples of layering.

Administrative separation

Systems separated by hard administrative boundaries, by definition, can not share all information due to business concerns. Administratively separate systems also typically have independent data plane resources, as each business will generally want to control their own set of devices. BGP (across ISPs) [101], Wiser [103], P4P [155], and VDX [114, 115] (Ch. 3) provide examples of administrative separation.

Internet-scale systems

Many modern systems require complex internet-scale performance optimization, but also need responsiveness. For example, internet-scale video performance optimization may take minutes to run [51, 117], yet if a new client joins the system between computations, they should not be forced to wait for the next round of the optimization, as they will likely stop watching the video. In order to combat slow global optimization, these modern systems introduce an additional control plane

that can run at a much faster timescale, that handles events like new user joins. Systems with this kind of timescale separation can not share information precisely when its needed (e.g., the slow control plane can not help the fast control plane when a new user arrives), thus they can not use coordination designs that require complete information sharing. These systems, by definition, have two control planes making decisions for shared data plane resources. Klein [132], Bohatei [43], C3 [51], VDN [117] (Ch. 4), etc. provide examples of internet-scale systems.

1.4 Recipes for control plane coordination

Here we place the three scenarios from §1.3 (layering, administrative separation, and internet-scale systems) into our design space. Figure 1.5 groups different systems into these scenarios. Combining our knowledge of coordination mechanisms from Figure 1.6, we derive the following simple set of design **recipes** for control plane coordination:

- Systems designed around *layering* should try using *transparency* for coordination.
- Systems designed around *administrative separation* should try using *priority ranking* for coordination.
- Systems designed for *internet scalability* should try using *hierarchical partitioning* for coordination.

Systems involving layering generally have a single entity that has full knowledge of all aspects of all layers. These systems are split for modularity, and likely gives up some performance. The entity in charge of all layers could add specialized code to each layer to regain this performance (cross layer optimization). This is what we defined as transparency (§1.1). We explore this scenario in Chapter 2.

Systems involving administrative separation by definition involve multiple different entities (e.g., businesses) controlling different parts of the system. Business concerns forbid these entities from sharing all information. Generally, each entity only make decisions for their own set of data plane resources. Priority ranking (§1.1) provides a way for entities to share their concerns (e.g., avoid using these paths within my network) with each other without needing to explicitly show the reasons for these concerns (e.g., these paths are double the cost of these other paths). We explore this scenario in Chapter 3.

Systems built for internet scalability involve one entity, but have a global and local control plane. While lacking the hard administrative boundaries found in the administrative separation scenario, internet-scale systems still have trouble sharing completely up-to-date, fine-grained information with each other due to granularity separation or timescale separation. Additionally, these systems generally have both the global and local control planes make decisions for shared data plane resources. Hierarchical partitioning (§1.1) has the global control plane take priority over the local control plane, and has the local control plane work within the "slack" provided by the global control plane. This provides an effective means of coordination between the two control planes. We explore this scenario in Chapter 4.

1.5 Scope

The focus of this thesis is to provide a set of recipes for control plane coordination when designing split control plane systems in a specific set of diverse scenarios. The primary focus is on three specific scenarios (layering, administrative separation, and internet-scale systems). Other scenarios may exist that require complex combinations of the coordination mechanisms presented, or perhaps alternate coordination mechanisms all together. Furthermore, while the design space presented in §1.2.3 uses two axes (*information sharing* and *shared resources*) that represent fundamental aspects of coordination (interfaces and decision making, respectively), other scenarios with alternate solutions may not fit cleanly within the design space as drawn. Interesting alternative scenarios likely are constrained to partial information sharing, but my require an additional axis beyond *shared resources* to differentiate them from administrative separation and internet-scale systems. We touch more on these limitations in §5.2.

1.6 Goals

In this thesis, our primary goal is to examine differences between different networked system involving split control planes, in order to understand why they are often solved with different coordination mechanisms. To this end, we argue that there are three goals fundamental to this work:

1. **Identify common scenarios that naturally lead to split control planes and *why* certain coordination mechanisms help eliminate adverse interactions in these scenarios:** We wish to show that there are key scenarios with wildly different properties (i.e., layering, administrative separation, and internet scalability; §1.3) that each lead to unique coordination mechanisms (transparency, priority ranking, and hierarchical partitioning, respectively; §1.4), due to fundamental differences between scenarios (constraints on information sharing and sharing of data plane resources; §1.2.3).
2. **Show that these coordination design recipes provide natural guidelines for system design:** We wish to show that knowledge of these coordination design recipes aids in solving real-world system design problems, and helps provides context to previous ad hoc designs.
3. **Provide insight into efficient implementations of control coordination mechanism:** We wish to show that control plane coordination mechanisms are not infeasible to build efficiently, in a variety of contexts.

1.7 Contributions

The contributions of this thesis come naturally from the goals:

- In order to ease future system building in scenarios built around split control planes, we provide a set of **recipes** for coordination (§1.4): layering systems should try transparency; administratively separate systems should try priority ranking; systems trying to achieve internet scalability should try hierarchical partitioning. To understand why these scenarios

lead to these coordination mechanisms, we provide an initial **design space** primarily built around these scenarios (§1.2.3). We show that these scenarios require different coordination mechanisms because they fundamentally differ in terms of what information can be shared, and whether they make decisions for disjoint or shared data plane resources. What information can be shared constrains the interface between control planes, and sharing data plane resources constrains the style of decision making.

- To both understand how to use our coordination design recipes in practice, as well as how to implement coordination mechanisms efficiently, we present a **case study** of systems we built in a variety of contexts (Chapters 2–4).
- We examine *layered* control planes, in the context of reconfigurable datacenter networks. While layering trades performance for modularity, there are no issues blocking information sharing between layers. We can regain performance though transparency (here "cross-layer optimization"), as we show with our emulator Etalon [116] (Chapter 2).
- We examine control planes that are *administratively* separate, in the context of content brokering. Not all information can be shared (due to business concerns), and there are no shared resources. Our system, VDX [114, 115] (Chapter 3), uses priority ranking (here a "marketplace-style interface"), between a content broker and CDNs, to remove many problems brokers and CDNs cause for each other, in terms of performance and cost.
- We examine control planes for *internet-scale systems*, in the context of live video delivery. While not all information can be shared because of timescale separation, data plane resources are completely shared between the control planes. Thus, our system, VDN [117] (Chapter 4), uses hierarchical partitioning (here "hybrid control") to provide the performance of a centralized controller, with the responsiveness of a distributed one.

The rest of this thesis focuses on this case study (Chapters 2–4) before concluding (Chapter 5).

Chapter 2 Control Planes in Different Layers: A Case Study in Reconfigurable Datacenters

In this chapter, we explore one example, in detail, where transparency can be used to overcome adverse interactions between control planes. Specifically, we look at how application-/transport-layer assumptions about the network can cause issues with in-network scheduling, in reconfigurable datacenters. For example, applications like HDFS make assumptions about what traffic patterns are simple to deliver over the network, which no longer hold true in reconfigurable datacenters.

We argue that these incorrect assumptions lead to decisions that are sub-optimal, and that building specialized versions of applications, endhost stacks, and switches (i.e., *c*ross-layer optimization) allows us to overcome these issues. Functionally, cross-layer optimization means building awareness into each layer about the specifics of the other layers. Doing this requires deep insight into how the other layers function and why they make certain decisions. As datacenter operators have full control over all aspects of their endhosts and network infrastructure, there are no fundamental limitations in understanding other layers. Thus, we argue that transparency (in this scenario usually called cross-layer optimization) is the correct way to avoid adverse interactions between endhost and in-network control planes. Figure 2.1 summarizes this in the context of our control plane coordination design space from §1.2.3.

Reconfigurable datacenter networks serve as an illustrative (and timely) context; increasing pressure for higher throughput, better response times, and lower cost in datacenters have lead to proposals augmenting traditional packet networks with very high bandwidth reconfigurable circuits. These proposals have focused on switch implementation or scheduling, not end-to-end challenges. In this chapter, we identify three key challenges: 1) rapid bandwidth fluctuation, 2) poor demand estimation, and 3) difficult-to-schedule workloads. Bandwidth fluctuation requires TCP to immediately jump in sending rate for <10 RTTs; poor demand estimation from shallow ToR queues leads to inefficient circuit schedules; and finally, certain application workloads are fundamentally difficult to schedule.

To overcome these challenges, we build an open-source reconfigurable datacenter network emulator, Etalon, for public testbeds. This emulator is a means to an end; using Etalon provides insight into the causes and effects of adverse control plane interactions in this context. We find solutions at different layers: we combat 1) bandwidth fluctuation with dynamic in-network queue resizing to ramp up TCP earlier, 2) poor demand estimation by communicating endhost stack buffer occupancy, leading to more accurate schedules, and 3) difficult-to-schedule workloads by

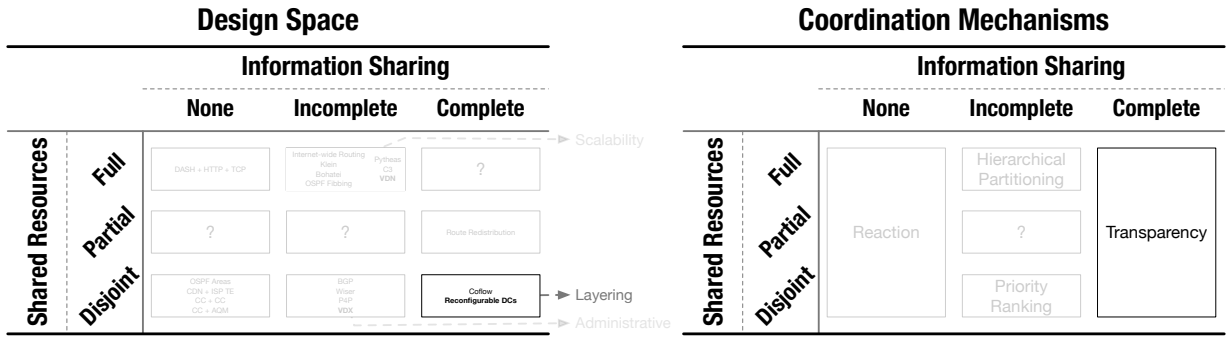


Figure 2.1: This chapter focuses on end-to-end challenges seen in reconfigurable datacenters due to layering, as shown with Etalon. Looking back at our design space (§1.2.3), layered systems (e.g., Etalon, Coflow) can share all information needed for coordination, but make decisions for their own data plane resources. These systems should coordinate using transparency (§1.1).

rewriting application logic (e.g., HDFS' write placement algorithm). We conclude that cross-layer optimization is necessary and provides the most benefit at higher layers.

2.1 Introduction

Modern datacenter (DC) applications have staggering compute and storage requirements, leading to increased pressure for high bandwidth, low latency, high port count, and low cost networks to connect them. Traditional packet switches are hitting CMOS manufacturing limits, unable to simultaneously provide high bandwidth and port counts [109]. Thus, researchers have proposed augmenting DCs with reconfigurable circuit switches (e.g., optical, wireless) that provide high bandwidth between racks on demand [28, 42, 52, 61, 62, 84, 97, 130, 150, 164]. These *reconfigurable DC networks* (*RDCNs*; hybrid circuit + packet networks), however, are less flexible than traditional networks, as adding/removing bandwidth has a non-trivial reconfiguration penalty during which the circuit switch is unavailable.

Prior work has generally focused on two thrusts: switch implementation [42, 52, 61, 62, 84, 97, 130, 150, 164], or scheduling [20, 95, 98]. While important, little focus has been on end-to-end challenges faced by real applications and network stacks. While some end-to-end problems on switches with millisecond-scale reconfiguration have been explored [42, 150], modern μs-scale switches [52, 97, 130] have changed the nature of these problems, as well as the solutions needed.

We identify three such challenges in modern RDCNs:

1. **Rapid bandwidth fluctuation:** while circuits need to be enabled for a long period of time relative to the reconfiguration penalty, circuit uptime may be only a few (e.g., <10) round-trip times (RTTs), causing rapid bandwidth fluctuation. TCP's additive increase is too slow to utilize the large (e.g., 10×) temporary bandwidth increase, leading to low circuit utilization (§2.4).

2. **Poor demand estimation:** RDCN schedulers require accurate traffic estimation to produce efficient schedules. Prior work suggests Top-of-Rack (ToR) switch queues as a possible source

of estimates [42, 95, 130]. We find that these queues must be shallow to provide low latency, meaning most demand is hidden from the scheduler on endhosts. This makes it difficult to differentiate large and small flows, leading to idle schedules (§2.5).

3. **Difficult-to-schedule workloads:** certain workloads are fundamentally difficult to schedule on RDCNs efficiently. Workloads with large, all-to-all flows (i.e., lacking skew and sparsity) waste time repeatedly reconfiguring the network (§2.6).

These challenges arise from broken assumptions made by endhosts about the network: TCP assumes bandwidth does not predictably fluctuate at short timescales, the network stack assumes the network doesn't perform better if it can see demand in advance, and applications assume that all traffic patterns are equally easy to deliver. Either all layers need additional specialization for this new network (cross-layer optimization), or interfaces need to be changed to accommodate different behaviors or expose more info. As one entity controls all endhosts / networks in the DC, cross-layer optimization is the easiest solution.

We use cross-layer optimization to overcome these challenges at the lowest layer possible, providing transparency, less deployment pain, and keeping higher layers general:

1. **Overcoming rapid bandwidth fluctuation with *dynamic in-network buffer resizing*:** Increasing ToR queue sizes in advance of a circuit start gives TCP time to ramp up its sending rate and fill the circuit as soon as it gets it (§2.4). *Cross-layer: [network understands TCP behavior]*

2. **Overcoming poor demand estimation with *endhost ADUs*:** An interposition library reports data waiting on endhosts to the scheduler by tracking the sizes of `write()`s / `send()`s in unmodified applications. The scheduler uses these sizes (though not the boundaries) to decide which flows will benefit most from transiting the circuit switch (§2.5). *Cross-layer: [network understands app behavior]*

3. **Overcoming difficult-to-schedule workloads by *rewriting application logic*:** Modifying applications to introduce skew and sparsity into workloads results in schedules that require less reconfiguration. We demonstrate this with HDFS' write placement algorithm (§2.6). *Cross-layer: [apps understand network behavior]*

We argue that these solutions operate at the lowest layer possible; dynamic buffer resizing *in-network* is enough to insulate TCP from bandwidth fluctuation, exposing demand (ADUs) in the *endhost stack* is the only way to provide proper estimates without greatly increasing switch queuing delay, and changing *application* behavior is the only way to fundamentally change the workload.

To evaluate the efficacy of these solutions, we design and implement an open-source RDCN emulator, *Etalon*[1], for use on public testbeds (§2.3). Tests on CloudLab's 40Gbps APT DC cluster [35, 41] show: 1) dynamic buffer resizing can reduce median packet latency by ~36%, 2) ADUs can reduce median flow-completion time by 8× for large flows, and 3) modifying HDFS can reduce tail write times by 9×. We conclude that while cross-layer optimizations involving higher layers are harder to implement (e.g., modifying individual applications), they may provide the greatest benefit.

[1]Named after an optical filter used for solar observation. Source code: http://github.com/mukerjee/etalon

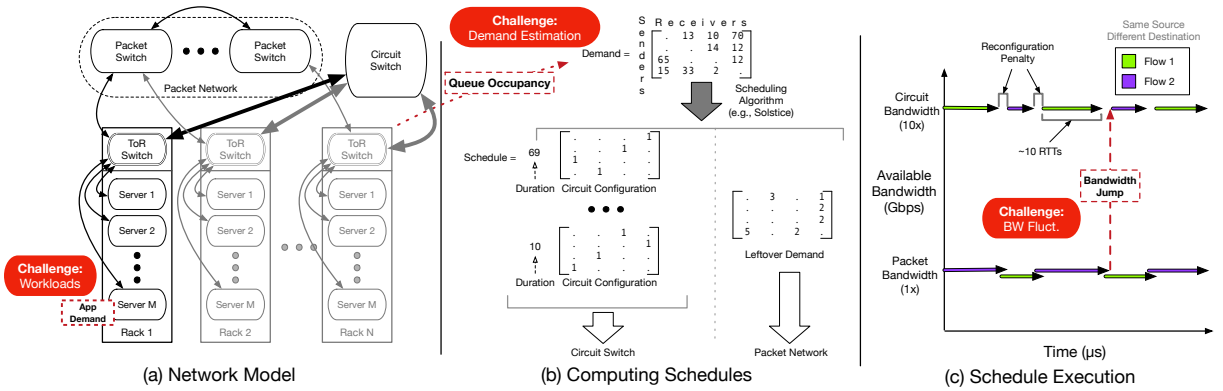


Figure 2.2: Overview of RDCNs. N racks of M servers connect to a low bandwidth packet network and a high bandwidth circuit switch. Application demand is sent through ToR switches to circuit or packet switches. This network requires scheduling, decomposing demand (e.g., ToR queue occupancy) into explicit circuit schedules and "leftovers" for the packet network. During schedule execution, circuit changes incur a reconfiguration penalty, which downs the circuit switch. Flows must use very brief bandwidth jumps. This model leads to three challenges.

To summarize, we make three contributions in this chapter:

1. We analyze three critical end-to-end challenges in RDCNs (rapid bandwidth fluctuation, poor demand estimation, and difficult-to-schedule workloads) caused by erroneous assumptions between layers.
2. We design solutions that require modifications at varying layers (in-network, network stack, application).
3. We design and implement an emulation platform, Etalon, for evaluating RDCNs end-to-end with real applications, finding that solutions at higher layers (while more painful to implement) lead to more benefit.

2.2 Setting

To better understand the challenges and solutions presented in this chapter, we first examine RDCNs in detail in Figure 2.2. While we use optical circuit switching [42, 97, 130, 150] as an illustrative example for the rest of the chapter, the results generalize to other reconfigurable technologies (e.g., free-space optics [52, 62], 60GHz wireless [61, 84, 164]). We eschew older millisecond-scale reconfigurable switches [42, 150] for modern μs-scale switches [52, 97, 130], as the nature of end-to-end challenges and solutions differ with timescale.

2.2.1 Network Model

We consider an RDCN of N racks of M servers, each containing a Top-of-Rack (ToR) switch (Figure 2.2(a)). ToRs connect racks to a packet network (one or more switches) and a single circuit switch. The packet switches are low bandwidth (e.g., 10Gbps), but can make forwarding decisions

for individual packets. The circuit switch is high bandwidth (e.g., 100Gbps), but makes forwarding decisions (i.e., sets up circuits) at much longer timescales to amortize a *reconfiguration penalty*.

We make the pessimistic assumption that during circuit reconfiguration no circuit links can be used, following prior work [97, 98, 130], allowing us to apply our results to a larger set of technologies. The packet switch, however, can be used at all times. Both switches source packets from $N \times N$ virtual output queues (VOQs) on the ToRs. The circuit switch is queue-less; it functions as a crossbar, only allowing configurations that form perfect matchings [20, 42, 97, 98, 130, 150] (i.e., a given sender is connected to exactly one receiver and vice-versa). Thus, at any point in time, the circuit switch may at most drain one VOQ on each ToR, whereas the packet switch may drain multiple VOQs.

2.2.2 Computing Schedules

Network scheduling in RDCNs is mapping rack-level demand to a set of circuit configurations (circuit switch port-matchings) with corresponding time durations. Any "leftover" demand is handled by the low-bandwidth packet switch (see Figure 2.2(b)). Borrowing terminology from prior work [130], we refer to a set of circuit configurations as a *week* of one or more variable-length *days* (individual circuit configurations), each followed by a *night* (down time from reconfiguration). Nights are generally 10-30μs [52, 97, 98, 130]. To allow for 90% link utilization, the average day length must be ≥9× the night length (e.g., 90-270μs). Weeks must be sufficiently long to amortize schedule computation (e.g., 2ms).

Scheduling is a three-step loop: 1) demand for the next week is estimated (e.g., through ToR VOQ occupancy), 2) an algorithm computes the schedule for the next week, and 3) the schedule is disseminated to the circuit switch. Scheduling algorithms for RDCNs (e.g., Solstice [98] and Eclipse [20]) use skew and sparsity in demand to minimize the number of circuit configurations.

2.2.3 Schedule Execution

Once a schedule is disseminated to the circuit switch, it runs the circuit configurations for their respective durations (see Figure 2.2(c)). After reconfiguration, a flow may transition from packet to circuit and vice-versa, but time spent on the circuit switch is likely only a few RTTs (e.g., <10). Flows must cope with these large (e.g., 10×) bandwidth variations.

2.2.4 Challenges

Three end-to-end challenges arise naturally (Figure 2.2):

1. **Rapid bandwidth fluctuation:** can TCP efficiently use a 10× increase in bandwidth in so few RTTs?
2. **Poor demand estimation:** efficient schedules require accurate demand estimates. Is ToR queue occupancy accurate enough for scheduling a week worth of demand (e.g., 2ms)?
3. **Difficult-to-schedule workloads:** schedulers require skew and sparsity for efficiency. Do all DC application workloads have these characteristics?

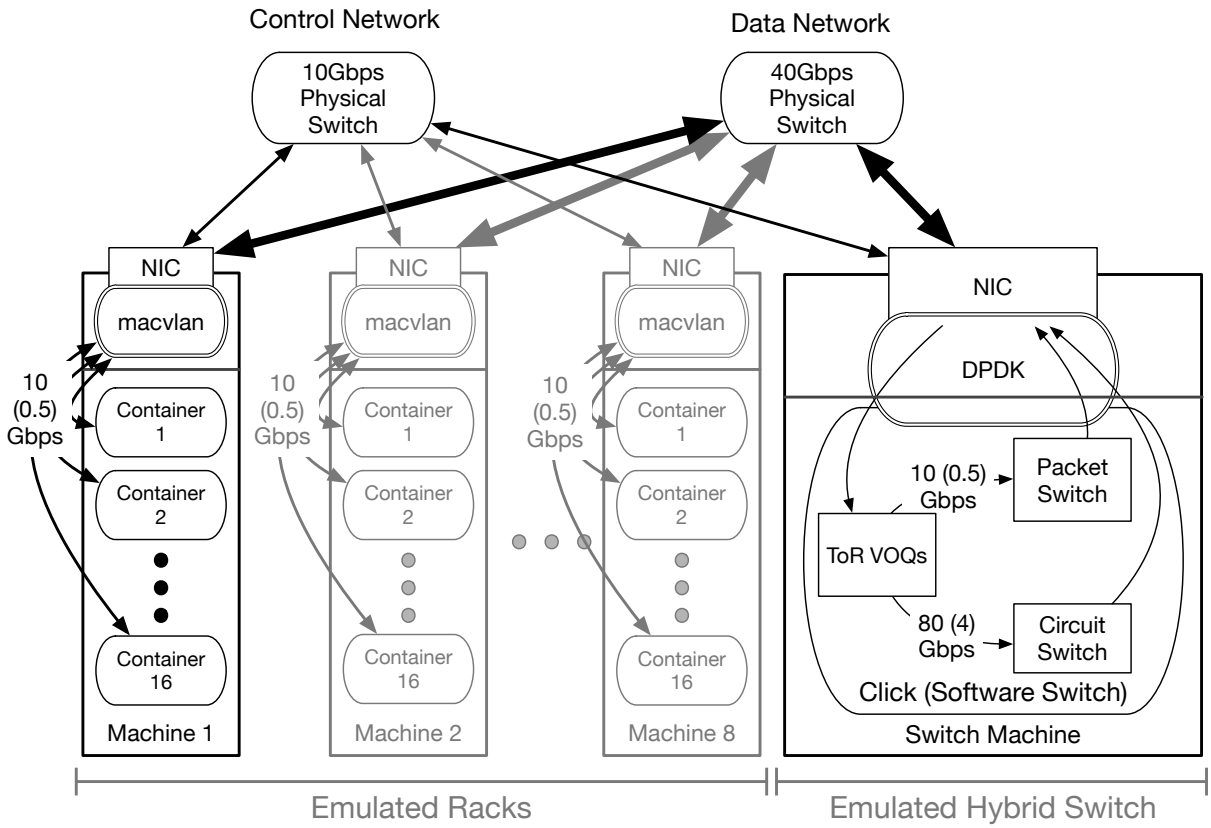


Figure 2.3: Overview of Etalon emulating 8 racks of 16 servers. Another machine emulates ToR VOQs, circuit & packet switches. Time dilation provides faster links.

2.3 Etalon

In this section, we present our open-source emulator, Etalon, which measures end-to-end performance of *real applications and endhost stacks* on emulated RDCNs in public testbeds.

2.3.1 Overview

Figure 2.3 presents an overview of Etalon. Each of the N physical machines emulates a rack of M servers using Docker containers [38, 110]. Containers are connected to the physical NIC using macvlan [39]. Macvlan virtualizes a NIC into multiple vNICs, connecting them with a lightweight layer-2 software switch. `tc` limits link bandwidths between the containers and the vswitch, emulating a server-to-ToR link. A separate physical machine serves as the emulated hybrid switch, running a software switch (Click [88]) using DPDK [40] to process packets at line rate. ToR VOQs are emulated in the software switch for convenience, making circuit and packet link emulation straightforward. We maintain a separate control network for convenience, although it is not strictly necessary.

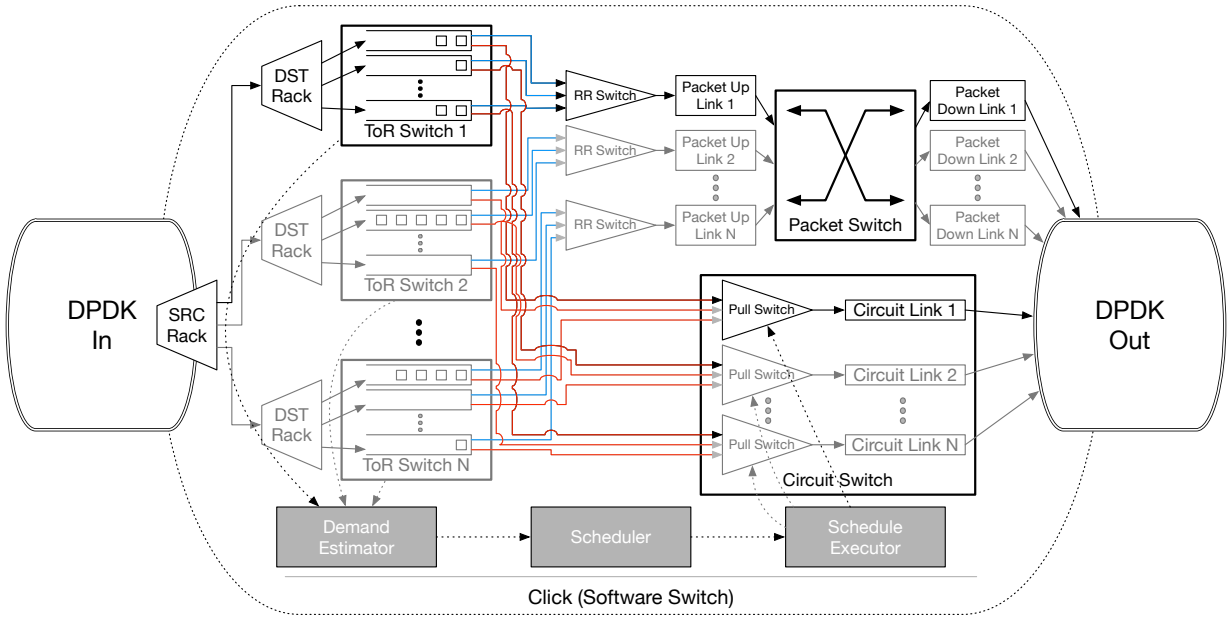


Figure 2.4: We emulate a circuit and packet switch using Click with DPDK. ToR VOQs are pulled round-robin by the packet switch. Circuit links pull from ToR VOQ based on current pull switch settings. The circuit schedule is computed using ToR VOQ occupancy and is executed by adjusting pull switch inputs.

2.3.2 Software Switch

Figure 2.4 shows the software switch's internals. Packets enter the switch via DPDK [40] and are sent to a ToR VOQ based on their *(source, destination)* rack pair. Packets are pulled from each VOQ by the packet switch or circuit switch.

Packet up link i is connected to the N VOQs in ToR i, pulling packets from these VOQs in a round-robin fashion. A packet pulled by a packet up link enters the packet switch, where it is multiplexed over a packet down link and exits using DPDK. If a packet would be dropped in the packet switch, it is held at the ToR VOQ (similar to PFC [71]).

Circuit link i is wired to the ith VOQ of each of the N ToRs via a *pull switch*. A settable "input" value on pull switch i connects circuit link i to exactly one VOQ at a time. After packets transit the circuit link, they exit using DPDK.

Our software switch contains three control elements (shown in gray): demand estimator, scheduler, and schedule executor. Demand estimator estimates rack-to-rack demand using ToR VOQ occupancy. The scheduler computes a schedule from this demand, which is run by the schedule executor by modifying the circuit link pull switches' "input" value. Our scheduler element is pluggable; we implement Solstice [98] as an example, but modify its objective to schedule maximal demand within a set window W (like Eclipse [20]), rather than scheduling all demand in unbounded time.

	Expected	Experimental Mean	Std. Dev.
Circuit day	180μs	180.25μs	0.04μs
Week length	1400μs	1400.02μs	0.05μs
Packet utilization	10Gbps	9.93Gbps	0.75Gbps
Circuit utilization	80Gbps	79.99Gbps	1.60Gbps

Table 2.1: Validating Etalon's timing and throughput.

2.3.3 Time Dilation

As the goal of Etalon is to emulate RDCNs on public testbeds, the software switch machine likely only has one high-speed NIC, yet we wish to emulate a switch with N high-speed ports. We solve this with *time dilation (TD)*. Originally proposed for VMs [58, 59, 148] (and recently containers [93, 157, 158]), TD provides accurate emulation of higher bandwidth links by "slowing down" the rest of the machine. We implement an open-source TD interposition library called LibVirtualTime (LibVT[2]) which applies TD to many common syscalls in unmodified applications. We catch: `clock_gettime()`, `gettimeofday()`, `sleep()`, `usleep()`, `alarm()`, `select()`, `poll()`, and `setitimer()`. Extending LibVT to other calls is trivial. We verify that common network benchmarks (iperf [73], iperf3 [74], netperf [65], sockperf [107], flowgrind [165, 166], ping) perform correctly with TD. We additionally limit CPU time for containers with respect to TD.

2.3.4 Testbed

All experiments are performed using Etalon on the public APT DC cluster [41] (CloudLab [35]), with 8 "racks", each containing 16 "hosts" (Figure 2.3). Each machine has an 8-core 2.1GHz Xeon and 16GB of RAM. Each machine is connected to a 40Gbps data network (with jumbo frames) and a 10Gbps control network, and uses TCP Reno.

We use a time dilation factor (TDF) of 20× to emulate an 8-port 10Gbps (0.5Gbps)[3] packet switch and an 8-port 80Gbps (4Gbps) circuit switch. Outside of TD, 0.5Gbps * 8 + 4Gbps * 8 = 36Gbps total traffic, below our 40Gbps physical link speed. Each per-container link is limited to 10Gbps (0.5Gbps). Packet switch up/down links have 5μs (100μs) delay each. The circuit links have 30μs (600μs) delay. While this delay is 3× higher than the packet links, prior work assumes a single circuit switch, requiring long fibers. We perform circuit link delay sensitivity analysis in §2.4.5.

We emulate a circuit switch with reconfiguration penalty (night length) 20μs (400μs) and run the Solstice [98] scheduler over W = 2ms windows (week length). To avoid out-of-order delivery, we disable the packet switch when the circuit is available for a rack pair. For the rest of this chapter all timing values and bandwidth values presented will be time dilated.

[2] Available: http://github.com/mukerjee/libVT

[3] values in parenthesis represent bandwidth/delay outside TD

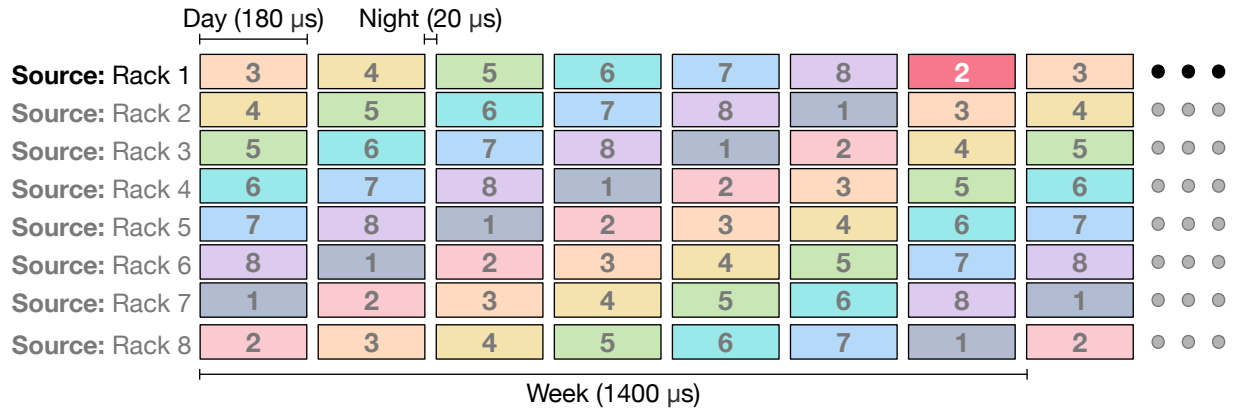


Figure 2.5: Strobe schedule used in §2.4 experiments. All *(source, destination)* pairs of racks can communicate 1/7th of the time. Traffic is generated from rack 1 to rack 2.

2.3.5 Validation

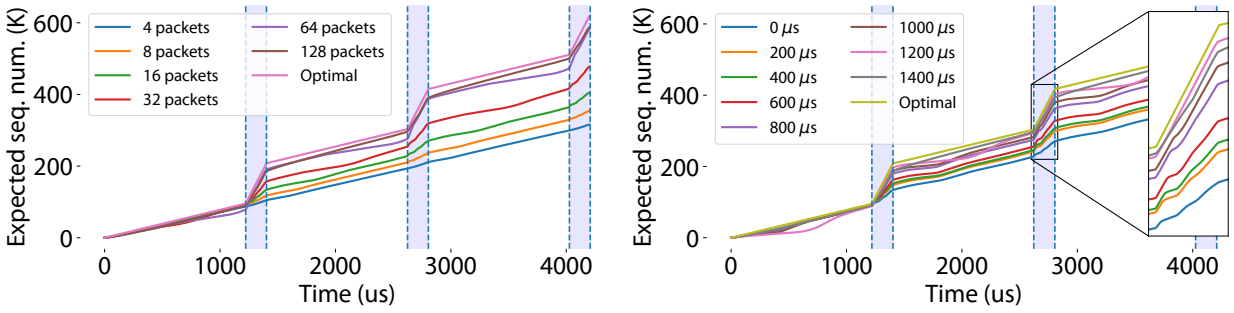
We validate Etalon using a strobe schedule (Figure 2.5) of seven days of minimal length (9 * night length, 180μs), while sending TCP traffic between all pairs of racks for 2 seconds. ACKs are diverted around the switch for this one experiment to avoid ACK loss. We find timing and bandwidth to be as expected, shown in Table 2.1.

2.4 Overcoming rapid bandwidth fluctuation with dynamic buffer resizing

While circuits need to be enabled for long periods of time relative to the reconfiguration penalty, this may be only a few (e.g., 3) RTTs, causing rapid bandwidth fluctuation. TCP's additive increase requires a longer timescale to make use of the extra (e.g., 8×) bandwidth. While TCP's interaction with RDCNs has been explored previously, it was on millisecond-scale reconfigurable switches (with thousands of RTTs per circuit configuration) [150], or required kernel/NIC modifications to pause/unpause flows according to the schedule [97, 130]. We explore an entirely *in-network* solution, dynamic buffer resizing, which to our knowledge has not been explored in the context of network scheduling.

2.4.1 Understanding the problem

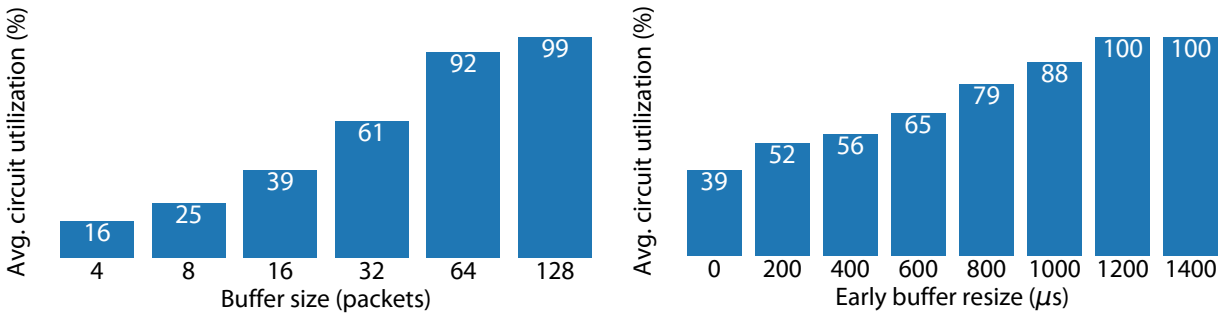
Modern reconfigurable switches can only be efficient if they make forwarding decisions for hundreds of packets at a time (e.g., for 80Gbps links / 9000 byte packets / reconfigured every 180μs), due to a reconfiguration penalty (e.g., 20μs). To avoid generating schedules with too many configurations (thus many reconfigurations), schedulers like Solstice [98] have a minimum day length parameter (180μs; 9 * night length in Etalon). Thus, in our worst case (minimal length circuit configuration), TCP flows are expected to immediately burst packets at 8× their rate for 3 RTTs



(a) Static buffer sizing

(b) Dynamic buffer resizing

Figure 2.6: Sequence plots for (a) buffer sizes / (b) how early buffers are resized. Circuit use is shaded.



(a) Static buffer sizing

(b) Dynamic buffer resizing

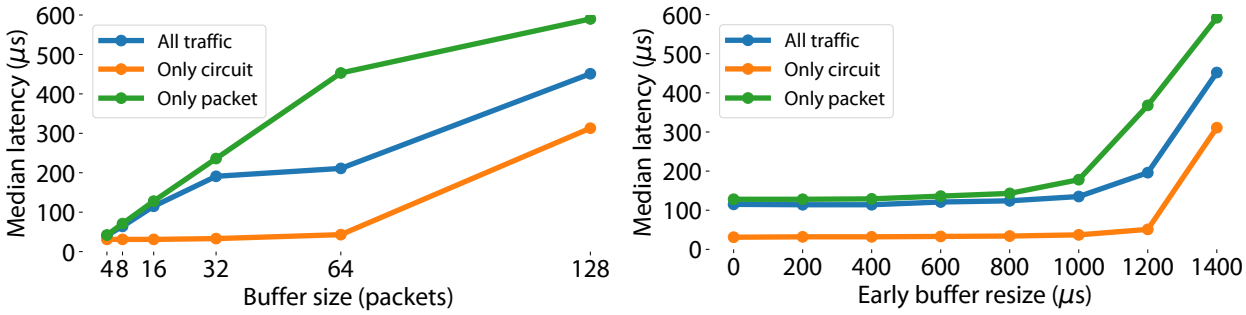
Figure 2.7: Circuit utilization as a function of (a) buffer size / (b) how early buffers are resized.

when a circuit starts. This is difficult for TCP Reno's one-packet-per-RTT additive increase, as we will show.

We illustrate the problem in an experiment. Using Etalon, we fix a strobe schedule of 7 days, covering all *(source, destination)* rack pairs (except where the source and destination are the same rack), shown in Figure 2.5. We fix the day length to the minimum, 180μs, thus, our schedule's week repeats every 1400μs. Solstice [98] would produce this schedule for an all-to-all workload (e.g., MapReduce's shuffle [37]). We generate 2-second-long TCP flows using flowgrind [165, 166] from rack 1 to rack 2. A fixed schedule and small flow count is illustrative, but limited. We relax these assumptions in §2.5.

Figure 2.6a shows three weeks versus the expected next TCP sequence number. We vary static ToR VOQ sizes from 4 to 128 packets, and plot an optimal line numerically, based on link bandwidths. Circuit uptime is shaded. The slope of each line is the flow's bandwidth; the area under the curve is data transmitted. As optimal shows, we expect an 8× bigger slope during circuit uptime. Circuit utilization is how well the slope matches optimal's slope during uptimes.

For small buffers, circuit utilization is low, while large buffers fare better. Figure 2.7a illustrates this concisely, showing circuit utilization directly. While TCP grows at one-packet-per-RTT regardless of buffer size, larger switch buffers, allow flows to have a "backlog" of packets queued up, draining during circuit uptime.



(a) Static buffer sizing

(b) Dynamic buffer resizing

Figure 2.8: Median latency for (a) buffer size / (b) how early buffers are resized, split per switch.

Finding the "proper" buffer size for a hybrid switch is difficult; common wisdom is to use the bandwidth-delay product (BDP), but the BDP is different for the packet and circuit switches (~4 packets and ~32 packets respectively). Using a weighted sum based on the schedule gives ~8 packets. As seen, none of these values provide maximum circuit utilization; 64+ packets are needed.

Queues this large have high latency. Figure 2.8a shows median latency, measured per-packet entering and leaving the software switch. We see that the packet switch latency grows sharply (as we already past the BDP for the packet switch), becoming high compared to latency over the circuit switch.

We want the best of both worlds: can we minimize latency and simultaneously maximizing our circuit utilization? No static buffer setting achieves this. We propose an entirely in-network solution to overcome this, dynamic buffer resizing.

2.4.2 Dynamic buffer resizing

We propose dynamically resizing ToR VOQ capacity to fix the effects of rapid bandwidth fluctuation on TCP. The key insight is bandwidth fluctuation within RDCNs is not arbitrary; it is part of a schedule and is known in advance.

With this knowledge, we can choose the ToR buffer sizes with respect to the packet switch or circuit switch BDPs in real time. By itself, a packet switch can effectively achieve full throughput with a very small buffer (e.g., 4 packets), but large buffers cause queuing delay (Figure 2.8a). And, by itself, a circuit switch needs larger buffers (e.g., 32+) to achieve full utilization, but incurs no queuing delay up to that point.

With dynamic buffer resizing we try to get the "best of both worlds" by keeping buffers small when the packet switch is in use and big when the circuit is in use. Doing this naively (i.e., resize buffers when the circuit comes up) provides little benefit; there is simply not enough time in one day for TCP to grow to fill the circuit link, regardless of how large you make the buffer. Data needs to be available *immediately* at circuit start (either buffered or via a high TCP send rate); ramping up post facto means circuit time is already wasted.

Instead, we dynamically resize ToR VOQs for a *(source, destination)* rack pair *in advance* of a circuit starting for that pair. In scenarios where this pair spends most time using the packet switch, small buffers are used to avoid additional latency. In advance of getting a circuit, this pair's VOQ

size increases, providing time to 1) queue up packets, and 2) ramp up TCP flows, leading to better circuit utilization.

Our buffer resize function has three parameters:

$$\texttt{resize}(s, b, \tau)$$

where s and b are the small and large buffer sizes in packets, and τ is how early a buffer should be resized in advance of the circuit start. For the rest of this chapter we use $s = 16$ and $b = 128$ to slightly favor throughput, although (4, 64) would be reasonable to slightly favor latency in Etalon. τ is a tradeoff; resizing too late means low circuit utilization, but resizing too early increases latency. We vary τ in experiments below. While the value of τ impacts the circuit utilization/latency tradeoff, we find that waiting to resize the buffer back to s after a circuit stops has no benefit, thus we always set buffers back to s immediately after circuit teardown.

2.4.3 Experiments

To test the efficacy of dynamic in-network buffer resizing, we repeat the experiments from §2.4.1 (strobe schedule, TCP flows between racks 1 and 2), but with resizing.

We vary τ from 0 to 1400μs at intervals of 200μs. These values correspond to the length of a day + night in our schedule (Figure 2.5). Thus, at $\tau = 0\mu$s buffers are always size $s = 16$, and at $\tau = 1400\mu$s buffers are always sized $b = 128$.

Figure 2.6b shows a sequence plot for the next expected TCP sequence number for the various τ and a calculated optimal based on link rates. Circuit uptimes are shaded. The earlier we resize, the higher bandwidth obtained, as expected.

Recall that the benefits of resizing come from 1) packet buildup in queues and 2) TCP ramp up. Both are illustrated in the inset in Figure 2.6b: when the circuit comes up, there is an initial region with high slope as built-up packets drain from the queue. Then there is a plateau as TCP waits for ACKs from these packets before sending more data (recall we disable the packet switch in advance of the circuit start to avoid out-of-order packets). The higher the slope after the plateau, the more TCP ramped up before the circuit start. We see both an increase in the length of the initial burst (how many packets built up) and the following slope, as we move to earlier resize times. Achieving line rate requires the initial queue-draining burst to be long enough to overcome the length of the plateau, and then TCP's sending rate equaling the link rate, effectively achieved by $\tau = 1400$.

Figure 2.7b show circuit utilization for the various τ. As expected from the sequence plot, circuit utilization increases rather dramatically when compared to a static buffer size of 16, eventually achieving full utilization for large τ.

The remaining question is whether this utilization comes at the cost of high latency, as it did for static buffers. Figure 2.8b shows how dynamic resizing affects median latency (we show 99th percentile latency in §2.4.4). Surprisingly, median latency is relatively flat until $\tau > 1000\mu$s. More concisely, 5/7ths of a week (1000 / 1400μs) can be spent with large buffers with negligible impact on median latency (even for packets sent over the packet switch), while doubling circuit utilization. **Takeaway:** comparing the results for $\tau = 1000\mu$s to a static buffer with similar throughput (64 packets), **dynamic buffering improves median latency by ~36%**, or for the same latency as a static buffer with 16 packets **increases circuit utilization by ~2×**.

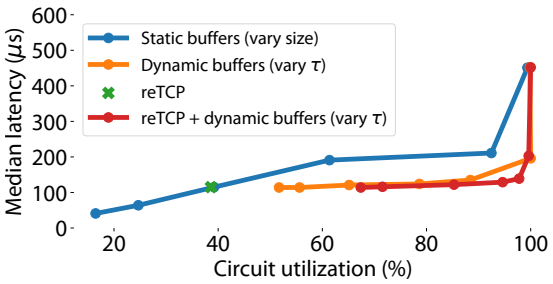


Figure 2.9: Comparing throughput to median latency for various configurations.

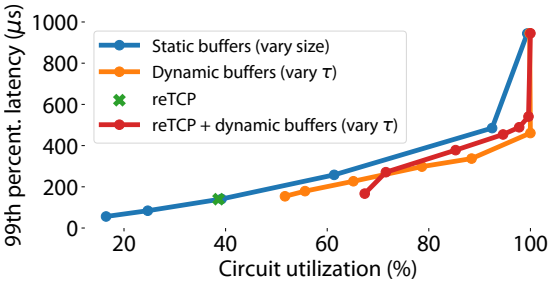


Figure 2.10: Comparing throughput to 99th percentile latency for various configurations.

2.4.4 Incorporating explicit network feedback

Dynamically resizing in-network buffers raised circuit utilization without raising latency (for $\tau \leq 1000\mu$s) and did not require endhost modification. Next we remove that limitation to see if incorporating explicit network feedback about circuit state into TCP offers additional improvement.

For some rack pair (S, D), we modify our software switch to set the ECN-echo (ECE) bit in the TCP headers of ACKs sent by D, if there is currently a circuit enabled from S to D.

We create a pluggable TCP congestion control module (reconfigurable DC network TCP: reTCP) which looks at this stream of ECE bits, multiplicatively increasing it's `cwnd` by $\alpha \geq 1$ on $0 \rightarrow 1$ transitions and decreasing by $0 \leq \beta \leq 1$ on $1 \rightarrow 0$ transitions. We set $\alpha = 2$ and $\beta = 0.5$ empirically. Intuitively, this provides higher circuit utilization; TCP will immediately have higher sending rate on circuit start.

reTCP additionally requires a single line kernel change, as the kernel only passes ECE flags to congestion control modules if ECN is enabled on the system. Enabling ECN lowers `cwnd` upon receiving an ECE marked packet, so we modify the kernel to pass the ECE flag even when ECN is disabled.

Results: Figure 2.9 shows the tradeoff of circuit utilization versus median latency for various static buffer sizes, various τ values (from above) for dynamic buffer resizing, reTCP, and reTCP + dynamic buffers with various τ values. These mechanism are beneficial when they are below and to the right of the "static buffers" curve.

Dynamic buffer resizing is an improvement over static buffers, but interestingly reTCP does not help unless paired with dynamic buffer resizing. Buffer resizing helps because it provides both a backlog of queued packets at circuit start and a higher sending rate for TCP. reTCP without resizing just provides the latter. Combining them helps, resulting in a higher TCP sending rate than dynamic buffers alone. This allows reTCP + dynamic buffers to eek out slightly more circuit utilization for the same τ values.

Figure 2.10 shows the same results for 99th percentile latency. Given that resizing forces flows to use a large buffer for a significant amount of time (e.g., 5/7ths of the schedule for $\tau = 1000\mu$s) it is surprising that tail latency is not worse than static buffers for comparable circuit utilization. Since some portion of time still uses small buffers (e.g., 2/7ths) even for large τ, this may be enough to keep low tail latency. Curiously, reTCP + dynamic buffers mostly does worse than just dynamic buffering in terms of tail latency. This is likely due to the fact that the jumps in `cwnd` caused

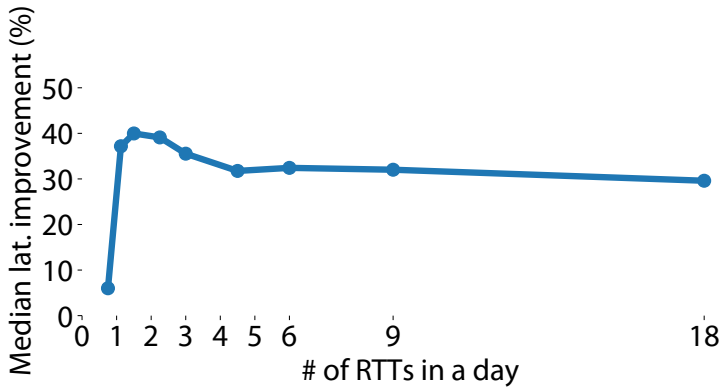


Figure 2.11: Improvement in latency for same circuit utilization for dynamic buffering versus static buffering, as a function of the number of RTTs in day.

by reTCP can sometimes occur when `cwnd` is already very large or very small leading to more variability, increasing tail latency.

2.4.5 Delay sensitivity analysis

Finally, we analyze how circuit link propagation delay affects the usefulness of dynamic buffer resizing. We use 30μs in previous experiments (§2.3). Varying link delay with fixed day length changes how many RTTs are in a day. In our experiments so far, 30μs delay and 180μs days gives 3 RTTs per day. How does changing link delay (and therefore # RTTs per day) affect the improvements in median latency (for the same circuit utilization) that we saw for dynamic buffer resizing (with τ = 1000μs) compared to static buffers?

Figure 2.11 shows the improvement in median latency as a function of RTTs per day. We see that increasing the number of RTTs (decreasing link delay) makes early resizing is less important, as there is now more time for TCP to ramp up after the circuit starts. Decreasing the number of RTTs (increasing link delay) makes early buffer resizing more important, as there is less chance to grow TCP during the day. Once days become shorter than an RTT, little improvement can be made.

2.5 Overcoming poor demand estimation with ADUs

Poor demand estimation leads to inefficient schedules (i.e., unnecessary reconfigurations or circuits that sit idle due to insufficient demand), an issue pointed out in scheduling work that is difficult to correct in-network [20, 95]. We argue that this must be solved at endhosts and demonstrate its impact on real flows over modern switches and schedulers.

2.5.1 Understanding problems with demand estimation

The quality of the schedules produced by an RDCN scheduler is largely based on the accuracy of the demand estimate. If demand estimation is limited by the size of shallow ToR VOQs (e.g., 16

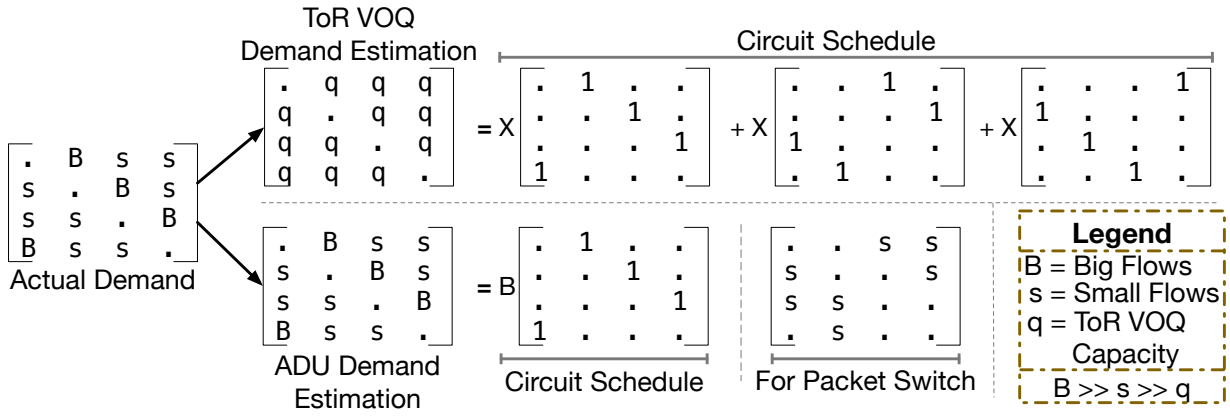


Figure 2.12: The problem of poor demand estimation: big and small flows can't be differentiated in shallow ToR VOQs, leading to long schedules with much idle time. Communicating endhost ADUs solves this.

packets as we suggest in §2.4), then the scheduler cannot tell which flow are large (and should be scheduled on the circuit switch) and which flows are small (and should be scheduled on the packet switch). Figure 2.12 illustrates this; ToR VOQ-based estimation produces long schedules requiring more reconfiguration penalties and idle times where the scheduler mistakenly assumed small flows were large.

Communicating the size of application data units (ADUs) sitting in endhost buffers eliminates this problem. We argue that proper demand estimation must be done in the endhost stack because scheduler overhead is amortized by scheduling a week of demand (3ms in ReacToR [97]; order .1-1ms in Eclipse [20]; 2ms here) at once. Gathering 2ms of traffic at 80Gbps would require per-port buffers for ~2000 (9000 byte) packets, which is likely impractical on a switch and would (greatly) increase packet latency. Using endhost information requires no switch modifications nor inflates latency.

2.5.2 Using ADUs

We gather ADU information by creating an interposition library (libADU) that sends flow five-tuples (addresses, protocol, ports) and ADU sizes to the demand estimator on the software switch, using TCP over our control network (see Figure 2.3). LibADU interposes on `write` and `send` to get ADU sizes, and `shutdown` and `close` to indicate when outstanding demand for this flow should be removed. We modify the demand estimator (Figure 2.4) to keep track of outstanding demand per-flow to ease cleanup on flow shutdown. Alternative solutions could track outstanding demand at the rack level, but would either need to tolerate potential inaccuracies after shutdowns or require more complex cleanup (e.g., timeouts). We additionally modify ToR VOQs to count the amount of application-layer bytes seen per-flow (with care given to avoid double-counting TCP retransmits). While individual ADU sizes are communicated to the scheduler, the scheduler we use (Solstice [98]) does not take advantage of knowing the boundaries between ADUs.

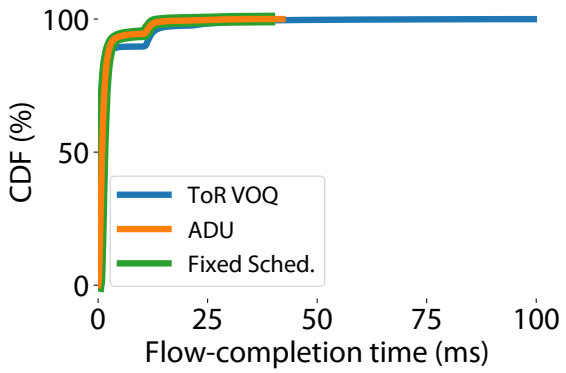


Figure 2.13: Small flow FCT (1 ring).

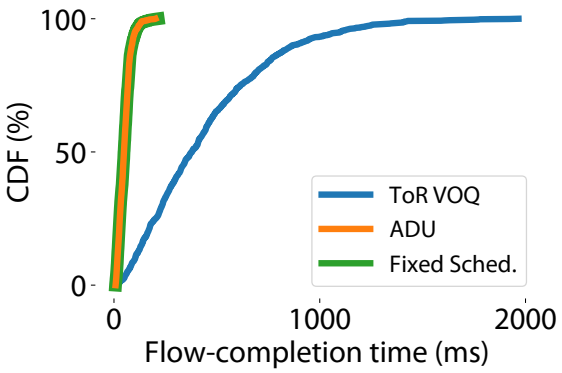


Figure 2.14: Big flow FCT (1 ring).

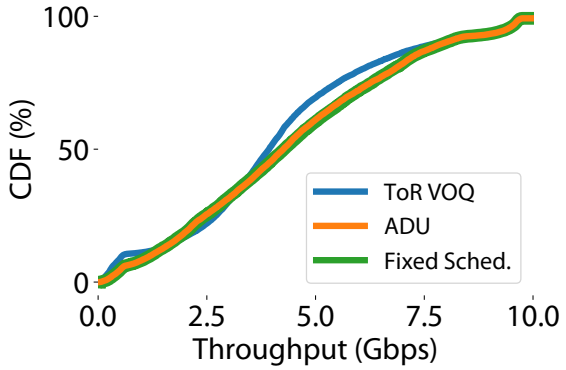


Figure 2.15: Small flow throughput (1 ring).

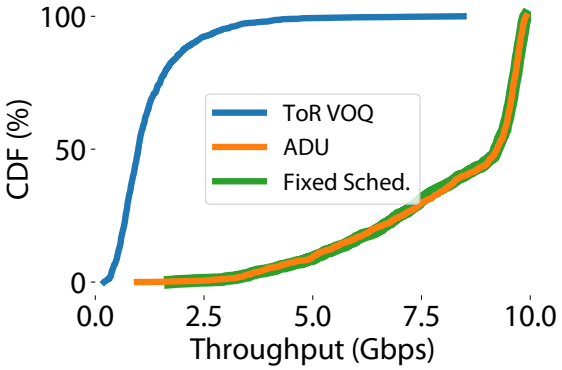


Figure 2.16: Big flow throughput (1 ring).

2.5.3 Experiments

Traffic generation: we test the impact of ADU-based demand estimation by using flowgrind [165, 166] to generate a workload of big and small flows. Big flows consist of 1, 000 to 10, 000 packets, chosen uniformly at random; small flows have 10 to 100. All packets are 9000 bytes. For each flow, we select a start time such that given their start times and sizes, all big flows from a source rack use 1/3 of the circuit bandwidth, and all small flows from a source rack use 1/3 of the packet bandwidth. *(source, destination)* rack pairs are chosen for the big flows so that they form a ring; the remaining racks send small flows. We run this workload for 2 seconds (after which we kill any unfinished flows), resulting in ~10,000 small flows and ~1,000 big flows.

Results: we compare ToR VOQ-based estimation to ADU-based estimation, as well as the single "fixed schedule" that our scheduler would output with perfect demand estimation (i.e., send the single ring of big flows over the circuit switch). We show separate flow-completion time (FCT) CDFs for small and big flows in Figures 2.13 and 2.14, respectively. We find that for small flows, there is not much room to do better, as they are already fairly well serviced by the packet switch, although ADU-based estimation does cut 99% latency by half. For large flows, however, the median (99%) FCT is cut by almost 8 (11)×. This is due to the smaller number of configurations per schedule, meaning fewer reconfiguration penalties/idle small-flow circuits (see Figure 2.12). We also find that ADU-based estimation tightly hugs the fixed schedule, implying that for this workload, ADUs

	ToR VOQ	VOQ + Resize	VOQ + reTCP	ADU	ADU + Resize	ADU + reTCP	Fixed Schedule	Packet (10Gbps)	Packet (20Gbps)	Packet (40Gbps)	Packet (80Gbps)
Median small FCT (ms)	1.52	1.57	1.60	1.51	1.58	1.59	1.49	2.19	1.05	0.71	0.65
99th small FCT (ms)	56.39	76.54	69.86	65.71	79.07	77.80	64.99	36.34	22.25	14.50	5.20
Median big FCT (ms)	1162.04	737.64	660.19	644.01	**110.61**	**102.11**	724.08	1680.35	1025.92	**46.95**	**45.30**
99th big FCT (ms)	1988.84	1917.55	1869.50	1815.74	**432.50**	**406.44**	1860.53	1998.27	1944.73	**134.29**	**114.77**

Table 2.2: Median and 99th percentile flow-completion time for small and big flows (2 ring workload).

provide optimal results. We show throughput for small and big flows in Figures 2.15 and 2.16, respectively. While the trends are the same, small flow throughput shows two peaks for ToR VOQ, likely due to the separation of small flows sent over the packet switch and those accidentally sent over the circuit switch. **Takeaway: ADU-based estimation improves median (tail) big FCT by 8 (11)×.**

2.5.4 Cumulative results

Next we test the techniques from this section (ToR VOQ, ADU, Fixed) together with the techniques from the last section (dynamic buffer resizing and reTCP + resize, both with $\tau = 1000\mu$s). We also try various higher bandwidth traditional packet networks (10, 20, 40, 80Gbps) without circuit switches. We generate the same workload as above, but create two rings of big flows instead of one. Two rings are more difficult to schedule than one ring, requiring two circuit configurations per week (and thus more reconfigurations). We update our "fixed schedule" to include both configurations.

Table 2.2 shows the median and 99th percentile FCT for small and big flows. As with the prior experiment, we find that for small flows, having better demand estimates or mechanisms to increase circuit utilization does not provide much benefit, as these flows are generally better served by the packet switch. Interestingly, all of our mechanisms (and the fixed schedule) increase the tail FCT for small flows compared to ToR VOQ, likely because ToR VOQ accidentally sends some small flows over the circuit switch, giving them more bandwidth than they would get if properly scheduled.

Small flow FCTs on increasingly high-bandwidth packet switches are even smaller, mainly due to increasing capacity as well as their finer-grained per-packet forwarding decisions when compared to circuit switches. Interestingly, while the setup using only a 10Gbps packet switch has median small FCT worse then all other scenarios, its tail small FCT is better than VOQ. This is likely due to less variability in the network, due to the lack of circuit ups/downs.

By accidentally scheduling small flows on the circuit, ToR VOQ starves big flows as we see in its big flow FCT. VOQ+Resize, VOQ+reTCP, ADU, and Fixed Schedule reduce the median (by ~1.5–1.8×), but not tail FCTs. For VOQ+Resize & VOQ+reTCP this is due to inefficient schedules computed from inaccurate VOQ-based demand estimation. For ADU & Fixed Schedule this is due to small static switch buffers that (while providing low per-packet latency) cannot provide the required high throughput to big flows.

We see a big drop (highlighted) in median and tail FCT for big flows with ADU+Resize & ADU+reTCP, ~10-11× over ToR VOQ. ADU+Resize & ADU+reTCP take advantage of ADU-based estimation leading to a repeating two configuration schedule serving only the big flows in this two

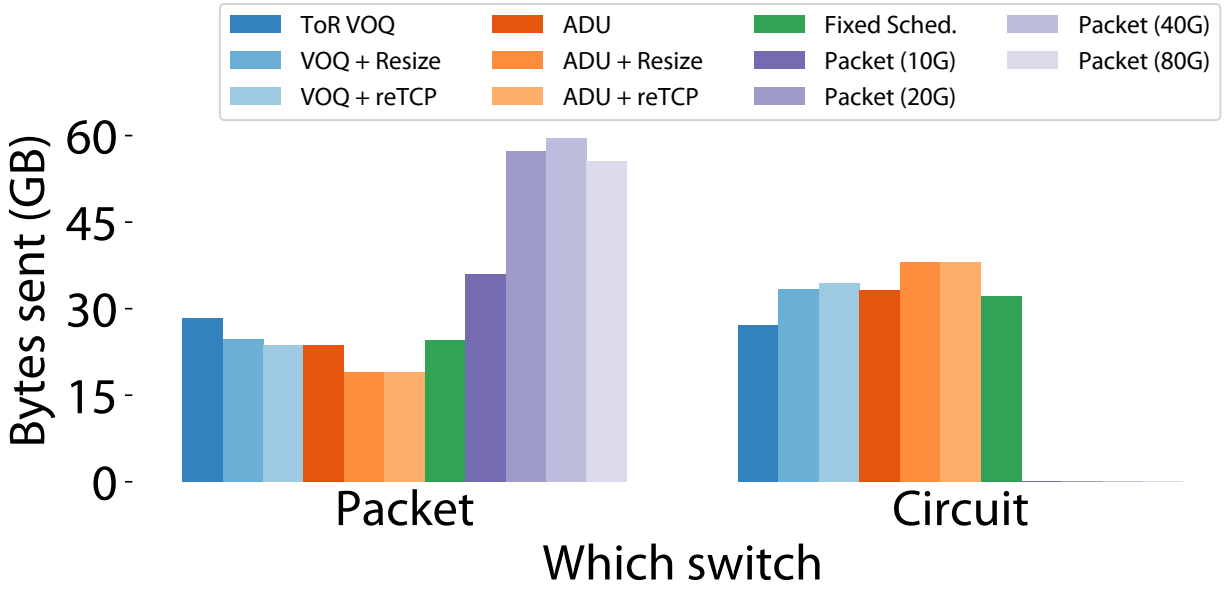


Figure 2.17: Packet and circuit utilization for various methods (2 ring workload). High circuit utilization is key to low flow-completion times in RDCNs.

ring workload; buffer resizing only happens to the big flow VOQs because of this schedule, and VOQs are resized early enough (given τ = 1000μs and week length 2000μs) to pin them to the large size. TCP takes advantage of the permanently large buffers in this workload to provide big flows much higher throughput than with either optimization on its own, greatly lowering FCT.

Replacing an RDCN with a higher-bandwidth traditional packet network (without a circuit switch) can eek out another 2 (3.5) × improvement (highlighted) over our best optimizations in median (tail) big FCT. This additional improvement is due to its finer-grained per-packet forwarding decisions. These improvements require a 4× jump in capacity over our RDCN packet switch (10Gbps to 40Gbps). CMOS manufacturing limits, however, are making it increasingly difficult to make these capacity jumps for high-port count packet switches [109], making RDCNs more attractive.

To better understand the results, Figure 2.17 shows packet switch and circuit switch utilization for each scenario. High circuit utilization is a key factor for low flow-completion time in RDCNs. We see that dynamic buffer resizing and reTCP help move significant traffic to the circuit switch, both for VOQ- and ADU-based estimation. Using a fixed schedule does similarly, but requires knowledge of the demand a priori. A 10G packet switch by itself on this workload is so overloaded that it can not send all bytes within the 2 second time limit, ending with outstanding data.

2.6 Overcoming difficult-to-schedule workloads with application-specific changes

Workloads that are not *skewed* (easily separable into big flows for circuit switch and small flows for packet switch) or *sparse* (few pairs of hosts communicate) are fundamentally difficult to schedule

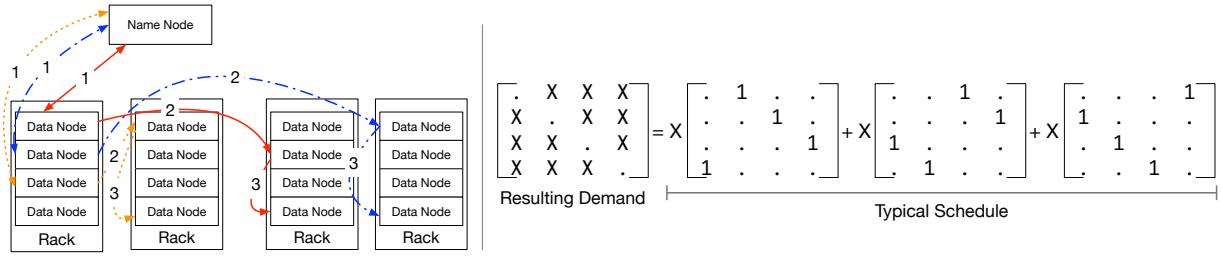


Figure 2.18: HDFS' default write placement algorithm. Data nodes communicate with the name node to learn where to write data replicas (1). The are told to write to themselves and two nodes in another randomly selected rack (2, 3). This results in all-to-all traffic that is difficult to schedule.

on RDCNs [98], as they require many disruptive switch reconfigurations. The techniques from §2.4 can help work around this, but it would be better not to need them in the first place. An ideal workload would require a single circuit schedule; in this case, there are no switch reconfigurations and techniques like dynamic buffer resizing and reTCP are not necessary because TCP would consistently see a large circuit link. Unfortunately, many existing distributed data center application workloads are not as skewed or sparse as we would like. However, in many cases, this is not a fundamental property of the application; their workloads are uniform simply because there was no reason not to be.

In preparation for running on a RDCN, a few simple application-layer modifications can greatly increase performance by introducing skew and sparsity. Applications amenable to this are those whose communication patterns are flexible rather than completely prescribed—that is, a source sending a large flow has multiple choices of recipient. Introducing skew does not require nodes to have very detailed knowledge of the network topology, simply which *circuit group* (e.g., rack) other nodes belongs to. With this information, the nodes in a circuit group should aim to minimize the number of other circuit groups to which they collectively send large flows. For example, in a ring, which can be serviced with a single circuit configuration, each group sends to exactly one other group. In the rest of this section, we use replica selection in HDFS [11] as a running example.

2.6.1 HDFS write placement difficulties and solutions

Original workload: HDFS is a distributed file system made up of racks of datanodes (DNs) and a namenode (NN) which coordinates access. Data is generally replicated on three different servers. In the current HDFS write placement algorithm (see Figure 2.18), when a client writes to HDFS, the client first contacts the NN and is told where to place (blocks of) the file. If the client happens to be a DN, then the client's machine will be the first replica (otherwise a random DN is chosen). Second and third replicas are picked in a different, randomly chosen rack. Different clients in the same rack may be instructed to write to different racks (as shown); the same is true for different files on the same client or blocks within the same file. This results in all-to-all workloads that require many circuit configurations to properly schedule, incurring more reconfiguration penalties.

Modified workload (reHDFS): we create a modified HDFS write placement algorithm for the NN, reHDFS (reconfigurable DC network HDFS), that introduces skew and sparsity by selecting replica racks as a function of the source rack, rather than randomly (Figure 2.19). For source rack

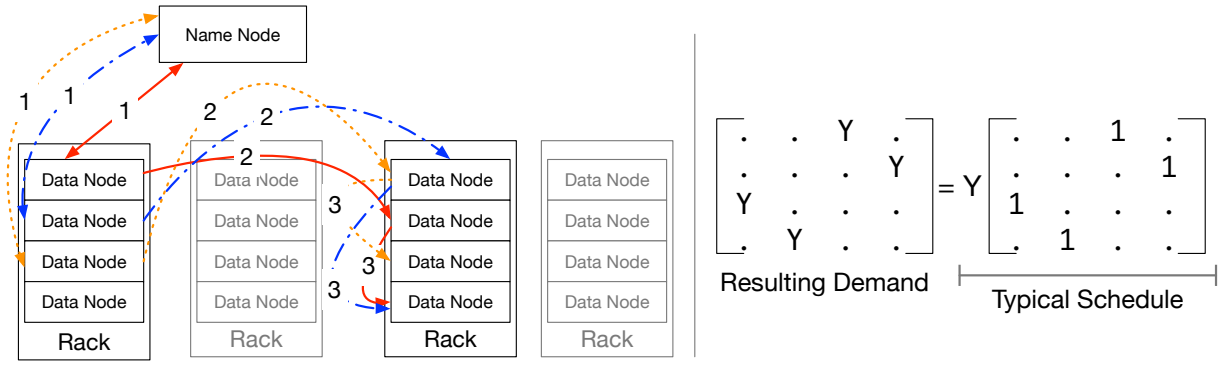


Figure 2.19: Our modified HDFS write placement algorithm, reHDFS. Replica rack selection is a function of the source rack (e.g., rack i writes to rack $i+2$), rather than random. This results in a single ring of demand that is easy to schedule.

i, the NN selects $(i+1)$ `mod` N as the replica rack. This ensures that each circuit group (rack i) only sends to one other circuit group (rack $i+1$). By choosing replicas in this way, the workload shifts from all-to-all to a single ring, which can be scheduled in a single circuit configuration. Concerns that this may cause availability issues during simultaneous rack failures can be addressed by changing this mapping function (e.g., to $i+2$ etc.) on short timescales (e.g., seconds).

2.6.2 Experiments

Methodology: We test reHDFS's modified write placement algorithm using an industry standard benchmark, DFSIO, in Intel HiBench [68]. DFSIO runs a Hadoop [10] (MapReduce [37]) job in which each of J mappers writes a file of X bytes to HDFS in parallel and then send statistics to a single reducer to output. We use $J = 64$ mappers to write 64 $X = 400MB$ files (25GBs of inter-rack traffic). This kind of workload might be seen during logging in big data applications or in scientific computing.

As we focus on improving network performance in our algorithm, we configure HDFS to use a 10GB RAM disk per host as its backing store to avoid being bottlenecked by slow mechanical disks. Running multiple virtualized Hadoop stacks (Hadoop, YARN, HDFS, JVM) per physical machine is difficult given the limited RAM (16GB) in our testbed machines. Instead, we run 8 mappers per physical machine (one per CPU core), sharing the underlying Hadoop stack to avoid this overhead. While this means tasks no longer have individual server-to-ToR link rate limiting, the lack of memory pressure makes this worthwhile. Machines with more RAM could avoid this limitation.

Results: Figure 2.20 shows the CDF of HDFS write time, for both HDFS and reHDFS. reHDFS reduces the median write time by ~4.7×, the 99th by ~7.6×, and the max by ~9×. Additionally, we reduce the job completion time for the DFSIO MapReduce task by 47%. **Takeaway: reHDFS improves write latency up to 9×.**

Cumulative results: Next, we look at how various combinations of current and previous optimizations (buffer resizing and reTCP + resizing, with $\tau = 1000\mu s$) perform under DFSIO. Combinations

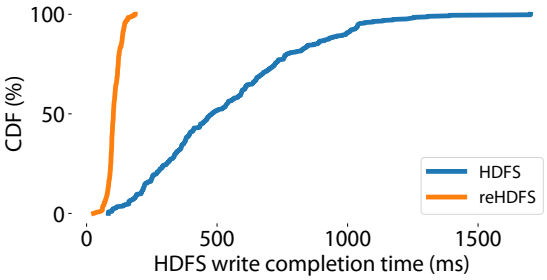


Figure 2.20: CDF of HDFS write completion time for HDFS and reHDFS for the DFSIO benchmark.

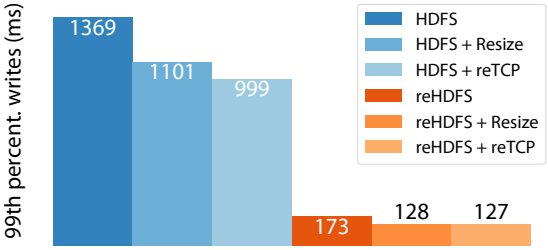


Figure 2.21: 99th percentile HDFS write times for the DFSIO benchmark.

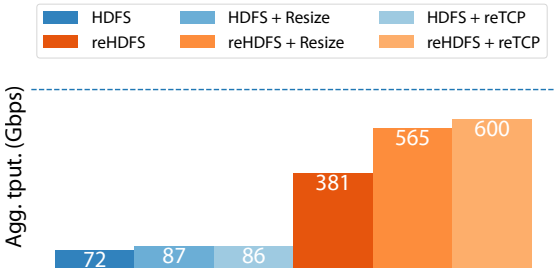


Figure 2.22: Aggregate HDFS write throughput. The dashed line is cluster capacity.

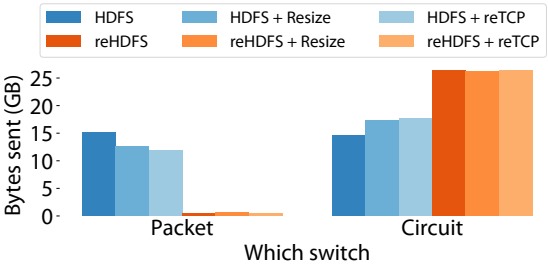


Figure 2.23: Packet and circuit utilization for the DFSIO benchmark.

involving ADUs are less interesting in this workload due to the lack of many small flows. We verify this for some combinations and leave them out for space.

Figure 2.21 shows the 99th percentile HDFS write times for various combinations of previous optimizations. Without application-layer changes, dynamic buffer resizing by itself (§2.4) provides a modest 20% improvement (similar to our previous results with a more controlled workload). Modifying HDFS' workload to increase skew and sparsity, however, yields a drastic improvement: reHDFS improves 99th percentile latency by 87% compared to HDFS and 84% compared to HDFS with dynamic buffer resizing. Adding buffer resizing to reHDFS gives only an additional 3% improvement over vanilla HDFS because the modified workload is close to ideal; applications that cannot reduce their circuit group fan-out as much as reHDFS will benefit more from buffer resizing and reTCP.

Figure 2.22 shows the aggregate HDFS write throughput for various combinations of previous optimizations, with a dashed line indicating maximum cluster bandwidth. Again, reHDFS provides the largest benefit (~5×), with others similar to the previous graphs (~1.6×). Interestingly, reHDFS + reTCP gets close to the maximum cluster capacity.

Figure 2.23 shows packet switch and circuit switch utilization for various combinations of previous optimizations, measured in the software switch. While the results are again similar, it is interesting to note how little data is sent over the packet switch for reHDFS-based optimizations, implying that reHDFS provides an astoundingly easier workload to schedule over RDCNs than vanilla HDFS. Finally, summing the bytes sent over both switches, we find that for HDFS-based methods send roughly 30GB of data, while reHDFS-based methods send about 27GB. Setting aside

the 25GB written to HDFS in both cases, we are left with a 60% reduction in the rest of the data transfered (e.g,. MapReduce results shuffle traffic, HDFS name node traffic, etc.).

Though we have focused on only one application in this section, our reHDFS results suggest a few key points that likely generalize:

1. As expected, application workloads have a big impact on RDCN performance.
2. There are real world applications that can be easily modified for RDCNs.
3. Though application-agnostic techniques like dynamic buffer resizing and reTCP are useful, modifying workloads to be more skewed and sparse can offer significantly more improvement, meaning application-specific changes are worth the extra effort.

2.7 Related work

While there has been much work on RDCN design [28, 42, 52, 61, 62, 84, 97, 130, 150, 164] and scheduling [20, 95, 98], however, no prior work has specifically focused on the end-to-end challenges presented in this chapter. Some do briefly touch on them:

Rapid bandwidth fluctuation: prior work briefly explores TCP ramp-up issues, but either does so on millisecond-scale reconfigurable switches with thousand RTT-long days (removing the issue) [150], or avoid sending packets over the packet switch unless absolutely necessary (using a modified endhost kernel pause protocol) [97]. We only require in-network changes to ramp up TCP, and allow flows to use the packet switch when a circuit is down.

Poor demand estimation: most RDCN designs briefly touch on demand estimation [20, 95, 97, 98, 150], but little work has looked at how demand error affects scheduling. Albedo [95] and Eclipse [20] propose *indirect routing*, but note its lack-luster performance. We argue that proper estimation is the solution and must involve the endhost stack.

Difficult-to-schedule workloads: While some prior work have hinted at opportunities for cross-layer application/network co-scheduling [84, 150], work with concrete examples have focused on making the network "application-aware" [95, 151, 154] (e.g., using MapReduce's job tracker for demand estimation), rather than making applications "network-aware" (i.e., modifying applications to suit the network), as we propose.

2.8 Summary

Recent work has shown the increasing need for augmenting traditional packet networks with reconfigurable circuit technologies in DCs [52, 84, 97, 98, 109, 130]. We show the existence of three key end-to-end challenges in such networks (RDCNs): 1) rapid bandwidth fluctuation, 2) poor demand estimation, and 3) difficult-to-schedule workloads. We overcome bandwidth fluctuation with dynamic in-network resizing of ToR queues, poor demand estimation by communicating endhost ADUs, and difficult-to-schedule workloads by rewriting application logic (e.g., HDFS' write placement algorithm). We design and implement an open-source RDCN emulator Etalon for use on public testbeds, as a means of understanding and evaluating the nature of these challenges,

and the efficacy of our solutions. We find that the cross-layer optimizations we propose are necessary, and that they provide the most benefit at higher layers. Etalon provides additional opportunities to explore in future work, e.g., utilizing ADU boundaries in scheduling, exploring multicast-enabled optical circuit switching (e.g., Blast [154]), or, more importantly, providing a cross-cutting evaluation across different RDCN designs, or an investigation of the challenges faced in future sub-μs RDCNs. We believe our experience speaks to the need for end-to-end system evaluations in future DC designs.

Within the context of this thesis, this chapter shows that systems with separate control planes due to layering do, in fact, benefit from using transparency to coordinate them. The fact that layering implies a lack of constraints on information sharing (as all layers are run by one entity), does allow for rich interfaces or specialization (here cross-layer optimization), greatly reducing adverse interactions between control planes. Next, we examine how having control planes run by different entities changes the nature of coordination between them.

Chapter 3 Control Planes in Different Businesses: A Case Study in Content Brokering

In this chapter, we explore one example, in detail, where priority ranking can be used to overcome adverse interactions between control planes. Specifically, we look at how decisions made by content brokers (i.e., which CDN to use) can, by accident, greatly impact profits for CDNs, and how CDNs can accidentally limit the performance opportunities made available to content brokers (i.e., by limiting which clusters can be used for a given client). Both of these issues are due to errant assumptions made by both about the other.

We argue that these incorrect assumptions lead to decisions that are sub-optimal (i.e., in terms of the quality/cost tradeoff brokers try to meet on behalf of content providers, and in terms of profits for CDNs), and that building a *m*arketplace-style interface between them alleviates these problems through joint decision making. As CDNs and brokers are separate businesses, they can not share all their information. Sharing a small amount of specific information is enough to remove most erroneous assumptions, as we will show. Since brokers and CDNs have independent data plane resources (i.e., clients and servers, respectively), our categorization in §1.2.3 leads us to coordinate using priority ranking (here a marketplace-style interface). Figure 3.1 summarizes this.

Functionally, a marketplace effectively provides a way for CDNs to priority rank their clusters with respect to different clients, allowing brokers to make decisions based on these priorities, without needing to understand how/why those priorities were computed. This allows CDNs to hide their most important data (e.g., internal costs, raw capacity values, etc.) from other CDNs and brokers.

Content brokering serves as an illustrative (and timely) context; various trends are reshaping Internet video delivery: exponential growth in video traffic, rising expectations of high video quality of experience (QoE), and the proliferation of varied content delivery network (CDN) deployments (e.g., cloud computing-based, content provider-owned datacenters, and ISP-owned CDNs). More fundamentally though, content providers are shifting delivery from a single CDN to multiple CDNs, through the use of a content broker. Brokers have been shown to invalidate many traditional delivery assumptions (e.g., shifting traffic invalidates short- and long-term traffic prediction), by not communicating their decisions with CDNs.

In this chapter, we analyze these problems using data from a CDN and a broker. We examine the design space of potential solutions, finding that a marketplace design (inspired by advertising exchanges) potentially provides interesting tradeoffs. A marketplace allows all CDNs to profit on

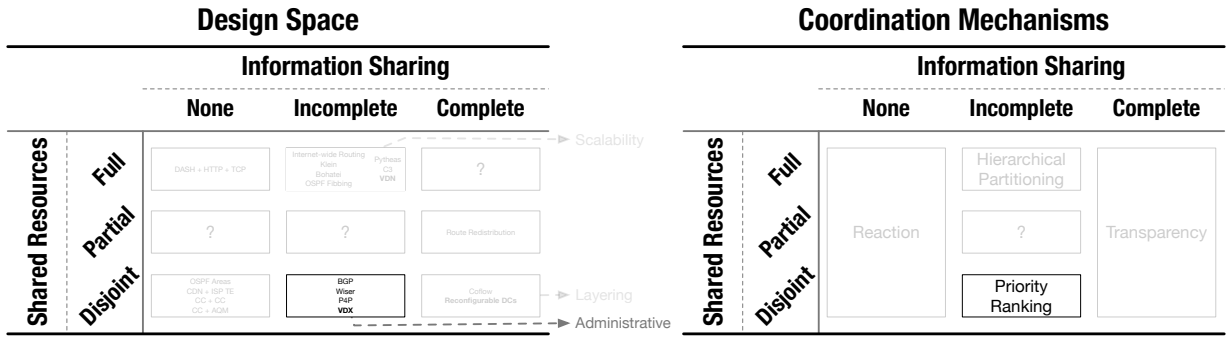


Figure 3.1: This chapter focuses on challenges in content brokering due to administrative separation, as shown with VDX. Looking back at our design space (§1.2.3), administratively separate systems (e.g., VDX, BGP, Wiser, P4P) have constraints on information sharing due to business concerns, but make decisions for their own data plane resources. These systems should coordinate using priority ranking (§1.1).

video delivery through fine-grained pricing and optimization, where CDNs learn risk-adverse bidding strategies to aid in traffic prediction. We implement a marketplace-based system (which we dub Video Delivery eXchange or `VDX`) in CDN and broker data-driven simulation, finding significant improvements in cost and data-path distance.

3.1 Introduction

Content delivery is constantly changing to meet the evolving challenges created by new workloads (e.g., streaming video), new actors (e.g., CDNs), new protocols (e.g., HTTP chunk-based video), new algorithms (e.g., video rate adaptation), and new demand (e.g., exponential growth in video traffic). Techniques introduced to accommodate these challenges have far-reaching repercussions on flows across all layers of the network stack. For example, the introduction of content delivery networks (CDNs) dramatically changed the traffic patterns that ISPs handled, clients' performance expectations, and the sheer volume of content that the Internet could deliver.

Content delivery is in the midst of another such major change. Until recently, major content providers (CPs) either contracted with a single CDN, such as Akamai [123], Level 3 [94], or CloudFront [8], or deployed their own CDN, such as Google [55] and Netflix [118]. The recent rise of CDN management services ("brokers"), such as Cedexis [26], Conviva [36], or NicePeopleAtWork [122], and CDN federation techniques [33, 121] has made it easier for content providers to enlist multiple CDNs to deliver content. Simultaneously, the rise of ISP CDNs (e.g., Comcast [12]) and proposals like running virtualized CDN nodes inside of ISPs [47, 49], are moving previous ISP-CDN tussle concerns [49, 82, 83, 128, 129] into the new context of a CDN-broker tussle.

At first glance, it may seem that the addition of brokers to content delivery is a minor change; however, brokering is a surprisingly complicated process. Our previous work [114] uses CDN and broker data to show there are significant issues in today's content brokering ecosystem, due to the lack of CDN-broker coordination in optimizing delivery objectives (e.g., cost, performance, etc.), that require fundamental changes. These issues, however, have not been widely identified as they

are hard to diagnose without both broker and CDN data. They have also yet to become widespread, as broker traffic is still a small (but growing) portion of overall CDN traffic.

In this chapter, we first characterize the kinds of problems brokers and CDNs face due to independent decision making by examining data from both a popular broker and a major CDN. Among the problems we uncover is that brokers may make traffic unpredictable for CDNs, making it difficult for CDNs to profit, due to combination of long-term CDN-CP contracts (months or years [6]) and flat-rate pricing. Additionally, despite having multiple clusters with similar performance, CDNs have no incentives to share this information with brokers today, limiting a broker's ability to optimize for certain CP goals, and to handle failures.

These problems lead us to three requirements needed for proper CDN-broker decision making: 1) CDNs need to replace today's flat-rate price model and reflect dynamic per-cluster prices to improve profitability; 2) CDNs need incentives for providing a fine-grained cluster-level view to brokers, allowing them to better optimize for CP goals; and 3) CDNs and brokers need to make decisions jointly, removing today's traffic unpredictability in the content delivery ecosystem.

Solutions that only address one of these requirements do not provide the right adoption incentives for CDNs, brokers, and CPs; CDNs only benefit from dynamic cluster-level pricing and traffic stability, but Brokers/CPs only benefit from cluster-level optimization. Addressing all the requirements simultaneously provides incentives to all parties. While similar to the well studied ISP-CDN collaboration problem [49, 82, 83, 128, 129], we argue CDN-broker collaboration is easier to achieve, as there are significantly fewer CDNs than ISPs, and business relationships are already more attuned to collaboration (CDNs and brokers both directly optimize content delivery under contract with CPs).

We address the above requirements directly by examining promising points in the design space. Simple tweaks to today's practices (e.g., providing brokers multiple clusters to choose from) do not meet all the requirements (and thus lack deployment incentives). In addition, multiparty transaction designs requiring all CDNs and brokers to agree are impractical. We find that a marketplace-like design represents a reasonable tradeoff. It meets the first two requirements while allowing CDNs to learn "bidding" strategies that likely provide them traffic predictability. A marketplace represents one possible solution, however, the focus of this work is that all parties (including clients) benefit from a content delivery service sold at much finer granularity than today. Any mechanism that supports these requirements may be sufficient.

We present a prototype marketplace design called Video Delivery eXchange (`VDX`). We leverage real-world traces obtained from a major CDN and a popular broker, as well as publicly available data from other CDNs, to build a CDN-scale simulation. We run our simulator across a variety of scenarios (e.g., differences in CDN deployment models, differences in country pricing) to better understand how relatively complex schemes like `VDX` can fine-tune the trade-off between performance and cost.

To summarize, we make the following contributions:

1. Identify the challenges created by the lack of joint decision making between brokers and CDNs by analyzing broker and CDN data.
2. Examine the design space, from simple tweaks (e.g., dynamic pricing or providing multiple potential clusters to brokers) to more complex designs (e.g., marketplace or multiparty transactions), evaluating their tradeoffs.

3. Evaluate a marketplace design (VDX), where all CDNs can profit on video delivery, in-depth, through CDN-scale simulations using data from both a broker and a CDN, finding significant improvements in cost and data-path distance.

3.2 Content Delivery: The Past and the Present

The arrival of CDNs has had a dramatic impact on the Internet. In this section, we explain how content delivery is again being reshaped for content providers by contrasting broker-based delivery with traditional CDN delivery.

3.2.1 Traditional Content Delivery

Content providers (CPs), such as ESPN, Netflix, and HBO, create or license content that users are interested in. In order to provide good "quality of experience" (QoE) (e.g., a combination of metrics such as average bitrate, buffering ratio, and join time [15]) to viewers around the globe, CPs would need to build massive amounts of infrastructure. Thus, most CPs rely on CDNs to provide reasonable QoE. CPs generate revenue through premium services and/or advertising, and try to minimize their delivery costs. CPs often pay CDNs based on bandwidth usage based on a 95/5 model [105].

CDNs deliver content to clients through clusters nearby (e.g., in datacenters, peering points, universities, large businesses, or ISP networks) to minimize latency and improve throughput. CDNs have a wide variety of deployment models: some deploy servers in a large number of geographic regions (e.g., Akamai [123]); others deploy in a small set of strategic regions (e.g., Level 3 and CloudFront) [91]; other "ISP CDNs" operate extremely locally, serving a single ISP's customers in a region (e.g., Comcast [12]). CDNs typically choose which cluster to serve a client request from based on network measurements or static assignments. Akamai, for example, uses latency and loss measurements from clusters to gateway routers in the network (not individual clients) [27, 96] to decide on an initial cluster assignment.

CDNs wish to provide reasonable performance to clients while minimizing their bandwidth and co-location (energy) costs. In a recent annual report [3], Akamai lists bandwidth costs as their largest cost ($150M/year) behind payroll, with slightly lower co-location costs ($126M/year). CDNs generally do not price their services to reflect costs at individual server locations (which may vary considerably; see §3.3.1), and, instead, use a flat-rate price across large geographic regions (e.g., continents) [9, 111] regardless of the actual delivery cost. Prices vary as CDNs typically negotiate individual contracts with CPs over long timescales (e.g., months, years [6]). This lack of fine-grained cost-aware pricing can lead to significant problems, as we show in §3.3.2.

Traditionally (see Figure 3.2), CPs contract with a single CDN and express very broad policy goals (e.g., what content can be served by the CDN). The CDN typically caches the content on front-end servers close to the clients (although more complicated caching structures also exist). Clients request an HTML page or video manifest from the CP's website that indicates which CDN to contact for the content. The CDN chooses which server to use for a given client, and provides a mechanism (e.g., DNS) for reaching the server. The client connects to this server and retrieves the content.

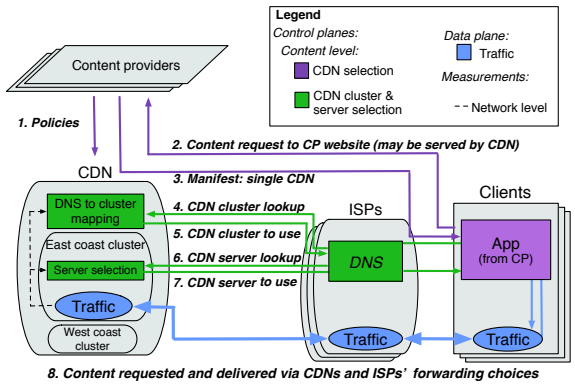


Figure 3.2: Traditional content delivery.

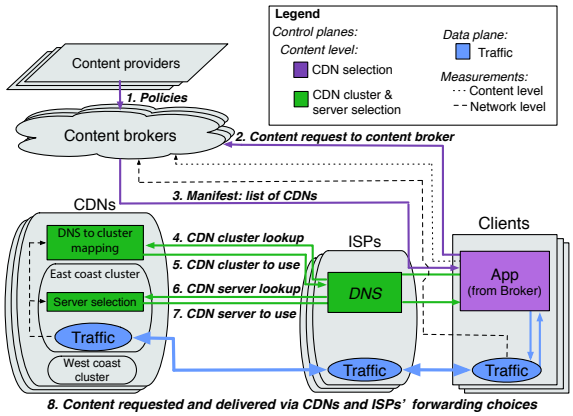


Figure 3.3: Brokered content delivery.

3.2.2 Brokers and Delivery Today

With increasing pressure from users in terms of QoE expectations, as well as the sheer volume of traffic, CPs have moved from a single-CDN delivery model to employing multiple CDNs [19, 134]. Due to variations in price and performance, spatially and temporally [99], CDN selection must be dynamic. Figure 3.3 shows brokered content delivery. From a client's perspective, brokers are a level of indirection for CDN selection; clients first ask a CP's broker which CDN to use, before querying a CDN's DNS server.

Brokers (e.g., Cedexis [26], Conviva [36], NPAW [122]) measure QoE within client applications (e.g., video players) and build predictive models to determine the best assignments of clients to CDNs (using CPs' QoE and cost goals), based on various factors (e.g., client's location, ISP, etc.) [51, 79]. Brokers not only select the initial CDN a client is assigned to, but also move clients between CDNs in real-time (e.g., mid-stream). Although a CP could function as a broker for its own content, independent brokers can leverage data across CPs' clients and CDNs to form a more complete view.

There is no explicit coordination between brokers and CDNs, a key point of tension addressed in this chapter (see §3.3). CDNs, however, implicitly see the effects of brokers' decisions when clients are suddenly moved to/away from their clusters. This increased traffic unpredictability, along with long-term CDN-CP contracts still based on flat-rate pricing, potentially makes the disparity between prices and internal CDN costs even worse. (see §3.3.2). Brokering also facilitates a wider variety of small-scale deployment models (e.g., regional CDNs, city-centric CDNs, etc.), although we have yet to see these types of CDNs in practice.

3.3 Potential Problems and Opportunities

Although brokers may appear to add a simple layer of indirection, they greatly complicate content delivery. Brokers and CDNs run independent control loops to maximize their own objectives, without explicitly communicating their decisions. These decisions directly impact one another, potentially leading to sub-optimal decisions for both parties.

Brokers have a global view of all client performance (app-level QoE) and costs for a CP, but can only make decisions by selecting which CDN to send a client to. Individual CDNs, conversely, have a large set of clusters to choose from but typically make their choice only on network level measurements, rather than QoE (each CDN would need to instrument CPs' software to get QoE). This mismatch of data richness (brokers) and selection richness (CDNs) leads to many potential problems.

We examine these problems separately, using data from a major CDN and a popular broker. The broker data allows us to understand when it uses different CDNs (e.g., over geographic regions, time, etc.). The CDN data allows us to understand its use of different server clusters. We distill these problems down to a short list of key requirements any proposed CDN-broker decision interface must meet to aid in CDN pricing, meet flexible performance goals, and provide traffic stability.

3.3.1 Traces

Broker: We collect trace data from a video delivery broker. The trace includes an entry for each client session containing the request arrival time, which video was requested, the average bitrate, session duration, the client city and AS, the initial CDN contacted, and the current CDN delivering the video. The data covers roughly an hour of off-peak requests (33.4K total) for one content provider (a music video streaming website). Even this small window illustrates many problems.

The data exhibits similar trends to those seen in other works [13]: video popularity follows a Zipf distribution, and the distribution of client cities follows a power-law. Most clients abandon almost immediately (around 78%). The distribution of bitrates is bimodal with peaks at the lowest and highest bitrate. The trace identifies three large CDNs (here "A," "B," and "C") directly and lists the rest as "other." CDN A is a CDN with clusters in many locations. CDN B and C deploy large amounts of capacity in a small number of locations. We investigate the effects of different deployment models in our evaluation (§3.7).

CDN: We collect Internet mapping data from a major CDN to compare performance estimates across its clusters. The data provides a score estimating the performance between blocks of client IP addresses and candidate CDN clusters. This score is a simple function of latency and packet loss. Measurement happens periodically and frequently (several times per minute) through pings from clusters to routers with large networks of clients behind them.

From the same CDN, we collect data on the average cost per byte delivered for the 20 countries with the highest volume of traffic, using client geolocation to bin requests into countries. We then compare them to the average delivery cost. We anonymize this data and present it in Figure 3.4.

3.3.2 Potential Problems for a CDN

Brokers create problems for CDNs for three main reasons: 1) brokers make load balancing difficult due to short-term traffic unpredictability, 2) brokers make provisioning difficult due to long-term traffic unpredictability, and 3) (broker-created) traffic unpredictability negatively impacts profits due to flat-rate CDN-CP contracts.

Short-term provisioning problems due to traffic unpredictability: Figure 3.5 shows a time-series graph of the percentage of client sessions in the broker trace, within 5 second intervals, that

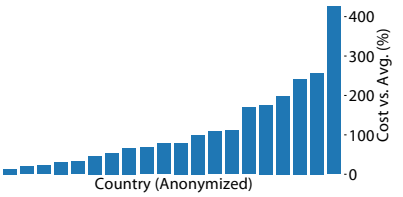


Figure 3.4: Average cost per byte serving clients geolocated in various countries relative to the average.

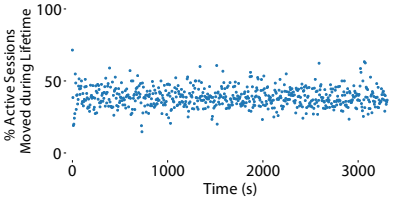


Figure 3.5: Sessions moved between CDNs by the broker in our trace in 5s intervals.

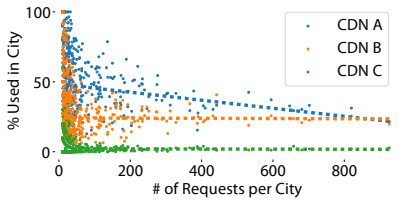


Figure 3.6: Broker's usage of CDNs, sorted by requests per city in the US. Dotted lines are best-fit linear regressions.

have been shifted from one CDN to another mid-stream. This value is surprisingly high throughout (averaging ~40%). We note, however, that at some points this dips to ~20% and at other times rises above ~60%. This indicates brokers not only often move traffic around, but the rate at which they do is highly variable. This potentially makes short-term provisioning (load balancing) difficult for CDNs.

Long-term provisioning problems due to traffic unpredictability: Figure 3.6 shows the utilization for the CDNs in the broker trace, plotted as a function of number of requests per city. The dotted lines are best-fit lines. We see from the best-fit lines that regardless of city size, CDN B and CDN C's usage does not change, whereas CDN A is strongly favored in smaller cities. This is perhaps unsurprising due to CDN A's broader geographic deployment. CDN A is also generally more expensive than CDN B and CDN C, suggesting that a broker will try to avoid CDN A where other options are available. Succinctly, brokers do not merely split traffic evenly among CDNs; traffic may be arbitrarily divided in geographic regions due to various factors and change over time.

This leads us to believe that brokers, as well as other CDNs, can cause a CDN difficulty in cluster planning and long-term provisioning (e.g., cluster location and capacity). For example, if a broker decides to stop using CDN A in big cities (e.g., CDN B deploys more servers) this will impact CDN A's future provisioning. If CDN B then raises its prices, the broker may move more traffic back to CDN A, again impacting future provisioning. In effect, in a brokered world, proper CDN provisioning becomes more difficult to achieve.

REQUIREMENT: Traffic Predictability
In order to provide more stability for CDNs, broker-controlled traffic must be more predictable. Thus, a proper CDN-broker decision interface must make decisions jointly (i.e., share information and decisions in both directions).

Pricing / cost disparities: Figure 3.7 illustrates a toy example of CDN pricing issues. Recall that CPs generally pay CDNs a contracted flat rate per traffic delivered (based on a 95/5 model [23, 105]) with price changes (e.g., 2 – 7×) depending on very coarse geographic regions (e.g., continents) [9, 111]. CDN Y can provide good performance at a low flat-rate contract price for the CP, for all clients except for the left-most one, who must be served by CDN X. Unfortunately for CDN X, this client is served by a very expensive cluster. CDN X's flat-rate contract price with the CP is more expensive than CDN Y, so a broker (unknowingly) avoids CDN X's cheaper clusters. This unfortunately means that CDN X actually loses money as the CP will pay CDN X at a price less than its cost.

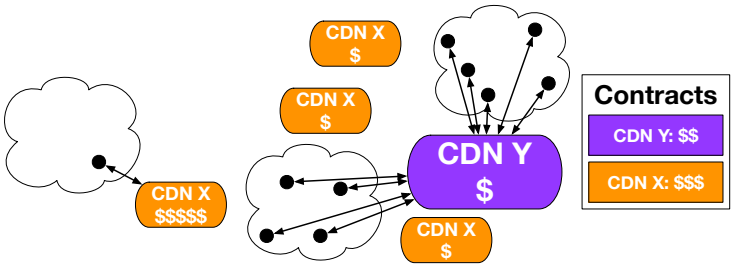


Figure 3.7: Brokers can greatly skew CDN traffic patterns making it difficult for CDNs to profit. Ovals represent CDN clusters, with "$" indicating the cost to the CDN. Rectangles represent CDN-CP contracts, with "$" indicating the price paid by the CP. Solid dots represent clients.

We see potential pricing problems like this occur at the country level. Figure 3.8 shows how the utilization of the CDNs in our broker trace differ in different countries. The remaining percentage of clients are serviced by other smaller CDNs. We show all countries that originated 100 or more requests in our trace, in random order. Note that utilization varies significantly: e.g., CDN B barely serves 7, yet almost entirely serves 8; CDN A is rarely used in 8, 11, and 15, etc.

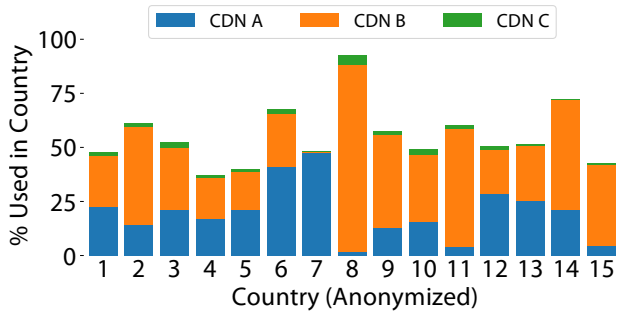


Figure 3.8: Broker's usage of CDNs for a sampling of countries based on request count.

Different countries around the globe can have markedly different bandwidth costs. Our CDN cost data shows up to a ~30× disparity in pricing between countries (Figure 3.4). CDN CloudFlare paints a similar picture, stating that when compared to Europe, North America, Asia, Latin America, and Australia, cost 1.5×, 7×, 17×, and 21× more respectively [34]. They further state that within a region, some transit ISPs may have an order of magnitude higher cost.

If, for example, Country 7 in Figure 3.8 is very expensive for a CDN, yet Country 8 is very cheap (with flat-rate pricing across both), CDN A will have trouble making a profit, whereas CDN B will easily make a profit. Therefore, unpredictable traffic from brokers may unintentionally cause disparities between pricing and cost, affecting CDN profits. Unfortunately, raising contract prices to recoup these profits will likely cause brokers to move even more traffic away from the CDN.

REQUIREMENT: Dynamic Cluster Pricing
In order to alleviate CDN pricing / cost disparities, a proper CDN-broker decision interface must allow CDNs to charge CPs (or brokers) prices reflecting their internal costs. In order to allow CPs/brokers to optimize over these prices, a price sharing mechanism that is fine-grained both spatially (per-cluster, not per-continent) and temporally (per-minute, not per-year) is needed.

3.3.3 Potential Problems for a Broker

CDNs create problems for brokers for two main reasons: 1) CDNs do not expose cluster-level information to brokers, limiting a brokers' ability to optimize performance and cost, and 2) CDNs make decisions that impact end-to-end delivery without information about all of the clients seen by brokers, directly affecting performance.

Limited optimization ability from lack of cluster-level view: CDNs and brokers do not explicitly consult with each other for decision-making; they lack an interface to facilitate this exchange. Brokers cannot gather much information about CDNs at a cluster level, and thus, treat each CDN as a black box function of {client location, client ISP, ...} → {performance, cost}. As CDNs typically map individual clients to one specific CDN cluster at a given time, when performance is inadequate based on the CP's objectives, a broker's only recourse is to switch CDNs (even if other better choices exist within the current CDN). Effectively, the granularity of change a broker can make is very coarse.

1 Alternative Choice	2 Alts.	3 Alts.	4 Alts.
77.8%	64.5%	53.7%	43.8%

Table 3.1: How often alternative CDN clusters with similar performance scores exist.

Table 3.1 shows how often there are alternative clusters with similar estimated performance (based on latency and loss measurements) in the CDN data. We find that on average there are four server clusters (i.e., 3 alternative choices) that have similar scores (within 25% of the best), yet typically only one choice is returned. This data indicates potential opportunities; as these clusters have similar performance estimates, brokers may be able to avoid switching CDNs due to inaccurate estimates, congestion, or failures (unlike today), to better meet CPs' performance goals. As explained above, today's flat-rate pricing discourages CDNs from making use of these alternative clusters if their costs are higher than the primary cluster.

Poor performance due to incomplete data: Both brokers and CDNs spend significant effort building maps of the Internet to predict performance between clients and servers. This is by no means a small task; in recent work [51] a broker claimed that they regularly handle 100M client sessions per day, 3M clients concurrently during peak hours, and 10s–100s of thousands of clients entering and exiting per minute. They also imply that this leads to 50–100 GB of new sample data to process per minute. Sharing mapping information could greatly improve the accuracy of the data as both CDNs and brokers have limited vantage points into the network. Namely, CDNs such

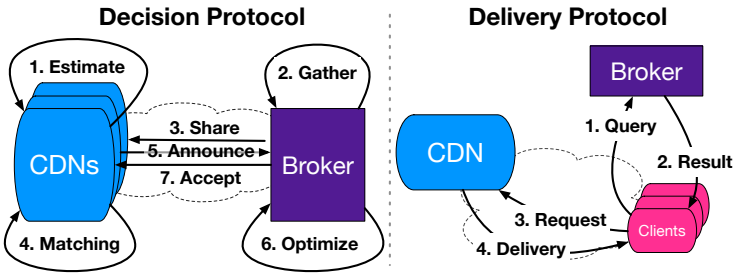


Figure 3.9: All CDN-broker joint-decision interfaces follow this basic structure, differing in how they implement steps.

as Akamai typically measure (in advance of connections) from clusters to gateway routers [96], whereas brokers generally only measure (during a connection) from clients to chosen CDN servers.

REQUIREMENT: Cluster-level Optimization
Brokers currently make decisions with an unnecessarily coarse view of CDNs, responding to issues by switching CDNs entirely, despite other reasonable clusters being available within the CDN. Clusters may also have different performance/cost tradeoffs leading to better CP goal optimization. Thus, a proper CDN-broker decision interface should provide brokers with cluster-level views of CDNs.

3.4 Exploring the Design Space

As seen, the problems from §3.3 motivate an explicit broker-CDN decision interface that provides: 1) cluster-level optimization, 2) dynamic cluster pricing, and 3) traffic predictability. We find many possible designs that meet some/all of these requirements have the same basic structure. While many of these designs are reasonable, we argue that only designs that meet all three requirements have adoption incentives for CDNs, brokers, and CPs.

3.4.1 Generalizing Designs

Building off today's CDN-broker interactions (as seen in §3.2), all designs we consider utilize the following two protocol structure:

Decision Protocol: Periodically (e.g., every few minutes, as brokers do today [51]), each design runs the seven steps below to update the mapping of clients to CDN (clusters) that maximizes CP goals (see Figure 3.9):

1. **Estimate**: CDN clusters estimate capacity, cluster costs, and cluster-to-client performance.
2. **Gather**: Brokers count clients, including meta-data (e.g., location).
3. **Share**: Brokers potentially send client (meta-)data to CDNs. We will see that designs that do not send client data make it difficult for CDNs to provide proper matchings (as they do not know which clients belong to which broker (or to no broker at all).

$$\max w_p * \sum_{r \in \text{Clients}, m \in \text{Matchings}_r} \text{Performance}(m) * U_{r,m}$$
$$- \quad w_c * \sum_{r \in \text{Clients}, m \in \text{Matchings}_r} \text{Cost}(m) * \text{Bitrate}(r) * U_{r,m}$$

subject to:

$$\forall r \in \text{Clients}, m \in \text{Matchings}_r : U_{r,m} \in \{0, 1\}$$
$$\forall r \in \text{Clients} : \sum_{m \in \text{Matchings}_r} U_{r,m} = 1$$
$$\forall l \in \text{Clusters}: \sum_{r \in \text{Clients}, m \in \text{Matchings}_{r,l}} \text{Bitrate}(r) * U_{r,m} \leq \text{Capacity}(l)$$

Figure 3.10: Example broker optimization problem.

4. **Matching**: CDNs match clients to one or more potential clusters, based on performance estimates, cluster costs, and capacities, as well as any client (meta-)data from brokers. Current algorithms such as Akamai's customized stable marriage algorithm [102], or simpler algorithms, such as ranking clusters that can provide adequate performance by their costs, provide plausible starting points.
5. **Announce**: Brokers receive CDNs' matching data (either pushed or pulled), and potentially performance, cost, and capacity information. We will see that designs that do not announce all three provide worse overall performance and cost.
6. **Optimize**: Brokers map clients to CDN (clusters) to meet CP goals, using the CDNs' matchings and current cost, capacity, and performance (including traditional application-level QoE) data/estimate. We show an example optimization ILP (to be solved by the broker), in Figure 3.10, that maximizes performance while minimizing cost (with weights w_p and w_c). The output is stored for the Delivery Protocol. Other optimizations are equally valid (e.g., utilizing client meta-data, or simpler greedy algorithms).
7. **Accept**: Brokers tells all CDNs which matchings were used, so CDNs can modify future matchings.

High-level designs differ solely on how they implement *Share*, *Matching*, and *Announce*. As brokers operate on behalf of CPs and can control client requests directly, all considered designs have the broker make the final decision.

Delivery Protocol: Whenever a client initiates content retrieval, the following protocol runs:

1. **Query:** Client queries broker for CDN (cluster).
2. **Result:** Broker returns CDN (cluster) chosen earlier.
3. **Request:** Client requests content from CDN (cluster).
4. **Delivery:** CDN (cluster) delivers data to the client.

Note that the most recent Decision Protocol results are used, and thus decision making does not slow down delivery. All designs use the same Delivery Protocol, thus we end our discussion of it here.

	Share	*Matching*	*Announce*	CO	DCP	TP	Runtime
`Brokered`		Single-Cluster					1 Round
`Multicluster`		Multi-Cluster	Performance	✓			1 Round
`DynamicPricing`		Single-Cluster	Cost		✓		1 Round
`DynamicMulticluster`		Multi-Cluster	Cost, Performance	✓	✓		1 Round
`BestLookup`		Multi-Cluster	Cost, Performance, Capacities	✓	✓		1 Round
`Marketplace`	Clients	Multi-Cluster	Cost, Performance, Capacities	✓	✓	Weak	1 Round
`Transactions`	Clients	Multi-Cluster	Cost, Performance, Capacities	✓	✓	Strong	Multi-Round

Table 3.2: Alternate designs for a CDN-broker decision making interface, and whether they meet the *Cluster-level Optimization* (CO), *Dynamic Cluster Pricing* (DCP), and *Traffic Predictability* (TP) requirements in §3.3.

3.4.2 Design Space

We now present alternate designs for CDN-broker decision making interfaces that differ in *Share*, *Matching*, and *Announce*, and the requirements from §3.3 (see Table 3.2).

`Brokered` (today's world §3.2): . CDNs and brokers share little information, and CDNs match clients to single clusters. As we have seen in §3.3 this does not meet our *Cluster-level Optimization*, *Dynamic Cluster Pricing*, or *Traffic Predictability* requirements.

`Multicluster`: CDNs provide multiple similar cluster options per client. From this, brokers learn rough performance values. This provides *Cluster-level Optimization*, but does not address *Dynamic Cluster Pricing* concerns, or provide *Traffic Predictability*.

`DynamicPricing`: CDNs share dynamic cost information with brokers. This fixes *Dynamic Cluster Pricing* concerns, but does not address *Cluster-level Optimization* or provide *Traffic Predictability*.

`DynamicMulticluster`: Combines `Multicluster` and `DynamicPricing`. This addresses both *Cluster-level Optimization* and *Dynamic Cluster Pricing*, but does not provide *Traffic Predictability*. Its major flaw is instability; as decisions are not made jointly, the clusters with the best performance-to-cost ratio are overwhelmed as specific cluster capacity values are unknown. This is similar to the instability seen in price-based routing schemes [54, 64, 86].

`BestLookup`: This design attempts to fix `DynamicMulticluster` by providing cluster capacity information to brokers. CDNs must build multiple potential client-to-cluster matchings without knowing which clients are being considered by the broker. If there are multiple brokers or significant non-broker traffic (as there is today), "overbooking" of traffic sources may still overwhelm capacity (e.g., a cluster with capacity 10 units may receive 9 units of traffic each from two brokers).

`Marketplace`: A marketplace-based design would view CDNs' matchings as *bids* for the brokers' resource (clients). When the Decision Protocol is run periodically, there is a single round of bidding for clients, in which all CDNs are first told about all clients (meta-)data. CDNs build more nuanced matchings, properly allocating capacity based on the received client data. Brokers optimize as before and return a list of accepted bids to CDNs. CDNs learn which bids are likely to be used over time (as they know which clients are associated with which broker and get explicit feedback on why bids fail), providing "weak" *Traffic Predictability*.

`Transactions`: After *Optimize*, the broker requests CDNs to commit the resources for the chosen client-to-cluster mapping. If any CDN disapproves the mapping, the mapping is withdrawn from all CDNs and a new mapping is computed. This provides stronger *Traffic Predictability* guarantees than `Marketplace` by making the process transaction-like. It is, however, unrealistic, as CDNs may never all approve the mapping. Thus, we do not consider it further.

3.5 Narrowing the Design Space

We present an evaluation of the different designs presented in §3.4. As it is not practical to deploy a multi-CDN-broker marketplace for evaluation, we focused on building a realistic simulator using CDN and broker data (§3.3.1), as well as other publicly available CDN data.

3.5.1 Simulation Overview

We simulate 14 world-wide CDNs and a broker focused on video delivery. We run one round of the Decision Protocol (§3.4.1) to determine our results, effectively building a "snapshot" of client-CDN cluster assignments. Time dynamics are less important as the Decision Protocol runs periodically (e.g., every few minutes) over all clients.

Clients: We use the client requests (with location and bitrate) from the broker data we received (§3.3.1). Client locations in the broker data are matched with client locations in the CDN data to allow us to use client-to-cluster performance (latency/loss) scores in the CDN data. Some client-clusters pairings do not have scores, so we extrapolate them by computing a linear regression of scores with respect to client-cluster distance.

We simulate an additional 3× this amount of clients as background traffic (e.g., other broker traffic or non-broker traffic) not optimized by this broker. While difficult to quantify (for both the CDN and broker), traffic today is predominantly non-brokered, but has been progressively changing.

Broker: We simulate a broker using the ILP in §3.4.1 as the optimization function, solved by Gurobi [60].

CDNs: Each CDN is defined by a list of cluster locations. We received world-wide cluster location information from one highly distributed CDN. We additionally inferred the locations of as many CDNs as we could find (13) on PeeringDB [127]. PeeringDB may underestimate cluster locations, but for the smaller CDNs we manually verify their locations based on information available on their websites.

CDN cluster locations and cost: Each cluster location has an associated bandwidth and co-location (energy) cost, expressed in dollars per bit. We generate bandwidth costs by choosing average costs for countries from the data in Figure 3.4, then assign bandwidth costs to specific clusters by drawing from a normal distribution centered on this mean, with standard deviation derived from CDN bandwidth cost data for the top 8 ISPs within the US. Co-location costs are based on the cost for the country, but decrease proportional to the logarithm of the number of CDNs in that location. This models the fact that more CDNs are located in places that are inexpensive to serve from.

	Cost	Score	Distance	Load	Congested
`Brokered`	136	132	297	9%	0%
`Multicluster (2)`	155	87	194	14%	27%
`Multicluster (100)`	171	85	141	20%	39%
`DynamicPricing`	126	148	318	11%	0%
`DynamicMulticluster`	115	122	219	40%	14%
`BestLookup`	94	108	166	14%	14%
`Marketplace`	93	112	178	23%	0%
`Omniscient`	86	111	172	48%	0%

Table 3.3: Comparing the different designs for various metrics in data-driven simulation. Lower values are better.

CDN contract price and capacity: Each CDN has a contract price that we use in flat-rate price designs. A CDN's contract price is the average price per bit for the CDN if it was individually offered all of the clients. Cluster capacity is assigned similarly; all clients are sent to each CDN individually and clusters are assigned 2× their received traffic as their capacity. We assume that in steady-state, clusters are provisioned with ample capacity. Clusters that did not see any clients take capacity from their closest neighbor with capacity. Designs that do not share cluster capacity information with brokers use the median cluster capacity (per-CDN) as an estimate.

CDN matching algorithm and bidding: For each client, a CDN selects a set of candidate clusters with scores at most 2× worse than the best score. If there is no other cluster with a score within 2× the best, the second best scoring cluster is considered a candidate cluster. Candidate clusters are sorted from lowest to highest cost, with the matchings prioritized in that order. The scores, costs, and capacities of CDNs are directly reflected in *Announce* (depending on the design) for simplicity. Real-world CDN matching algorithms could change over time to find risk-averse strategies. We avoid this for simplicity.

Designs: We compare the designs presented in §3.4.2. The *Matching* algorithm in `Multicluster` (2), (100), and `Marketplace`, produces 2, 100, and 100 alternative clusters respectively. `Omniscient` exposes all CDN data to the broker.

Metrics: We compare designs using *Cost*, *Score*, *Distance*, *Load*, and *Congested* as metrics. *Cost*, *Score*, and *Distance* are the median cost, score, and geographic cluster-to-client distance over all clients (lower is better). Load is the median cluster load over all CDN clusters that saw any traffic. Congested is the percentage of clients sent to clusters that have greater than 100% load.

3.5.2 Results

In Table 3.3, we summarize the results. `Brokered` serves as our baseline; Both `Multicluster` designs provide better performance at the expense of increased cost (as the first cluster bid is always the cheapest one according to the CDN Matching algorithm above); additional clusters may provide better performance but will not be cheaper than the first cluster. Both designs also overload clusters (as these designs only estimate cluster capacities) while optimizing for performance.

DynamicPricing marginally saves cost by dropping performance (without overloading clusters), by only exposing one cluster, but allowing the broker to optimize with knowledge of the CDN costs. This shows that just avoiding expensive clusters is not good enough; a balance of expensive and cheap clusters are needed provide good performance. DynamicMulticluster and BestLookup do better than Brokered in both performance and cost (with BestLookup spreading load better), but still overloads some clusters (given their inaccurate capacity info). Marketplace does very similar to BestLookup but avoids overloading any clusters as it has accurate capacity info. Omniscient provides similar results, with the lowest cost overall. We see the same trends in the CDFs of cost, score, and distance (not presented).

From these results, we see that BestLookup and Marketplace are promising points in the design space. However, Marketplace better meets the requirements from §3.3 as it is less likely to overload clusters. By addressing all the requirements, we argue that Marketplace is more aligned with adoption incentives than BestLookup. To better understand the tradeoffs inherit in a marketplace design, we additionally evaluate a concrete implementation of a marketplace system, which we dub Video Delivery eXchange (VDX).

3.6 VDX in Detail

We now discuss the design of our system VDX, which creates a *marketplace* where CDNs express performance and cost concerns at the cluster level, by programmatically sending bids to a broker (similar in spirit to an advertising exchange), to serve clients in specific locations. Brokers may use multiple bids both across and within CDNs to maximize the CPs' QoE and cost goals, while allowing CDNs to be paid appropriately. While this design is complex, it represents a plausible point in the design space with reasonable tradeoffs, although other valid choices exist as well. The large variation in CDN internal cluster cost (Figure 3.4), however, may warrant complex designs.

Below, we fill out the key details that define the instance of our Marketplace design, describe a few simple examples, and discuss some system challenges such as failures and fraud.

3.6.1 Decision Protocol Details

Share: Brokers send client (meta-)data to CDNs. The specific format may vary depending on the requirements of the marketplace; we use the simple format:

$$[share\_id, location, isp, content\_id, data\_size, client\_count].$$

Each share contains an opaque *share_id* for use in *Matching* and *Announce*. This format can easily be extended to include other meta-data, e.g., client device type, depending on CP's optimization goals.

Announce: CDNs send the output of *Matching* ("bids") to brokers for optimization. Similar to *Share*, the format of each bid should be specialized to the needs of the marketplace. We use the simple format:

$$[cluster\_id, share\_id, performance\_estimate, capacity, price]$$

Each bid includes a *cluster_id* (an opaque id known only between the broker and the CDN), a *share_id* from *Share*, performance estimates from *Estimate*, cluster capacity, and a price related to

internal cost. CDNs may express policy by choosing not to bid on certain client announcements (e.g., certain videos cannot be served from certain CDN clusters, etc.), or by announcing modified performance, capacity, or price values. The broker trusts that these values are accurate. CDNs that continuously provide inaccurate values can be held accountable by lowering the priority of their bids, or by taking legal action if this goes against their contract.

Accept: Brokers communicate the results of *Optimize* to CDNs, including CDNs that "lost" the auction. This allows CDNs to understand which bids were accepted, allowing the CDN to prepare different bids (e.g., ones with lower prices, higher performance estimates, etc.) for the next round of bidding. Once again, the format should meet the needs of the exchange. We use the simple format:

$$[cluster\_id, share\_id, performance\_estimate, capacity, price]$$

The accept format is likely the same as the bid format.

3.6.2 Examples

VDX addresses many of the problems faced in §3.3. Three such fixes are: 1) VDX uses per-cluster pricing. This addresses the scenario in Figure 3.7. CDN X loses money as only its expensive cluster is used. With VDX's per-cluster pricing, CDN X can bid using its cheap clusters at a competitive price. 2) Traffic unpredictability (Figure 3.5) is greatly reduced in VDX as CDNs are explicitly involved before brokers move any traffic. 3) Applications with non-standard QoE metrics (e.g., latency agnostic applications) are easy to accommodate by having CDNs send bids that do not prioritize latency.

3.6.3 Challenges and Limitations

"Weak" traffic predictability: Although better than today's world, VDX's marketplace design only provides weak traffic predictability, as it runs a single phase of bidding. Traffic may move more quickly than some CDNs want, or a particularly bad set of bids may be accepted in tandem, leading to overloaded clusters. An optimal design would require all CDNs to agree on the broker's allocation (effectively multiparty consensus), which is impractical (§3.4). We argue instead that, in VDX, CDNs can learn risk-averse bidding strategies over time that will likely provide traffic predictability. Modeling these strategies with game theoretic frameworks (similar to those looking at CDN pricing [136], ISP transit pricing [138], or CDN-ISP collaboration [82, 83]) provides an interesting future research direction.

Failures and poor performance: With many different entities, Decision Protocol failures may seem difficult to combat. If a CDN has a failure, the rest of the system still continues to work. Failures or poor performance in the Delivery Protocol are handled using a variety of recovery mechanisms (e.g., moving clients mid-stream), as is done today. As brokers solely exist to optimize performance, when a broker fails, CP software can always fail gracefully to ignoring the broker and request content from a given CDN directly.

Fraud: CDNs that consistently send fraudulent bids (or fail often) can be marked as "bad" using a reputation system. Their bids can be handled at lower priority in the brokers' decision process.

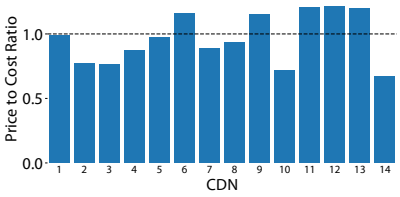


Figure 3.11: Per-CDN price to cost ratio for `Brokered` (less than 1.0 means profit loss).

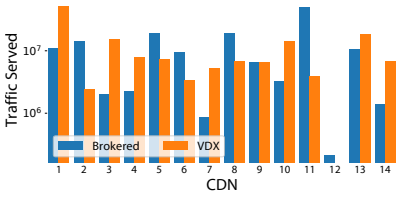


Figure 3.12: Per-CDN traffic for `Brokered` and `VDX`.

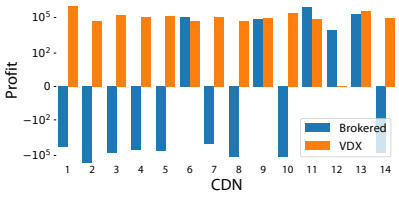


Figure 3.13: Per-CDN profits for `Brokered` and `VDX`.

Brokers that fail to provide clients to CDNs can be handled similarly. CDNs that collude with other CDNs or brokers can be handled similarly, or through legal action if CDN-broker contracts become commonplace (see §3.8).

Scalability limitations: For scalability, instances of `VDX`'s marketplace would most likely need to focus on specific geographic regions, content providers, or content types. However, this division comes at a cost: limiting the broker's view limits the quality of the optimization. Federating these different marketplaces (as well as those run by different brokers) remains an open question.

3.7 Evaluating a Marketplace Design

We evaluate `VDX` using the same simulation methodology from §3.5.1, focusing on comparing `Brokered` to the potential benefits of `VDX`'s `Marketplace` design, in three broad categories: 1) *data driven* which focuses on simulating `VDX` on real data, 2) *scenarios* which augment the data driven simulation, and 3) *microbenchmarks* which adjust knobs within `VDX` to look for trade-offs.

3.7.1 Data Driven

We answer the following two questions:

1. *How do different CDN deployment models compare? Does brokering really treat different CDNs differently?* `Brokered` makes it harder for distributed CDNs to make profits; `VDX` provides fairness.
2. *Do countries see pricing issues?* `Brokered` causes country-level pricing issues (some entirely unprofitable). `VDX` is cost-aware, moving traffic to cheaper ones, and charging appropriately.

CDN-level Pricing Differences

Here we examine how brokering today affects individual CDNs. In Figure 3.11, we show the ratio of flat-rate contract price to cost for `Brokered`. Recall that we compute contract price as an average over all clusters when the CDN is offered the entire workload. We markup this price by 20% to ease later comparison. If the price to cost ratio is less than 1.0, the CDN is losing money on delivery.

Most CDNs do not profit on *brokered video delivery* in our model of a flat-rate world, which may accurately represents the hardships present in some CDNs quarterly filings [5, 6, 104, 119, 120].

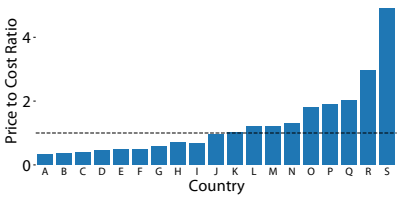


Figure 3.14: Per-country price to cost ratio for `Brokered` (less than 1.0 means profit loss).

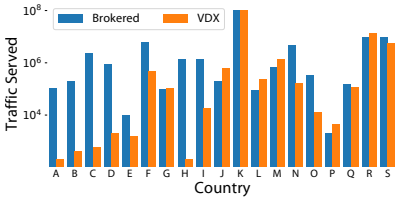


Figure 3.15: Per-country traffic for `Brokered` and `VDX`.

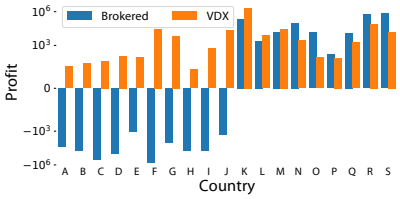


Figure 3.16: Per-country profits for `Brokered` and `VDX`.

Video delivery is traditionally hard to profit on, given its high-bandwidth, low-"importance-per-bit" nature. While video delivery makes up a large portion of CDNs' cost [6], only a subset of it is brokered video delivery. The trend, however, towards using brokers for video delivery, even among small CPs, is rapidly accelerating [134], making these issues even more pertinent.

Examining the CDNs that make a profit, we note that they are all centrally deployed CDNs mainly used in locations where costs are cheap. CDNs profit in `Brokered` only if they use clusters that are cheaper than their contract price. Today's world disincentivizes building large distributed CDNs, as distributed CDNs are more likely to be picked by brokers due to their better performance, yet their larger geographical presence potentially leads to higher cost variability.

Figure 3.12 shows traffic allocation across CDNs. Although CDN 12's cheap clusters are used by our broker, it does not actually serve much traffic. More distributed CDNs, such as CDN 1, have more variability in cluster cost as they are in many more remote regions (see Figure 3.4). Because of this, CDN 1 has an expensive flat-rate price (i.e., median cluster cost), so it is avoided by `Brokered` in favor of the comparably cheaper CDN 11. Moving to `VDX` allows CDN 1's prices to reflect individual cluster costs, allowing `VDX` to use CDN 1's cheaper clusters while avoiding its expensive ones.

Figure 3.13 crisply illustrates this switch. Here we plot each CDN's profits in `Brokered` and `VDX`. In `Brokered`, profit is a markup factor (1.2) times the contract price minus internal CDN cost. `VDX` uses the internal cost as the price, meaning profit is just the markup factor (1.2) times the cluster cost minus the cost. In `Brokered` many expensive CDN clusters are (unknowingly) used, leading to significant deficits for many CDNs in this flat-rate price model. `VDX`'s per-cluster cost model effectively levels competition, allowing each CDN to make profits, regardless of its deployment style.

Country-level Cost Differences

We examine the same data per-country. In Figure 3.14, we see that putting clusters in certain countries is more profitable; namely, countries L–S are easy to profit in, but countries A–J are where CDNs are losing money.

Interestingly we see different patterns in per-country traffic (Figure 3.15) than in per-CDN traffic (Figure 3.12). Country use is mostly even for `Brokered`. `VDX`, however, avoids the most expensive countries (A-E). This implies that `VDX` is sending traffic originating within these countries to clusters in cheaper countries. This may be reasonable in places like Europe, where neighboring countries are geographically close.

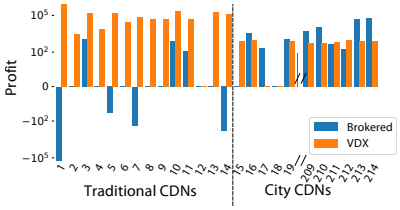


Figure 3.17: Profits for 200 "city-centric" CDNs added to our trace.

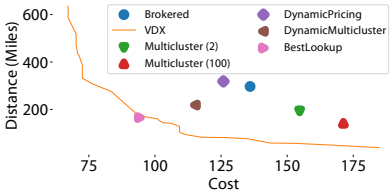


Figure 3.18: Adjusting optimization to balance performance vs. cost.

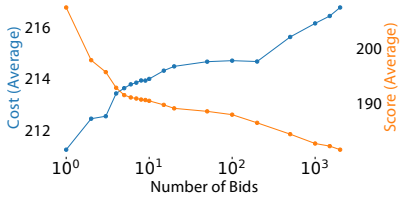


Figure 3.19: Adjusting bid counts vs. cost and score (lower is better).

Figure 3.16 shows CDN profits, calculated similar to Figure 3.13. Here it becomes very clear that in `Brokered` CDNs in countries A–J are losing money, but with `VDX`, CDNs are able to profit even within these expensive countries, as CDNs can now be properly paid for using their expensive clusters.

3.7.2 Scenarios

Here, we answer the following question: *How do hundreds of "city-centric" CDNs affect established CDNs in today's brokered world?* "City-centric" CDNs are always profitable, while traditional CDNs lose money. `VDX` provides a fair playing field.

As previously explained, brokering (both what we see today, as well as our vision of a marketplace), allows for more varied, specialized CDNs. With brokering, CDNs no longer need to provide global coverage (as brokers can stitch together many smaller CDNs), allowing for a rise of "city-centric" CDNs. We model the advent of CDN proliferation by generating 200 single-cluster CDNs to add to our trace. Each cluster is drawn randomly from the CDN location data we collected from PeeringDB [127]. As these clusters are co-located with other CDNs, they drive down the co-location cost in our model.

We show the profits for these CDNs in Figure 3.17 (the city CDNs not shown have similar profits). We find that many traditional CDNs continue to do poorly in `Brokered` as they do in Figure 3.13, while some are sent no traffic at all, but the city CDNs always profit. This is because the cost of their single cluster is always equal to their contract price (as it is their average price), and thus they profit. `VDX` levels out the playing field, allowing traditional CDNs to properly compete.

3.7.3 Microbenchmarks

We answer the following two questions:

1. *How much control do CPs have over `VDX`'s cost / performance trade-off?* Points on `VDX`'s trade-off curve outperform most other designs.
2. *How much impact does CDN bid count have on performance and cost?* Bid count can improve performance, but generally has diminishing returns.

Understanding the performance / cost trade-off

In Figure 3.18 we vary the cost weight w_c in the optimization function run by our broker (see §3.4.1). Not only can VDX lower the cost by ~44% while keeping distance equivalent to Brokered, it can instead lower distance by ~74% while keeping cost equivalent. At the knee of the curve, it can simultaneously lower cost and distance by ~31% and ~40% respectively. There are similar trade-offs that can be made with most other designs.

Number of Bids

Here we vary the number of bids that CDNs submit for every client location. We show its effects on the average cost and score in Figure 3.19. As bids are sorted based on cost, increasing the number of bids should allow better performance (lower score) at higher cost. Interestingly, the largest increase in performance (drop in score) is just achieved by adding the second bid. Having two choices provides much benefit for brokers in meeting CPs goals, but as we have seen, having many more choices and tuning the trade-off is likely more important.

3.8 Discussion

Adoption incentives: While CDNs have incentives to use dynamic cluster pricing (as it removes discrepancies between cluster price and cost), CPs may be hesitant to change their contracts. Similarly, while cluster-level optimization is incentivized for CPs/brokers (to better meet CP goals), CDNs may balk at the idea of providing brokers any additional control (although brokers already ultimately decide which CDN clients go to). We argue that requiring both (seen in a few designs in §3.4.2), provides enough incentive for both CDNs and CPs/brokers.

VDX's marketplace requires very little change to the existing "ecosystem," rather than the creation of an entirely new one (e.g., CDN federation, which inherently requires competitors working together). Furthermore, a marketplace design provides incentives to both large and small CDNs, as it allows both to compete on equal footing (§3.7.2). More nuanced CDN pricing schemes (e.g., low-but-variable pricing combined with high-but-flat pricing, similar to Amazon EC2 [7]) could offer CPs more control in meeting their goals, while retaining similarity to today's flat-rate pricing.

Lack of ISP integration: The lack of ISP integration is a purposeful limitation of this work. First, while there has been much spirited work looking into the ISP-CDN tussle [49, 82, 83, 128, 129], there has been little work focused on the CDN-broker tussle [114]. We view these works as orthogonal to ours, potentially fitting together into a single delivery ecosystem. Additionally, the lines between ISPs and CDNs are becoming much more blurred as large ISPs run their own CDNs (e.g., Comcast [12]), purchase CDN systems from vendors like Huawei [69] or Akamai [4, 139], or allow CDNs to run *virtual servers* within the ISP [47, 49].

Evolving the ecosystem: VDX's marketplace makes it much easier for CPs to meet their goals across a wide array of CDNs. In today's world, CPs sign contracts with CDNs directly, even if they use a broker. We do not need to assume this for VDX's marketplace. Similar to the evolution of online advertising networks, specific CP-CDN contracts could be removed to much more easily meet CP goals (by using many more CDNs), as well as lower the barrier of entry for new CDNs. If

CP-CDN contracts are removed, we expect CP-Broker contracts and CDN-Broker contracts to replace them. Additional intermediary players (e.g., geographic CDN aggregators) may pop up in the ecosystem (similar to ad networks).

Extensions to non-video content: Although VDX is designed with video delivery in mind (as that is where brokers are seen today), there is nothing inherently video-specific about its marketplace. While the optimization done by a broker on behalf of the CP would need to be adjusted, we expect that VDX could be extended to cover different types of content and applications.

The true impact of cost savings: 31% bandwidth and co-location cost savings may seem small, but would save Akamai ~$22.7M per quarter [6]. While clients would also benefit from multiple cluster choices (decreasing cluster distance), the pressing issue is that many of the parties involved in video delivery are having difficulty making much money from it, with some losing money on it [119, 120] or experiencing slowing revenue growth [6, 104]. If video delivery could be assured to be profitable (§3.7.1), that is significantly more impactful than cost savings.

3.9 Related work

Collaboration in content delivery: The most relevant related work looks at widening interfaces in content delivery through collaboration. This includes alternative CDN designs, such as federated Telco-CDNs [13] and P2P-CDN hybrids [13, 163], and the potential benefits of CDN-ISP collaboration [49, 128, 129]. Some focus on the mathematical basis of joint collaboration [82, 83]. These works show that ISPs can aid CDNs in assigning clients to CDN clusters.

Experience Oriented Network Architecture (EONA) [78] argues abstractly that content owners and infrastructure owners should collaborate to improve end clients' QoE. Though similar, we focus on concrete problems faced by CDNs and brokers, and how to fix them.

Other collaboration proposals: Other work on ISP-P2P collaborations [22, 155] or ISP-ISP collaborations [103] are also related in terms of their designs. Both have an actor (an ISP) communicate a set of preferences (i.e., costs) over a set a set of resources (ISP paths), which are then chosen by another actor (an application / another ISP). Neither, however, treat this as a marketplace where bids change over time to strategically match performance / cost goals.

Route Bazaar [24] is more closely related in design. It allows customers to build end-to-end ISP paths using a marketplace. Tuangou [138] propose customer ISPs collaborate to share the cost of upstream provider service. While similar, neither are directly applicable to CDN-broker collaboration.

Online marketplaces: We note strong parallels to online marketplaces, in particular those related to advertising. The most useful for our context are survey papers tracking the evolution from one-on-one contracts to ad networks (e.g., Google AdWords [56]) to ad exchanges (e.g., DoubleClick by Google [57]) [125, 141, 161]. Work in other networking domains have also decried flat-rate pricing in the context of inter-datacenter transfers [76]. Finally, different auction-style pricing mechanisms have been applied to cloud-computing [153, 162].

3.10 Summary

The introduction of brokers into CDN-based content delivery may have caused many issues for both CDNs and brokers due to the lack of explicit joint decision making. Using data from both a broker and a CDN, we show that: 1) brokers need cluster-level info to best meet CP goals, 2) CDNs are not being fairly paid due to the lack of cluster-level pricing, and 3) traffic patterns are unpredictable. We argue that there is a rich design space that solves all three problems, with a marketplace-inspired design providing potentially nice tradeoffs. We design a marketplace-based system called VDX that allows all CDNs to profit on video delivery, improving cost and data-path distance.

Within the context of this thesis, this chapter shows that systems with separate control planes due to administrative separation do, in fact, benefit from using priority ranking to coordinate them. The fact that administrative separation implies constraints on information sharing (as some information must be kept private for business concerns), makes the use of priority ranking appealing. With priority ranking (here a marketplace-style interface), each business can explain how they would like data plane resources used (e.g., map this client to this cluster), without explaining why (e.g., due to private internal cost concerns). This greatly reduces adverse interactions between control planes. Next, we examine how even systems operated by a single entity may have information sharing constraints (e.g., due to timescale separation), and how this changes the nature of coordination between them.

Chapter 4 Control Planes in Internet-Scale Systems: A Case Study in Live Video

In this chapter, we explore one example, in detail, where hierarchical partitioning can be used to overcome adverse interactions between control planes. Specifically, we look at how building high-quality, responsive internet-scale live video delivery requires combining global and local control planes. In this scenario, both control planes make specific decisions on how to use the same data plane resources (here how to route live video through the distribution network).

This chapter highlights two key differences from the previous chapters: 1) even within one business, there are scenarios that constrain information sharing, and 2) the nature of coordination designs differ between systems depending on if their data plane resources are separate or the same.

With VDX (Ch. 3), we saw that coordinating using transparency is untenable between two business, requiring more complex priority ranking schemes. Here, we find that systems that require internet-scale optimization may run at inappropriate timescales (or granularity) to handle failures, and thus, require an additional, fast timescale (or fine-grained) localized control plane. Despite being run by the same business, this *t*imescale separation (or *g*ranularity separation) may constrain information sharing, precluding such systems from using transparency for coordination, as compared to systems like Etalon (Ch. 2).

Systems with disjoint data planes use priority ranking to come to a decision mutually beneficial to both control planes (e.g., VDX). Systems with shared data plane resources generally prioritize the global control plane's decision except in unusual cases (e.g., during failures), when the local control plane's decision is used instead (hierarchical partitioning, as explained in Ch. 1). Thus, while systems like VDX need a relative complex mechanism (bidding) to build a joint decision, the system presented in this chapter, VDN, only needs a way to choose between the global or local decision. This can lead to simpler designs.

We solve the issues presented in our live video delivery scenario with hierarchical partitioning (here *h*ybrid control), achieving the performance of a centralized control plane, and the responsiveness of distributed control plane. We build a system, VDN, where a centralized (global) control plane has priority in how it decides to use data plane resources (CDN clusters), but in case of short-term failures or new client joins, a distributed (local) control plane for an individual CDN cluster ("regions" in hierarchical partitioning) can decide to temporarily override the global decision, using slack present in the network. Thus, we argue that hierarchical partitioning (here hybrid control) is a very reasonable way to build a high performance, responsive internet-scale live video

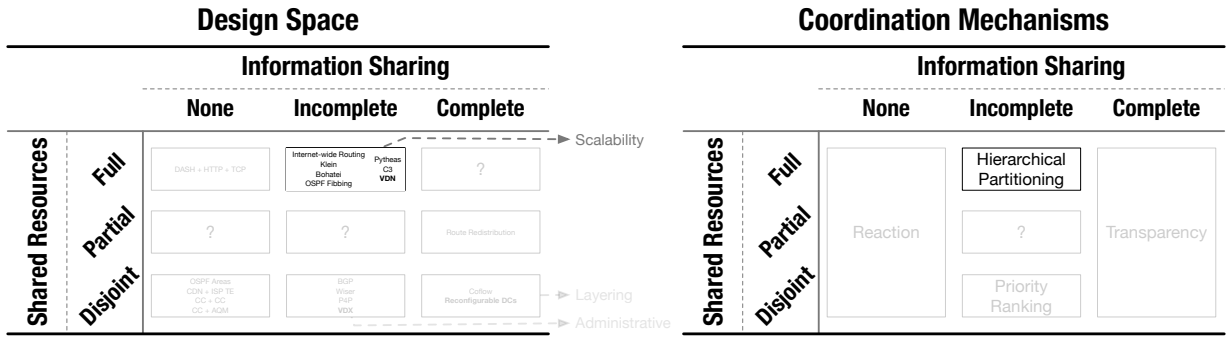


Figure 4.1: This chapter focuses on challenges in live video delivery when building an internet-scale system, as shown with VDN. Looking back at our design space (§1.2.3), internet-scale systems (e.g., VDN, internet-wide routing, Klein, Bohatei, OSPF Fibbing, Pytheas, C3) have constraints on information sharing between a global and local control plane, due to the granularity or timescale separation they see at scale. Their control planes make decisions for a shared set of data plane resources. These systems should coordinate using hierarchical partitioning (§1.1).

delivery system. Figure 4.1 summarizes this in the context of our control plane coordination design space from §1.2.3.

Live video delivery serves as an illustrative (and timely) context: Live video delivery was estimated to reach a peak of 50 Tbps in 2015 [2]. 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 [99] 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 [89], a requirement of both operators and users. We introduce VDN, a practical approach to a live 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.

4.1 Introduction

Demand for live video is increasing by 4–5× every three years and peak live video streaming rates were estimated to reach 50 Tbps in 2015 [2]. This demand spans wildly different types of videos (professionally-produced and user-generated) and workloads ("mega-events" to long-tail). On one end of the spectrum, the 2014 World Cup (a professionally-produced live video mega-event) streamed several terabits per second [137], which is estimated to be 40% of all Internet traffic during that time [135]. At the other extreme, 55 million Twitch users [145] watch more than 150

billion minutes of user-generated long-tail live video each month, generated by over 1 million users, making it the fourth largest Internet traffic producer in the US [144, 146].

This diversity and volume makes live video delivery a complex challenge. The challenges are further complicated, however, as users, CDNs, and the network environment impose additional constraints: Users demand high quality (e.g., high bitrate, instant start-up (join) times, and low buffering ratios) [16]; CDNs want to minimize their delivery costs and respond to issues in real-time [123]; and, the wide-area network environment requires designs that handle common latency variations and communication failures. In summary, we need a live video control plane that: 1) **scales** to the diversity and volume of live video streaming seen by today's largest CDNs, 2) enables proactive control over user **quality** and CDN delivery **cost** at fine-grained timescales, 3) achieves **real-time responsiveness** to minimize join times and failure response, despite wide-area network delays and failures.

Previous solutions (e.g., traffic engineering or overlay multicast) fail to meet all of these requirements. State-of-the-art traffic engineering systems [66, 75] 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, as we will show. Overlay multicast systems [25, 32, 77, 90], while focusing on individual stream optimization, overlook issues that arise due to the lack of coordination across concurrent, independent, high-bandwidth streams, as we will also show. Internet-scale, video-specific systems like Conviva's C3 [51] use client-side analytics to pick the best CDN for a given client at a given time, but do not optimize actual data delivery, which we target in this chapter.

In order to address these challenges, we propose a new system, called *video delivery network* (VDN). VDN is built around the idea that centralized optimization of live stream routing (as opposed to today's distributed CDNs) can greatly improve user quality, as shown in previous analysis [99]. Centralization alone is not enough, however, as wide-area latencies and slow optimization times result in slow join times and failure recovery. Thus, VDN combines the fine-grained proactive control of centralization with the resilience and responsiveness of distributed control in a *hybrid* approach.

We evaluate VDN using a large-scale trace-driven simulation based on real live video sessions, as well as a small-scale 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 or reduces delivery costs by 2× compared to current CDNs. We scale VDN to 10,000 different videos (each with multiple streaming clients), and also 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.

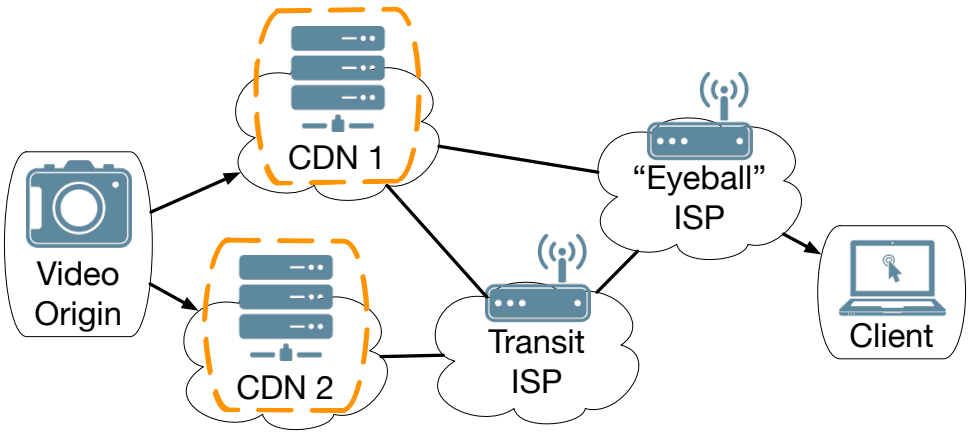


Figure 4.2: Entities involved in live video distribution. Unlike Conviva's C3 [51], which focuses on clients, we focus on optimizing CDNs.

4.2 Motivation

4.2.1 Setting

CDN background: We focus on optimizing CDNs for HTTP-based *live video delivery*. Each entity on the video delivery path (see Figure 4.2) can be independently optimized (e.g., clients in Conviva's C3 [51]), 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 [1].

CDN structure: Figure 4.3 presents the high-level structure of a CDN's video delivery system [89, 123, 143]. 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 [21]. In Figure 4.3, 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.) [123]. When a client's request for a particular video 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.

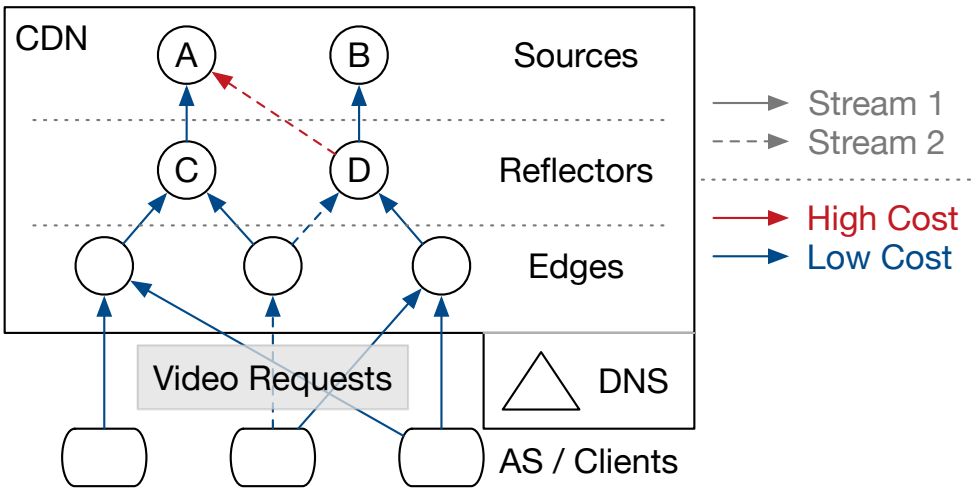


Figure 4.3: CDN live content distribution [123].

This has been the design used by Akamai for live streaming since 2004 [89], externally observed in 2008 [143] and referenced by Akamai in 2010 [123]. We confirm this holds today [21].

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); 2) To reduce load on DNS, CDNs group different videos together, reducing granularity [89, 143]; and 3) CDNs today update DNS mappings using heuristics [21, 89, 123, 143], 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) [143], 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. We explore this in our evaluation.

Video aggregation: Both popular and unpopular videos are binned into groups called "portsets" to reduce the load on DNS [89, 143], while also reducing control granularity.

Heuristic-based mapping algorithm: A monitoring system collects performance and load information and, based on this knowledge, updates the DNS system every minute [123]. 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 [21, 89, 123, 143]. It is implied that the mapping of edge clusters to reflectors is done similarly [123], 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 [143]. Additionally, an

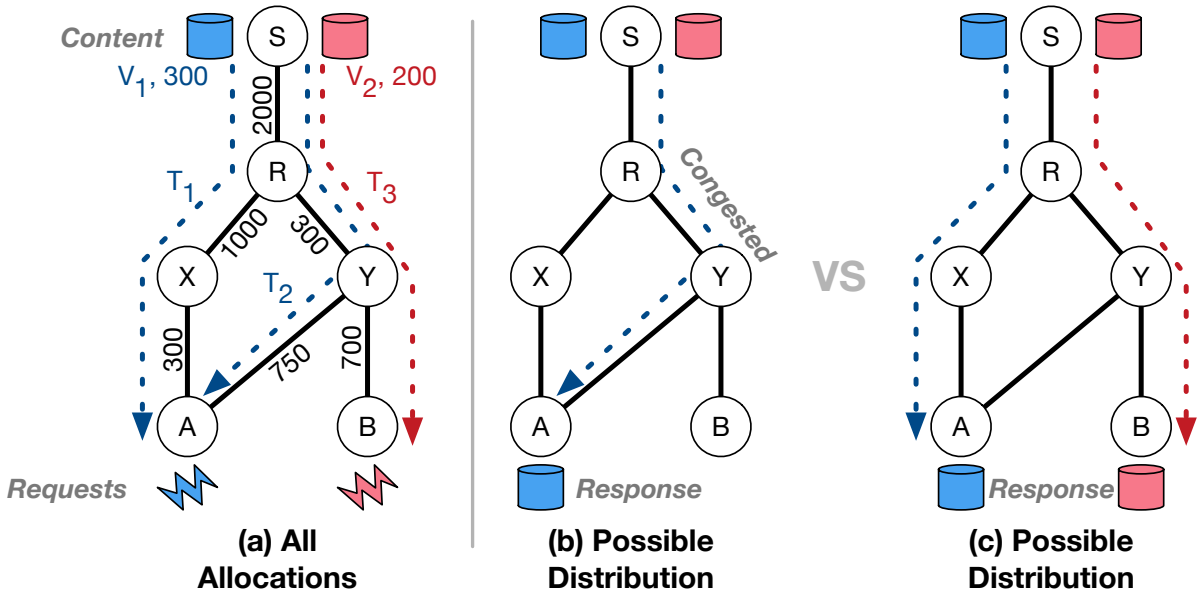


Figure 4.4: Motivating central coordination.

analysis of video traces shows that mapping requests based on a global view of the network [99] could provide major benefits for end-users, implying there are opportunities for improvement.

4.2.2 Design goals

A proper live video delivery system has two jobs: 1) coordinate the selection of distribution trees for each video 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.

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 [137], (b) *TV-style channels* serving 100K users [70], and (c) *"long tail" user channels* (e.g., Twitch, Ustream) serving 1-10,000 users [18]. (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 [143] and 2000 edge clusters [48], beyond the scale of today's largest CDNs. Such scale is challenging as finding the optimal placement complex (slow) integer programming.

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).

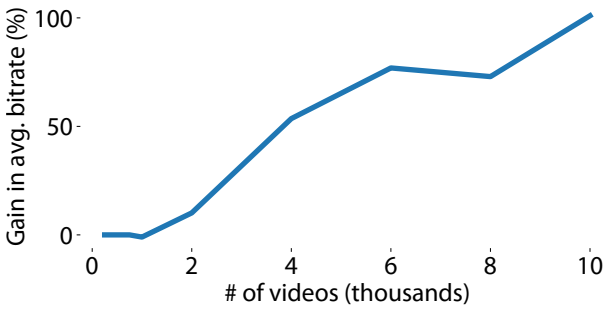


Figure 4.5: The importance of coordinating streams generalizes to larger systems. This graph shows the gain of our system compared to a multicast-style approach as we'll see in §4.7.

4.2.3 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 [99]. In this section we focus on the reasons for these benefits.

Coordination: Throughout this chapter, we use coordination to mean the ability to consider all individual streams simultaneously. As mentioned, modern CDNs have difficulty with this as they both aggregate videos and get "locked in" to decisions due to DNS TTLs [143]. Figure 4.4 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 client/AS A and V_2 to B. Three possible distribution trees exist: T_1, T_2 and T_3 (Figure 4.4a). We present two feasible distribution strategies in Figure 4.4b and c. In Figure 4.4b only client A is served, whereas in Figure 4.4c 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 4.5 compares a system with a global view that places streams independently without coordination (OM in §4.7) to one that has a global view *and* coordinates streams (VDN in §4.7) 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 4.6, two videos are encoded as low quality (400 Kbps) and high quality (1500 Kbps) versions. Due to bandwidth constraints (Figure 4.6a), we must deliver both over link RY. We present two ways to allocate bandwidth in Figure 4.6b and c. Despite fairly allocating bandwidth between the streams, Figure 4.6b does worse overall, as both clients only receive the low quality stream. Figure 4.6c 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.

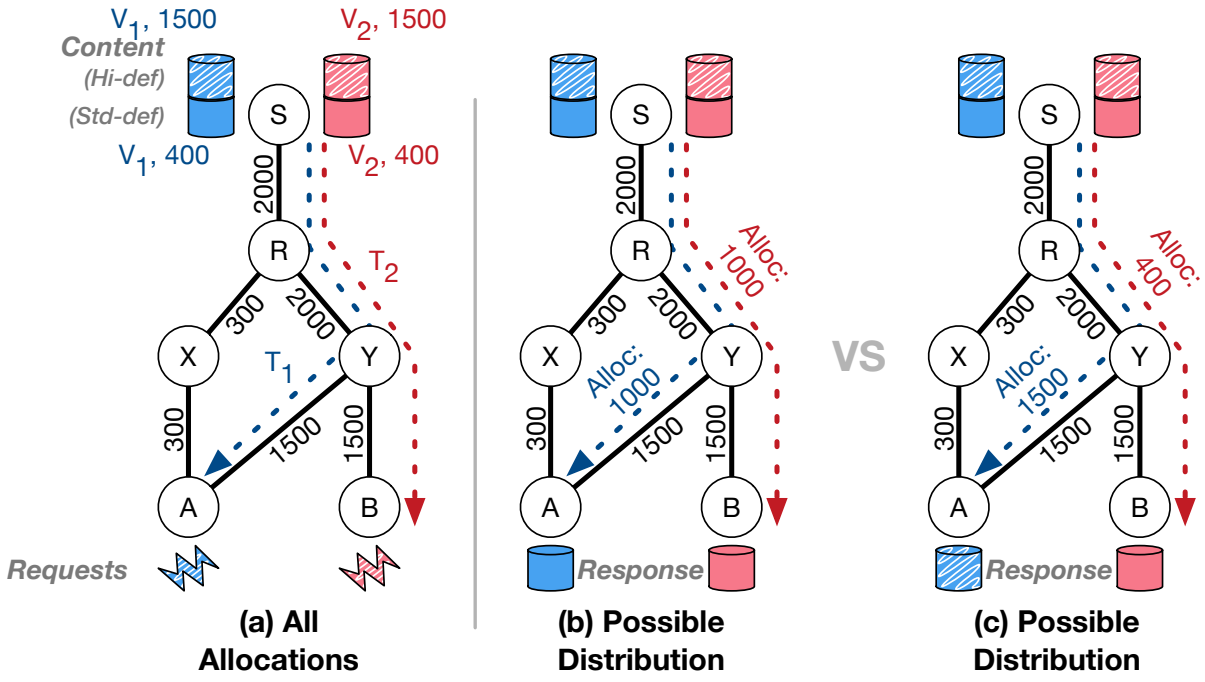


Figure 4.6: Motivating app-specific optimization.

4.2.4 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 slow join times. Additionally, centralized optimization using an integer program can take quite long (e.g., 10s of seconds), making join times terrible. A distributed scheme, by comparison, provides low join times and fast failure recovery, as nearby clusters could react to requests immediately. We argue, however, that a distributed scheme is challenged to provide the high quality demanded by users at reasonable cost, due to the lack of coordination (§4.2.3).

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 control loops by exploiting properties of our highly structured topology (§4.2.1) and by keeping track of alternate paths ("slack") in the network with enough capacity for each video channel (§4.4).

4.3 VDN system overview

Our solution, "video delivery network" (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 based on global state, while individual clusters simultaneously make decisions quickly based on distributed state. VDN

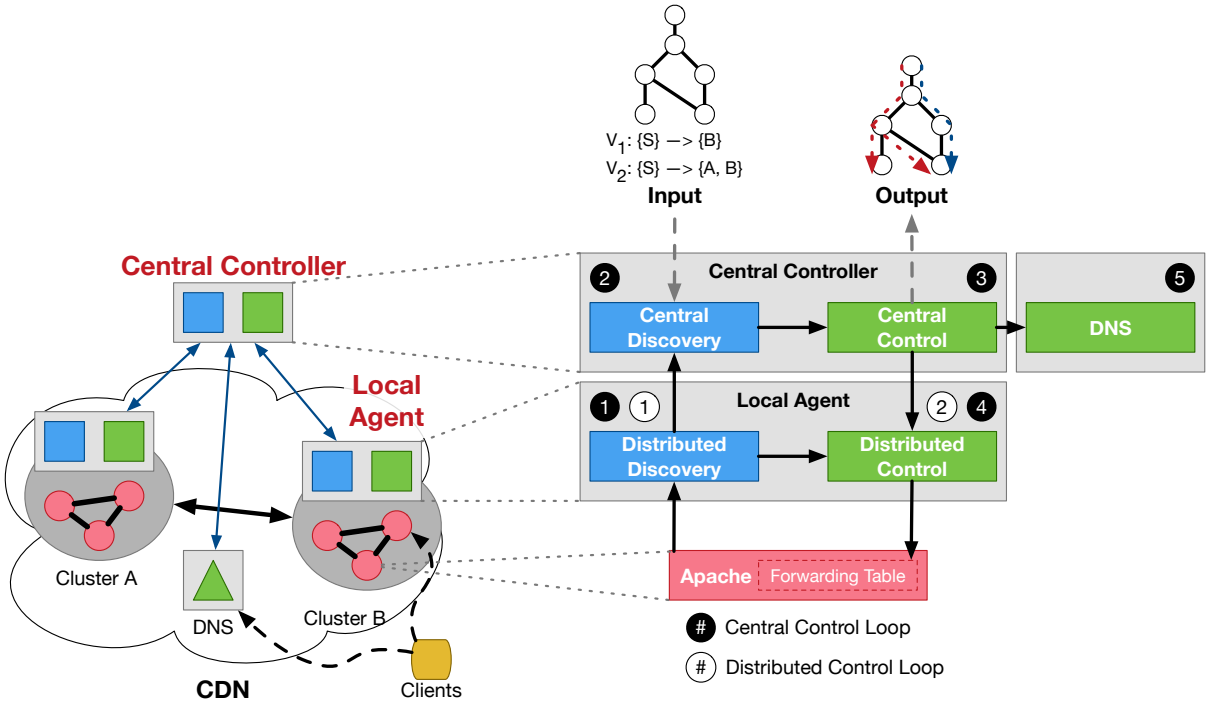


Figure 4.7: VDN system overview.

treats each cluster as an atomic unit (as in Figure 4.7) and controls the distribution of live video from sources to clients; traffic management within a cluster is outside the scope of this work.

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* (§4.4.2). Reflectors pick source clusters similarly. The video chunk follows this path in reverse. Eventually, centralized control updates the clusters' forwarding tables (§4.4.1 and 4.5).

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.

4.3.1 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 *discovery* subsystem that tracks incoming requests and topology information, and (2) a *control* subsystem that computes path assignments based on network state.

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 §4.5. This loop runs continuously and operates on the timescale of tens 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.

4.4 Hybrid control

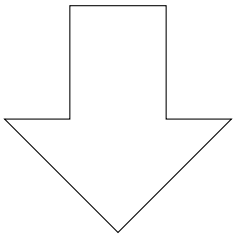
Running two control loops in tandem can lead to challenges that destroy any benefit that either control loops would have had individually, resulting in a "worst of both worlds" scenario, as hinted in §4.2.4. When distributed decision-making is used, hybrid control handles this by only considering end-to-end paths in the network that were purposely left with "slack" (additional 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.

4.4.1 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 §4.5 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 [92]. 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

Routing Information Base

Central / Distributed	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



Forward Information Base

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

Table 4.1: 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.

a *routing information base* (RIB) specific to that cluster, as shown in Table 4.1. 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 central control 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 central control 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 [100, 131, 142], and is thus considered out of scope in this work.

4.4.2 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

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

Table 4.2: Example of the distributed state table used in Algorithm 1.

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 aid in reacting to large changes. Each cluster periodically (e.g., every second) sends all connected clusters at the next lower layer (see Figure 4.3) 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 Table 4.2) 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.

Reacting to large changes: If distributed 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.

If the central control "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 (Table 4.2).

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

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                                        */
```
$useful := \emptyset$
for *parent* **in** *parents* **do**
  if $d(v,b)_{via\_parent} == \min(d(v,b))$ **and**
   $c(v,b)_{via\_parent} > b$ **then**
    $useful = useful \cup \{parent\}$
  end
end
return *pick_at_random(useful)*

Algorithm 1: Distributed control algorithm.

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 (Table 4.1). 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 central 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 distributed/central 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.

4.4.3 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 [53, 85, 106]. 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.

4.5 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. 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, Priority$_o > 0$, set by operators indicating how important it is to serve o.

Topology: Our network topology (see Figure 4.9a) is a directed graph made of server clusters (sources, reflectors, and edges as explained in §4.2) 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 [123], 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.).

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

$$
\begin{aligned}
&\max\ w_s * \textstyle\sum_{l \in L_{AS}, o \in O} \text{Priority}_o * \text{Request}_{l,o} * \text{Serves}_{l,o} \\
&-\quad w_c * \textstyle\sum_{l \in L, o \in O} \text{Cost}(l) * \text{Bitrate}(o) * \text{Serves}_{l,o} \\
&\textbf{subject to:} \\
&\forall l \in L, o \in O : \text{Serves}_{l,o} \in \{0, 1\} \\
&\forall l \in L \qquad\ : \textstyle\sum_o \text{Bitrate}(o) * \text{Serves}_{l,o} \leq \text{Capacity}(l) \\
&\forall l \in L, o \in O : \textstyle\sum_{l' \in \text{InLinks}(l)} \text{Serves}_{l',o} \geq \text{Serves}_{l,o}
\end{aligned}
$$

Figure 4.8: Integer program at the controller.

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 (performance) $w_s \geq 0$ to strike a balance between performance and cost.

Formulation: Figure 4.8 presents our problem formulation. The optimization takes the following as input (and treats them as constants): w_s, w_c, Priority, Request, 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; e.g., prioritize high bitrate requests or prioritize satisfying as many low bitrate requests as possible) in §4.7.1. 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 4.9 gives an example input with two channels V_1 and V_2, with stream bitrates 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 4.9a shows the topology, link capacities, and costs. Link *YA* has a relatively high cost of 10. Figure 4.9b 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

Video	Bitrates (Kbps)	Priorities	Requests (at Start)
V_1	[200, 800]	[1, 1]	(A, 800), (B, 800)
V_2	[300, 900]	[1, 100]	-

Service Weight	1000
Cost Weight	0.1

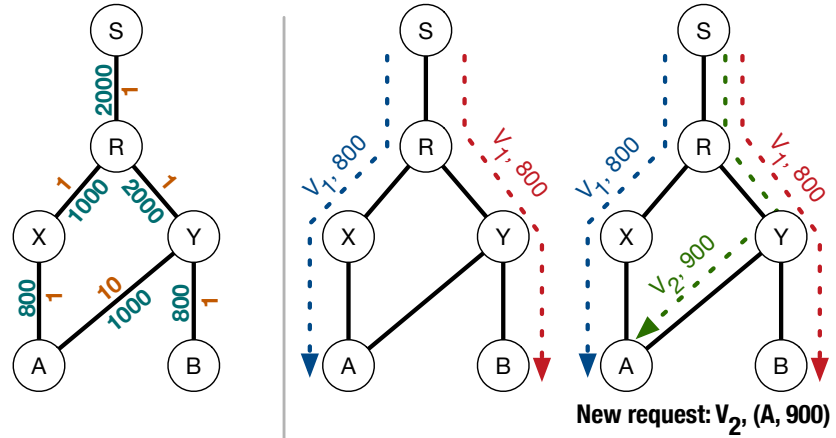


Figure 4.9: Example input and output of the centralized optimization.

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.

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 4.10 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 §4.2.2), 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 [124], making this a reasonable target.

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

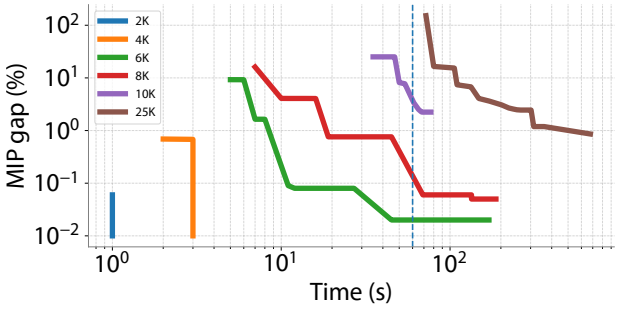


Figure 4.10: The centralized optimization MIP gap shows rapid improvement in a short time-frame, even for large numbers of videos.

4.6 Prototype implementation

Control plane: We build a prototype central controller that uses Gurobi [60] to solve the integer program. For trace-driven experiments, we run the controller on an r3.8xlarge EC2 instance [7]. For end-to-end experiments, we use a smaller number of nodes than trace-driven, thus the controller runs comfortable on a much slower 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 measured ~10ms latency.

Data plane: We build a prototype data plane for our end-to-end experiments using Apache [46] running on t2.small EC2 instances. Our data plane uses standard Apache modules: *mod_proxy* configures nodes as reverse HTTP proxies and *mod_cache* 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.

4.7 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.

4.7.1 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 [159], 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 [123] consisting of 4 source clusters, 10 reflectors, and 100 edge clusters. Akamai has roughly 1,800 clusters (1,000 networks) located worldwide [48], 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 §4.2.1).

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

	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 4.3: 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 4.4: 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 4.5: Heavy-Tail trace.

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.)

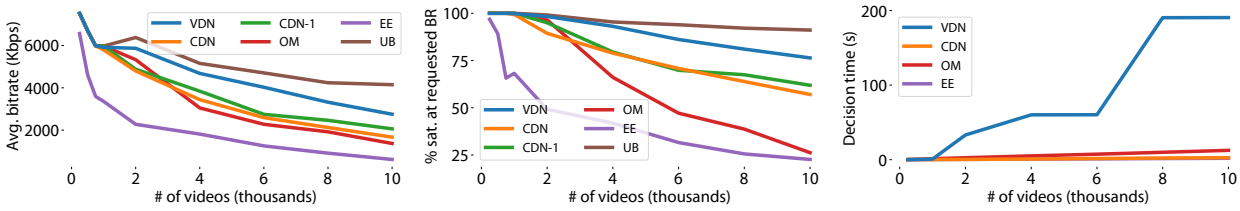
In addition, we assign each link a *cost*, loosely modeling a CDN's cost for transferring data on that link. Source-reflector and reflector-edge link costs vary from 10 to 50 units over a normal distribution. Links from edge clusters to client groups are handled differently: half have a cost of 0, since CDNs often strike deals with edge ISPs [21]; 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 [21, 89, 123, 143]. 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 [99, 143] and a high-level description [21, 123]): 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 [143]. CDN assumes that server selection is stored in a DNS cache with a TTL of 30 seconds [143]. 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 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.)



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

Figure 4.11: Scaling load: increasing the number of videos.

Trace results

Tables 4.3, 4.4, and 4.5 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.

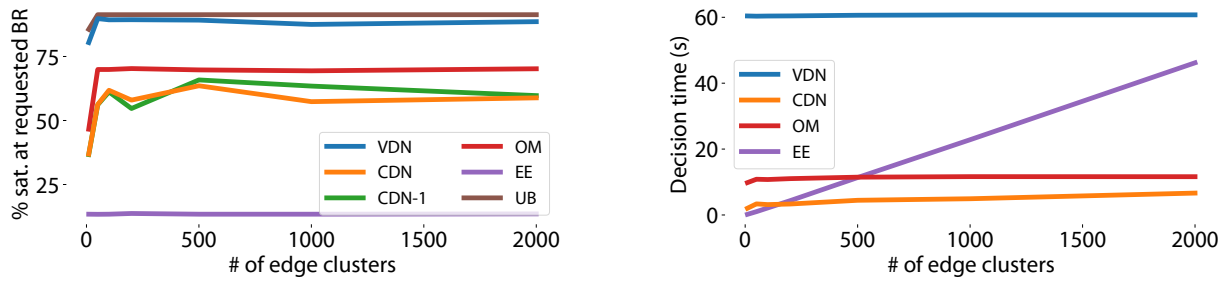
Heavy-Tail is the toughest workload to coordinate. VDN provides a 1.7× improvement in quality, while serving 24% more clients at their requested bitrate. This is because other schemes react to requests individually; DNS-based schemes like CDN get "locked in" to decisions until DNS records time-out, making 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.

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.

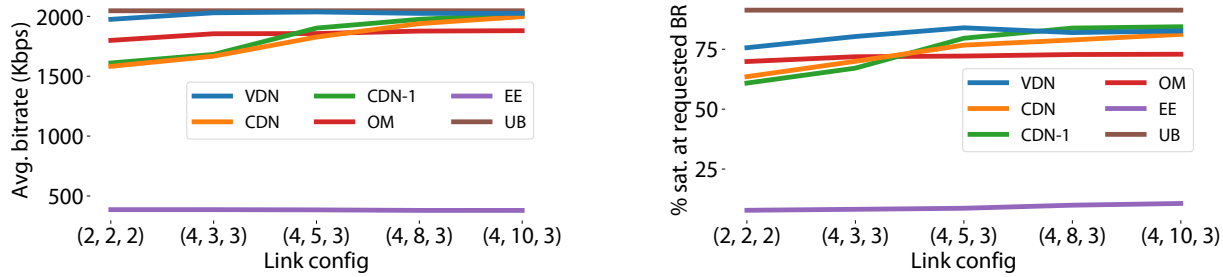
Number of videos: In Figure 4.11, we augment Heavy-Tail with increasing numbers of videos and requests, keeping the video/request ratio, topology, and capacity constant. As we stress the system, it becomes more difficult to place videos. Thus, coordination becomes more important with less spare capacity 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 4.11a) and the number of clients satisfied at the requested bitrate (Figure 4.11b). CDN-1 does marginally better than CDN, but a system with path-level optimization and per-stream coordination (e.g.,



(a) % at requested bitrate

(b) Processing time

Figure 4.12: Scaling network size: increasing the number of edge clusters.



(a) Avg. client bitrate

(b) % at requested bitrate

Figure 4.13: Topology sensitivity: (x, y, z) indicates reflectors are connected to x sources, edge clusters to y reflectors, and client groups to z edge clusters.

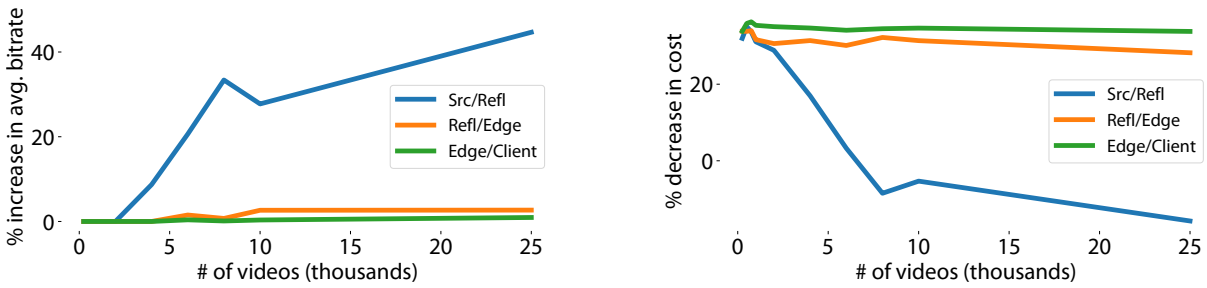
VDN) does substantially better, as expected. Interestingly, OM satisfies fewer clients than CDN, most likely due to OM grabbing key resources early, starving later clients.

As expected, VDN's improved assignments come at the cost of longer decision times (Figure 4.11c). 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 §4.5). 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 4.12). We see that VDN maintains the ability to satisfy roughly 90% of clients at their requested bitrate (effectively the naive upper bound) in 60 seconds for this workload (as opposed to Heavy-Tail, which required 190 seconds; Figure 4.11c).

Topology sensitivity: Next, we explore the impact of the network topology on the designs. 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 (bottleneck placement).

Network connectivity: Figure 4.13 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



(a) Avg. client bitrate

(b) Cost

Figure 4.14: Bottleneck location: improvement over CDN with the bottleneck between source/reflector links, reflector/edge cluster links, and edge cluster/client links.

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.

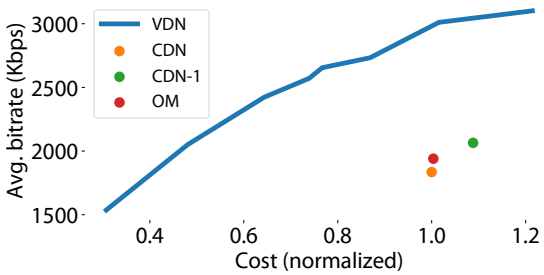
Bottleneck placement: We evaluate the impact of the location of the capacity bottleneck. We begin with the topology described in §4.7.1; the aggregate capacity from sources to reflectors is 4 Gbps, from reflectors to edge clusters is 30 Gbps, and from edge clusters to client groups is 900 Gbps (denoted $(4, 30, 900)$ and named Source Constrained). We now construct two additional topologies: Reflector Constrained $(400, 30, 900)$ and Edge Constrained $(4000, 3000, 900)$.

Figure 4.14 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, 21]). In all three cases, VDN improves average bitrate (Figure 4.14a). It also reduces cost up through 6,000 videos (Figure 4.14b), 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.

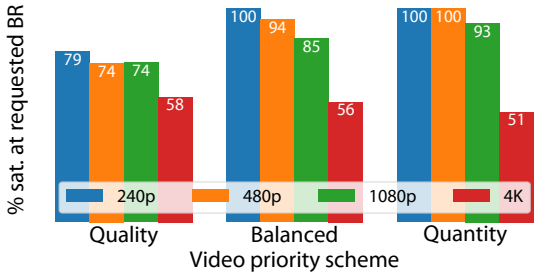
Customizing VDN

Quality vs. cost: By adjusting the weight of the global cost term in the objective function, operators can tune the quality/cost tradeoff. Figure 4.15a 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 $(priority = bitrate)$, Balanced $(priority = 1)$, and Quantity $(priority = 1/bitrate)$. Quality favors serving high bitrate streams; Quantity favors serving as many streams as possible. Figure 4.15b shows the percentage of satisfied requests for



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



(b) Quality vs. quantity: impact of video prioritization strategies

Figure 4.15: VDN gives operators fine-grained control.

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 video is delivered, but what is delivered (e.g., ensuring premium customers *always* receive HD streams even when many free customers request SD streams).

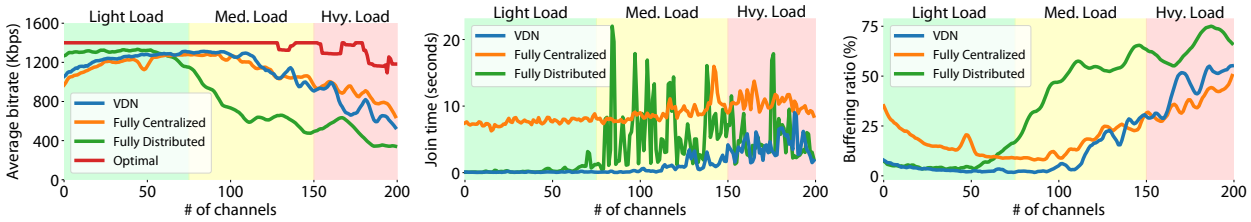
4.7.2 End-to-end experiments

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 [7], each representing a cluster, configured in a three-tiered CDN topology (described in §4.2). 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 §4.4 and Fully Centralized uses only the central controller in §4.5. 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 ~150Mbps, 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.



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

Figure 4.16: Client-side quality in testbed: increasing the number of videos.

Quality of experience

Figure 4.16a 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 4.16b, 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 4.16c shows buffering ratio. Despite having good quality overall, Fully Centralized has a much worse buffering ratio due to its lack of responsiveness.

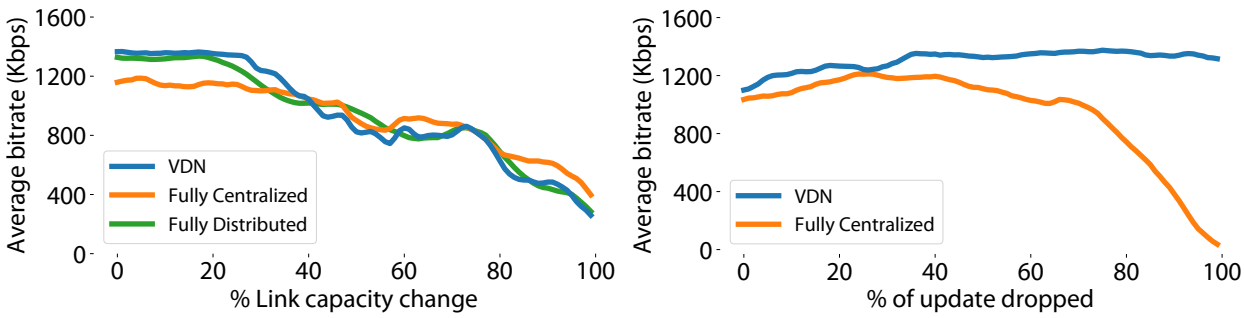
Coping with network events

Figure 4.17a shows the effects of link fluctuations. We select 25% of links at random, degrade their capacity by increasing amounts (using `tc`), and measure the performance 10 seconds after adding 10 channels. We see that all three systems perform similarly.

Figure 4.17b 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.

4.8 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 4.15a and 4.15b). Additionally, VDN provides a centralized point of management to



(a) Link fluctuation

(b) Updates dropped

Figure 4.17: VDN handles network issues without much degradation.

adjust link costs and video priorities. Furthermore, as seen in §4.7, 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 [21, 89, 123, 143]. 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 [14].

4.9 Related work

Content delivery networks: Large- (e.g., [89, 123]) and medium-scale (e.g., [50, 152]) CDN systems have explored various design choices, including peer-to-peer, hybrid [63, 156], centralized, or hierarchical architectures [67] as well as their tradeoffs [160]. 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 [25, 32, 77, 90] focuses on how to best organize individual streams. However, they do not perform extensive coordination across video streams. P2P-based approaches [90] can potentially benefit VDN, but may cause additional issues with hybrid control (e.g., loops) as they complicate the topology.

Traffic engineering: Recent work [30, 45, 66, 75] 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 [80], metrics [14, 16], cross-CDN optimization (e.g., [99]), and CDN-selection strategies (e.g., [51]). Our work focuses on end-to-end delivery and provides a practical system design.

4.10 Summary

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.

Within the context of this thesis, this chapter shows that systems with separate global and local control planes for achieving internet scalability do, in fact, benefit from using hierarchical partitioning to coordinate them. As internet-wide scaling in many systems requires timescale separation or granularity separation, there are constraints on information sharing (as some information will either be stale or coarse), making the use of hierarchical partitioning appealing. With hierarchical partitioning (here hybrid control), the global control plane can make a slow (or coarse) decision as to how the shared data plane resources should be used, while the local control plane can make a quick (or fine-grained) decision when needed (e.g., during new client joins, failures). The fact that the global control plane has priority over the local control plane, and that the local control plane only makes decisions using slack in the network relative to the global decision, greatly reduces adverse interactions between control planes.

Chapter 5 Conclusions

This dissertation focuses on eliminating adverse interactions between control planes in independent network systems. While systems that deal with these issues have previously been designed ad hoc, the key contribution of this thesis is a set of *recipes* for interfaces between control planes that alleviate these issues. These recipes provide means for control planes to coordinate their decisions in ways that make their individual concerns (e.g., performance with respect to differing metrics, cost, etc.) apparent to other control planes.

Recipes are chosen based on 1) how much information can be shared, and 2) whether or not both control planes make decisions for the same set of shared resources (i.e., have overlap in their data plane devices). These two pieces are essential because constraints in information sharing effectively limit what interface can be built, and having disjoint/shared resources determines the possible styles of decision making (e.g., joint, coarse + fine, etc.).

Here we touch on takeaways, shortcomings, and future work, before concluding.

5.1 Takeaways

The core contributions of this thesis are: 1) identifying a set of key scenarios that consistently appear across systems that, by definition, lead to split control planes, 2) an understanding of why certain scenarios naturally lead to specific coordination mechanisms, and 3) a case study showing how to implement these key coordination mechanisms effectively.

Scenarios that lead to split control: This thesis focuses on three core scenarios that lead to split control planes: 1) layering, 2) administrative separation, and 3) internet-scale systems. While there are likely other scenarios that also lead to split control planes, these three appear overwhelmingly in prior work.

Layering is when control is split for modularity, potentially giving up performance. Layering is used throughout the network stack (e.g., splitting applications, TCP, IP), as well within the application layer (e.g., splitting Hadoop [10] from HDFS [11]). Regaining lost performance in layered system is often done through cross-layer optimization (i.e., specializing two layers for one another). While cross-layer optimization works well in practice, it tends to be fragile. As each layer becomes tightly bound to one another, they give up much of the modularity that layering provided initially. A better solution would be to come up with a more expressive interface between layers. Defining such an interface, however, tends to be very challenging.

Administrative separation is when control is split across companies. This scenario is quite common (e.g., BGP running between ISPs, CDN server selection + ISP traffic engineering, etc.). A key

concern is that some information must be kept private for business reasons. Thus, administratively separate systems generally need specialized coordination mechanisms that allow them to share some subset of information (e.g., BGP ingress preferences), without necessarily explaining how that information was derived (e.g., from internal path costs). Both control planes make decisions for their own disjoint sets of resources.

Internet-scale systems have started using split control planes to aid in scaling. These systems tend run a global control plane in tandem with several local control planes in order to approximate an optimal (global) decision while providing fast (local) response times (e.g., for new clients or failures). We see these systems in video delivery (VDN [117], C3 [51]), DDoS prevention (Bohatei [43]), cellular core routing (Klein [132]), as well as other contexts. Some of these systems employ a timescale split (i.e., slow global control and fast local control), while others employ a granularity split (i.e., coarse global control and fine local control). All systems, however, have both global and local control make decisions for the same set of shared resources.

Recipes for split control plane coordination: These scenarios each have very different mechanisms for coordination between control planes. We examine why these different scenarios lead to these distinct mechanisms, leading us to a set of design recipes for split control plane coordination (i.e., if your scenario looks like this, try this coordination mechanism).

If your system is layered, use transparency. We define transparency to be a coordination interface between control planes where all information that needs to be shared, to come up with the best decision for both control planes, can be shared. Layering is a natural candidate for transparency because usually all layers are controlled by one entity, and are just split for modularity.

If your system is administratively split, use priority ranking. We define priority ranking to be a coordination interface between control planes where a ranked list of decisions for how to use one control plane's set of resources is provided to another control plane. While these decisions are ranked in the order that the first control plane would like them to be used, the first control plane does not provide information as to how these decisions were computed. This allows them to keep key information private for business reasons. The key insight is that administratively split systems are constrained in what information they can share, and that priority ranking is an expressive means of coordination that respects this constraint.

If your system is split for internet scalability, use hierarchical partitioning. We define hierarchical partitioning to be a coordination interface between control planes where a global control plane optimizes overall performance for a shared set of resources and multiple local control planes focus on local performance and responsiveness. These two control planes may have differing timescales (i.e., global control runs a slow, but fine-grained optimization over all resources, whereas local control comes to a fast, but sub-optimal decision) or differing granularity (i.e., global control runs a fast, but coarse allocation of resources, whereas local control refines resource allocation within its region). The key insight is that hierarchical partitioning provides scalability because it decouples global optimality concerns from local responsiveness. Generally, a single control plane at internet-scale has to choose between high performance *or* responsiveness. A split global/local control plane can provide a better tradeoff.

Beyond the three scenarios we look at in detail, building a design space helps us come to understand one additional recipe:

If your system can share no information, use reaction. We define reaction to be a rudimentary form of coordination between control planes which lack an explicit interface for decision making. They instead learn of each other's decisions by observing their effects in the data plane. Systems that can not share any information must use reaction by definition, but we find in practice very few systems have such strong constraints. Most prior work that uses reaction (e.g., CDN server selection + ISP traffic engineering, competing congestion control algorithms, brokered video delivery today), do so because reaction is easier to implement compared to an explicit coordination interface. We argue that many reaction-based systems can be improved with the addition of a coordination interface, as we show with brokered video delivery (Ch. 3).

Why do these scenarios lead to such varied coordination mechanisms? Coordination between control planes can be thought of in two steps: 1) information is shared, and 2) a decision is computed. It is clear that constraints on what information can be shared directly impacts step 1. We argue that whether or not decisions across control planes are made for the same set of resources impacts how an overall decision is computed. Namely, decision making styles seen in internet-scale systems (e.g., coarse decisions + refinement, or slow fine-grained global decisions + fast local updates) inherently don't make sense in scenarios where decisions are made for disjoint resources (e.g., what does refinement mean for two control planes that don't have shared resources). Thus, we argue that the underlying reason why these scenarios lead to these mechanisms is because of constraints on *information sharing* and whether they make decisions for *shared resources*.

Split control planes in practice: We build three systems that exemplify these recipes.

We show that transparency can regain performance in systems split for layering using our emulator Etalon, in the context of reconfigurable datacenter networks (Ch. 2). Reconfigurable datacenter networks face a variety of end-to-end challenges due to their combined circuit and packet networks. We show that there are useful cross-layer optimization techniques that can combat many of these problems (e.g., dynamic in-network buffer resizing to limit the impact the impact of bandwidth fluctuation, or making application-specific modifications to alleviate difficult-to-schedule workloads).

We show that priority ranking can improve administratively separated systems compared to using reaction. We show this with VDX, in the context of content brokering (Ch. 3). Brokers and CDNs today don't share any information (i.e., they simply use reaction as a rudimentary form of coordination). We propose a marketplace-style design using priority ranked bids to allow CDNs to provide rough cost and performance estimates at a much finer cluster-level granularity. This allows brokers to better optimize performance and cost for content providers, while simultaneously allowing CDNs to increase revenue.

We show that hierarchical partitioning can improve the scaling properties of internet-wide systems with VDN, in the context of live video delivery (Ch. 4). VDN employs a timescale split between a global and local control plane to achieve near-optimal performance, while being able to quickly react to new client joins and network failures. Hierarchical partitioning provides local control with enough information to make decisions quickly, while simultaneously affording global control enough time to perform its slower (internet-scale) video optimization. The key insight is that it separates these varying timescale concerns into two control planes. VDN avoids poor performance interactions by having global control have priority over local control, and by having local control always use slack in the network in its decisions.

5.2 Shortcomings

While we provide an initial view of the design space for split control plane coordination, we do not argue that we have explored all parts of the space. The core contribution of this thesis are recipes for coordination, showing that certain extremely common scenarios lead to very specific coordination mechanisms. Thus, in this section we discuss scenarios and mechanism that have been missed in this initial view of the design space, as well as the lack of a theoretical framework to examine why each coordination mechanism works properly.

Scenarios and coordination mechanisms outside of those presented: While we present three common split control scenarios (layering, administrative separation, and internet-scale systems) and recipes that map each scenario to different coordination mechanisms (transparency, priority ranking, and hierarchical partitioning, respectively), there are some systems that either fall outside of these scenarios or don't fit their respective recipes.

For example, routers may provide TCP with explicit feedback about path congestion (explicit congestion notification, ECN [133]), by setting a single bit in a packet. One could argue that this is all the information needed to understand congestion has occurred, and thus, this is coordination using transparency. There is, however, much more information that this router has that would be useful for TCP in combating congestion. For example, which router on the path experienced congestion? How many packets are sitting in queues on this router? At what point will this router drop packets? It is clear, therefore, that there are some information sharing constraints.

Single bit ECN markings are used because routers are *line-rate systems*. Line-rate systems need to send control plane messages potentially on a per-packet basis. Thus, such systems conflate control plane timescales with data plane timescales. This clearly constrains how complex message generation tasks can be, as we've seen in this example. Line-rate systems represent a scenario that we have not pursued in this thesis. While such systems represent an interesting scenario for coordination, they tend to not result in interesting coordination mechanism because they are so drastically constrained, similar to how systems that can not share any information must use reaction.

More generally, splitting the design space based on information sharing and shared resources is purposefully coarse; we provide an initial intuition as to how control planes may be split and how to coordinate between them despite this split, but paint a picture with broad strokes. Simply saying the amount of information sharing between two control planes is "complete," "some," or "none" glosses over some subtlety, claiming that designs that share one bit of information and designs that share *all but one* bit of information should be considered equivalent.

Thus, we argue that there likely more scenarios to uncover in this space, as well as more coordination mechanisms, by looking at both axes in finer granularity, as well as perhaps considering additional axes. These additional scenarios / mechanisms likely have constraints on information sharing, as these constraints led us to our more complex mechanisms (priority ranking and hierarchical partitioning). Despite this, our recipes that map specific scenarios to specific coordination mechanism are likely always correct, by their definitions.

A control theoretic approach to the coordinating split control planes: This thesis focuses on coordination across split network control planes through the lens of a system builder; while the

focus of this work is providing recipes for coordination in certain common scenarios, there is less focus on building a theoretical framework as to *why* each coordination mechanism works. Control theory may provide interesting avenues to help ground the presented coordination mechanisms into a theoretical framework.

5.3 Future work

We briefly point to open issues found during this thesis, leaving them to future work:

Network verification across control planes: This thesis is concerned with interactions between control planes that operate independently. While some of these split control planes are logically thought of as part of the same system (e.g., two BGP instances running in different ISPs), some of these control planes are generally thought of logically distinct pieces (e.g., endhost stack layering). While verifying the correctness of individual protocols (e.g., routing within an SDN-run ISP [87]) may be feasible, it is significantly harder to verify that all implicit / explicit interactions between control planes lead to the correct outcome.

Performance guarantees in hierarchical partitioning: Chapter 4 focused on VDN, where global control makes long timescale decisions for performance/cost optimization, while local control makes short timescale decisions for responsiveness to new client joins or failures. Thus, VDN coordinates using hierarchical partitioning. While we show through performance evaluation that VDN's hierarchical partitioning works correctly in a testbed, providing theoretical guarantees that there are no pathologically bad situations that decimate performance or responsiveness has yet to be done. Given the large difference in timescale, the fact that global control has priority over local control, and that local control only makes use of slack in the network, nice theoretical properties may be possible to prove.

Designing marketplace-style interfaces with multiple brokers: Chapter 3 focused on VDX, a marketplace built around CDNs and a content broker. VDX levels the playing field for CDNs when there is a single broker. There is still work to be done to understand what complex problems arise when additional brokers are added to the marketplace. For example, CDNs will likely need to send bids to multiple brokers with overlapping capacity utilization (i.e., if all bids are accepted the CDN will be overloaded, but if only some bids are accepted, the CDN will be fine). If CDNs become overloaded, they will not be able to meet their performance estimates. How should brokers handle this scenario? Should bids be considered hints or guarantees? Additionally, work must be done to ensure that both brokers and CDNs are unable to cheat the system.

Reconfigurable datacenters: While fairly well studied over the past decade, there are still fundamental open problems in reconfigurable DCs. There has yet to be an "apples-to-apples" comparison of different switch designs in terms of technologies (e.g., optical, 60GHz, free-space optics), features/limitations (e.g., circuits can be perfect matchings only, indirection routing, multicast), and timescales (e.g., millisecond reconfiguration delay, microsecond, future nanosecond). In Chapter 2, we've shown that the nature of end-to-end challenges change based on these features (e.g., bandwidth fluctuations only affect TCP when we reach microsecond timescales); investigating

which combinations provide optimal properties, using our open-source emulator Etalon, can provide a principled approach to reconfigurable switch design, rather than just applying trendy new technology.

More specific open problems are also interesting, e.g., remote direct memory access (RDMA) has greatly improved DC performance by removing the remote CPU from the critical path. What changes are needed in reconfigurable DCs to support RDMA or RDMA-like primitives?

Can machine learning-based scheduling algorithms seen at the application-layer (e.g., for MapReduce tasks) apply to circuit scheduling in reconfigurable DCs? What challenges arise from the 6 orders of magnitude shift in timescale, and does the resulting scheduler look similar to prior (hand-built) schedulers like our work Solstice [98]?

Network / endhost co-design: Chapter 2 provides a cursory glance at end-to-end challenges that arise if fundamental assumptions about the network change, without making changes to the endhost stacks and applications. This need for network / endhost co-design appears in a variety of scenarios. A simple example would be adding multicast to datacenter networks (as has been proposed for reconfigurable DCs, for example); not only does this break assumptions for in-network packet / flow scheduling algorithms, it also provides new performance opportunities for low-level scheduling and for applications (e.g., HDFS could cut replication traffic in half using multicast).

Our work on reconfigurable datacenters opens up interesting questions about making applications "network-aware" (as opposed to making the network "application-aware" [95, 151, 154]), a problem that will likely become increasing important. While we manually modify HDFS to provide an easier-to-schedule workload for the network, the key insight was that applications make assumptions about how easy it is for the network to transmit different workloads. Are there a better set of primitives (e.g., anycast, multicast) that could be built into applications to better automate network-awareness?

Furthermore, some datacenter network designs propose combining multiple heterogeneous networks. While MultiPath TCP has been explored with heterogeneous paths in the wide-area, its impact has yet to be explored over heterogeneous paths at DC timescales.

Finally, the advent of re-programmable FPGA-based NICs and re-programmable switches (e.g., P4-based [126] or Mellanox Spectrum Linux-based [108]) provides a strong indicator that changes to network functionality is imminent. Thus, a strong understanding of network / endhost co-design will be required to overcome the challenges laid out in this thesis.

5.4 Final remarks

This thesis presents the idea that regardless of how clever a system is that solves a network control problem, it doesn't exist in isolation. Other systems may interact with it in unexpected ways, leading to a variety of issues in varying metrics (e.g., performance, correctness, cost). Thus, having a mechanism to express concerns about decisions made by other control planes (*coordination*) is paramount; control plane coordination should be a first-order design decision, not an after-thought.

While there are an overwhelming variety of issues that arise from the lack of control plane coordination in different contexts, we find that there are a small set of coordination mechanisms

that appear again and again in practice. Each of these mechanisms (reaction, transparency, priority ranking, and hierarchical partitioning) satisfies very different needs in terms of interface design (i.e,. how much information can be shared) and decision making style (e.g., hierarchical partitioning may do coarse grain + fine grain decision making over shared resources).

These four mechanisms prove to be interesting points in the design space, as they not only represent drastically different solutions, but they also tend to be employed in drastically different scenarios (e.g., layering, administrative separation, internet-scale systems). For each scenario there is significant prior work (in varied contexts) that makes use of the same coordination mechanisms, leading us to simple design recipes for control plane coordination in these scenarios. Our work forms a case study across these very different scenarios, providing insight in how to build these coordination mechanisms efficiently, in practice.

While split control planes and how to coordinate between them is not new problem, we argue that in this thesis, we've taken the first steps towards providing guidelines to consider when designing systems, whereas designs today have been built ad hoc. Control plane coordination is likely going to become an increasingly apparent problem as more systems apply SDN-like techniques, eking out increased performance through long-timescale internet-scale centralized optimization. These internet-scale systems will need some form of coordination (i.e., hierarchical partitioning) to provide both high performance and responsiveness.

Bibliography

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

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

[3] Akamai. Financial reports, 2015. URL: http://www.ir.akamai.com/phoenix.zhtml?c=75943&p=irol-reportsannual.

[4] Akamai. Akamai, 2017. URL: https://www.akamai.com/.

[5] Akamai. Akamai reports first quarter 2017 financial results, May 2017. URL: https://seekingalpha.com/pr/16818558-akamai-reports-first-quarter-2017-financial-results.

[6] Akamai. Akamai technologies 2017 q1 earnings summary, May 2017. URL: https://seekingalpha.com/filing/3541289.

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

[8] Amazon. Amazon cloudfront, 2016. URL: https://aws.amazon.com/cloudfront/.

[9] Amazon. Amazon cloudfront pricing, 2016. URL: https://aws.amazon.com/cloudfront/pricing/.

[10] Apache. Hadoop, 2018. URL: https://hortonworks.com/apache/hadoop/.

[11] Apache. Hadoop HDFS, 2018. URL: https://hortonworks.com/apache/hdfs/.

[12] arstechnica. It's not a fast lane but comcast built a cdn to charge for video delivery, 2014. URL: http://arstechnica.com/information-technology/2014/05/its-not-a-fast-lane-but-comcast-built-a-cdn-to-charge-for-video-delivery/.

[13] Athula Balachandran, Vyas Sekar, Aditya Akella, and Srinivasan Seshan. Analyzing the potential benefits of cdn augmentation strategies for internet video workloads. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 43–56, New York, NY, USA, 2013. ACM.

[14] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. A quest for an internet video quality-of-experience metric. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 97–102, New York, NY, USA, 2012. ACM. URL: http://doi.acm.org/10.1145/2390231.2390248, doi:

10.1145/2390231.2390248.

[15] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *SIGCOMM '13: Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM Request Permissions, August 2013.

[16] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *Proc. ACM SIGCOMM*, pages 339–350. ACM, 2013.

[17] Hari Balakrishnan, Hariharan S Rahul, and Srinivasan Seshan. An integrated congestion management architecture for internet hosts. *ACM SIGCOMM*, 29(4):175–187, 1999.

[18] Andrew Bashore. Twitch stats. URL: http://stats.twitchapps.com/.

[19] Bizety. The multi-cdn strategy, 2014. URL: https://www.bizety.com/2014/05/09/multi-cdn-strategy/.

[20] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *ACM SIGMETRICS Performance Evaluation Review*. ACM, 2016.

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

[22] Eric W Burger and Jan Seedorf. Rfc 5693: Application-layer traffic optimization (alto) problem statement, 2009. URL: https://tools.ietf.org/html/rfc5693.

[23] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content delivery networks*, volume 9. Springer Science & Business Media, 2008.

[24] Ignacio Castro, Aurojit Panda, Barath Raghavan, Scott Shenker, and Sergey Gorinsky. Route bazaar: Automatic interdomain contract negotiation. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, Kartause Ittingen, Switzerland, 2015. USENIX Association. URL: https://www.usenix.org/conference/hotos15/workshop-program/presentation/castro.

[25] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *ACM SIGOPS Operating Systems Review*. ACM, 2003.

[26] Cedexis. Cedexis, 2017. URL: https://www.cedexis.com/.

[27] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 167–181, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2785956.2787500, doi:10.1145/2785956.2787500.

[28] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, and Xitao Wen. OSA: An Optical Switching Architecture for Data Center Networks and Unprecedented Flexibility. In *Proc. USENIX NSDI*, April 2012.

[29] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.

[30] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *SIGCOMM CCR*, 41(4):98, 2011.

[31] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM*. ACM, 2014.

[32] Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM computer communication review*, 31(4):55–67, 2001.

[33] Cisco. Cisco Report: The CDN Federation - Solutions for SPs and Content Providers To Scale a Great Customer Experience, 2012. URL: {http://www.cisco.com/c/dam/en_us/about/ac79/docs/sp/CDN-Federation_Phase-2-Pilot.pdf}.

[34] CloudFlare. Bandwidth costs around the world, 2016. URL: https://blog.cloudflare.com/bandwidth-costs-around-the-world/.

[35] CloudLab. CloudLab, 2018. URL: https://www.cloudlab.us/.

[36] Conviva, 2015. URL: {http://www.conviva.com}.

[37] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[38] Docker. Docker, 2018. URL: https://www.docker.com/.

[39] Docker. Get started with Macvlan network driver, 2018. URL: https://docs.docker.com/engine/userguide/networking/get-started-macvlan/.

[40] DPDK. DPDK, 2018. URL: https://dpdk.org.

[41] Emulab. APT cluster, 2018. URL: https://www.aptlab.net/.

[42] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *Proc. ACM SIGCOMM*, August 2010.

[43] Seyed Kaveh Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense. In *USENIX Security Symposium*, pages 817–832, 2015.

[44] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[45] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *Communications Magazine, IEEE*, 40(10):118–124, 2002.

[46] Apache Foundation. Apache HTTP Server Project. URL: http://httpd.apache.org/.

[47] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. Pushing cdn-isp collaboration to the limit. *SIGCOMM Comput. Commun. Rev.*, 43(3):34–44, July 2013.

[48] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. Pushing cdn-isp collaboration to the limit. *ACM SIGCOMM CCR*, 43(3), 2013.

[49] Benjamin Frank, Ingmar Poese, Georgios Smaragdakis, Anja Feldmann, Bruce M Maggs, Steve Uhlig, Vinay Aggarwal, and Fabian Schneider. Collaboration opportunities for content delivery and network infrastructures. *Recent Advances in Networking*, 1:305–377, 2013.

[50] Michael J Freedman. Experiences with coralcdn: A five-year operational view. In *Proc. USENIX NSDI*, 2010.

[51] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. C3: Internet-scale control plane for video quality optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 131–144, Oakland, CA, May 2015. USENIX Association. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/ganjam.

[52] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 216–229. ACM, 2016.

[53] Soudeh Ghorbani and Matthew Caesar. Walk the line: consistent network updates with bandwidth guarantees. In *Proc. HotSDN*, pages 67–72. ACM, 2012.

[54] Richard J Gibbens and Frank P Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35(12):1969–1985, 1999.

[55] Google. Google cloud cdn, 2016. URL: https://cloud.google.com/cdn/.

[56] Google. Adwords, 2017. URL: https://adwords.google.com/.

[57] Google. Doubleclick by google, 2017. URL: https://www.doubleclickbygoogle.com/.

[58] Diwaker Gupta, Kashi Venkatesh Vishwanath, and Amin Vahdat. Diecast: Testing distributed systems with an accurate scale model. In *Proc. USENIX NSDI*, 2008.

[59] Diwaker Gupta, Kenneth Yocum, Marvin McNett, Alex C Snoeren, Amin Vahdat, and Geoffrey M Voelker. To infinity and beyond: time warped network emulation. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 1–2. ACM, 2005.

[60] Gurobi. Gurobi Optimization, 2018. URL: http://www.gurobi.com/.

[61] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *Proc. ACM SIGCOMM*, August 2011.

[62] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 319–330, New York, NY, USA, 2014. ACM. URL: http://doi.acm.org/10.1145/2619239.2626328, doi:10.1145/2619239.2626328.

[63] Dongsu Han, David Andersen, Michael Kaminsky, Dina Papagiannaki, and Srinivasan Seshan. Hulu in the neighborhood. In *Proc. COMSNETS*, pages 1 –10, January 2011. doi:10.1109/COMSNETS.2011.5716501.

[64] Dongsu Han, Robert Grandl, Aditya Akella, and Srinivasan Seshan. Fcp: a flexible transport framework for accommodating diversity. *ACM SIGCOMM Computer Communication Review*, 43(4):135–146, 2013.

[65] HewlettPackard. netperf, 2018. URL: https://github.com/HewlettPackard/netperf.

[66] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proc. ACM SIGCOMM*, 2013.

[67] Cheng Huang, Angela Wang, Jin Li, and Keith W Ross. Measuring and evaluating large-scale cdns. In *Proc. ACM IMC*, 2008.

[68] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51. IEEE, 2010.

[69] Huawei. Huawei, 2017. URL: https://www.huawei.com/.

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

[71] IEEE. 802.1Qbb – Priority-based Flow Control, 2018. URL: https://1.ieee802.org/dcb/802-1qbb/.

[72] International Telecommunication Union. Introduction to CCITT Signalling System No. 7, 1988. URL: http://www.itu.int/rec/T-REC-Q.700/en.

[73] iperf. iperf, 2018. URL: https://iperf.fr/.

[74] iperf3. iperf3, 2018. URL: https://iperf.fr/.

[75] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *Proc. ACM SIGCOMM*, 2013.

[76] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 73–86. ACM, 2016.

[77] John Jannotti, David K Gifford, Kirk L Johnson, M Frans Kaashoek, et al. Overcast: reliable multicasting with on overlay network. In *Proc. 4th conference on Symposium on Operating System Design & Implementation*, 2000.

[78] Junchen Jiang, Xi Liu, Vyas Sekar, Ion Stoica, and Hui Zhang. EONA: Experience-Oriented Network Architecture. In *HotNets-XIII: Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM Request Permissions, October 2014.

[79] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. Cfa: A practical prediction system for video qoe optimization. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 137–150, Berkeley, CA, USA, 2016. USENIX Association. URL: http://dl.acm.org/citation.

cfm?id=2930611.2930621.

[80] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. ACM CoNEXT*, 2012.

[81] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *NSDI*, volume 1, page 3, 2017.

[82] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. Cooperative content distribution and traffic engineering. In *Proceedings of the 3rd International Workshop on Economics of Networked Systems*, NetEcon '08, pages 7–12, New York, NY, USA, 2008. ACM. URL: http://doi.acm.org/10.1145/1403027.1403030, doi:10.1145/1403027.1403030.

[83] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. Cooperative content distribution and traffic engineering in an isp network. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 239–250, New York, NY, USA, 2009. ACM. URL: http://doi.acm.org/10.1145/1555349.1555377, doi:10.1145/1555349.1555377.

[84] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways To De-Congest Data Center Networks. In *Proc. ACM HotNets-VIII*, October 2009.

[85] Naga Praveen Katta, Jennifer Rexford, and David Walker. Incremental consistent updates. In *Proc. HotSDN*. ACM, 2013.

[86] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[87] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 49–54. ACM, 2012.

[88] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[89] Leonidas Kontothanassis, Ramesh Sitaraman, Joel Wein, Duke Hong, Robert Kleinberg, Brian Mancuso, David Shaw, and Daniel Stodolsky. A transport layer for live streaming in a content delivery network. *Proceedings of the IEEE*, 92(9):1408–1419, 2004.

[90] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Operating Systems Review*. ACM, 2003.

[91] USC-Network Systems Lab. CDN Geographic Coverage, 2015. URL: {http://usc-nsl.github.io/cdn-coverage/}.

[92] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. URL: http://doi.acm.org/10.1145/279227.279229, doi:10.1145/279227.279229.

[93] Jereme Lamps, David M Nicol, and Matthew Caesar. Timekeeper: A lightweight virtual time system for linux. In *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 179–186. ACM, 2014.

[94] Level3 Communications, 2017. URL: http://www.level3.com/en/products/content-delivery-network/.

[95] Conglong Li, Matthew K Mukerjee, David G Andersen, Srinivasan Seshan, Michael Kaminsky, George Porter, and Alex C Snoeren. Using indirect routing to recover from network traffic scheduling estimation error. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, pages 13–24. IEEE Press, 2017.

[96] Philip A Lisiecki, Cosmos Nicolaou, and Kyle R Rose. Scalable, high performance and highly available distributed storage system for internet content, November 24 2009. US Patent 7,624,169.

[97] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit Switching Under the Radar with REACToR. In *Proc. USENIX NSDI*, April 2014.

[98] He Liu, Matthew K Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M Voelker, David G Andersen, Michael Kaminsky, et al. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, page 41. ACM, 2015.

[99] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. In *Proc. ACM SIGCOMM*, pages 359–370, 2012.

[100] Yong Liu, Honggang Zhang, Wenyu Gong, and Don Towsley. 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*, volume 4, pages 2543–2553. IEEE, 2005.

[101] K. Lougheed and Y. Rekhter. Border Gateway Protocol (BGP). RFC 1105 (Experimental), June 1989. Obsoleted by RFC 1163. URL: http://www.ietf.org/rfc/rfc1105.txt.

[102] Bruce M Maggs and Ramesh K Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[103] Ratul Mahajan, David Wetherall, and Thomas E Anderson. Mutually controlled routing with independent isps. In *NSDI*, 2007.

[104] MarketWatch. Akamai technologies profit falls, stock suffers, July 2015. URL: http://www.marketwatch.com/story/akamai-technologies-profit-falls-stock-suffers-2015-07-28.

[105] MaxCDN. Which cdn pricing model costs less, 2016. URL: https://www.maxcdn.com/blog/cdn-pricing-models/.

[106] Rick McGeer. A safe, efficient update protocol for openflow networks. In *Proc. HotSDN*, pages 61–66. ACM, 2012.

[107] Mellanox. sockperf, 2018. URL: https://github.com/Mellanox/sockperf.

[108] Mellanox. Spectrum Linux Switch, 2018. URL: http://www.mellanox.com/page/products_dyn?product_family=262&mtag=switchdev.

[109] William M Mellette, Alex C Snoeren, and George Porter. P-fattree: A multi-channel datacenter network topology. In *HotNets*, pages 78–84, 2016.

[110] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014. URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[111] Microsoft. Content delivery network (cdn) pricing, 2016. URL: https://azure.microsoft.com/en-us/pricing/details/cdn/.

[112] J Moy. Rfc 1247: Ospf version 2, 1991. URL: https://tools.ietf.org/html/rfc1247.

[113] J. Moy. OSPF Version 2. RFC 2328 (INTERNET STANDARD), April 1998. Updated by RFCs 5709, 6549, 6845, 6860, 7474. URL: http://www.ietf.org/rfc/rfc2328.txt.

[114] Matthew K Mukerjee, Ilker Nadi Bozkurt, Bruce Maggs, Srinivasan Seshan, and Hui Zhang. The impact of brokers on the future of content delivery. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 127–133. ACM, 2016.

[115] Matthew K Mukerjee, Ilker Nadi Bozkurt, Devdeep Ray, Bruce M Maggs, Srinivasan Seshan, and Hui Zhang. Redesigning cdn-broker interactions for improved content delivery. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '17)*, pages 68–80. ACM, 2017.

[116] Matthew K Mukerjee, Daehyeok Kim, and Srinivasan Seshan. Overcoming end-to-end challenges in reconfigurable datacenter networks. In *Submission to SIGCOMM '18*, 2018.

[117] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 311–324, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2785956.2787475, doi:10.1145/2785956.2787475.

[118] Netflix. Netflix open connect, 2016. URL: https://openconnect.netflix.com/en/.

[119] Limelight Networks. 2015 annual report, 2015. URL: http://investors.limelight.com/.

[120] Limelight Networks. 2017 first quarterly, 2017. URL: http://investors.limelight.com/.

[121] Ben Niven-Jenkins, Francois Le Fauchuer, and Nabil Bitar. RFC 6707: Content Distribution Network Interconnection (CDNI) Problem Statement, 2012. URL: {https://tools.ietf.org/html/rfc6707}.

[122] NPAW. Nicepeopleatwork (npaw), 2017. URL: https://www.nicepeopleatwork.com/.

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

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

[125] OpenX. Ad exchange vs. ad network: How do they compare?, 2017. URL: http://openx.com/whitepaper/ad-exchange-vs-ad-network-how-do-they-compare/.

[126] P4 Language Consortium. P4, 2018. URL: https://p4.org/.

[127] PeeringDB. Peeringdb, 2017. URL: https://www.peeringdb.com/.

[128] Ingmar Poese, Benjamin Frank, Simon Knight, Niklas Semmler, and Georgios Smaragdakis. PaDIS emulator: an emulator to evaluate CDN-ISP collaboration. In *SIGCOMM '12: Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, August 2012.

[129] Ingmar Poese, Benjamin Frank, Georgios Smaragdakis, Steve Uhlig, Anja Feldmann, and Bruce Maggs. Enabling content-aware traffic engineering. *SIGCOMM Computer Communication Review*, 42(5), September 2012.

[130] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang-Chen Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating Microsecond Circuit Switching into the Data Center. In *Proc. ACM SIGCOMM*, August 2013.

[131] Ravi Prasad, Constantinos Dovrolis, Margaret Murray, and KC Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 17(6):27–35, 2003.

[132] Zafar Ayyub Qazi, Phani Krishna Penumarthi, Vyas Sekar, Vijay Gopalakrishnan, Kaustubh Joshi, and Samir R Das. Klein: A minimally disruptive design for an elastic cellular core. In *Proceedings of the Symposium on SDN Research*, page 2. ACM, 2016.

[133] K Ramakrishnan, S Floyd, and D Black. Rfc 3168: The addition of explicit congestion notification (ecn) to ip, 2001. URL: https://tools.ietf.org/html/rfc3168.

[134] Dan Rayburn. Current state of the cdn market: Diy, pricing trends, competitive dynamics, 2016. URL: http://blog.streamingmedia.com/wp-content/uploads/2016/05/2016CDNSummit-Rayburn-Pricing.pdf.

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

[136] Yang Song, Arun Venkataramani, and Lixin Gao. On the cdn pricing game. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 339–344. IEEE, 2013.

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

[138] Rade Stanojevic, Ignacio Castro, and Sergey Gorinsky. Cipt: Using tuangou to reduce ip transit costs. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 17:1–17:12, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/2079296.2079313, doi:10.1145/2079296.2079313.

[139] Volker Stocker, Georgios Smaragdakis, William Lehr, and Steven Bauer. The growing complexity of content delivery networks: Challenges and implications for the internet ecosystem. *Telecommunications Policy*, 2017.

[140] Thomas Stockhammer. Dynamic adaptive streaming over http–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[141] Brett Stone-Gross, Ryan Stevens, Apostolis Zarras, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. Understanding fraudulent activities in online ad exchanges. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 279–294. ACM, 2011.

[142] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, pages 39–44, New York, NY, USA, 2003. ACM. URL: http://doi.acm.org/10.1145/948205.948211, doi:10.1145/948205.948211.

[143] Ao-Jan Su and Aleksandar Kuzmanovic. Thinning akamai. In *Proc. ACM IMC*, 2008.

[144] Trefis Team. How twitch fits in amazon's strategy. URL: http://www.forbes.com/sites/greatspeculations/2014/08/28/how-twitch-fits-in-amazons-strategy/.

[145] Twitch. URL: http://twitch.tv.

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

[147] K Varadhan. Rfc 1403: Bgp ospf interaction, 1993. URL: https://tools.ietf.org/html/rfc1403.

[148] Kashi Venkatesh Vishwanath, Diwaker Gupta, Amin Vahdat, and Ken Yocum. Modelnet: Towards a datacenter emulation environment. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 81–82. IEEE, 2009.

[149] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *ACM SIGCOMM*. ACM, 2015.

[150] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: Part-time Optics in Data Centers. In *Proc. ACM SIGCOMM*, August 2010.

[151] Guohui Wang, TS Ng, and Anees Shaikh. Programming your network at run-time for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 103–108. ACM, 2012.

[152] Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek S Pai, and Larry L Peterson. Reliability and security in the codeen content distribution network. In *Proc. USENIX ATC, General Track*, 2004.

[153] Q. Wang, K. Ren, and X. Meng. When cloud meets ebay: Towards effective pricing for cloud computing. In *2012 Proceedings IEEE INFOCOM*, pages 936–944, March 2012. doi:10.1109/INFCOM.2012.6195844.

[154] Yiting Xia, TS Eugene Ng, and Xiaoye Steven Sun. Blast: Accelerating high-performance data analytics applications by optical multicast. In *Proc. IEEE INFOCOM*, pages 1930–1938. IEEE, 2015.

[155] Haiyong Xie, Y Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4p: Provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362, 2008.

[156] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, and Heung keung Chai. A cdn-p2p hybrid architecture for cost-effective streaming media distribution. *Computer Networks*, 44:353–382, 2004.

[157] Jiaqi Yan and Dong Jin. A virtual time system for linux-container-based emulation of software-defined networks. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 235–246. ACM, 2015.

[158] Jiaqi Yan and Dong Jin. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 27. ACM, 2015.

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

[160] Minlan Yu, Wenjie Jiang, Haoyuan Li, and Ion Stoica. Tradeoffs in cdn designs for throughput oriented traffic. In *Proc. ACM CoNEXT*, pages 145–156. ACM, 2012.

[161] Shuai Yuan, Jun Wang, and Xiaoxue Zhao. Real-time bidding for online advertising: measurement and analysis. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*, page 3. ACM, 2013.

[162] H. Zhang, H. Jiang, B. Li, F. Liu, A. V. Vasilakos, and J. Liu. A framework for truthful online auctions in cloud computing with heterogeneous user demands. *IEEE Transactions on Computers*, 65(3):805–818, March 2016. doi:10.1109/TC.2015.2435784.

[163] Mingchen Zhao, Paarijaat Aditya, Ang Chen, Yin Lin, Andreas Haeberlen, Peter Druschel, Bruce Maggs, Bill Wishon, and Miroslav Ponec. Peer-assisted content distribution in akamai netsession. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 31–42. ACM, 2013.

[164] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao, and Haitao Zheng. Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers. In *Proc. ACM SIGCOMM*, August 2012.

[165] Alexander Zimmermann, Arnd Hannemann, and Tim Kosse. Flowgrind-a new performance measurement tool. In *IEEE GLOBECOM*, pages 1–6. IEEE, 2010.

[166] Alexander Zimmermann, Arnd Hannemann, and Tim Kosse. flowgrind, 2018. URL: http://www.flowgrind.net/.