Refactoring Network Control and Management: A Case for the 4D Architecture

Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers,
Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, Hui Zhang
{dmaltz,acm,yh,jibin,hzhang}@cs.cmu.edu
gisli@ru.is jrex@cs.princeton.edu albert@research.att.com xie@nps.edu

February, 2005
CMU-CS-05-???

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was sponsored by the NSF under ITR Awards ANI-0085920, ANI-0331653, and ANI-0114014. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of AT&T, NSF, or the U.S. government.

Keywords: Network control plane, centralized versus distributed control, decision plane, dissemination plane, routing design, routing protocols

Abstract

We argue for the refactoring of the IP control plane to support network-wide objectives and control. We put forward a design that refactors functionality into a novel 4D architecture composed of four separate planes: decision, dissemination, discovery and data. All decision-making logic is moved out of routers along with current management plane functions to create a logically centralized decision plane, where network-level objectives and policies are specified and enforced by direct configuration of states on individual network elements. Pulling much of the control state and logic out of the routers enables both simpler protocols, which do not have to embed decision-making logic, and more powerful decision algorithms for implementing sophisticated goals. Remaining on the routers is a wafer-thin class of intrinsically distributed control functions. These support the discovery plane, consisting of elementary functions to discover topology and network state, and the dissemination plane, consisting of elementary functions to distribute explicit instructions to manipulate the data plane forwarding mechanisms.

This paper motivates the need for a new approach to network control and management, describes the 4D architecture, and sketches the design space and challenges posed by the architecture. As a first exploration of the design space and its challengs, we have constructed a working prototype that implements the 4D architecture. Through evaluation of this prototype on 9 different network topologies derived from production networks, we demonstrate that (i) the 4D architecture can achieve *subsecond* reconvergence times upon single link or router failures and can adequately deal with other failure scenarios including *network partition*; (ii) the 4D architecture is able to implement a network design intent such as a reachability matrix more robustly than currently possible; and (iii) the 4D architecture does not introduce excess overhead.

Contents

1 Introduction

Most data networks were designed to provide best-effort communication among cooperating users. This has resulted in a box-centric architecture in which each switch participates in a distributed control protocol that *embeds* the path computation logic. For example, in IP networks, the path computation logic is governed by distributed protocols such as Dijkstra's SPF or Bellman Ford. In Ethernet networks, the path computation logic is in the Spanning Tree protocol [1]. Today's data networks, controlled by numerous institutions and operated in diverse environments (e.g., data center, access, enterprise, and service provider), must support network-level objectives and capabilities far more sophisticated than best-effort packet delivery, including traffic engineering, survivability, security, and policy enforcement. Retro-fitting these network objectives on the box-centric architecture has led to bewildering complexity, with diverse state and logic distributed across numerous network elements and management systems. This complexity is responsible for the increasing fragility of IP networks and the tremendous difficulties facing people trying to understand and manage their networks.

Continuing on the path of incremental evolution would lead to additional point solutions that exacerbate the problem. Instead, we advocate re-architecting the control and management functions of data networks from the ground up. We believe that a clean-slate approach based on sound principles will, at the minimum, provide an alternative perspective and shed light on fundamental trade-offs in the design of network control and management functions. More strongly, we believe that such an approach is *necessary* to avoid perpetuating the substantial complexity of today's control plane.

To guide our design, we start from a small set of principles: *network-level objectives*, *network-wide views*, and *direct control*. These principles lead us to a 4D architecture that refactors network functionality into four components—the *data, discovery, dissemination, and decision planes*. The decision plane creates a network configuration that satisfies network-level objectives. It has a network-wide view of the topology and events, and it has direct control over the operation of the data plane. No decision logic is hardwired in protocols distributed among switches. The output of the decision logic is communicated to switches by the dissemination plane. Our study investigates an *extreme design point* where control and management decisions are made in a logically centralized fashion. By pulling all the decision logic out of the network elements, we enable both simpler protocols and more sophisticated algorithms for driving the operation of the data plane. In addition, we believe that the technology trends toward ever-more powerful and inexpensive computing platforms make our design point attractive in practice.

Despite the many potential advantages of the 4D architecture, key questions remain: **Is the 4D architecture *feasible?*** Centralizing the decision-making logic at servers raises natural concerns about responsiveness, scalability, and reliability [2, 3]. **Are the advantages of the 4D architecture *realizable?*** As discussed in Section 6, our architecture introduces new capabilities, such as network-wide reachability policies, and zero device-specific configuration of routers/switches. Can these capabilities be realized without introducing significant complexity?

To answer these questions, we have implemented a prototype of the 4D architecture. We chose to target the first 4D prototype at controlling IPv4 networks, and we base our evaluations on topologies taken from production enterprise networks that vary in size from 10 to 100 routers. However, we believe that our architecture is applicable to a wide range of networks: (i) networks with different packet-forwarding paradigms, such as longest-prefix-matching forwarding (IPv4 and IPv6), exact-match forwarding (Ethernet), and label switching (MPLS and VLAN/Ethernet) and (ii) networks in different environments, including data centers, enterprise networks, access/metro networks, and service provider backbones. Using our current prototype, we demonstrate that it is feasible to give IP networks more sophisticated capabilities while making them fully self-configuring, comparable to switched Ethernet networks. Further, we believe that applying the 4D architecture to Ethernet will enable Ethernet networks to retain their simplicity while obtaining the scalability and functionality seen in IP networks[4].

The contributions of this paper are threefold. First, we synthesize fundamental design principles capturing the essence of the problem of network control and management. We present a 4D network architecture founded on these principles, and we articulate why the 4D architecture potentially has major advantages over existing designs. Second, through evaluation of responsiveness and overhead of our prototype, we demonstrate the viability of the 4D architecture and specifically validate the viability of the novel refactoring and relocating of control functions. Third, we demonstrate new and valuable functionality facilitated by the 4D architecture that is currently unattainable in traditional networks.

2 Network Control and Management Today

In today's data networks, the functionality that controls the network is split into three main planes: (i) the *data plane* that handles the individual data packets; (ii) the *control plane* that implements the distributed routing algorithms across the network elements; and (iii) the *management plane* that monitors the network and configures the data-plane mechanisms and control-plane protocols.

While the original IP control plane was designed to have a *single* distributed algorithm to maintain the *forwarding* table in the data plane, today's IP data, control and management planes are far more complex. The data plane needs to implement, in additional to next-hop forwarding, more functions such as tunneling, access control, address translation, and queueing. The states used to implement these functions are governed by multiple entities and have to be configured through a rich set of individual, interacting commands. Even for the forwarding state, there are usually multiple routing processes running on the same switch. While there are many dependencies among the states and the logic updating the states, most of the dependencies are *not* maintained automatically. For example, controlling routing and reachability today requires complex arrangements of commands to tag routes, filter routes, and configure multiple interacting routing processes, all the while ensuring that no router is asked to handle more routes and packet filters than it has resources with witch to cope. A change to any one part of the configuration can easily break other parts. The problem is exacerbated as packet delivery cannot commence until the routing protocols create the necessary forwarding tables, and the management plane cannot reach the control plane until the routing protocols are configured. Resolving this catch-22 requires installing a significant amount of configuration information on IP routers before deployment.[1] Studies of production networks show them requiring hundreds of thousands of lines of low-level configuration commands distributed across all the routers in the network [5]. These configurations and the dynamic forwarding state they generate require a myriad of ad hoc scripts and systems in the management plane to validate, monitor, and update them. The result is a complex and failure-prone network.

We present two examples that illustrate the network fragility caused by today's complex and unwieldy control and management infrastructure. The examples illustrate how the lack of coordination between routing and security mechanisms can result in a fragile network, and how today's control and management infrastructure makes it difficult to properly coordinate the mechanisms.

2.1 Reachability Control in Enterprise Networks

Today, many enterprise networks attempt to control which hosts and services on their network can communicate (i.e., reach each other) as part of their security strategy [5]. They implement their strategies using a combination of routing policy and packet filters, but this approach is fraught with peril even in simple networks.

Consider the example enterprise network in Figure 1. The company has two locations, A and B. Each location has a number of "front office" computers used by the sales agents (AF1-2 and BF1-2). Each

[1]This problem is so profound in today's networks that whenever possible remote routers/switches are plugged into telephone modems so that the Public Switched Telephone Network provides a management communication path of last resort.

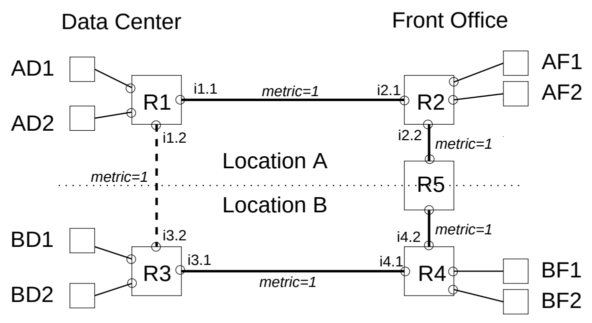

Figure 1: Enterprise network with two locations, each location with a front office and a data-center.

location also has a data center where servers are kept (AD1-2 and BD1-2). Initially, the two locations are connected by a link between the front office routers, R2 and R4, over which inter-office communications flow. The Interior Gateway Protocol (IGP) metric for each link is shown in italics. The company's security policy is for front-office computers to be able to communicate with other locations' front office computers and the local data center's servers, but not the data center of the other location. Such policies are common in industries like insurance, where the sales agents of each location are effectively competing against each other even though they work for the same company. The security policy is implemented using packet filters on the routers controlling entrance to the data centers to drop packets that violate the policy. Interface i1.1 is configured with a packet filter that drops all packets from the BF subnet, and interface i3.1 drops all packets from the AF subnet.

The network functions as desired, until the day when the data-center staff decides to add a new, high-capacity dedicated link between the data centers (shown as a dashed line between R1 and R3 — perhaps they have decided to use each other as remote backup locations). It seems reasonable that with packet filters protecting the entrances to the data centers, the new link between data centers should not compromise the security policy. However, the new link changes the routing such that packets sent from AF to BD will travel from R2 to R1 to R3 to BD — completely avoiding the packet filter installed on interface i3.1 and violating the security policy. When the designers eventually discover the security hole, probably due to an attack exploiting the hole, they would typically respond by copying the packet filter from i3.1 to i3.2, so it now also drops packets from AF. This filter design does plug the security hole, but it means that if the front office link from R2 to R4 fails, AF will be unable to reach BF. Even though the links from R2 to R1 to R3 to R4 are all working, the packet filter on interface i3.2 will drop the packets from subnet AF.

In this example, the problems arise because the ability of a network to carry packets depends on the routing protocols and the packet filters working in concert. While routing automatically adapts to topology changes, there is no corresponding way to automatically adapt packet filters or other state. It could be argued that a more "optimal" placement of packet filters, or the use of multi-dimensional packet filters (i.e., filters that test both source and destination address of a packet) would fix the problems shown in this example. However, as networks grow in size and complexity from the trivial example used here for illustrative purposes, finding these more optimal placements and maintaining the multitude of multi-dimensional packet filters they generate requires developing and integrating entirely new sets of tools into the network's management systems. Since these tools will be separate from the protocols that control routing in real time, they will perpetually be attempting to remain synchronized with routing protocols by trying to model and guess the protocols' behavior.

In contrast, the 4D architecture simply and directly eliminates this entire class of problems. The 4D architecture allows the direct specification of the desired "reachability matrix" and automated mechanisms for simultaneously setting both the forwarding-table entries and packet filters on the routers based on the

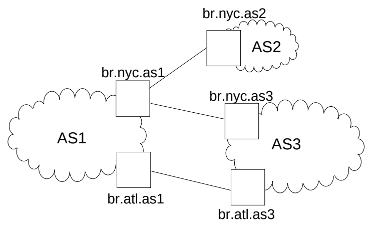

Figure 2: Autonomous Systems (ASs) peering with each other via eBGP sessions. AS1 must place packet filters on its ingress links to prevent AS3 from sending packets to destinations for which AS1 has not agreed to provide transit.

current network state.

2.2 Enforcing Peering Policies in Transit Networks

Routing policy is based on the premise that a router that does not announce a route to a destination to a peer will not be sent packets for that destination by that peer. However, the routing system does nothing to prevent an unscrupulous peer from sending packets to that destination anyway. Enforcing routing policy is nearly impossible with today's control and management planes.

Figure 2 shows an example of three Autonomous Systems (ASs) peering with each other via three eBGP sessions (one eBGP session along each of the links shown in the figure). Assume that AS1 is a major transit network, and it announces a route to destination d in its eBGP session with AS2. If AS1's policy is to not provide AS3 with transit service for d, it does not announce d in its eBGP sessions with AS3. However, if AS3 wishes to be unscrupulous (e.g., use AS1 for transit service without paying), it can assume AS1 does know a way to d (e.g., so AS1's own customers can reach d). If AS3 sends packets for d to br.nyc.as1, they will definitely be delivered, as br.nyc.as1 must have a route to d in order to handle legitimate traffic from AS2.

Enforcing routing policy requires installing packet filters to drop packets to destinations which have not been announced as reachable. As the announcements received by an AS, and the AS's own topology, change over time the announcements sent by the AS will change and the packet filters must be moved correspondingly. Implementing such functionality by adding another ad hoc script to the management plane is essentially impossible today. Even if it were possible to write a script that snoops on the eBGP announcements sent to each neighboring border router and installs packet filters on the ingress interface as appropriate, the script would be extremely dangerous as it would not properly order the packet filter installation/removal with the BGP announcements. For example, it would be bad to announce to a neighbor border router a route to a destination before removing the packet filters that drop the packets sent to the destination.

Beyond ordering issues, transit networks handle a large number of destinations, and each packet filter applied to an interface consumes forwarding resources and reduces the effective capacity of the interface. It might be desirable to move packet filters into the network whenever possible, away from the ingress interfaces, so that one packet filter can enforce the BGP policy for multiple ingress interfaces.

Enforcing routing policy requires dynamically placing packet filters to respond to the continually changing routes selected by that policy. Correctly and optimally placing the filters requires that the placement be synchronized with the announcement of routing decisions and that the placement algorithms have access to the complete routing topology of the network. The 4D architecture provides the primitives and abstractions needed to implement correct placement strategies and support placement optimization algorithms.

2.3 Same Problems, Many Guises

The global Internet is composed of many separate data networks, designed and managed by different people and organizations with different goals. Individual networks serve radically different purposes; in addition to the familiar backbone networks, there are access, metro, enterprise and data-center networks. In each of these settings, the network administrators struggle to "program" their networks, integrating a diverse set of technologies and protocols, and artfully setting the configurable parameters that determine the network's functionality and dynamics.

While the specific context, technology, and mechanisms may change from network to network, there is commonality among the problems. For example, while Ethernet was initially designed to run on a shared medium, it has since evolved into a networking technology with a full package of data plane, control plane, and management plane to rival IP. Just as IP has many routing protocols to compute FIB state, Ethernet has many variations of the spanning tree protocol [6]. Just as IP networks have mechanisms like MPLS to control the paths packets take, Ethernet has VLANs (and VLANs-in-VLANs). Just as IP networks have needed to implement sophisticated functionality like traffic engineering, security policies and fast restoration, these same needs are being required of Ethernet in many contexts, such as enterprises, data centers [7], and metro/access networks [8]. Just as ad hoc management capabilities need to be overlaid on top of IP control plane, achieving advanced functionality in Ethernet networks has led to increasingly ad hoc and complex management systems. The systems must operate outside Ethernet's control plane, often coming into conflict with it.

We argue the key to solving the problems illustrated above is creating a way for the architectural intent and operational constraints governing the network to be expressed directly, and then automatically enforced by configuring all the data-plane mechanisms on the individual routers/switches. Until this occurs, we expect the design and operation of robust networks to remain a difficult challenge, and the state of the art to remain a losing battle against a trend where ever richer and more complex state and logic are embedded in distributed protocols or exposed through box-level interfaces.

3 The 4D Architecture

Rather than exploring incremental extensions to today's control and management planes, we propose a *clean-slate* redesign of the division of functionality. We believe that a green-field approach based on sound principles is necessary to avoid perpetuating the substantial complexity in today's design. We have developed the 4D architecture as an *extreme design point* that moves all responsibility for network control and management to a logically-centralized decision plane. We deliberately chose an extreme design as we believe that it crystallizes the issues, so that exploring the strengths and weaknesses of this architecture will lead to important network-level abstractions and a deeper understanding of the essential functionality needed in the individual routers and switches. In this section, we propose our vision of the desired architecture — in later sections we explain its implementation.

3.1 Design Principles

The rich literature on the complexity of today's control and management planes has led us to the following three principles that we believe are essential to dividing the responsibility for controlling and managing a data network:

Network-level objectives: Running a robust data network depends on satisfying objectives for performance, reliability, and policy that can (and should) be expressed separately from the low-level network elements. For example, a traffic-engineering objective could be stated as "keep all links below 70% utilization, even under single-link failures." A reachability policy objective could be stated as "do not allow

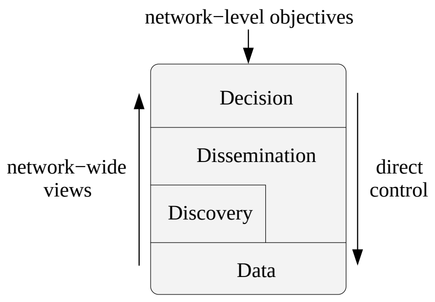

Figure 3: New 4D architecture with network-level objectives, network-wide views, and direct control

hosts in subnet B to access the accounting servers in subnet A.” Today’s network requires these goals to be expressed in low-level configuration commands on the individual routers, increasing the likelihood that the objectives are violated due to semantic mistakes in translating the network-level objectives into specific protocols and mechanisms.

Network-wide views: Timely, accurate, network-wide views of topology, traffic, and events are crucial for running a robust network. The network-wide view must accurately reflect the current state of the data plane, and include accurate information about each device, including its name, resource limitations, and physical attributes. However, today’s control plane was *not* designed to provide these network-wide views, forcing substantial retro-fitting to obtain them. Instead of adding measurement support to the system as an afterthought, we believe providing the information necessary to construct a complete, consistent, network-wide view should be one of the primary functions of the routers and switches.

Direct control: Satisfying network-level objectives is much easier with direct control over the configuration of the data plane. The decision logic should not be hardwired in protocols distributed among switches. Rather, only the output of the decision logic should be communicated to switches. Early data networks embed the path computation logic in simple distributed protocols. For example, in IP networks, the path computation logic is governed by distributed protocols such as Dijkstra’s SPF or Bellman Ford. In Ethernet networks, the path computation logic is determined by the Spanning Tree protocol[1]. Since it is difficult to extend the distributed control protocols to support more sophisticated network-level objectives such as traffic engineering or reachability, the management plane is used to implement these additional capabilities. With only indirect influence over the network, today’s management plane must replicate the state and logic of the control plane and perform a complex inversion of the functionality. The problem would be much easier to solve if the management plane could compute the forwarding tables and install them in the routers. For direct control to be meaningful, it must be complete. If configuration commands or multiple entities can affect the state in the network elements, then yet more entities are required for auditing (and correcting) the settings [9, 10, 11] to ensure the network-level objectives are upheld.

In addition to these three principles, any design must also consider traditional systems requirements, such as scalability, reliability, and consistency. Our three principles attempt to capture the issues specific to the control and management of networks. By separating the network-specific issues from the traditional systems requirements, we can apply existing techniques from other areas of distributed computing research to the traditional systems problems while exposing for closer scrutiny the network-specific ones.

3.2 New 4D Network Architecture

Although the three principles could be satisfied in many ways, we have deliberately made the 4D architecture an extreme design point where all control and management decisions are made in a logically centralized fashion by servers that have complete control over the network elements. The routers and switches only have the ability to run network discovery protocols and accept explicit instructions that control the behavior of the data plane, resulting in network devices that are auto-configurable. Our proposed network architecture has the following four components, as illustrated in Figure 3:

Decision plane: The decision plane makes *all* decisions driving network-wide control, including reachability, load balancing, access control, security, and interface configuration. Replacing today's management plane, the decision plane operates in *real time* on a network-wide view of the topology, the traffic, and the capabilities and resource limitations of the routers. The decision plane uses algorithms to turn network-level objectives (e.g., reachability matrix, load-balancing goals, survivability requirements, planned maintenance events) directly into the packet-handling state that must be configured into the data plane (e.g., forwarding table entries, packet filters, queueing parameters). The decision plane consists of multiple servers called decision elements that connect directly to the network.

Dissemination plane: The dissemination plane provides a robust and efficient communication substrate that connects routers and switches with decision elements. While the dissemination plane may use the same set of physical links as the data packets, the dissemination paths are maintained separately from the data paths so they can be operational without requiring configuration or successful establishment of routes or paths in the data-plane. In contrast, in today's networks, control and management data are carried over the data paths, which need to be established by routing protocols before they can be used. The dissemination plane moves management information created by the decision plane to the data plane and state identified by the discovery plane to the decision plane, but does not create state itself.

Discovery plane: The discovery plane is responsible for discovering what physical entities make up the network and creating logical identities to represent those entities. The discovery plane defines the scope and persistence of the identities, and carries out the automatic discovery and management of the relationships between them. This includes box-level discovery (e.g., what interfaces are on this router? How many FIB entries can it hold?), neighbor discovery (e.g., what other routers does this interface connect to?), and discovery of lower-layer link characteristics (e.g., what is the interface's capacity?). The network-wide views used by the decision plane are formed from the information learned by the discovery plane. In contrast, in today's IP networks, the only automatic mechanism is neighbor discovery between two preconfigured and adjacent IP interfaces; physical device discovery and associations between entities are driven by configuration commands and external inventory databases.

Data plane: The data plane handles individual packets based on the state that is *output* by the decision plane. This state includes the forwarding table, packet filters, link-scheduling weights, and queue management parameters, as well as tunnels and network address translation mappings. The data plane may also have fine-grain support for collecting measurements [12] on behalf of the discovery plane.

The 4D architecture embodies our three principles. The decision-plane logic operates on a network-wide view of the topology and traffic, with the help of the dissemination plane in collecting the measurement data, to satisfy network-level objectives. The decision plane has direct control over the operation of the data plane, obviating the need to model and invert the actions of the control plane. Pulling much of the control state and logic out of the routers enables both simpler protocols, which do not have to embed decision-making logic, and more powerful decision algorithms for implementing sophisticated goals.

3.3 Advantages of the 4D Architecture

Our 4D architecture offers several important advantages over today's division of functionality:

Higher robustness: By simplifying the state and logic for network control, and ensuring the internal consistency of the state, our architecture greatly reduces the fragility of the network. The 4D architecture raises the level of abstraction for managing the network, allowing network administrators to focus on specifying network-level objectives rather than configuring specific protocols and mechanisms on individual routers and switches. Network-wide views provide a conceptually-appealing way for people and systems to reason about the network without regard for complex protocol interactions among a group of routers. Moving the state and logic out of the routers also facilitates the creation of new, more sophisticated algorithms for computing the data-plane state that are easier to maintain and extend.

Better security: Security objectives are inherently network-level goals. For example, the decision plane can secure the network perimeter by installing packet filters on all border routers. Managing network-level objectives, rather than the configuration of individual routers, reduces the likelihood of configuration mistakes that can compromise security. The security of a 4D system depends primarily on securing the dissemination plane that forms the communication channels between the routers/switches and the decision plane, and securing the decision plane itself. Reachability to the decision elements can be restricted to the routers/switches inside the network, reducing the likelihood of a compromise or attack on the decision plane.

Accommodating heterogeneity: The same 4D architecture can be applied to different networking environments but with customized solutions. For example, in an ISP backbone with many optimization criteria and high reliability requirements, the decision plane may consist of several high-end servers deployed in geographically distributed locations. A data-center environment with Ethernet switches may require only a few inexpensive PCs, and still achieve far more sophisticated capabilities (e.g., traffic engineering with resilience) than what spanning tree or static VLAN configuration can provide today.

Enabling of innovation: Separating the network control from the routers and protocols is a significant enabler for innovation. The decision plane can incorporate new algorithms and abstractions for computing the data-plane state to satisfy a variety of network-level objectives. In addition, moving the control functionality out of the router software enables new players (e.g., the research community and third-party software developers) to contribute to the creation of these algorithms. The 4D architecture provides mechanisms in which ideas such as "regions" from the NewArch Project's New Internet Architecture [13] can be realized.

4 Design and Implementation

This section discusses a number of the interesting design questions that arise in turning the principles and planes of the 4D architecture into a working prototype that controls an actual network. Our approach in creating the prototype was to choose simplicity over performance, taking natural design decisions to their logical conclusions. Section 5 shows that even the simplistic mechanisms we have chosen result in a prototype with very reasonable performance.

4.1 Target Platform

Unlike other control/management architectures that are tied to specific data planes, the 4D architecture could be used to control a variety of different networks. For instance, the 4D architecture could be used as the control plane for an Ethernet-based network (replacing the spanning tree protocol, bridge learning algorithms, and VLAN management) , or a router-based IP network (where it replaces the control and management planes).

For the evaluation of the viability of the 4D architecture in this paper we have chosen to prototype the control of a routed IPv4 network. Our choice is partially motivated by the potential impact our design could have on routed networks, but also by the significant number of tools available for profiling and investigating such networks (e.g. traceroute, ping) allowing us to validate the functionality of our prototype. For ease of programming, we use x86 based PCs running a standard Linux kernel as our routers. The code for our prototype runs in user-space, manipulating the FIB entries and packet filters in the kernel tables when commanded to do so by the remote decision elements.

Successful deployment of the 4D architecture is transparent to end-systems; all the end-to-end tools will continue to function as normal. Hosts connected to the prototype network are unmodified. The decision elements, where all decisions with network-wide implications are made, are servers connected to the network at various points. Every router in the network runs code implementing the data plane, the discovery plane, and dissemination plane. Every decision element runs code implementing the discovery, dissemination and decision plane.

4.2 Neighbor Discovery

The problem of neighbor discovery is not unique to our design, but it does impact the time required to respond to link failures. Because of its importance, some link layers (e.g., SONET) provide extremely rapid link liveness checking as a fundamental primitive to support neighbor discovery and maintenance. However, many others link layers (e.g., Ethernet) do not.[2] In environments where link-layer neighbor discovery is not available, it is commonly implemented with the explicit exchange of HELLO packets. Such a hello based protocol has been shown to be viable for supporting sub-millisecond restoration times [14] . In our prototype, which is implemented using Ethernet interfaces, the discovery plane code on each router sends 28-byte long HELLO messages out every interface every 30 ms. Failure to receive a HELLO from neighbor for 100 ms is treated as indicating that the link or neighbor has failed. Each router uses the results of neighbor discovery to create a Link-state Advertisement (LSA) that contains the status of its interfaces and the identity of the router(s) connected to the interfaces. Each DE periodically polls the routers, requesting that the routers send their current LSAs to the DE. Whenever a router detects a change in a neighbor's status, it sends a triggered LSA to every DE that has recently polled it.

4.3 Communication Between DEs and Routers

The 4D architecture physically separates the decision plane logic that controls the network from the routers that comprise the network. The dissemination plane provides means for the decision elements to communicate with each and every router. The problem is unusual in several ways. First, the dissemination paths must be able to boot strap themselves, ideally requiring zero prior configuration on the routers. Second, the dissemination plane will be used to carry a wide variety of information. Some information, like LSAs, might need to be distributed to multiple places (e.g., all of the decision elements), while other information, like commands from a DE to a router, cannot be silently lost. On the one hand, the potential design space is very large, as there is tremendous freedom in designing the dissemination plane; on the other, the dissemination plane's sole function is to distribute state (data and instruction) and can therefore employ standard distributed systems techniques.

One option for the dissemination plane would be to use normal IPv4 routing tables to direct management information along the dissemination paths. Another option would be to have routers run a spanning tree protocol among themselves, with decision elements serving as roots of the trees. However, spanning tree protocols have been reported to have slow reconvergence [4]. For our prototype, we have chosen to implement the dissemination plane using source routes. Each management message is segmented into dissemination frames, and each frame carries in its header the identity of the source, destination, and the series of routers/switches/DEs through which it must pass.

Decision element beacons: Our design leverages the fact that there are fewer decision elements than routers to simplify the design of the dissemination plane. Each decision element sends a beacon identifying itself as a decision element every 500 ms. The beacons are flooded throughout the network in an orderly fashion using the Dynamic Source Routing Protocol [15]. We use a single beacon packet to serve three

[2]Some modern Ethernet interfaces provide proprietary neighbor detection primitives, but we do not have these available to us.

separable purposes: setting up source routes in the network, polling LSAs from the routers, and serving as DE heartbeats. When a router or decision element receives a beacon, it appends its identity to a source route that is built up inside the beacon and then forwards a copy of the beacon out each interface except the one which it has received the beacon from.

Creating dissemination paths and collecting topology information: Whenever a router receives a beacon or a message from a DE update), it remembers the source route contained in the beacon/message for use as a path to reach DE that sent the beacon/message (i.e., routers store one route to each DE). Receipt of a beacon also causes each router to send its current LSA back to the DE that originated the beacon. When a router has information it must send to a DE (e.g., a triggered LSA) it uses the source route from the last message from the DE. When a DE has information to send to a router, it uses the LSAs it has received to assemble a view of the network topology, and can then compute a source route to the router.

Coping with lost management messages: It is always possible for congestion to cause management messages to be lost.[3] To avoid the silent loss of a state update, when a router receives a message from a DE it sends the DE an acknowledgement. The prototype's dissemination plane retransmits state updates that have not been acknowledged after a fixed timeout (set at 250 ms). Our prototype dos not send ACKs for LSAs or retransmit lost LSAs.

A more interesting problem arises when a link or router fails. The router detecting the failure notifies the decision elements by immediately sending a triggered LSA with its new neighbor information. In some cases the route to a decision element is across the failed link. However, most often (always for link failure) at least one router adjacent to a failed link or router will have a working source route to each DE, so any failure generally results in a least one triggered LSA reaching each DE. For the routers whose path to the DE is broken, we adopt the simplest possible approach, namely to simply wait for the next DE beacon — when the next beacon floods the network, the router will learn a new path to the DE. We reduce the likelihood of a long latency for flushing out any "stranded" LSAs by having the DE send a "triggered" beacon whenever it receives a triggered LSA. To bound overhead, triggered beacons are rate limited to one per 20 ms.

4.4 Computing FIB Entries and Packet Filters

The 4D architecture places upon the decision plane the responsibility for calculating all the state required by the data plane, including FIB entries and packet filters. In conventional route calculation, the actual computation of the routes is quite simple. For example, OSPF applies Dijkstra's algorithm to a set of LSAs and BGP applies a 7-step ranking function to a set of known routes. The complexity of conventional approaches lies in coming up with ways of filtering LSAs, setting link weights, tagging, scoring, and dropping routes, etc. so that when the simple computation is performed the results put the data plane into a state that happens to meet the objectives of the network designer. In the 4D architecture, we turn the problem around. We provide the designer with the ability to directly express their network-level objectives, and accept that a more complex computation may be required to generate routes that meet these requirements.

Traffic-engineering [16] and redundant/resilent path planning [17] are two problems for which a number of algorithms are known for computing routes that obey network-level objectives. In this paper, we illustrate another type of capability: the specification of a reachability matrix as a network-level objective. That is, network designers can specify sets of source and destination subnet pairs that should or should not

[3]A more sophisticated implementation would give messages carried by the dissemination plane higher priority than normal data packets to reduce the chance of loss, but loss is still possible. Our prototype sends management messages with the same priority as data packets.

be allowed to exchange packets. Using this matrix, designers can precisely and unambiguously specify the reachability (security policies) they wish their network to enforce. The decision plane will compute routes so that packets can flow between any two subnets connected to the network and not prohibited by the reachability matrix, while installing packet filters where needed to ensure the reachability matrix is enforced.

In our current prototype, whenever a DE receives an LSA indicating a change in the network, it begins a 40 ms hold-down timer to allow time for other LSAs spawned by the same event to reach the DE. When the timer expires, it begins to compute entries for the FIB table on each router by assembling the most current LSA from each router into a matrix representing the physical connectivity of the data links in the network (the weight of each link is set at one). The Floyd-Warshall algorithm is run on the matrix to compute the all-pairs shortest paths for the network. The results are processed to produce a FIB entry for each destination for each router in the network such that packets will follow the shortest paths to their destinations. For each source destination subnet pair that should not be able to reach each other, we initially place a packet filter to drop that traffic on the interface closest to the destination. We then use a greedy heuristic to optimize the filter placement and combine filters by “pulling” the filters towards the source of forbidden traffic until further pulling would require duplicating the filters.

The running-time of the FIB calculation algorithm just described is $O(N^3 + I + ND)$ where N is the number of routers in the network and the N^3 term comes from the Floyd-Warshall algorithm. I is the total number of interfaces in the network, and the term arises from the processing of the all-pair shortest paths matrix to determine which interface should be used for each destination. D is the number of destination subnets known by the network, and the ND term represents the need to generate for each router a FIB entry for each of the D destinations.[4] In a production system an incremental algorithm could be used instead of Floyd-Warshal, reducing the N^3 term to an amortized $O(N \log N)$. Still, as shown in Section 5 the measured running times for the simpler algorithm of the prototype are within acceptable ranges.

The 4D architecture exposes the logic that controls the network, feeding it with the inputs it requires and implementing its decisions. However, the question remains how best to organize this logic. For example, some decision plane algorithms can run in real-time, but there should not be a requirement that all algorithms must run in real-time. There must also be clean ways to mix algorithms that control different facets of the network state. One possible approach is developing algorithms that allow network-level objectives to be separated into optimization objectives and mandatory objectives. This decomposition enables potentially slow-running optimization algorithms to complete their computation, while allowing fast-running algorithms to react quickly to ensure that mandatory objectives are always met. For example, a network with both traffic engineering and reachability matrix objectives might simultaneously use a slow-running linear program to identify a range of link weights that keep the link utilization in the network within some factor of the optimal level and our filter placement algorithm to ensure that the reachability matrix is obeyed exactly at all times.

4.5 Coordinating Multiple Decision Elements

The 4D architecture takes the extreme design point of logically centralizing all decision making; in our prototype this means all decisions are made by a single decision element. This begs the question of how to handle the failure of the decision element or a partition of the network that leaves some routers cut off from the decision element. In designing a system to cope with these failures, we can take advantage of the

[4]While the number of destination subnets D is likely to be small in many enterprise networks, it is currently over 150,000 in transit backbone networks. However, various address aggregation schemes can be used by the DEs to subsume many destination subnets into a single route, so the ND would be replaced by the total number of aggregated routes in the network, which could be much smaller.

specific characteristics of networks and their failure models. For example, single link and router failures are the most common, followed by correlated failures when a fiber bundle or shared resource goes down. The simultaneous failure of multiple pieces of widely separated equipment is extraordinarily rare. Further, even when all decision elements fail, the network does not loose the ability to forward packets — all that is lost is the ability to react to further changes.

To cope with DE failures our prototype implements a version of the hot-standby model. At any time a single master decision element takes responsibility for configuring the data plane on each router in the network, but multiple decision elements are plugged into the network. Each of the standby DEs gathers information from the network and performs the computations exactly the same way as the master [5]. However, the standbys do not send out the results of their computations.

The prototype selects the master DE using a very simple election protocol based on the DE beacons. At boot time, each DE begins sending periodic beacons and receiving LSAs from the routers in the network. The DE listens for beacons from other DEs for at least three times the beacon period (i.e., 1.5 s). The DE decides it should be the master and begin sending commands to routers if, after this waiting period, it has the highest priority of all received beacons. When the master DE receives a beacon from a DE with higher priority than its own, it immediately switches to a hot standby and ceases sending commands to routers. The two implications of this election protocol are that (1) when the master DE dies it may be 1.5 s before another DE becomes master, and (2) when a DE with higher priority than the current master boots up, the current master will immediately stop sending commands to routers, creating a window of up to 1.5 s when no DE is willing to send commands to routers since the new master is not yet sure that it is the winner of the election.

4.6 Interacting with Other Networks

In this paper, we have focused primarily on controlling the state of a single Autonomous System or enterprise network, as there is tremendous complexity in intra-domain problems [5]. Yet, the local network will need to connect to other networks. This means that the 4D architecture must provide a way to secure the dissemination paths: defining the boundary of the local network and preventing routers in neighboring networks from injecting commands into the local network. It must also accept routing advertisements from outside networks, so the decision plane can decide how to direct packets to destinations outside the local network.

To uphold our principle of direct control, we must solve these problems with minimal configuration on the routers and leveraging the DE-based path computation. The conventional approach to defining a network's boundary is to explicitly configure the border routers (BRs) to know which of their interfaces are internal and which face the "outside world." They then use a specially configured routing protocol session, like eBGP, to accept routes from the BRs of neighboring networks. In our prototype, we solve the problem of defining the network boundary and securing the dissemination paths by assuming each router in the network is configured via a USB key or flash card with a single security certificate. This certificate can be common across all devices, so it is not a device-specific configuration. It is needed to authenticate the routers that are part of the network and the decision elements that are authorized to control the network. As a part of neighbor discovery, each router and decision element uses its security certificate to decide whether the neighbor is part of the same network or not, and it reports this information as part of its LSA. The dissemination plane does not forward DE beacons out interfaces that face external routers, and it does not forward source routed messages that arrive on those interfaces.

[5]So during normal operation when the network is not partitioned, these DEs would converge to the same FIB entries, achieving "eventual consistency" among the DEs without the overhead of replicating FIB entries from the master DE to the standbys.

Our prototype does not explicitly model the exchange of route advertisements with external networks, although this can be achieved by using RCP's [18] approach, where eBGP is used to collect routes from neighboring networks and feed them to an engine in the decision plane that computes inter-domain routes.

5 Evaluation

Centralizing the decision-making logic at servers raises natural concerns about responsiveness, scalability, and reliability [2, 3]. In this section, we will show that the 4D architecture is a *feasible* approach for controlling a network by examining the behavior of our admittedly simplistic 4D prototype. Our goal is *not* to show that the prototype performs faster or better than any particular tweaking of an OSPF/EIGRP/BGP implementation. Rather, we will show that its performance is viable and roughly comparable to the performance seen in today's production networks.

5.1 Evaluation Environment and Scenarios

We have implemented our prototype in the Emulab [19] environment using Linux-based PCs as IPv4 routers. All PCs run the standard Emulab Linux v2.4 kernel. All PCs, including those used for the compute-intensive decision elements, have clock speeds between 600 and 850 MHz — these were state-of-the-art machines in 1999. Each Emulab PC has four 100 Mbps Ethernet interfaces that can be used to create network topologies. A fifth control interface is invisible to the prototype and used only for experiment administration (e.g., starting and stopping the experiment). To enable post-experiment correlation of events from the log file on each PC, all PCs synchronize their clocks by running the Network Time Protocol over their control interfaces, and we verify that each PC's clock is synchronized to within 2 ms of the others. A more sophisticated implementation of the 4D architecture could take advantage of the availability of synchronized clocks, but the prototype does not.

In each experiment we run, the routers are initialized to have no FIB entries and no knowledge of their neighbors. The routers discover their own interfaces and any connected neighbors, and disseminate the information to the decision elements. The decision elements elect a master, and the master installs FIB state into each of the routers. To verify that our software has performed correctly, we have run traceroute, ping, and sent user data across the network before, during, and after failure events.

The network topologies used in this section are those of production enterprise networks obtained through analysis of the networks' router configuration files [5]. Due to the four interface limit on Emulab PCs, routers in the production networks that have more than 4 interfaces are modeled by chaining together Emulab PCs to create a "supernode" with enough interfaces (e.g., a router with 8 interfaces will be represented by a string of 3 Emulab PCs). Multipoint interfaces in the production networks are handled similarly, with a string of Emulab PCs used to represent the multipoint link. This model increases the number of nodes and interfaces in our Emulab scenarios, meaning the results reported here are a conservative estimate of how our prototype would perform on the actual production networks. As these are enterprise networks, we do not configure the Emulab infrastructure to add artifical propagation delay or packet loss to the links. The topologies we use in our experiments vary in size from one with 10 Emulab PCs (as routers) and 11 links to one with 100 Emulab PCs and 149 links. On topologies with less than 20 routers, we insert 3 single-homed decision elements into the topology. On larger topologies, we insert 5 single-homed decision elements.

5.2 Response to Single Failures

The most common failure scenarios involve the failure of either a single link or a single router. In this subsection, we examine the 4D prototype's response to these scenarios.

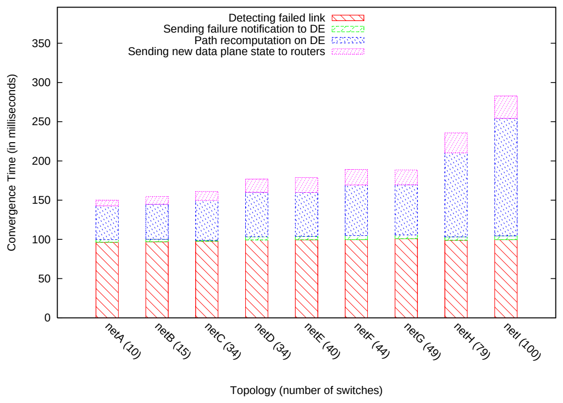

Figure 4: Reconvergence time after a single link failure averaged over 25 runs per network. Number of routers in the network is given in parentheses next to the network name.

In each experiment, we bring up a topology of routers and decision elements and wait for the topology to stabilize. We then cause either a node or a link to fail. To fail a router, we send the binary a signal to terminate, causing the router to abruptly stop participating in the network and to remove all routes from its routing table. To fail a link, we bring down the interface on one of the routers. We have implemented the router binary to ignore any send errors from a downed interface, so link failures are only found through the neighbor discovery protocol.

For each network topology, we randomly select 5 of its links to fail. For each of these links, we then run 5 separate experiments where we fail the link, for a total of 25 runs. The procedure for the router failure experiments is the same, only with 5 routers selected rather than 5 links. This results in 25 experiments in which a single node fails.

There are four phases of network convergence. First, the time between a failure and the time adjacent routers discover the failure. Second, the time for the dissemination plan to carry triggered LSAs announcing the failure to the master decision element. Third, the time for the decision element to calculate new FIB updates for the routers. Fourth, the time for dissemination plane to carry FIB updates to each router and for the routers to install the updates.

Figure 4 shows the mean network reconvergence time after a link failure in 9 different networks. The reconvergence time is measured from the time the link is taken down to the time the last router installs its update, at which point all routers will have working routes to all destinations. Each bar is divided into four stripes, one for each phase of the reconvergence. For instance, convergence took an average of 153 ms in network A. Of that time, 96 ms were spent discovering that the link failed, 3 ms were spent transmitting an LSA to the decision element, 44 ms were spent in recomputing configurations for all routers, and 10 ms were spent sending the configuration to each router and installing the configuration.

For all topologies, reconvergence time is acceptably low. In topologies of less than 50 routers, convergence time is below 200 ms. For the two larger topologies, convergence time is less than 300 ms. These convergence times are comparable to those of link-state networks, where sub-second reconvergence is considered excellent [20, 21].

The mean reconvergence time for the 9 networks varies between 153 and 286 ms. Of that time, 100 ms is spent detecting the link failure, and another 40 ms is spent while the decision element's hold-down timer[6] runs out. The problems of neighbor discovery and batching LSAs via a hold-down timer, which

[6]The purpose of the hold-down timer in both fully distributed control planes and in the 4D architecture is to prevent the network from recalculating routes before all the "bad news" has arrived. In essence, this timer trades response time to failures for a reduction in the number of network state changes.

	Minimum	Mean	Maximum
Discovery	91	96	107
LSA send	1	3	10
Computation	41	44	51
Confi g write	8	10	14

Table 1: Minimum, mean, and maximum times (in ms) for each of the four phases of convergence for 25 link failure experiments on network A (10 routers).

	Minimum	Mean	Maximum
Discovery	89	99	108
LSA send	2	4	12
Computation	143	149	162
Confi g write	15	28	40

Table 2: Minimum, mean, and maximum times (in ms) for each of the four phases of convergence for 25 link failure experiments on network I (100 routers).

	Minimum	Mean	Maximum
Discovery	91	97	106
LSA send	2	9	17
Computation	32	40	50
Confi g write	8	11	18

Table 3: Minimum, mean, and maximum times (in ms) for each of the four phases of convergence for 25 router failure experiments on network A (10 routers).

represent over 75% of the time spent in reconvergence in our prototype, are common to both the 4D architecture and fully-distributed architectures, so it is reasonable to expect both architectures to have comparable performance. Many proposals to reduce reconvergence time in conventional architectures begin with faster neighbor discovery [21] or adding hardware support for link status detection. Those techniques are equally applicable to the 4D architecture.

Tables 1 and 2 zoom in on the results from two topologies, networks A and I, respectively. Each table provides the mean, maximum, and minimum values for each phase of convergence over the 25 experiments performed. Removing the 40 ms hold down timer, the mean computation time scales from 4 ms in network A to 109 ms in network I, which has 10 times more routers. This scaling is smaller than predicted by the $O(N^3)$ complexity of the Floyd-Warshall implementation.

The reconvergence time for the nine networks after a single router fails is quite similar to those of the link failure scenarios, both the totals and the breakdowns. The largest difference is that it takes slightly longer for all the LSAs to reach the decision element because when a router fails, up to 4 other routers send LSAs announcing that their link to the failed router has gone down. Comparing Table 3, which presents the reconvergence times for a router failure in network A, to Table 1 illustrates the point.

5.3 DE Failures and Network Partitions

A potential concern with the 4D architecture is that by centralizing decision making at a single Decision Element (DE), loss of communication with the DE leaves the network without the ability to react to additional changes to the network. As explained above, our experimental setup adds reliability to the system by attaching 3 or 5 DEs the network, depending on the size of the network

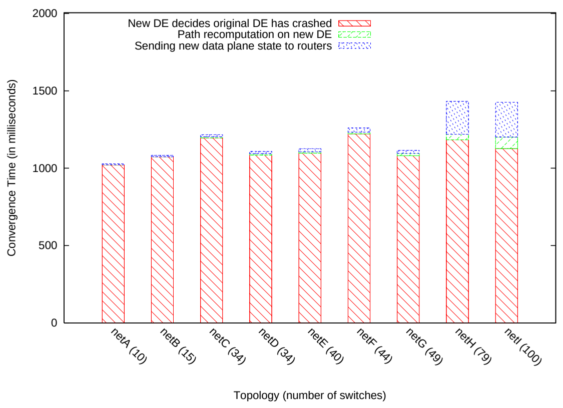

Figure 5: Mean reconvergence time after the failure of the master decision element averaged across five runs on each network.

DE Failure: Failure of any DE but the master DE is harmless, since in our prototype the other DEs are only hot stand-bys. To evaluate the effect of the failure of the master DE, we run experiments in which the network is allowed to come up and converge, and then the master DE is shutdown without warning.

Figure 5 shows the time required for a new DE to take control of the network after the master DE fails, broken out into the steps involved. The first step is detecting that the old master has failed. Once this occurs, the situation is the same as the reconvergence scenarios described previously. To verify that the network is in a known-good state, the new master computes routes for each switch, and then writes the configurations to the switches. [7] As expected, the time for a new master DE to take control is dominated by the time required for the election. With the election algorithm requiring that three beacons be missed before a new master takes over and beacons being sent every 500 ms, determining that the master DE has failed takes 1-1.5 s. If this is too long a window to be without a DE, changes to either beacon frequency or number of missed beacons could decrease the detection time, limited only by the propagation delay between DEs (the cost of additional beacons is minimal).

Network Partition: If events should cause the network to partition, only one of those partitions will contain the master DE and, by the definition of a network partition, the switches in other partitions will be unable to communicate with the master DE. Just as network designers can choose to build a topology that is more or less resistant to partition (e.g., a ring versus a mesh), the designers have the freedom to add DEs to their network until all likely partitions include at least one DE.

To evaluate the response of our prototype to a partition, for each network we select five scenarios involving failures of links and/or switches that result in creation of at least two partitions. We then run each scenario five times. To analyze the response to the partition, we separately consider the partition that contains the original master DE from the other partitions.

As shown in Figure 6, the partition with the original master DE responds in essentially the same manner as the single-failure reconvergence scenarios examined earlier, with two significant changes. First, there is greater variance in how long it takes all the switches to discover which neighbors are missing. The variation exists because our experimental administration scripts cannot instantaneously create a partition: depending on the network, it takes between 0 and 200 ms to shutdown the links and switches that create the partition.

Figure 7 shows the reconvergence time in the partitions that do not contain the original master DE. In these partitions, the response is essentially the same as when a master DE fails (cf. Figure 5). Again,

[7] One possible optimization to this process is for the new master to query each switch for its configuration and only perform writes on switches that require state changes.

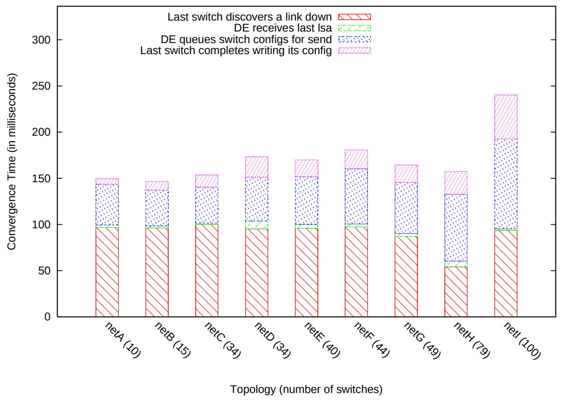

Figure 6: Reconvergence time in the main network after a partition event.

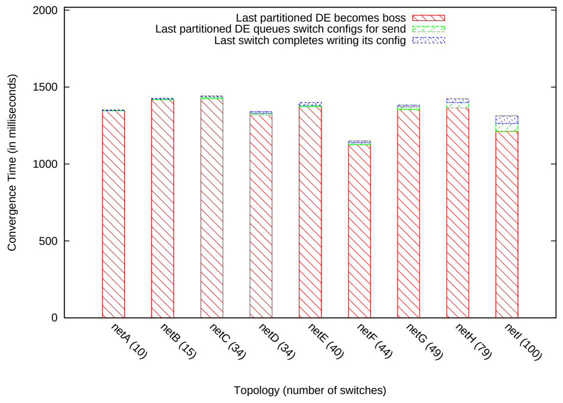

Figure 7: Reconvergence time in the partitioned network after a partition event.

there is small additional variation in detecting the absence of a master DE since the partition is not created instantaneously, and beacons from the original master DE can find their way into the new partition as the partition is being created.

These results show that loss of communication with a decision element, attributed to either decision element failure or a network partition, does not result in catastrophic failure even with our simplistic replication and election strategies. In fact, the failure of a DE does not cause any noticable change to data crossing the network unless there are additional events before a new master DE can be elected.

5.4 Overhead of the 4D Architecture

To be scalable, it is critical that the dissemination-plane does not consume excessive bandwidth in carrying data between the routers and the decision elements. Figure 8 shows the average number of bytes of management information sent by the active DE during five, 120 second single-link failure scenarios. DE's, like all entities running the discovery plane code, send 33 HELLO packets of 28 bytes out each interface each second — we omit this fixed overhead from the figure. All DE's, whether the master or not, send beacons every 500 ms, but these beacons are a negligible contribution to the overhead. Only the master DE sends configurations to the routers, and this is the only component of sent bytes that varies with network. At worst, when all routers in the network are completely empty, approximately 28 bytes per route per router must be sent. The volume of configuration data does not vary directly with network size because the use of delta encoding means that the size of the updates is proportional to the number of routes that change, which is a function of the topology.

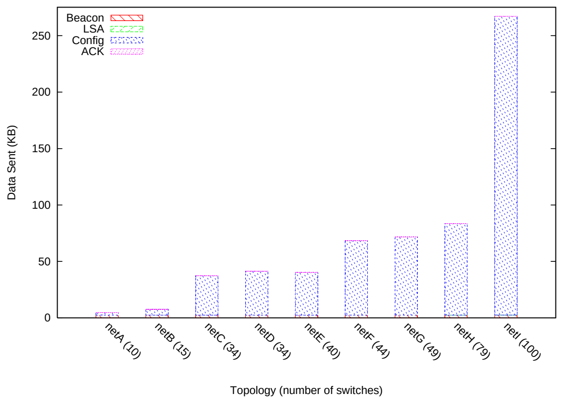

Figure 8: Bytes sent by the master DE in each network, averaged over 5 runs.

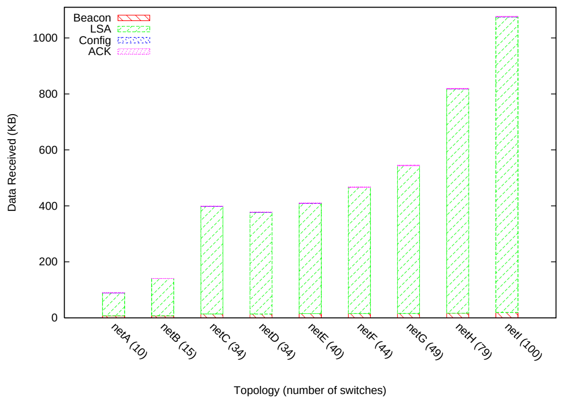

Figure 9: Bytes received by each DE in each network, average over 5 runs.

Figure 9 shows the average number of bytes of management information received by each DE during the same scenarios. The bytes received are dominated by topology collection, which scales linearly with the number of interfaces in the network, consuming roughly 32 bytes per interface per router per LSA. Variation in volume of LSAs received among networks of similar size is due to variation in the number of links in the topologies. The absolute count of bytes involved is not large: receiving an LSA from every router in a 100 router network totals 32 KB.

6 New Capabilities of the 4D Architecture

In this section, we describe one example of how the expressive power of the 4D architecture can be used to change the way network reachability is conveyed and enforced. The data center example in section 2.1 shows what can go wrong in today's practice of reachability control, and here we use the same example to illustrate how the 4D architecture avoids the problem through joint control of packet filtering and routing.

We implement the example by connecting 13 PCs, 5 as routers and 8 as hosts, to form a network as shown in Figure 1. We conduct two experiments, each running through the example scenario. In the first experiment, we use a traditional control plane, running the Quagga [22] OSPF daemon on the five routers to calculate routes for the network and manually add packet filters, just as a real network operator would. In the second experiment, the network is controlled using our 4D prototype, with the decision element given a 2-line reachability specification describing the desired security policy. During both experiments we use data traffic sent at 240 packets/second to test whether the network's security policy is obeyed: one flow that should be permitted (AF1 sending to BF1) and one flow that should be forbidden (AF1 sending

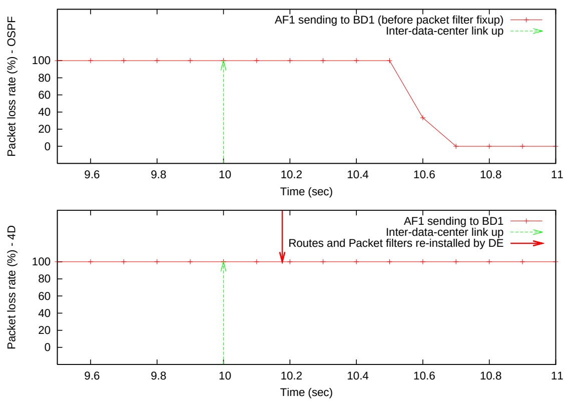

Figure 10: Impact of adding new link (R1 to R3) on traffic from AF1 to BD1, which should be dropped to obey security policy. Conventional network (top figure), 4D network (bottom figure).

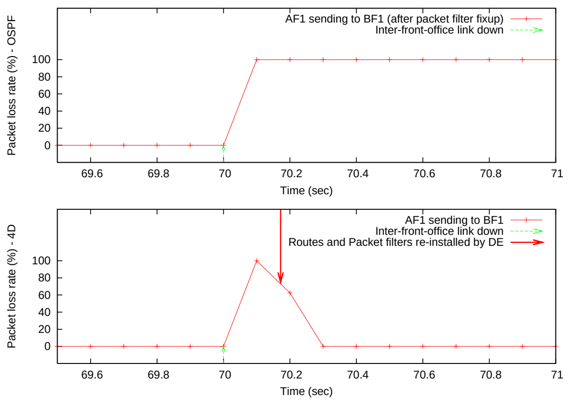

Figure 11: Impact of link failure (R2 to R5) on traffic from AF1 to BF1, which should be permitted. Conventional network (top figure), 4D network (bottom figure).

to BD1).

In the conventional network, the manually calculated and placed filters are insufficient to block traffic forbidden by the security policy. In the 4D architecture, routes and packet filters are calculated together, with the network's global reachability specification taken into account. Figure 10 plots the loss rate versus time during the period surrounding the introduction of the link between A and B's data centers. The conventional network's response is shown in the top graph, and our prototype's behavior is shown in the bottom graph. Initially, traffic from AF1 to BD1 is blocked by packet filters. In the conventional network, the filter is manually placed by the network architect on interface i3.1 to prevents A's front office traffic from reaching BD. In the 4D archtecture, the packet filter is placed automatically by the decision logic. As a result, both networks show the desired loss rate of 100% for the forbidden traffic. At time 10, the new link between the data centers is added. In the conventional network, OSPF responds to the additional link by recomputing routes and within 600 ms has redirected traffic from AF to BD over the new link, bypassing the packet filter on interface i3.1 and allowing forbidden traffic to pass through the network. In the 4D network, both new routes *and new packet filter placements appropriate for those routes* are computed and loaded into the routers within 200 ms. Since the new filters are loaded simultaneously with the new routes, not a single packet of forbidden traffic is leaked.

The inability of conventional networks to jointly control packet filters and routing means that packet filters must be placed pervasively throughout the network, but this can rob the network of its ability to resilently carry desired traffic. Figure 11 plots the loss rate versus time for traffic sent from A's front office to B's front office during a period when the link between the two front offices goes down. Again, the top graph depicts the loss rate on the conventional network while the bottom graph depicts the loss rate for a 4D-controlled network. In this scenario, the link between the two data centers remains up. To patch the security hole described in the previous paragraphs, the conventional network has manually-placed packet filters on both i3.1 and i3.2 to drop traffic from AF to BF.

When the link between front offices goes down at time 70, the flow from AF to BF on the conventional network immediately begins losing packets. Even though OSPF successfully reconverges to route traffic from one front office to the other via the data center link, the packet filter on interface i3.2 prevents front office traffic from getting through, causing permitted traffic to be incorrectly thrown away. In a conventional network, this outage would persist until a human operator diagnosed the problem and altered the configuration files (hopefully remembering to replace the filters when the outage ends, to avoid reopening the security hole disscused above). In the 4D-controlled network, the decision element recalculates both routing and packet filter placement simultaneously, placing a filter on i2.1 to block traffic from AF to BD, but allowing traffic from AF to BF to pass through. Within 300 ms of the link failure, connectivity is restored in the 4D network.

This example illustrates how all the principles underlying the 4D architecture work together to solve a problem. The network designer specifies a network-level objective to the decision plane. The decision plane uses a network-wide view to compute packet filter placements and FIB contents that meet the objectives, and then, using its ability to directly control the routers, the decision plane configures the routers to instantiate the solution without introducing undesirable transients behaviors.

7 Related Work

Today's networks are managed using large numbers of separate tools for tasks like DoS mitigation [23], traffic engineering [24, 25], QoS provisioning [26], etc. While very useful for specific tasks, each tool focuses on a small portion of the configuration state, and they do not consider the interactions among multiple mechanisms. Our architecture avoids these concerns by defining better abstractions and enabling decision elements with network-wide views to directly control all network state.

The importance of network management in creating robust networks is taking hold in the research community [27]. Rexford and colleagues [2] advanced an architecture that makes the network control plane "wafer-thin," but they provided no details of how such an architecture could be implemented and left open many questions as to whether their architecture is even feasible [3]. We have formulated a set of concrete principles and proposed a detailed architecture. In addition, we have implemented a prototype of the architecture, and report on its performance and capabilities.

The 4D architecture we propose is consistent with recent trends for using measurement data to drive network operations [28, 29], open interfaces for router software and hardware [30, 31, 32, 33, 34, 35], and recent proposals for moving path computation to dedicated servers [18, 36, 37, 38]. Our proposal takes these trends to their natural conclusion by elevating network discovery to a first-order function of the routers and switches, moving network control completely out of the router/switch software, and having the decision plane go beyond path computation to handle all aspects of network control.

8 Summary

We make three contributions in this paper. First, we synthesize three fundamental design principles: *network-level objectives*, *network-wide views*, and *direct control*, that capture the essence of the problem of network control and management. Based on these principles, we propose a novel 4D architecture that refactors network functionality into four components—the *data, discovery, dissemination, and decision* *planes*. Second, we implement a prototype of the 4D architecture and conduct a comprehensive performance study. Our study demonstrates the viability of the 4D architecture, specifically with respect to responsiveness, reliability, and overhead. Finally, we demonstrate that the 4D architecture enables sophisticated new network functionality (such as meeting network-wide reachability objectives) that is difficult to achieve in traditional networks.

References

[1] LAN/MAN Standards Committee of the IEEE Computer Society, *IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Common specifications Part 3: Media Access Control (MAC) Bridges*, 1998.

[2] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, pp. 59–64, November 2004.

[3] D. Clark, "Public review of 'Network-wide decision making: Toward a wafer-thin control plane'," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[4] A. Myers, E. Ng, and H. Zhang, "Rethinking the service model: Scaling Ethernet to a million nodes," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[5] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, August 2004.

[6] LAN/MAN Standards Committee of the IEEE Computer Society, *802.1Q IEEE Standards for Local and metropolitan area networks Virtual Bridged Local Area Networks*, 2003.

[7] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh, "Viking: A multi-spanning-tree Ethernet architecture for metropolitan area and cluster networks," in *Proceedings of IEEE Infocom 2004*, March 2004.

[8] "Yipes." http://www.yipes.com.

[9] A. Feldmann and J. Rexford, "IP network configuration for intradomain traffic engineering," *IEEE Network Magazine*, pp. 46–57, September/October 2001.

[10] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, "The cutting EDGE of IP router configuration," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2003.

[11] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proc. Networked Systems Design and Implementation*, May 2005. To appear.

[12] G. Varghese and C. Estan, "The measurement manifesto," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2003.

[13] D. Clark, K. Sollins, J. Wroclawski, D. Katabi, J. Kulik, X. Yang, R. Braden, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa, "FINAL TECHNICAL REPORT New Arch: Future generation Internet architecture." Available from http://www.isi.edu/newarch/.

[14] G. Hjalmtysson, P. Sebos, G. Smith, and J. Yates, “Simple IP restoration for IP/GbE/10GbE optical networks,” *Postdeadline paper PD-36, OFC 2000*, March 2000.

[15] D. B. Johnson, D. A. Maltz, and J. Broch, *Ad Hoc Networking*, ch. The Dynamic Source Routing Protocol for Multi-HopWireless Ad Hoc Networks, pp. 139–172. Addison-Wesley, 2001.

[16] B. Fortz and M. Thorup, “Internet traffic engineering by optimizing OSPF weights,” in *Proc. IEEE INFOCOM*, March 2000.

[17] M. Alicherry and R. Bhatia, “Pre-provisioning networks to support fast restoration with minimum over-build,” in *Proc. IEEE INFOCOM*, March 2004.

[18] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, “The case for separating routing from routers,” in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2004.

[19] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, “An integrated experimental environment for distributed systems and networks,” in *Proc. Operating Systems Design and Implementation*, pp. 255–270, December 2002.

[20] H. Villfor, A. Hedlund, E. Aman, T. Eriksson, and T. Ahlstrom, “Operator experience from ISIS convergence tuning,” in *Proceedings of RIPE 47*, Feb 2004. Available as http://www.ripe.net/ripe/meetings/ripe-47/presentations/ripe47-routing-isis.pdf.

[21] C. Alaettinoglu, V. Jacobson, and H. Yu, “Towards milli-second IGP convergence.” Expired Internet Draft, Nov 2000. Available as http://www.packetdesign.com/news/industry-publications/drafts/convergence.pdf.

[22] “GNU Quagga routing software.” http://www.quagga.net.

[23] “Arbor Networks Peakflow.” http://www.arbornetworks.com/products_sp.php. Last visited 1/2005.

[24] “Cariden MATE framework.” http://www.cariden.com/products/. Last visited 1/2005.

[25] “OpNet SP Guru.” http://www.opnet.com/products/spguru/home.html. Last visited 1/2005.

[26] R. Chadha, G. Lapiotis, and S. Wright, “Policy-based networking,” *IEEE Network Magazine*, vol. 16, pp. 8–9, 2002.

[27] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, “A knowledge plane for the Internet,” in *Proc. ACM SIGCOMM*, pp. 3–10, 2003.

[28] M. Grossglauser and J. Rexford, “Passive traffic measurement for IP operations,” in *The Internet as a Large-Scale Complex System*, Oxford University Press, 2005. To appear.

[29] S. Bhattacharyya and S. Moon, “Network monitoring and measurements: Techniques and experience.” Tutorial at *ACM SIGMETRICS’2002*. http://ipmon.sprint.com/pubs_trs/tutorials/Network-Measurement-And-Monitoring.pdf.

[30] G. Hjalmtysson, “The Pronto platform - a flexible toolkit for programming networks using a commodity operating system,” in *Proc. International Conference on Open Architectures and Network Programming (OPENARCH)*, March 2000.

[31] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullmann, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, and J. Hartman, “A NodeOS interface for active networks,” *IEEE J. Selected Areas in Communications*, March 2001.

[32] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click modular router,” *ACM Trans. Computer Systems*, August 2000.

[33] M. Handley, O. Hudson, and E. Kohler, “XORP: An open platform for network research,” in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, October 2002.

[34] A. Doria, F. Hellstrand, K. Sundell, and T. Worster, *General Switch Management Protocol (GSMP) V3*. Internet Engineering Task Force, 2002. RFC 3292.

[35] “Forwarding and Control Element Separation Charter.” http://www.ietf.org/html.charters/forces-charter.html.

[36] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, “The SoftRouter architecture,” in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[37] A. Farrel, J.-P. Vasseur, and J. Ash, “Path computation element (PCE) architecture.” Internet Draft draft-ash-pce-architecture-00.txt, September 2004.

[38] K. Lakshminarayanan, I. Stoica, and S. Shenker, “Routing as a service,” Tech. Rep. UCB-CS-04-1327, UC Berkeley, 2004.