DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet

Jiayue He, Rui Zhang-Shen, Ying Li, Cheng-Yen Lee, Jennifer Rexford, Mung Chiang
Princeton University
{jhe, rz, yingli, chenglee, jrex, chiangm}@princeton.edu

ABSTRACT

Running multiple virtual networks, customized for different performance objectives, is a promising way to support diverse applications over a shared substrate. Despite being simple, a static division of resources between virtual networks can be highly inefficient, while dynamic resource allocation runs the risk of instability. This paper uses optimization theory to show that adaptive resource allocation can be stable and can maximize the aggregate performance across the virtual networks. In the DaVinci architecture, each substrate link periodically reassigns bandwidth shares between its virtual links; while at a smaller timescale, each virtual network runs a distributed protocol that maximizes its own performance objective independently. Numerical experiments with a mix of delay-sensitive and throughput-sensitive traffic show that the bandwidth shares converge quickly to the optimal values. We demonstrate that running several custom protocols in parallel and allocating resource adaptively can be more efficient, more flexible, and easier to manage than a compromise "one-size-fits-all" design.

1. INTRODUCTION

The Internet was designed to provide the same simple packet-delivery service for all applications. In practice, different applications can have different performance objectives; for example, voice-over-IP and gaming traffic perform better over low-delay paths, whereas large file transfers perform better over high-bandwidth paths.

1.1 Case for Adaptive Network Virtualization

Researchers have proposed that future networks can run multiple *virtual networks*, each customized for a particular traffic class, with the substrate providing *separate resources* for each virtual network [1, 2, 3]. Today, network virtualization is moving from fantasy to reality, as major router vendors start to support both router virtualization (to run multiple virtual routers in parallel on a single router) [4, 5] and router programmability (to run customized protocols) [6, 7]. These features can start being deployed within a single institution, such as an enterprise network, a Virtual Private Network (VPNs), or an Internet Service Provider (ISP), and ultimately across multiple institutions [3].

Despite having great potential for supporting customized protocols, network virtualization faces a fundamental challenge of efficiently sharing the underlying network resources. At one extreme, each virtual network can be assigned a *static* share of the resources, and each substrate link can ensure isolation by rate limiting the traffic belonging to each virtual link that traverses it. Static resource allocation is simple, but inefficient. For example, a congested virtual network would have to drop packets, even if other virtual networks are idle. As another example, suppose a virtual network optimized for small end-to-end delay became congested; then, users might ironically experience lower end-to-end delay by using another virtual network optimized for different performance goals.

Static allocation of resources across the virtual networks could easily cause network virtualization to perform *worse* than existing solutions, such as overlays. At the other extreme, unconstrained sharing of the resources can lead to instability. Aided by optimization theory, we show that *adaptive network virtualization*, with resource allocation responding to the demands faced by the virtual networks, can lead to a stable solution *maximizing* the aggregate utility of all the virtual networks. In our architecture, resource shares are *periodically* reassigned between virtual networks; between reassignments of the resource shares, each virtual network makes efficient use of its allocated share of the resources using a distributed protocol. The small-timescale isolation between virtual networks ensures the stability of the overall system, while the longer-timescale adaptation of resource shares ensures efficiency.

ACM CoNEXT 2008, December 10-12, 2008, Madrid, SPAIN

1.2 Leveraging Technology Trends

Adaptive network virtualization leverages and extends several trends in networking research, from the long history of research on Quality-of-Service (QoS) techniques, to recent work on overlay networks and the emerging interest in network virtualization.

While traditional QoS techniques aim to provide performance guarantees, adaptive network virtualization maximizes the aggregate performance objective across all traffic classes. Our approach is very different from the Intserv [8] architecture and QoS-routing protocols [9, 10], which offer *per-flow* performance guarantees. Adaptive network virtualization is more similar to DiffServ [11] and Type-of-Service (ToS) routing [12], which manage resources at the granularity of traffic classes. There are also key differences between adaptive virtualization and DiffServ. In DiffServ, routers schedule among the queues based on static priority or fixed weights. In contrast, adaptive network virtualization assigns the scheduling weights *dynamically*. Extensions to Diffserv provide separate forwarding tables for each traffic class, in the form of "multi-topology routing" [13]. Recent research has shown that running two instances of OSPF, with link weights tuned to different application performance objectives, offers significant performance benefits over a single protocol instance [14]. *This paper takes these ideas one step further by allowing each virtual network to run customized protocols.*

Overlay networks commonly consist of end hosts or middleboxes that communicate over tunnels that span the underlying Internet. Similar to adaptive network virtualization, overlay networks can run customized routing or congestion-control protocols [15, 16, 17, 18]. In addition, an overlay can perform its own admission control and packet scheduling to manage its own traffic [19]. Overlays partially shift the responsibility to select routes and forward packets to the end hosts. Without support from the underlying network, however, overlays rely on frequent measurements that add significant system overhead [15], and multiple overlay networks can interact in harmful ways [20]. This begs the question of whether the underlying network should support multiple packet-delivery services. *This paper takes these ideas to their logical conclusion by running the customized protocols inside the network and providing them separate shares of the underlying resources.*

Adaptive network virtualization leverages a variety of virtualization technologies [21, 4, 5, 6, 7] to build customized virtual networks. There are two broad usages of virtual networks: experimental research facilities and platforms for commercial services. The research community has proposed to build experimental testbeds that run multiple virtual networks in parallel [22, 23]. To ensure that experiments are repeatable, static resource partitioning is the natural choice for experimental testbeds. Commercial institutions, on the other hand, are interested in maximizing revenue. In this scenario, efficient usage of the underlying resources becomes more important, thus this paper proposes to dynamically adapt the resources shares. A few position papers have advocated that commercial institutions run customized protocols [1, 2, 3], though presently no commercial institution has deployed such a system. In addition, previous research focused on high-level issues such as economic incentive, while this paper presents an architecture for efficiently managing resources between multiple virtual networks.

1.3 The DaVinci Architecture

Adaptive network virtualization is a rich and challenging research problem. As a first step, this paper focuses on how a *single* substrate network can support multiple traffic classes, each with a different performance objective, using network virtualization. In addition, each virtual network is assumed to run customized packet-delivery protocols that *optimize* a performance objective, formulated as a convex function of network parameters. We focus on how the virtual networks share link bandwidth, as opposed to (say) CPU resources, because the sharing of link bandwidth has the most direct impact on packet-delivery services. Since all virtual networks are run by the same institution, we assume that each virtual network does not maliciously try to acquire more resources (e.g., by injecting bogus traffic) to harm the performance of the other virtual networks.

This paper shows a single institution can *maximize aggregate performance* across multiple traffic classes if:

- Each virtual network runs *customized* protocols that *implicitly optimize* a convex performance objective. Fortunately, these distributed protocols can often be derived by leveraging optimization theory [24]. Each protocol can be designed and run *independently* of other virtual networks.

- At each link, the substrate periodically reassigns bandwidth shares between virtual networks based on *local* information, including current congestion levels and the performance objectives of the virtual networks. On a smaller timescale, a *traffic shaper* enforces the bandwidth allocations for each virtual link.

Under this combined system, optimization theory shows that the bandwidth shares converge to *optimal* resource allocation among the traffic classes. We refer to the combined system as DaVinci—Dynamically Adaptive Virtual Networks for a Customized Internet.

The design and evaluation of DaVinci leverages, and extends, previous research on using optimization theory to derive new network protocols (see survey paper [24]).

Previous work shows how to use optimization decomposition to design protocols that implicitly maximize a performance objective, such as maximizing throughput [25] or minimizing delay [26]. In this paper, we show that multiple such protocols can *share* an infrastructure with limited resources and collectively maximize aggregate performance. This result, itself derived using optimization decomposition, provides further motivation for designing customized protocols based on optimization theory.

The rest of the paper is organized as follows. Section 2 discusses why multipath traffic management, customized protocols, and separate resources are all important to DaVinci. Section 3 describes how the DaVinci architecture ties these features together. Section 4 models the DaVinci architecture, including the bandwidth-allocation algorithm and the proof that the overall system converges to maximize aggregate performance. Numerical experiments on the adaptation of bandwidth shares are presented in Section 5 for delay-sensitive and throughput-sensitive traffic classes. Finally, Section 6 concludes and discusses future work.

2. DAVINCI DESIGN DECISIONS

Traffic management controls how much traffic traverses each path in a network, and is achieved by congestion control, routing protocols, and traffic engineering in today's Internet. In DaVinci, each virtual network runs its own traffic-management protocols. DaVinci has support for flexible splitting of traffic over multiple paths, custom traffic-management protocols in each virtual network, and separate resources to provide isolation between traffic classes. Multipath traffic management and customized protocols allow each traffic class to efficiently utilize its bandwidth, while separate resources provide isolation between the traffic classes.

Each subsection motivates DaVinci's design decisions through simple examples using the two-link, two-node topology in Figure 1. For our examples, we consider two different traffic classes: inelastic delay-sensitive traffic and elastic throughput-sensitive traffic. Elastic traffic adapts sending rate based on the available bandwidth, while inelastic traffic has a fixed amount of demand independent of the available bandwidth. We assume the delay-sensitive traffic is trying to minimize average user delay, and it has a fixed amount of demand. The links in Figure 1 have disparate delays and capacities, so that the delay-sensitive traffic clearly prefers the low-propagation-delay link, while the throughput-sensitive traffic clearly prefers the high-bandwidth link.

2.1 Multiple Paths

In the Internet today, routing protocols select a *single path* between two end hosts. In contrast, DaVinci advocates traffic to be forwarded over *multiple paths* between two end hosts.

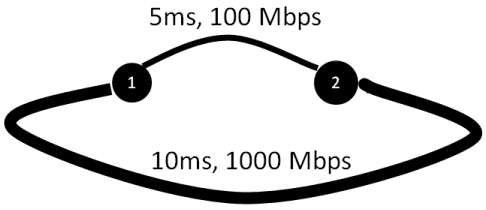


Figure 1: Topology with two paths between a pair of nodes.

To illustrate the importance of forwarding over multiple paths, we contrast multipath routing with single-path routing on the topology in Figure 1. First, we consider the case of a network carrying only throughput-sensitive traffic. Since the throughput-sensitive traffic is elastic, under single-path routing, the high-bandwidth link will be selected and 1000Mbps of traffic will be carried. When multipath routing is available, however, both paths can be used simultaneously and 1100Mbps of traffic can be carried.

Second, we consider the case of a network carrying only delay-sensitive traffic. For this example, we assume that the queuing delay is low relative to propagation delay when the link load is below capacity. When the delay-sensitive traffic has less than 100 Mbps of traffic, all traffic is routed on the low-delay path for both single-path and multipath routing. When the delay-sensitive traffic has between 100 Mbps and 1000 Mbps of traffic, all traffic is routed on the high-delay path for single-path routing. If multipath routing is available, then the delay-sensitive traffic would route 100 Mbps on the low-delay path, and thus experience lower average delay. When the delay-sensitive traffic has between 1000 Mbps and 1100 Mbps of traffic, single-path routing can only support a fraction of the traffic while multipath routing can support all.

As shown with the simple example, *multipath traffic management leads to better performance for both traffic classes than single-path traffic management.* Moving to multipath routing requires both control-plane and data-plane support. Fortunately, the key ingredients such as path diversity, computing splitting percentages, and directing packets onto multiple paths in specified ratios already exist [27]. However, there still remains the challenge of balancing the tradeoff between efficiency (i.e., using short paths) and avoiding shared bottlenecks (i.e., using disjoint paths). For efficiency, a network can select the shortest m paths between two nodes, but these paths are likely to share many links. To avoid shared bottlenecks, a network can choose *disjoint* paths, but

this can easily lead to long paths which consume extra resources. See [28] for a discussion of the tradeoff.

2.2 Customized Protocols

Today's Internet runs the same traffic-management protocols for all traffic. In this paper, we advocate running customized traffic-management protocols for each traffic class. Each traffic class (possibly including multiple applications) has its own performance objective. For example, the throughput-sensitive traffic's performance objective can be to maximize throughput. On the other hand, the delay-sensitive traffic's performance objective can be to minimize the average delay. Since end-to-end delay is the sum of propagation delay and queueing delay over a path, delay-sensitive traffic prefers low propagation-delay paths and small queues. In contrast, throughput-sensitive traffic prefers high-bandwidth paths and tends to drive to fuller queues.

To understand the importance of customized protocols, we consider running protocols optimized for different performance objectives on the two-node topology in Figure 1. For simplicity, we assume single-path routing. First, we consider the case of a network carrying only throughput-sensitive traffic. When the traffic-management protocol is optimized to maximize throughput, 1000 Mbps can be carried. In contrast, only 100 Mbps will be carried when the traffic-management protocol is optimized to minimize delay. Second, we consider the case of a network carrying less than 100 Mbps of delay-sensitive traffic. When the traffic-management protocol is optimized to minimize delay, the low-delay path is selected. In contrast, the high-delay path will be selected when the traffic-management protocol is optimized to maximize throughput.

Our simple example demonstrates that *each traffic class benefits from having traffic-management protocols customized to its own performance objective*. Today, routers already support "multi-topology routing" [13], where routers can run multiple instances of the same routing protocol with configurable parameters tuned to each traffic class. Further protocol customization is possible with *router programmability*: a flexible and extensible way to implement a variety of distributed traffic-management protocols. Though not yet a commercial reality, vendors have announced plans to support programmable routers [6, 7].

2.3 Separate Resources

When multiple traffic classes coexist over the same network, each traffic class could control a subset of resources at each node and link, as shown in Figure 2. At each node, each traffic class has a portion of the underlying node resources (such as CPU and memory). At each link, each traffic class can consume a portion of the bandwidth of each link. Packets arriving at an edge router are first classified to a particular traffic class, then directed to the appropriate outgoing link and the queue associated with that traffic class.

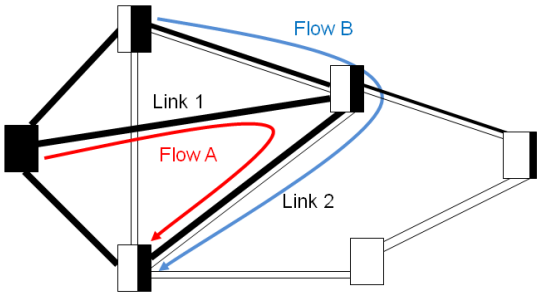


Figure 2: The shaded regions identify the portion of node and link resources allocated to one traffic class. The remaining resources are allocated to the second traffic class.

To show customized protocols alone are insufficient, we consider a system with *a single queue*, but supporting *customized traffic-management protocols*. Without separate resources, the throughput-sensitive traffic can consume all the bandwidth in the network, thus not leaving enough room for the delay-sensitive traffic. In the topology in Figure 1, even if there is sufficient space for delay-sensitive traffic across the two links, the delay-sensitive traffic might not get its preferred path. Similarly, separate resources alone are insufficient, as the previous subsection illustrated, a protocol optimized to maximize throughput will not necessarily minimize delay.

At each link, one possible choice is to use a *work-conserving scheduler*, where it would transmit extra packets for one traffic class, if the other traffic class has an empty queue. Consider the case where two flows (belonging to traffic class A and B respectively) share a link in Figure 2. If link 1 is idle, then a work-conserving scheduler would transmit additional packets for flow A. This would cause additional congestion on link 2, and incorrectly signal that the traffic class A needs more resources on link 2, leading to resources taken away from traffic class B. Consequently, the distribution of resources between the two traffic classes may be inefficient. The same problem would not arise with a non-work-conserving scheduler, which does not take advantage of an empty queue.

As shown by the simple examples, *the elastic throughput-sensitive traffic can easily overwhelm the inelastic delay-sensitive traffic without the separation of resources*. With the separation of resources, the delay-sensitive traffic is more likely to be routed on its preferred path(s). Technologies for packet classification, separate queues, and link scheduling have existed for more than ten years

[8, 11, 12]. More recently, separate link resources and forwarding tables are used to establish Virtual Private Networks (VPNs) [21]. In addition, router vendors have started supporting virtualization to subdivide node resources such as CPU and memory [4, 5].

3. DAVINCI ARCHITECTURE

In this section, we introduce the basic building blocks of DaVinci and how they work together. In DaVinci, each traffic class is carried on its own virtual network with customized traffic-management protocols. Virtual networks are constructed over the physical network (which we refer to as the substrate network) by first subdividing each physical node and each physical link into multiple virtual nodes and virtual links. The substrate runs schedulers that arbitrate access to the shared node and link resources, to give each virtual network the illusion that it runs on a dedicated physical infrastructure. Table 1 summarizes the key notation.

Symbol	Meaning
C_l	Capacity of substrate link l.
$\mathbf{y^{(k)}}$	Bandwidth assigned to VN k. Computed by substrate network.
$\mathbf{z^{(k)}}$	Path rates for VN k.
$\lambda^{(\mathbf{k})}$	Indication of VN k's satisfaction. Computed by substrate network. Used by substrate to compute $\mathbf{y^{(k)}}$

Table 1: Summary of key notation. VN represents Virtual Network.

3.1 Packet Classification and Forwarding

All data packets are handled by the substrate at the behest of the virtual networks. At an edge node of the substrate, data packets are directed to the appropriate virtual network using packet classification or some other form of "user opt-in" technique. Users may connect to a virtual network in a variety of ways, such as establishing a tunnel to a virtual node, configuring a Web browser to use a virtual node as a proxy, or DNS redirection [29, 30, 31].

Within the same virtual network, each edge virtual node may have *multiple (virtual) paths* for reaching another virtual node, as shown in Figure 3. To distinguish between multiple paths within the same virtual network, edge virtual nodes encapsulate the packets with *labels*. Virtual nodes can then populate *label tables* based on the paths they computed. A packet can then be directed onto a specific path using the label tables [32]. If the traffic-management protocols are sensitive to out-of-order packets (like TCP is today), then packets belonging to the same flow can be directed onto the same path based on packet header information using hashing.[1]

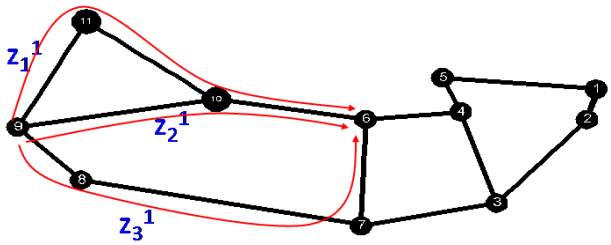


Figure 3: There are three paths between two virtual nodes 9 and 6. The superscript denotes the nodal pair number and the subscript enumerates the paths.

3.2 Node and Link Computations

Consider an instance of DaVinci with N virtual networks, denoted by superscript (k), where $k = 1, 2, ..., N$. Each virtual network consists of virtual nodes that each have a share of the CPU and memory of the corresponding substrate nodes for running *customized* distributed traffic-management protocols. In particular, each virtual node of virtual network (k) (residing at substrate node i) computes a *path rate* $z_j^{(k)i}$ that determines the amount of traffic directed over path j, based on the allocated bandwidth $y_l^{(k)}$.

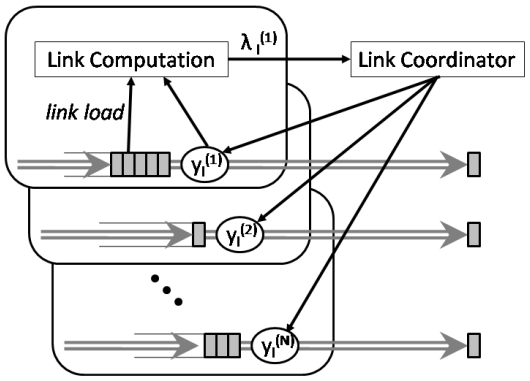


Figure 4: Each substrate link computes bandwidth shares for the virtual links that traverse it.

Substrate link l monitors the load on each virtual link that traverse it to compute $\lambda_l^{(k)}$: an indication of virtual

[1]When splitting ratios change with time, it is better to use *consistent hashing* [33], where packets belonging to the same flow are consistently mapped to the same path, while new flows are placed on a path to best approximate the splitting ratios [34].

network k's satisfaction with its assigned bandwidth on link l. As shown in Figure 4, the substrate feeds $\lambda_l^{(k)}$ to a *link coordinator*. The link coordinator periodically computes the bandwidth shares $\mathbf{y_l}$, taking care to ensure that $\sum_k y_l^{(k)} = C_l$, where C_l is the capacity of the substrate link. At a smaller timescale, the substrate has a *scheduler* for each virtual link that serves incoming data packets based on the bandwidth share $y_l^{(k)}$.

The main building blocks of DaVinci—router virtualization, packet encapsulation, and non-work-conserving scheduler—are readily available today. The main novelty of DaVinci is (i) the way these components are combined and (ii) how the link coordinator adapts the bandwidth shares to ensure that the system maximizes aggregate performance. These aspects of DaVinci derive directly from using optimization theory to model network virtualization, as illustrated in the next section.

4. MODELING AND ANALYSIS OF DAVINCI

In this section, we first present an optimization problem that represents the performance objective and constraints of each virtual network, and then through the technique of primal decomposition derive the bandwidth-share adaptation performed by the substrate. Then we prove stability and optimality of DaVinci under sufficient conditions. We conclude the section with a discussion of DaVinci's benefits and limitations.

4.1 Virtual Networks: Customized Protocols

Optimization theory has been applied to derive a large variety of distributed network protocols as surveyed in [24]. In particular, the many variants of TCP congestion control (running distributedly on end hosts) can be reverse-engineered as implicitly solving an optimization problem. Each optimization problem has a performance objective, which we denote by $U^{(k)}(\cdot)$ for virtual network k. The shape of the performance objective function depends on the particular traffic class. As one example, delay-sensitive traffic may wish to choose paths with low propagation-delay and keep the queues small to reduce queuing delay. As another example, throughput-sensitive traffic may wish to maximize aggregate user utility, as a function of rate. Different utility functions correspond to different degrees of elasticity, user satisfaction, or fairness [35].

The objective function $U^{(k)}(\cdot)$ may depend on both path rates $\mathbf{z}^{(k)}$ and virtual link capacity $\mathbf{y}^{(k)}$. The traffic-management protocols running in each virtual network can be viewed as solving the following optimization problem:

$$\begin{array}{ll} \text{maximize} & U^{(k)}(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}) \\ \text{subject to} & \mathbf{H}^{(k)}\mathbf{z}^{(k)} \preceq \mathbf{y}^{(k)}, \\ & g^{(k)}(\mathbf{z}^{(k)}) \preceq \mathbf{0}, \\ & \mathbf{z}^{(k)} \succeq \mathbf{0}, \\ \text{variables} & \mathbf{z}^{(k)} \end{array} \quad (1)$$

We let $z_j^{(k)i}$ to take on any non-negative value, which implicitly assumes there is flexible splitting between the multiple paths. The objective is subject to a capacity constraint and possibly other convex constraints $g^{(k)}(\mathbf{z}^{(k)})$. The capacity constraint requires the link load to be no more than the allocated bandwidth. To compute the link load, we require a mapping between links and paths:

$$H_{lj}^{(k)i} = \begin{cases} 1, & \text{if path } j \text{ of source-destination pair } i \\ & \text{in virtual network } k \text{ uses link } l \\ 0, & \text{otherwise.} \end{cases}$$

Then $\mathbf{H}^{(k)}\mathbf{z}^{(k)}$ are the virtual link loads, which we also denote $\mathbf{r}^{(k)}$.

Given its own bandwidth shares, each virtual network can run distributed protocol(s) to maximize its own performance objective. A distributed protocol that implicitly maximizes (1) can be derived through *optimization decomposition*. Decomposition is the process of breaking a single optimization problem into smaller problems that are solved at virtual nodes and virtual links respectively, possibly with message passing between them. While the details of the distributed traffic-management protocol depends on the objective function of (1), the protocols all have a similar overall structure. The distributed traffic-management protocols running in each virtual network require only simple computations such as additions and multiplications.

Each virtual edge node updates the path rates based on the local performance objective, the congestion level on its virtual paths, and possibly other constraints. While performance objectives and other constraints can differ, all virtual networks are subject to the bandwidth constraint. Let $s_l^{(k)}$ denote the *congestion price* for link l of virtual network k.[2] In TCP congestion control, the link congestion prices are summed up over each path and interpreted as end-to-end packet loss or delay [36]. The substrate network also uses $s_l^{(k)}$ to determine bandwidth shares, as shown in the following subsection.

In DaVinci, each substrate link updates $s_l^{(k)}$ on behalf of the virtual links as follows:

$$s_l^{(k)}(t+T) = \left[s_l^{(k)}(t) - \beta^{(k)} \left(y_l^{(k)}(t) - r_l^{(k)}(t) \right) \right]^+, \quad (2)$$

where t is time and T is at the same timescale as the longest Round Trip Time (RTT) of the network, *e.g.*,

[2]If we consider $\mathbf{z}^{(k)}$ to be the primal variable, then $\mathbf{s}^{(k)}$ is the dual variable corresponding to the capacity constraint.

$100ms$. Let $r^{(k)} = H^{(k)}z^{(k)}$ be the link load of virtual network k. As seen in (2), $s_l^{(k)}$ is updated for virtual network k based on the difference between the virtual link load and virtual link capacity. The stepsize β moderates the magnitude of the update, and reflects the tradeoff between convergence speed (large stepsizes) and stability (small stepsizes). The $[\ ]^+$ implies that s_l must be nonnegative.

Using these link prices, each virtual network can run its own distributed traffic-management protocols. In particular, each virtual network differs in how it updates the path rates at each virtual router. The path rate update depends on the performance objective of a virtual network. Examples of distributed traffic-management protocol for delay-sensitive traffic with different properties can be found in [26, 37, 38]. For throughput-sensitive traffic, examples of distributed protocols derived from optimization can be found in [25, 39].

4.2 Substrate: Bandwidth Adaptation

The previous subsection demonstrates how each virtual network can maximize its own performance objective by efficiently utilizing the resources assigned to it. This subsection deals with how the substrate should periodically rebalance the allocations between the virtual networks. The goal of the substrate network is to optimize the aggregate utility of all the virtual networks.

$$\begin{array}{ll} \text{maximize} & \sum_k w^{(k)} U^{(k)}(\mathbf{z}^{(k)}, \mathbf{y}^{(k)}) \\ \text{subject to} & \mathbf{H}^{(k)}\mathbf{z}^{(k)} \preceq \mathbf{y}^{(k)}, \forall k \\ & \sum_k \mathbf{y}^{(k)} \preceq \mathbf{C}, \\ & g^{(k)}(\mathbf{z}^{(k)}) \preceq \mathbf{0}, \forall k, \\ & \mathbf{z}^{(k)} \succeq \mathbf{0}, \forall k \\ \text{variables} & \mathbf{z}^{(k)}, \mathbf{y}^{(k)}, \forall k \end{array} \tag{3}$$

where $w^{(k)}$ is the weight the substrate assigns to represent virtual network k's importance.[3] The precise computations are summarized in Figure 5.

First, the substrate determines how satisfied each virtual network is with its allocated bandwidth. Congestion price $s_l^{(k)}$ is one indicator that a virtual network may want more resources. Yet, congestion alone does not capture the entire picture. For example, a delay-sensitive network may have no congestion because it wants to maintain small queues, but that does not necessarily mean the resources should be taken away. Similarly, a throughput-sensitive network may be congested because it wants to keep queues relatively full in order to maintain high bandwidth utilization, but that does not necessarily mean it should be assigned further resources. So the increase in virtual link bandwidth is also based on the differential increase in virtual network k's performance from acquiring more bandwidth. Thus, the virtual network k's satisfaction on link l ($\lambda_l^{(k)}$) is the sum of the congestion price ($s_l^{(k)}$) and the partial derivative of the performance objective relative to the bandwidth allocation ($\partial U^{(k)}(\cdot)/\partial y_l^{(k)}$).

Next, the substrate determines how much bandwidth virtual network k should have on link l, denoted by $v_l^{(k)}$. The substrate increases $v_l^{(k)}$ proportional to virtual network k's satisfaction on link l ($\lambda_l^{(k)}$) and the relative importance $w^{(k)}$ of virtual network k. The stepsize α moderates how much the new bandwidth allocation is relative to the past value. Similar to β in (2), it reflects the tradeoff between convergence speed and stability. The $[\ ]^+$ ensures that all traffic classes are allocated a nonnegative amount of bandwidth.

Finally, since each $v_l^{(k)}$ is adjusted independently, the sum $\sum_k v_l^{(k)}$ might exceed the substrate's capacity C_l on link l. Therefore it is necessary to renormalize the vector $\mathbf{v}_l$, so that the final bandwidth shares $\mathbf{y}_l$ satisfy the capacity constraint. A common way to renormalize is to minimize the Euclidean distance (denoted by $||\cdot||$) between the feasible region $\sum_k y_l^{(k)} \le C_l, y_l^{(k)} \ge 0, \forall k$, and the original point $\mathbf{v}_l$.

Although the bandwidth share computations presented in Figure 5 may look complex, they are simple to do in practice. To perform the bandwidth allocation, each substrate link also needs to know the performance objectives of all virtual networks, which is reasonable if they belong to the same institution. It is important to note the desirable fact that the bandwidth shares at each link are computed with only *local* link loads, thus requiring no message passing. If $U(\cdot)$ is differentiable, $\frac{\partial}{\partial y_l^{(k)}} U^{(k)}(\mathbf{z}^{(k)}, \mathbf{y}^{(k)})$ has a closed form, and the computations at each substrate node are simple. Conse-

---

Let the converged value of $s_l^{(k)}$ be denoted $s_l^{*(k)}$. This is the input for updating $\mathbf{y}_l$.

$$\begin{array}{rcl} \lambda_l^{*(k)}(t) & = & s_l^{*(k)}(t) + \frac{\partial}{\partial y_l^{(k)}} U^{(k)}(\mathbf{z}^{*(k)}, \mathbf{y}^{*(k)}) \\ v_l^{(k)}(t+T_s) & = & [y_l^{(k)}(t) + \alpha(w^{(k)})(\lambda_l^{*(k)}(t))]^+ \\ y_l^{(k)}(t+T_s) & = & \text{argmin}_{y_l^{(k)}} ||y_l^{(k)} - v_l^{(k)}(t+T_s)||, \\ \text{subject to} & & \sum_k y_l^{(k)} \le C_l, y_l^{(k)} \ge 0, \forall k \end{array} \tag{4}$$

where T_s is the time period between bandwidth assignments, usually several orders of magnitude larger than T, *e.g.*, 10s.

Figure 5: The bandwidth share allocation algorithm at link l.

[3] If the substrate wants to give virtual network k strict priority, then $w^{(k)}$ can be assigned a value several orders of magnitudes larger than the other weights.

quently, the processing overhead is modest. In addition, since each substrate link only needs to store the previous values of $\mathbf{s}_l$ and $\mathbf{v}_l$, the additional memory consumption is low.

4.3 Convergence and Optimality

Given that each virtual network is acting independently, the key question is whether the virtual networks, together with the bandwidth share adaptation performed by the substrate network, actually maximize the overall performance objective of the substrate network.

THEOREM 1. *The bandwidth allocation algorithm (4), together with each traffic class solving (1), converges to maximize (3) under the following conditions:*

1. *The problem (3) is a convex optimization.*
2. *The bandwidth allocation is updated after convergence of the primal variables* $\mathbf{z}$ *and the dual variables* $\mathbf{s}$ *in each subproblem (1).*
3. *The parameter* α *in (4) is diminishing with time.*

Proof: Applying standard primal decomposition techniques [40, 41], (3) can be decomposed into N subproblems, each in the form of (1), keeping $\sum_k \mathbf{y}^{(k)} \preceq \mathbf{C}$ as a constraint in the master problem. Each subproblem can be solved for both its original variables $z_l^{(k)}$ and Lagrangian multipliers (technical term for congestion prices) $s_l^{(k)}$ introduced to relax the capacity constraint per link for traffic class k. Then the master problem solves for $\mathbf{y}^{(k)}$ using a gradient update, which corresponds to (4). The weighing factors $w^{(k)}$ do not affect the optima of (1), but they do scale the Lagrangian multiplier as well as $\frac{\partial}{\partial y_l^{(k)}} U^{*(k)}(\mathbf{z}^{*(k)}, \mathbf{y}^{(k)})$ at the optima. Consequently, the weights $w^{(k)}$ in (3) are reflected in (4). From [40, 41], the bandwidth share allocation algorithm (4) converges to the maximum of (3) if the problem is convex and the parameters $\alpha(t)$ are diminishing with time. ■

Theorem 1 has two implications. First, (3) maximizes the aggregate performance across all traffic classes at equilibrium. Second, the adaptive bandwidth allocation algorithm only relies on *local information*.

There are three major assumptions that lead to the theorem: convexity of the optimization problem (maximizing concave function over convex constraint set), timescale of adaptation, and proper selection of α. Concave objective functions apply to most performance objectives [24], so the problem formulation is still broad. Convexity of the capacity constraint is true if traffic can be split flexibly among multiple paths (when they exist). The choice of α will impact the speed of convergence: a smaller α means convergence is slower, but a larger α can cause divergence. We take a closer look at how to select α in the next section.

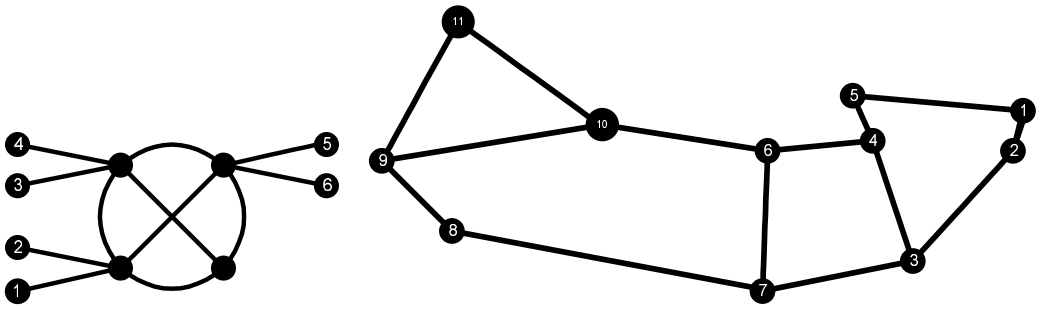
Finally, the timescale of adapting the bandwidth shares must be chosen judiciously. Theorem 1 assumes that each virtual network converges before the next round of bandwidth-share allocation. We observe in simulation that each virtual network takes a few tens of iterations to converge [25, 26], so a timescale separation of a few tens (corresponding to bandwidth share adapted about every ten seconds) should be sufficient. However, the convergence is usually asymptotic, and therefore reaches a value close to equilibrium in only a few iterations, so a smaller timescale separation may be sufficient in practice. In a real operating environment, the traffic demand may change over time, and fast bandwidth adaptation is important for efficient utilization of the links.

5. NUMERICAL EXPERIMENTS

While useful for proving that DaVinci's bandwidth shares converge to optimal values, optimization theory only offers loose bounds on the rate of convergence. In addition, convergence is only guaranteed for diminishing stepsize, while traffic demand is assumed to be constant after time zero. When the stepsize becomes sufficiently small, the network loses its ability to react to changes. In practice, traffic varies over time and the network needs to adapt the bandwidth shares constantly. Therefore, a constant stepsize is more practical. In this section, we use numerical experiments to extend the theoretical results of the previous section. Our experiments consider an example instance of DaVinci with two virtual networks running in parallel. One virtual network carries inelastic delay-sensitive traffic, and the other virtual network carries elastic throughput-sensitive traffic. This section studies the convergence rate of the bandwidth shares and associated sensitivity to parameters. In addition, we examine the evolution of the bandwidth shares when traffic patterns shift and links fail.

5.1 Experimental Set-up

The convergence of DaVinci depends on the convergence of the individual virtual networks (on a small timescale) and the bandwidth shares (on a bigger timescale). Previous work has shown how to tune distributed traffic-management protocols to converge within a few tens of iterations [25]. Assuming the timescale separation between the convergence of congestion price $\mathbf{s}$ and the adaptation of bandwidth shares $\mathbf{y}$, we study the stability of $\mathbf{y}$ using the converged values of $\mathbf{s}$. Consequently, we do not simulate the distributed protocols running inside each virtual network, and instead use the values of $\mathbf{s}$ computed by solving the optimization problems directly. Since the computation of $\mathbf{y}$ does not



(a) Access-Core topology (b) Abilene topology

Figure 6: Two realistic topologies.

involve packet-level operations, we study the evolution of $\mathbf{y}$ in the MATLAB environment.

For a quantitative understanding of the bandwidth share adaptation, we consider a concrete example where virtual network 1 is delay sensitive, and virtual network 2 is throughput sensitive. In our experiments, the performance objectives of each traffic class take on specific forms. Following [26], the delay-sensitive traffic's objective is to minimize average end-to-end delay:

$$\sum_i \sum_j z_j^{i(1)} \sum_l H_{lj}^i (p_l + f(u_l^{(1)}))$$

where p_l is the propagation delay on link l, and $f(\cdot)$ approximates the queueing delay, as a function of the link utilization $u_l^{(1)} = (\mathbf{H}^{(1)}\mathbf{z}^{(1)})_l / y_l^{(1)}$. In our simulations we use $f(u) = p_0 \exp(u)$, where $p_0 = 1ms$. In [26], a piece-wide linear approximation is used. Note that the delay-sensitive traffic is inelastic, so it has a fixed demand rate that needs to be met. Following [25], the throughput-sensitive traffic's objective is to minimize:

$$\sum_i \log(\sum_j z_j^{i(2)}) - q \sum_l \exp(u_l^{(2)})$$

where each source i is maximizing its utility as a logarithmic function of its sending rate. To avoid congestion, each link penalizes high link utilization with an exponential function, and $u_l^{(2)}$ is defined similarly to $u_l^{(1)}$. Following [25], we set $q = 0.5$, to strike a balance between maximizing utility and minimizing congestion.

We experiment with the two networks in Figure 6 in addition to the two-node topology in Figure 1, in order to study realistic topologies with greater path diversity. On the left is a tree-mesh topology, which is representative of a common network structure: a full mesh core with access networks on the edge. Of the twelve possible source-destination pairs, 1–3, 1–5, 2–4, 2–6, 3–5, and 4–6 are chosen, and for each source-destination pair, the three paths with the smallest number of hops are chosen as possible paths. All links have 100 Mbps of bandwidth. The edge links have 5ms of propagation delay while the core links have 10ms of propagation delay. On the right is the Abilene backbone network [42]. Of the many possible source-destination pairs, we choose 1–6, 3–9, 7–11, and 1–11. For each source-destination pair, we choose the four minimum-hop paths as possible paths. All links have 1 Gbps of bandwidth, and we estimate the propagation delays based on the physical distance between nodes. *In general, the same trend is observed across all topologies, so we only show plots of a single topology for each experiment.*

5.2 Setting the Weights w

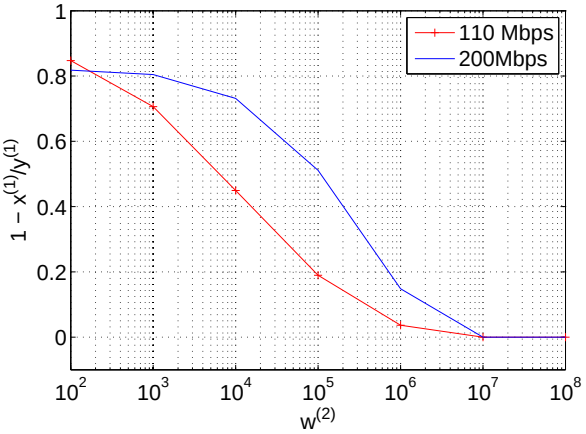


Figure 7: Effect of sweeping $w^{(2)}$ on two-node topology for two delay-sensitive traffic demands.

Before we examine convergence of the bandwidth shares, we discuss how to set the weight $w^{(1)}, w^{(2)}$ of the two virtual networks, using the simple topology in Figure 1 as an example. We fix $w^{(1)} = 1$ and sweep values of $w^{(2)}$ for two different demand values $x^{(1)}$ of the delay-sensitive traffic, 110 Mbps and 200 Mbps. Using an optimization solver, we explicitly solve for the optimal bandwidth share allocations at equilibrium. Since the delay-sensitive traffic penalizes high link utilization, we plot the percent excess bandwidth allocated to the

delay-sensitive network (Figure 7). We observe that when more weight is given to the throughput-sensitive traffic, the delay-sensitive traffic is only allocated the bandwidth it needs to satisfy the demand, i.e., $1 - x^{(1)}/y^{(1)} = 0$. When less weight is given to the throughput-sensitive traffic, however, the delay-sensitive traffic is allocated more bandwidth than it needs on the long-delay link. The excess bandwidth allows delay-sensitive traffic to keep the queues small and thus the end-to-end delay small. The same trend is observed across demand values and topologies. For the remaining experiments, we set $w^{(1)} = 1, w^{(2)} = 10^5$.

5.3 Convergence of Bandwidth Shares

In this experiment, we set the volume of the delay-sensitive traffic to be 110 Mbps for the topology in Figure 1. The ideal bandwidth share assigned to the delay-sensitive traffic is 100 Mbps on the low-delay link (link 1) plus 32 Mbps on the high-delay link (link 2).

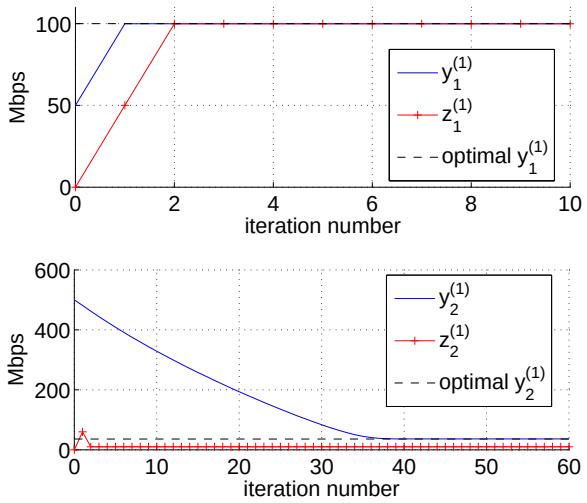


Figure 8: Convergence of bandwidth shares on the two-node topology.

As seen in Figure 8, the delay-sensitive traffic is initially assigned 50% of the bandwidth, thus the y-axis intercepts are 500 Mbps and 50 Mbps respectively. The initial link loads are set to zero, then the link loads jump to 50 Mbps and 60 Mbps respectively after one iteration to satisfy the demand of 110 Mbps. From the top plot in Figure 8, we observe that after one iteration, the delay-sensitive traffic is assigned all of the bandwidth on the low-delay link. This is due to the large difference between the delay properties of the two links. After two iterations, the load is maintained at 100 Mbps on this link. From the bottom plot in Figure 8, we observe that the delay-sensitive traffic's bandwidth share is consistently reduced on the high-delay link until reaching the ideal value.

Similarly, the throughput-sensitive traffic is also initially assigned 500 Mbps on the high-delay link and 50 Mbps on the low-delay link. On the low-delay link, the bandwidth share for the throughput-sensitive traffic drops to 0 Mbps. On the high-delay link, the bandwidth share for the throughput-sensitive traffic is increased to consume most of the idle bandwidth. Overall, the bandwidth is efficiently utilized by the two traffic classes, independent of the initial conditions. Similar behavior is observed across topologies, demand values, and initial values.

5.4 Sensitivity of Stepsize α

The tunable stepsize α controls how much $\mathbf{y}$ reacts to changes in λ (Equation (4)). We study the convergence rate of bandwidth shares for *constant* α, where convergence is defined as being within 0.1% of the optimal bandwidth shares. In particular, we are interested in studying the sensitivity to α.

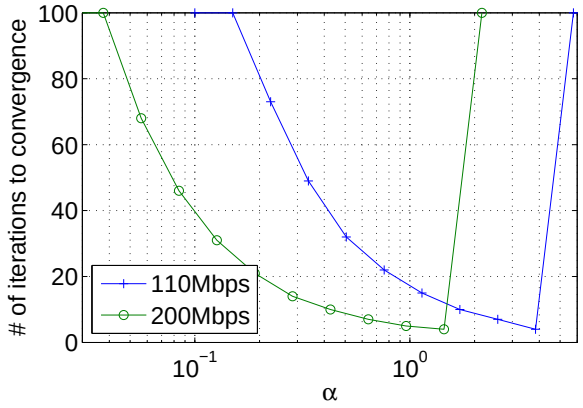


Figure 9: The rate of convergence versus α for Abilene topology.

This experiment sweeps the values of α to record the rate of convergence, for two delay-sensitive traffic demand values (Figure 9). We observe the following: First, the bandwidth shares do converge to their ideal values for constant α. Second, for demand of 110 Mbps, convergence in under 100 iterations occurs for α values between 0.103 and 5. In particular, above an α value of 5, the bandwidth shares may not converge, the reason being as α gets large, there is a tendency to overshoot beyond the feasible region every single iteration. Below an α value of 5, the rate of convergence decreases as we move to smaller values of α. Similar behavior is observed for a demand of 200 Mbps, with the "good" α values being slightly smaller. In practice, simulations should be run to tune α value for fast convergence. If oscillatory behavior is observed, the α value can always be decreased to ensure convergence. For our remaining experiments, we set $\alpha = 0.2$.

5.5 Traffic and Topology Dynamics

Another caveat of Theorem 1 is that the traffic demand and substrate topology are fixed. In this subsection, we examine how the bandwidth shares shift when traffic demand changes and links fail.

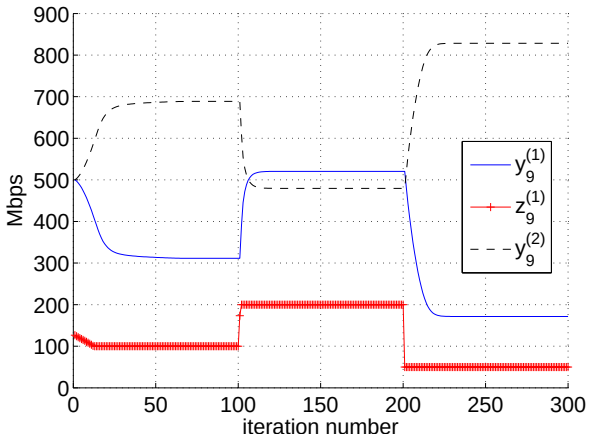


Figure 10: Effect of changing traffic demand on Abilene topology.

First we consider the impact of the delay-sensitive traffic increasing from 100 Mbps to 200 Mpbs at iteration 100, and then dropping to 50 Mbps at iteration 200. In Figure 10, we select to plot bandwidth shares for link 9 of the Abilene topology which carries a single flow of delay-sensitive traffic. When the delay-sensitive traffic volume increases, bandwidth is taken from the throughput-sensitive traffic in order to satisfy the demand of the delay-sensitive traffic. When the delay-sensitive traffic volume decreases, idle bandwidth from the delay-sensitive traffic is assigned to the throughput-sensitive traffic.

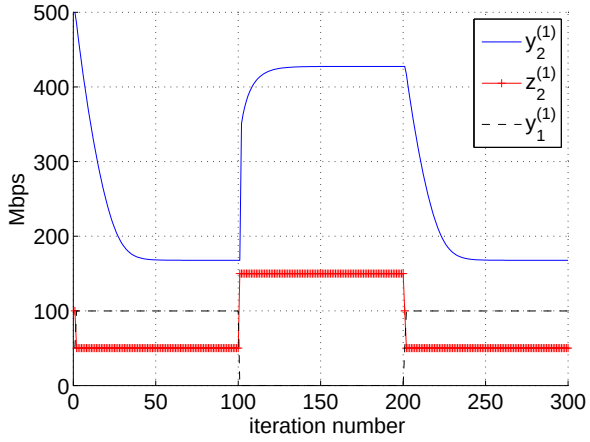


Figure 11: Effect of low-delay link failing in the 2-node topology.

Second, we consider the impact of the low-delay link failing at iteration 100 in the 2-node topology. The delay-sensitive traffic demand is set at 150 Mbps. In Figure 11, we plot the path rates of the delay-sensitive traffic. Initially, the path rates are split between the low-delay link and the high-delay link. We observe that immediately after the failure, the high-delay link is carrying 150 Mbps. In addition, the delay-sensitive traffic gains about 250 Mbps extra bandwidth share on the high-delay link when it loses the low-delay link. After the failed link recovers at iteration 200, the traffic patterns and bandwidth shares return quickly to their original values.

6. CONCLUSIONS AND FUTURE WORK

We present DaVinci: a simple, flexible, and efficient architecture for supporting multiple traffic classes. In DaVinci, each virtual network runs customized traffic-management protocols, and a per-link bandwidth coordinator adjusts bandwidth shares across virtual networks. The substrate computes bandwidth shares entirely based on local link loads, imposing no message-passing overhead. A non-work-conserving shaper at each link ensures that the virtual networks are isolated between bandwidth share computations.

It is interesting to note that primal decomposition is used to derive an adaptive virtualization architecture. This is in contrast to the standard usage of dual decomposition for congestion control protocols [24]. Aided by optimization theoretic tools, we prove that the bandwidth shares converge for diminishing stepsize, while our numerical experiments demonstrate that the bandwidth shares converges *quickly* for a range of *constant* stepsizes. In addition, our experiments show that the bandwidth shares adapt quickly to traffic shifts and link failures.

We believe that making network virtualization adaptive is promising as a future architecture, though many open challenges remain. For example, DaVinci assumes each virtual network runs optimal protocols, but simpler protocols that compromise on optimality may be preferable in practice. One interesting direction is to quantify the tradeoff between stability, optimality, and overhead. As another example, DaVinci assumes each virtual network makes efficient usage of its assigned bandwidth, but virtual networks may exhibit greedy and malicious behaviors, especially if they are controlled by multiple parties. One possibility is to introduce economic incentives that will encourage efficient behavior, such as associating the link congestion price with payments.

Acknowledgments

We would like thank Dave Andersen, Umar Javed, Martin Suchara, Vytautas Valancius, Dan Wendlandt, and

Yaping Zhu for their feedback on earlier drafts of this paper. This work has been supported in part by NSF grants CNS-0519880 and CCF-0448012, and Cisco grant GH072605.

7. REFERENCES

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, pp. 34–41, April 2005.

[2] J. S. Turner and D. E. Taylor, "Diversifying the Internet," in *Proc. IEEE GLOBECOM*, November 2005.

[3] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *ACM SIGCOMM Computer Communication Review*, pp. 61–64, January 2007.

[4] D. McPherson, D. O'Leary, D. Ward, E. Brendel, O. Aruj, P. Agarwal, R. Hartani, and S. Poretsky, "Core Network Design and Vendor Prophecies," in *Proc. NANOG*, June 2003.

[5] "Juniper Networks: Intelligent Logical Router Service." `http://www.juniper.net/solutions/literature/white_papers/200097.pdf`.

[6] "Cisco opening up IOS," December 2007. `http://www.networkworld.com/news/2007/121207-cisco-ios.html`.

[7] "Partner Solution Development Platform Opens Opportunity to Accelerate the Pace of Network Innovation with JUNOS Software," December 2007. `http://www.juniper.net/company/presscenter/pr/2007/pr-071210.html`.

[8] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview." RFC 1633, June 1994.

[9] S. Chen and K. Nahrstedt, "An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions," *IEEE Network Magazine*, November/December 1998.

[10] F. Kuipers, T. Korkmaz, M. Krunz, and P. V. Mieghem, "Overview of constraint-based path selection algorithms for QoS routing," *IEEE Communication Magazine*, pp. 50–55, December 2002.

[11] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services." RFC 2475, October 1998.

[12] I. Matta and A. U. Shankar, "Type-of-service routing in datagram delivery systems," *IEEE J. on Selected Areas in Communications*, vol. 13, pp. 1411–1425, October 1995.

[13] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF." RFC 4915, June 2007.

[14] K.-W. Kwong, R. Guerin, A. Shaikh, and S. Tao, "Improving service differentiation in IP networks through dual topology routing," in *Proc. CoNEXT*, December 2007.

[15] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proc. Symposium on Operating Systems Principles*, October 2001.

[16] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Networking*, December 2006.

[17] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashock, and J. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *Proc. Operating Systems Design and Implementation*, October 2000.

[18] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the Internet using an overlay multicast architecture," in *Proc. ACM SIGCOMM*, August 2001.

[19] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz, "OverQoS: An overlay based architecture for enhancing Internet QoS," in *Proc. Networked Systems Design and Implementation*, September 2004.

[20] R. Keralapura, C.-N. Chuah, N. Taft, and G. Iannaccone, "Race conditions in coexisting overlay networks," *IEEE/ACM Trans. Networking*, vol. 16, no. 1, pp. 1–14, 2008.

[21] E. Rosen and Y. Rekher, "BGP/MPLS VPNs." RFC 2547, March 1999.

[22] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI Veritas: Realistic and controlled network experimentation," in *Proc. ACM SIGCOMM*, August 2006.

[23] J. S. Turner, "A proposed architecture for the GENI backbone platform," in *Proc. Architecture for Networking and Communications Systems*, 2006.

[24] M. Chiang, S. H. Low, R. A. Calderbank, and J. C. Doyle, "Layering as optimization decomposition," *Proceedings of the IEEE*, January 2007.

[25] J. He, M. Suchara, M. Bresler, M. Chiang, and J. Rexford, "Rethinking Internet Traffic Management: From Multiple Decompositions to A Practical Protocol," in *Proc. CoNEXT*, December 2007.

[26] U. Javed, M. Suchara, J. He, and J. Rexford, "Multipath protocol for delay-sensitive traffic," in *Proc. the First International Conference on COMmunication Systems and NETworkS (COMSNETS)*, January 2009.

[27] J. He and J. Rexford, "Towards Internet-wide Multipath Routing," *IEEE Network Magazine*, March 2008.

[28] J. He, M. Suchara, M. Bresler, J. Rexford, and M. Chiang, "From Multiple Decompositions to TRUMP: Traffic Management Using Multipath Protocol," March 2008. in submission to IEEE/ACM Transactions on Networking, `www.cs.princeton.edu/~jrex/papers/conext07-long.pdf`.

[29] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, "OCALA: An architecture for supporting legacy applications over overlays," in *Proc. Networked Systems Design and Implementation*, May 2006.

[30] H. V. Madhyatha, A. Venkataramani, A. Krishnamurthy, and T. Anderson, "Oasis: An overlay-aware network stack," in *Proc. ACM SIGOPS*, January 2006.

[31] "GENI opt-in working group." `http://geni.net/wg/opt-in-wg.html`.

[32] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture." RFC 3031, January 2001.

[33] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proc. ACM STOC*, June 1997.

[34] I. Avramopoulos, J. Rexford, D. Syrivelis, and S. Lalis, "Counteracting discrimination against network traffic." Princeton University Computer Science Technical Report TR-794-07, August 2007.

[35] J. Mo and J. C. Walrand, "Fair End-to-end Window-based Congestion Control," *IEEE/ACM Trans. Networking*, vol. 8, pp. 556–567, October 2000.

[36] S. H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Trans. Networking*, vol. 11, pp. 525–536, August 2003.

[37] J. Pongsajapan and S. Low, "Reverse engineering TCP/IP-like networks using delay-sensitive utility functions," in *Proc. IEEE INFOCOM*, May 2007.

[38] Y. Li, M. Chiang, A. R. Calderbank, and S. Diggavi, "Optimal delay-rate-reliability tradeoff in networks with composite links," in *Proc. IEEE INFOCOM*, May 2007.

[39] X. Lin and N. B. Shroff, "Utility Maximization for Communication Networks with Multi-path Routing," *IEEE Trans. Automatic Control*, vol. 51, May 2006.

[40] L. S. Lasdon, *Optimization Theory for Large Systems*. Macmillian, 1970.

[41] D. P. Bersekas, *Nonlinear Programming*. Athena Scientific, second ed., 1999.

[42] Abilene Backbone. `http://abilene.internet2.edu/`.