Tesseract: A 4D Network Control Plane

Hong Yan[†], *David A. Maltz*[‡], *T. S. Eugene Ng*[§], *Hemant Gogineni*[†], *Hui Zhang*[†], *Zheng Cai*[§]
[†]*Carnegie Mellon University* [‡]*Microsoft Research* [§]*Rice University*

Abstract

We present Tesseract, an experimental system that enables the *direct control* of a computer network that is under a single administrative domain. Tesseract's design is based on the 4D architecture, which advocates the decomposition of the network control plane into *decision*, *dissemination*, *discovery*, and *data* planes. Tesseract provides two primary abstract services to enable direct control: the *dissemination service* that carries opaque control information from the network decision element to the nodes in the network, and the *node configuration service* which provides the interface for the decision element to command the nodes in the network to carry out the desired control policies.

Tesseract is designed to enable easy innovation. The neighbor discovery, dissemination and node configuration services, which are agnostic to network control policies, are the only distributed functions implemented in the switch nodes. A variety of network control policies can be implemented outside of switch nodes *without* the need for introducing new distributed protocols. Tesseract also minimizes the need for manual node configurations to reduce human errors. We evaluate Tesseract's responsiveness and robustness when applied to backbone and enterprise network topologies in the Emulab environment. We find that Tesseract is resilient to component failures. Its responsiveness for intra-domain routing control is sufficiently scalable to handle a thousand nodes. Moreover, we demonstrate Tesseract's flexibility by showing its application in joint packet forwarding and policy based filtering for IP networks, and in link-cost driven Ethernet packet forwarding.

1 Introduction

We present Tesseract, an experimental system that enables the *direct control* of a computer network that is under a single administrative domain. The term direct control refers to a network control paradigm in which a *decision element* directly and explicitly creates the forwarding state at the network nodes, rather than indirectly configuring other processes that then compute the forwarding state. This paradigm can significantly simplify network control.

In a typical IP network today, the desired control policy of an administrative domain is implemented via the synthesis of several indirect control mechanisms. For example, load balanced best-effort forwarding may be implemented by carefully tuning OSPF link weights to indirectly control the paths used for forwarding. Inter-domain routing policy may be indirectly implemented by setting OSPF link weights to change the local cost metric used in BGP calculations. The combination of such indirect mechanisms create subtle dependencies. For instance, when OSPF link weights are changed to load balance the traffic in the network, inter-domain routing policy may be impacted. The outcome of the synthesis of indirect control mechanisms can be difficult to predict and exacerbates the complexity of network control [1].

The direct control paradigm avoids these problems because it forces the dependencies between control policies to become explicit. In direct control, a logically centralized entity called the decision element is responsible for creating all the state at every switch. As a result, any conflicts between the policy objectives can be detected at the time of state creation. With today's multiple independent and distributed mechanisms, these conflicts often only appear *in vivo* after some part of the configuration state has been changed by one of the mechanisms.

The direct control paradigm also simplifies the switch functionality. Because algorithms making control decisions are no longer run at switches, the only distributed functions to be implemented by switches are those that discover the neighborhood status at each switch and those that enable the control communications between the decision element and the switches. Thus, the switch software can be very light-weight. Yet, sophisticated control algorithms can be easily implemented with this minimal set of distributed functions.

The Tesseract (a tesseract is a 4-dimensional cube) system is based on the 4D architecture that advocates the decomposition of the network control plane into the *decision*, *dissemination*, *discovery*, and *data* planes. Tesseract implements two services to enable direct control:

Dissemination service: The dissemination service provides a logical connection between decision element and network switch nodes to facilitate direct control. The dissemination service only assumes network nodes are pre-configured with appropriate keys and can discover and communicate with direct physical neighbors. The dissemination service thus enables plug-and-play bootstrapping of the Tesseract system.

Node configuration service: The node configuration service provides an abstract packet lookup table interface that hides the details of the node hardware and software

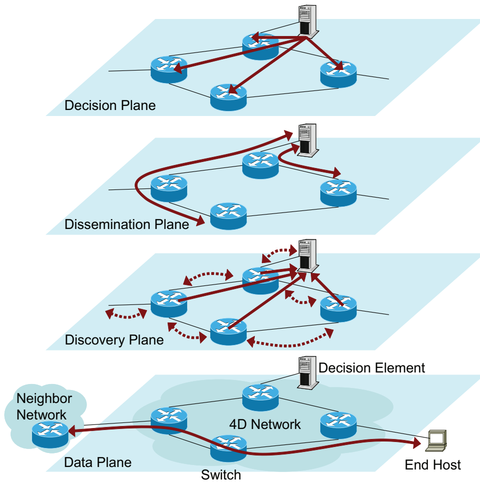

Figure 1: The 4D architectural concepts.

from the decision element. Each table entry contains a packet matching rule and the corresponding control actions. The decision element issues commands to the node configuration service through the logical connection provided by the dissemination service.

This paper presents the design, implementation, evaluation, and demonstration of the Tesseract system. To guide our design, we explicitly select a set of goals and devise solutions to address them. We deploy Tesseract on Emulab [2] to evaluate its performance. We show how Tesseract can rapidly react to link, node, and decision element failures and efficiently re-configure network switches in response. Also, micro-benchmark experiments show that the system can easily handle the intra-domain routing control for a thousand-node network. We then demonstrate Tesseract's flexibility by showing its applications in joint packet forwarding and policy based filtering in IP networks, and in link cost driven Ethernet packet forwarding.

2 From the 4D Architecture to Tesseract Design Goals

This section explains the key concepts in the 4D architecture. Since the 4D architecture describes a very large design space, we present the design goals we used to guide our design of the specific Tesseract system.

2.1 The 4D Architectural Concepts

The 4D architecture advocates decomposing the network control plane into four conceptual components: *decision*, *dissemination*, *discovery*, and *data* planes. These conceptual components are illustrated in Figure 1 and are explained below. For an in-depth discussion of the 4D architecture, please refer to [3].

Data plane: The data plane operates in network switches and provides user services such as IPv4, IPv6, or Ethernet packet forwarding. The actions of the data plane are based on the state in the switches, and this state is controlled solely by the decision plane. Example state in switches includes the forwarding table or forwarding information base (FIB), packet filters, flow-scheduling weights, queue-management parameters, tunnels and network address translation mappings, etc. The arrow in the figure represents an end-to-end data flow.

Discovery plane: Each switch is responsible for discovering its hardware capabilities (e.g., what interfaces are on this switch and what are their capacities? How many FIB entries can the switch hold?) and its physical connectivity to neighboring switches. A border switch adjacent to a neighboring network is also responsible for discovering the logical connectivity to remote switches that are reachable via that neighbor network (in today's environment, this may be implemented by an eBGP session). The dotted arrows in the figure represent the local communications used for discovering connectivity. The information discovered is then reported to the decision element in the decision plane via the logical connections maintained by the dissemination plane. The solid arrows in the figure represent these reporting activities. For backward compatibility, end hosts do not explicitly participate in the discovery plane.

Dissemination plane: The dissemination plane is responsible for maintaining robust logical channels that carry control information between the decision element and the network switches. The arrows in the figure represent the paths used by the logical channels. While control information may traverse the same set of physical links as the data packets in the data plane, the dissemination paths are maintained separately from the data paths so they can be operational without requiring configuration or successful establishment of paths in the data plane. In contrast, in today's networks, control and management information is carried over the data paths, which need to be established by routing protocols before use. This creates a circular dependency.

Decision plane: The decision plane consists of a logically centralized decision element that makes *all* decisions driving network control, such as reachability, load balancing, access control, and security. The decision element makes use of the information gathered by the discovery plane to make decisions, and these decisions are sent as commands to switches via the dissemination plane (shown as arrows in the figure). The decision element commands the switches using the node configuration service interface exposed by the network switches. While logically centralized as a single decision element, in practice multiple redundant decision elements may be used for resiliency.

2.2 Tesseract Design Goals

Tesseract is based on the general 4D architectural concepts, but these concepts admit a wide variety of design choices. We used the following goals to guide our decisions while designing Tesseract, and these goals can be roughly grouped into three categories. The first category concerns system performance and robustness objectives:

Timely reaction to network changes: Planned and unplanned network changes, such as switch maintenance and link failures, can cause traffic disruption. Tesseract should be optimized to react to network changes quickly and minimize traffic disruption.

Resilient to decision plane failure: Tesseract should provide built-in support for decision plane redundancy so that it can survive the failure of a decision element.

Robust and secure control channels: The logical channels for control communications maintained by Tesseract should continue to function in the presence of compromised switches, decision elements or failed links/nodes.

The next set of goals concern making Tesseract easy to deploy:

Minimal switch configuration: The Tesseract software on each switch should require no manual configuration prior to deployment except for security keys that identify the switch. We do, however, assume that the underlying switch allows Tesseract to discover the switch's properties at run-time.

Backward compatibility: Tesseract should require no changes to the end host software, hardware, or protocols. Thus, Tesseract can be deployed as the network control system transparently to the end users.

The final set of goals concerns making Tesseract a flexible platform:

Support diverse decision algorithms: Tesseract should provide a friendly platform on which diverse algorithms can be easily implemented to control networks.

Support multiple data planes: Tesseract should support heterogeneous data plane protocols (e.g., IP or Ethernet). Thus, the system should not assume particular data plane protocols and the dissemination service should be agnostic to the semantics of the control communications.

3 Design and Implementation of Tesseract

In this section, we present the design and implementation of Tesseract. We first provide an overview of the software architecture, and then discuss each component of the system in detail.

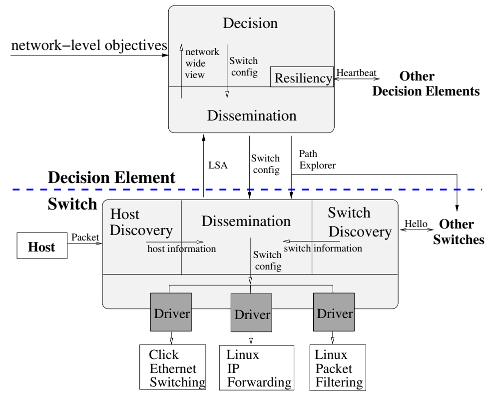

Figure 2: High-level overview of Tesseract.

3.1 System Overview

The Tesseract system is composed of two applications implemented on Linux. These applications are called the `Switch` and the Decision Element (`DE`). Figure 2 illustrates the software organization of these applications.

The discovery plane implementation currently deals only with neighbor node discovery. It includes two modules, one for discovering hosts connected to the switch and the other for discovering other switches. The switch discovery module exchanges hello messages with neighbor switches to detect them, and creates Link State Advertisements (LSAs) that contain the status of its interfaces and the identities of the switches connected to the interfaces. The generated LSAs are reported to `DE` via the dissemination plane. To avoid requiring changes to hosts, the discovery plane identifies what hosts are connected to a switch by snooping the MAC and IP addresses on packets received on the interfaces that are not connected to another switch.

The dissemination plane is cooperatively implemented by both `Switch` and `DE`. The dissemination service is realized by a distributed protocol that maintains robust logical communication channels between the switches and decision elements.

`Switch` leverages existing packet forwarding and filtering components to implement the data plane. `Switch` interacts with `DE` in the decision plane through the node configuration service interface. The interface is implemented by data plane drivers, which translate generic configuration commands from `DE` into specific configurations for the packet forwarding and filtering components.

`DE` implements the discovery, dissemination and decision planes. The discovery and dissemination plane functions are as outlined above. The decision plane constructs an abstract network model from the information reported by the switches and computes switch configuration commands for all the switches based on the specific

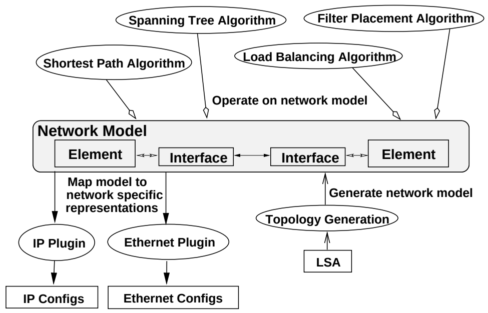

Figure 3: The network model separates general purpose algorithms from network specific mechanisms.

decision algorithm used. The computed switch configuration commands are sent to the switches via the dissemination service.

3.2 Decision Plane: Versatility, Efficiency and Survivability

The decision plane implements a platform for the deployment of network control algorithms. In addition, it implements mechanisms that enable the replication of the decision logic among multiple decision elements (DEs) so that DE failures can be tolerated.

Support diverse network control algorithms: In designing the decision plane, our focus is not to hard-wire sophisticated network decision logics into the system. Instead, our goal is to make the decision plane a friendly platform where any network control algorithm can be easily integrated and used to control any suitable network technology. Towards this end, we introduce an abstract network model to separate generic network control algorithms (e.g., shortest path computation, load balancing) from network specific mechanisms (e.g., IP, Ethernet).

Figure 3 illustrates the abstract network model. The model consists of node element and link interface objects, and is constructed from information discovered and reported by switches (e.g. LSA) through the dissemination service. Operating on this model, Tesseract currently implements four generic algorithms: incremental shortest path, spanning tree, joint packet filter/routing (Section 5.1), and link-cost-based traffic engineering (Section 5.2). Finally, technology-specific plug-ins translate the general control decisions into network specific configuration commands that are sent to switches via the dissemination service. These commands are then processed by the node configuration service at individual switches.

As an example, we implement an incremental shortest path algorithm [4] on the abstract network model, and the same code can be used to generate either IP routing table in IP networks or Ethernet forwarding entries in Ethernet.

Efficient network event processing: The DE must efficiently handle multiple simultaneous network changes, which the DE will receive as events communicated over the dissemination plane. We chose a different event processing architecture than that used in typical implementation of OSPF, where a hold-down timer is used to delay the start of route recomputation after an event arrives to force the batching of whatever events arrive during the hold-down window.

Instead, the Tesseract DE uses a *push timer*. The DE runs a decision thread that processes all queued events to update the network-wide view, starts the push timer as a deadline for pushing out new switch configuration commands, and then enters its computation cycle. After the compution of new forwarding state finishes, the DE will immediately push out the new commands if the push timer has expired, if the event queue is empty, or if the queued events do not change the network-wide view used in the computation. Otherwise, the DE will dequeue all pending events and re-compute.

We use a push timer instead of a fixed hold-down timer for two reasons. In the common case where a single link fails, the push timer avoid unnecessary waiting. The first LSA announcing the failure starts the route recomputation, and subsequent LSAs announcing the same failure do not change the network-wide view and so are ignored. In the less common case of multiple failures, a push timer may result in recomputation running more than once for the same event. However, since recomputation has latency on the same order as typical hold-down timers and DEs are unlikely to be CPU-limited, it is reasonable to trade off extra computation for faster reconvergence.

The DE also records the state that has been pushed to each switch and uses delta-encoding techniques to reduce the bandwidth required for sending configuration commands to the switches. Acknowledgments between DE and the node configuration service on each switch ensure the delta-encoded commands are received.

Provide decision plane resiliency: Our decision plane copes with DE failures using hot-standbys. At any time a single master DE takes responsibility for configuring the network switches, but multiple DEs can be connected to the network. Each standby DE receives the same information from the switches and performs the same computations as the master. However, the standby DEs do not send out the results of their computations.

The master DE is selected using a simple leader election protocol based on periodic DE heartbeats that carry totally ordered DE priorities. Each DE has a unique priority, and at boot time it begins flooding its priority with a heartbeat message every heartbeat period (e.g., 20 ms). Each DE listens for heartbeats from other DEs for at least five times the heartbeat period (we assume that 5 times heartbeat period will be greater than the maximum latency of a packet crossing the network). After this waiting period, the DE that has the highest priority among all received heartbeats decides to be the master and begins sending commands to switches. When the master DE receives a heartbeat from a DE with a higher priority than its own, it immediately changes into a standby DE and ceases sending commands to switches. A DE also periodically floods a path explorer message, which has the effect of triggering switches to reply with their current state. In this way, a new DE can gather the latest switch state. Switches simply process commands from any DE. Authentication is handled by the dissemination plane and is discussed next.

3.3 Dissemination Plane: Robustness and Security

The goal of the dissemination plane is to maintain robust and secure communication channels between each DE and the switches. With respect to robustness, the dissemination plane should remain operational under link and node failure scenarios. With respect to security, the network should remain operational when a switch or even a DE is compromised.

Observing that the traffic pattern in dissemination plane is few-to-many (switches communicate not with each other, but only with the DEs), we adopt an asymmetric design where the dissemination module at a DE node implements more functionality than the dissemination module at a switch.

Dissemination plane design overview: Tesseract's dissemination plane is implemented using source routes. Each control message is segmented into dissemination frames, and each frame carries in its header the identity of the source, destination, and the series of switches through which it must pass. We choose a source routing solution because: (1) It requires the minimal amount of routing state and functionality in each switch. Each switch needs only to maintain the routes to the DEs. (2) Source routes provide very flexible control over routing, as a different path can be specified for each destination, making it easy to take advantage of preferred paths suggested by the decision plane. (3) Combining source routing with the few-to-many communication pattern enable us to design a dissemination plane with desirable security properties, as discussed below. To protect control communications from user data traffic, the queuing of dissemination frames is separate from user data traffic and dissemination frames have higher transmission priority. To protect the source-routes from being misused by adversaries inside the network, we encrypt them at each hop before they are forwarded.

Threat model: Tesseract is designed to cope with the following threats: (1) Adversaries can compromise a switch, gaining full control over it including the ability to change the way dissemination packets are forwarded; (2) A compromised switch can piggyback data on packets to collude with other compromised switches downstream; (3) A compromised switch can peek into dissemination plane data to try to learn the network topology or location of critical resources; and (4) Adversaries can compromise a DE and use it to install bad forwarding state on the switches.

Bootstrapping security: The Tesseract trust model is based on a *network certificate* (i.e., a signed public key for the network) — all the other keys and certificates are derived from the network certificate and can be replaced while network continues operating. Switches will accept commands from any DE holding a DE certificate that is signed by the network certificate. The private key of the network certificate is secret-shared [5] among the DEs, so that any quorum of DEs can cooperate to generate a new DE certificate when needed.

When a switch is first deployed, the network certificate and a DE certificate are installed into it. This is done by plugging a USB key containing the certificates into each switch or as part of the default factory configuration of the switch before it is deployed in the field. The switch then constructs a DeviceID, which can be as simple as a randomly generated 128-bit number, and a private/public key pair. The switch stores the network and DE certificates, its DeviceID, and its key pair into nonvolatile memory. The switch then encrypts the information with the public key of the DE, and writes it back onto the USB key. When the USB key is eventually inserted into a DE, the DE will have a secret channel to each device and a list of the valid DeviceIDs. As each switch communicates with a DE for the first time, it uses ISAKMP [6] and its private/public keys to establish a shared-secret key known only by that switch and the DE. All subsequent dissemination plane operations use symmetric cryptography.

Computing dissemination plane routes: Dissemination plane routes are computed by each decision element flooding a path explorer message through the network. To ensure fast recovery from link failures, the path explorer is sent periodically every 20 ms in our prototype, and can be triggered by topology updates.

Onion-encryption (or encapsulated encryption) is used in path explorers to support dissemination security. The DE initiates the path explorer by embedding its DeviceID as the source route and flooding it over all its ports. When a switch receives the path explorer, it (1) optionally verifies the route to the DE contained in the path explorer; (2) records the source route; (3) encrypts the existing source route using the secret key it shares with the DE that sent the path explorer; (4) appends its own DeviceID to the path explorer in plain text; and (5) floods the path explorer out its other interfaces. Path explorers carry sequence numbers so that switches can avoid unnecessary re-flooding.

To send data to a DE, a switch uses the encrypted source route it recorded from a path explorer sent by that DE. When an upstream switch receives the message, it decrypts the source-route using its secret key. This reveals the ID of the next hop switch along the path to the DE. By successive decryption of the source route by the on-route switches, dissemination plane packets are delivered to the DE. Since the DE knows the secret-key of every switch, it can construct an onion-encrypted route to any switch it desires.

As part of the negotiation of its secret key over ISAKMP, each switch learns whether it is required to perform the optional source route verification in step (1) before forwarding a path explorer. If verification is required, the switch first checks a cache of source routes from that DE to see if the source route has already been verified. If the source route is not known to be valid, the switch forwards the source route to the DE in a signed VERIFY packet. Since the DE knows the secret keys of all the switches, it can iteratively decrypt the source route and verify that each hop corresponds to link it has learned about in an LSA. Once verified, the DE sends a VERIFYOK message to the switch using the extracted source route, confirming the validity of the route. The DE confirmation is signed with a HMAC computed using the secret key of the destination switch to prevent it from being tampered or forged.

Security properties: The optional verification step exposes a classic trade-off between security and performance. In Tesseract, we provide a dissemination plane with two different levels of security. The network operator can choose the semantics desired.

The basic security property is that a compromised switch cannot order other switches to install invalid forwarding state or forge LSAs from other switches. This is achieved by each switch having a secret key shared only with the DE.

If path explorers are *not* verified before being forwarded, a compromised switch can forge path explorers that artificially shorten its distance to the DE and attract dissemination plane traffic from other switches (e.g., so the attacker can drop or delay the traffic). Compromised switches can also communicate with each other over the dissemination plane to coordinate attacks.

If path explorers *are* verified before being forwarded, a compromised switch cannot lie about its distance to the DE. Also, compromised switches are prevented from communicating arbitrarily over the dissemination plane unless they are directly connected. This is because the DE will not validate a source route that originates and ends at switches. A switch also cannot discover the identity or connectivity of another switch that is two or more hops away. This prevents attackers from identifying and targeting critical resources in the network.

The cost of the extra security benefits provided by verifying source routes is the extra latency during reconvergence of the dissemination plane. If a link breaks and a switch receives path explorers over a source route it has not previously verified, it must wait a round-trip time for the verification to succeed before the switches downstream can learn of the new route to the DE. One approach to minimize this penalty is for the DE to prepopulate the verified source route tables of switches with the routes that are most likely to be use in failure scenarios. A triggered path explorer flooded by the DE in response to link failure will then quickly inform each switch which preverified routes are currently working.

Surviving DE compromise: As a logically centralized system, if a DE were compromised, it could order switches to install bad forwarding state and wreck havoc on the data plane. However, recovery is still possible. Other DEs can query the forwarding state installed at each switch and compare it to the forwarding state they would have installed, allowing a compromised or misbehaving DE to be identified. Because the private key of the network certificate is secret-shared, as long as a quorum of DEs remain uncompromised they can generate a new DE certificate and use the dissemination plane to remotely re-key the switches with this new DE certificate.

Notice that while a compromised DE can totally disrupt data plane traffic, it *cannot* disrupt the dissemination traffic between other DEs and the switches. This is one of the benefits of having control traffic traversing a secured dissemination plane that is logically separate from paths traversed by data packets. Once re-keyed, the switches will ignore the compromised DEs.

As a point of comparison, in today's data networks recovering from the compromise of a management station is hard as the compromised station can block the uncompromised ones from reaching the switches. At the level of the control plane, the security of OSPF today is based on a single secret key stored in plain-text in the configuration file. If any switch is compromised, the key is compromised, and incorrect LSAs can be flooded through the network. The attacker could then DoS all the switches by forcing them to continuously rerun shortest path computation or draw traffic to itself by forging LSAs. Since a distributed link-state computation depends on all-to-all communications among the switches, one alternative to using a single shared key is for each switch to negotiate a secret key with every other switch. Establishing this $O(n^2)$ mesh of keys requires every switch to know the public key of every other switch. Both key establishment and revocation are more complex when compared to the direct control paradigm of Tesseract.

3.4 Discovery Plane: Minimizing Manual Configurations

The discovery plane supports three categories of activities: (1) providing the DE with information on the state of the network; (2) interacting with external networks and informing the DE of the external world; and (3) bootstrapping end hosts into the network.

Gathering local information: Since misconfiguration is the source of many network outages, the 4D architecture eliminates as much manually configured state as possible. In the long term vision, the switch hardware should self-describe its capabilities and provide run-time information such as traffic load to the discovery plane. The current Tesseract implementation supports the discovery of physical switch neighbors via periodic HELLO message exchanges. Switches are identified by the same DeviceID used in the dissemination plane.

Interacting with external networks: The DE directs the border switches that peer with neighbor networks to begin eBGP sessions with the neighbor switches. Through this peering, the DE discovers the destinations available via the external networks. Rather than processing the BGP updates at the switches, the switches simply report them to the DE via the dissemination service, and the DE implements the decision logic for external route selection. The DE sends the appropriate eBGP replies to the border switches, as well as configuring external routes directly into all the switches via the dissemination service. RCP [7] has already demonstrated that the overall approach of centralized BGP computation is feasible, although they continue to use iBGP for backward compatibility with existing routers.

It is important to note that an internal link or switch failure in a Tesseract network does not lead to massive updates of external routes being transmitted from the DE to the switches. The reason is that external routes identify only the egress points. External and internal routes are maintained in two separate tables and are combined locally at switches to generate the full routing table. This is identical to how OSPF and BGP computed routes are combined today. In general, an internal link or switch failure does not change external routes and thus no update to them is necessary.

Bootstrapping end hosts: For backward compatibility, end hosts do not directly participate in Tesseract discovery plane.

In networks running IP, the discovery plane acts as a DHCP proxy. The DE configures each switch to tunnel DHCP requests to it via the dissemination service. Whenever a host transmits a DHCP request, the DE learns the MAC address and the connection point of the host in the network. The DE can then assign the appropriate IP address and other configuration to the host.

In networks operating as a switched Ethernet LAN, the discovery plane of a switch reports the MAC address and the connection point of a newly appeared end host to the DE. The DE then configures the network switches appropriately to support the new host. Section 5.2 describes how we use Tesseract to control a switched Ethernet LAN and provide enhancements.

3.5 Data Plane: Support Heterogeneity

The data plane is configured by the decision plane via the node configuration service exposed by the switches. Tesseract abstracts the state in the data plane of a switch as a lookup table. The lookup table abstraction is quite general and can support multiple technologies such as the forwarding of IPv4, IPv6, or Ethernet packets, or the tunneling and filtering of packets, etc.

Tesseract's data plane is implemented using existing Linux kernel and Click components. For each component, we provide a driver to interface the component with the Tesseract decision plane as shown in Figure 2. The drivers model the components as lookup tables and expose a simple `WriteTable` interface to provide the node configuration service to the DE. For example, when the DE decides to add or delete an IP routing or Ethernet forwarding table entry, it sends a `add_table_entry` or `delete_table_entry` command through the `WriteTable` interface, and the driver is responsible for translating the command into component-specific configurations. This allows the algorithms plugged into the DE to implement network control logic without dealing with the details of each data-plane component. We implemented three drivers and describe their details next.

Linux IP forwarding kernel: The Linux kernel can forward packets received from one network interface to another. To determine the outgoing network interface, the Linux kernel uses two data structures: a Forwarding Information Base (FIB) that stores all routes, and a routing cache that speeds up route search. As in all Tesseract data plane drivers, the driver for Linux IP forwarding kernel implements the `WriteTable` interface. The driver interprets commands from the DE, creates a `rtentry` structure with the route to add or delete, and invokes the `ioctl` system call to modify the FIB. We set `proc/sys/net/ipv4/route/min_delay` to zero so that the routing cache is flushed immediately after the FIB is modified.

Click router: We use Click for forwarding Ethernet frames. The driver for Click includes two parts: an implementation of the `WriteTable` interface, and a Click element package called the `4DSwitch` that is integrated into Click. The implementation of `WriteTable` parses commands and executes those commands by exchanging control messages with the 4DSwitch element in the Click process via a TCP channel. The `4DSwitch` element maintains an Ethernet forwarding table and updates the table according to the received control messages. To control the data forwarding behavior of Click, the `4DSwitch` element overrides the Click `Element::push` function and directs incoming traffic to the outgoing port(s) specified in the `4DSwitch` forwarding table.

netfilter/iptables: Tesseract uses netfilter/iptables to implement reachability control in IP networks. The driver for netfilter/iptables translates commands into iptables rules (e.g., `-A FORWARD -s 10.1.1.0/24 -d 10.1.2.0/24 -i eth0 -j DROP`) and forks an iptables process to install the rules.

3.6 Decision/Dissemination Interface

In designing the interface between the decision plane and the dissemination plane, there is a tension between the conflicting goals of creating a clean abstraction with rigid separation of functionality and the goal of achieving high performance with the cooperation of the decision and dissemination planes.

The key consideration is that the dissemination plane must be able to function independently of the decision plane. Our solution is to build into the dissemination plane a completely self-contained mechanism for maintaining connectivity. This makes the dissemination plane API very simple, giving the basic decision plane only three interface functions: `Send(buf,dst)`, which sends control information to a specific switch, `Flood(buf)`, which floods control information to all switches, and `RegisterUpCall(*func())`, which identifies the decision plane function that handles incoming information.

However, to optimize the performance of the dissemination plane, we add two interface functions: `LinkFailure(link)`, which the DE uses to identify a known failed link to the dissemination plane so the dissemination plane can avoid it immediately, and `PreferredRoute(dst,sourceRoute)`, which the DE uses to suggest a specific source route for carrying control information to switch `dst`. This solution enables a sophisticated DE to optimize the dissemination plane to its liking, but also allows the simplest DE to fully function.

4 Performance Evaluation

In this section, we evaluate Tesseract to answer the following questions: How fast does a Tesseract-controlled network converge upon various network failures? How large a network can Tesseract scale to and what are the bottlenecks? How resilient is Tesseract in the presence of decision-element failures?

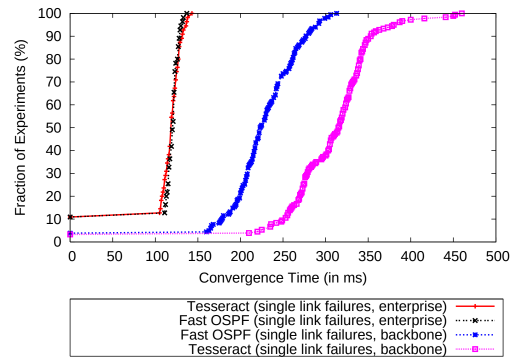

Figure 4: CDF of convergence times for single link failures in enterprise and backbone networks. We pick one link to fail at a time and we enumerate all the links to get the distribution of convergence times. The zero convergence times are caused by failures disconnecting switches at the edge of the network.

4.1 Methodology

We perform both emulation and simulation experiments. We use Emulab to conduct intra-domain routing experiments using two different topologies. The first topology is an ISP backbone network (AS 3967) from Rocketfuel [8] data that spans Japan, U.S., and Europe, with a maximum round trip delay of 250 ms. The other is a typical enterprise network with negligible propagation delay from our earlier study [9].

Emulab PCs have 4 interfaces each, so routers that have more than 4 interfaces are modeled by chaining together PCs to create a "supernode" (e.g., a router with 8 interfaces will be represented by a string of 3 Emulab PCs). As a result, the backbone network is emulated by 114 PCs with 190 links, and the enterprise network is emulated by 40 PCs with 60 links. For each Tesseract experiment, there are 5 decision elements — these run on "pc3000" machines that have a 3GHZ CPU and 2GB of RAM. To inject a link failure, we bring down the interface with the `ifconfig down` command. To inject a switch failure, we abruptly terminate all the relevant software running on a switch.

So that we evaluate the worst-case behavior of the control plane, we measure the time required for the *entire* network to reconverge after an event. We calculate this network convergence time as the elapsed time between the event occurring and the last forwarding state update being applied at the last switch to update. We use Emulab's NTP (Network Time Protocol) servers to synchronize the clocks of all the nodes to within 1 millisecond.

As a point for comparison, we present the performance of an *aggressively tuned* OSPF control plane called Fast OSPF. Fast OSPF's convergence time represents the best possible performance achievable by OSPF and it is determined by the time to detect a link failure and the one way propagation delay required for the LSA flood. Such uniform and aggressive tuning might not be practical in a real network as it could lead to CPU overload on older routers, but Fast OSPF serves as a useful benchmark.

We implemented Fast OSPF by modifying Quagga 0.99.4 [10] to support millisecond timer intervals. There are four relevant timers in Quagga: (1) the hello timer that sets the frequency of HELLO messages; (2) the dead timer that sets how long after the last HELLO is received is the link declared dead; (3) the delay timer that sets the minimum delay between receiving an LSA update and beginning routing computation; and (4) the hold-down timer that sets the minimum interval between successive routing computations. For Fast OSPF, we use hello timer = 20 ms, dead timer = 100 ms, delay timer = 10 ms (to ensure a received LSA is flooded before routing computation begins), and 0 ms for the hold-down timer. Tesseract uses the same hello and dead timer values to make direct comparison possible. There is no need for the delay timer or the hold-down timer in Tesseract.

4.2 Routing Convergence

Common concerns with using a logically centralized DE to provide direct control are that reconvergence time will suffer or the DE will attempt to control the network using an out-of-date view of the network. To evaluate these issues, we measure intra-domain routing convergence after single link failures, single switch failures, regional failures (i.e., simultaneous multiple switch failures in a geographic region), and single link flapping.

Single link failures: Figure 4 shows the cumulative distribution of convergence times of Tesseract and Fast OSPF for all single link failures in both topologies (Some convergence times are 0 because the link failure partitioned a stub switch and no forwarding state updates were required). First, consider the enterprise network scenario where the network propagation delay is negligible. For Fast OSPF, which represents an ideal target for convergence time, its performance is primarily a function of the link failure detection time, which is controlled by the dead timer value (100 ms), and the time to compute and install new routes. Even though Tesseract has a single DE machine compute all the routes, its performance is nearly identical to that of Fast OSPF, thanks to the usage of an efficient dynamic shortest path algorithm and the delta encoding of switch configurations. The only observable difference is that Tesseract's convergence time has a slightly larger variance due to the variability of the

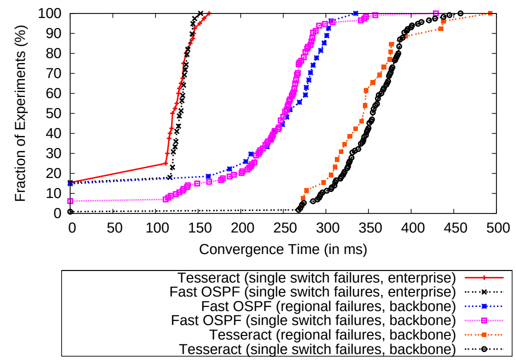

Figure 5: CDF of convergence times for single switch failures and regional failures.

dynamic shortest path algorithm on different failed links.

In the backbone network scenario, propagation delay becomes an important factor as switch-to-switch RTT ranges from 1 ms to 250 ms. Tesseract's convergence requires the link state update to be transmitted to the DE and the new switch configurations to be transmitted back to the switches. On the other hand, Fast OSPF only requires the one-way flooding of the link state update. This is why Tesseract's convergence time is roughly a one-way delay slower than Fast OSPF. In return, however, the direct control paradigm enabled by Tesseract allows other control functions, such as packet filtering, to be implemented together with intra-domain routing in a simple and consistent manner.

Switch failures and regional failures: Next, we examine the convergence time under single switch failures and regional failures. To emulate regional failures, we divide the backbone topology into 27 geographic regions with each region containing a mean of 7 and a maximum of 26 switches, and we simultaneously fail all switches in a region.

Figure 5 compares the cumulative distributions of convergence times of Tesseract and Fast OSPF on switch and regional failures. In the enterprise network, again, the performance of Tesseract is very similar to that of Fast OSPF. In the backbone network, the difference between Tesseract and Fast OSPF is still dominated by network delay, and both are able to gracefully handle bursts of network state changes. There are two additional points to make. First, Fast OSPF has more cases where the convergence time is zero. This is because the 10 ms delay timer in Fast OSPF is acting as a hold-down timer. As a result, Fast OSPF does not react immediately to individual link state updates for a completely failed switch and sometimes that can avoid unnecessary configuration changes. In Tesseract, there is no hold-down timer, so it reacts to some link state updates that are ultimately inconsequential. Second, in some cases, Tesseract has faster convergence time in regional failure than in single

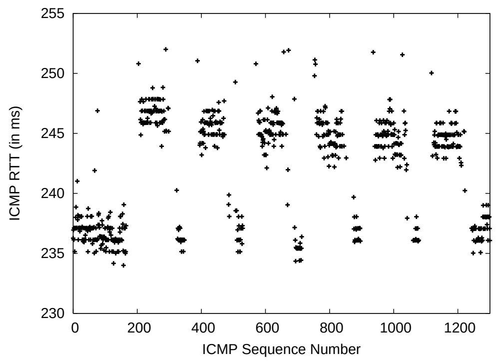

Figure 6: Effects of link flapping on ICMP packets sent at a rate of 100 packets/sec.

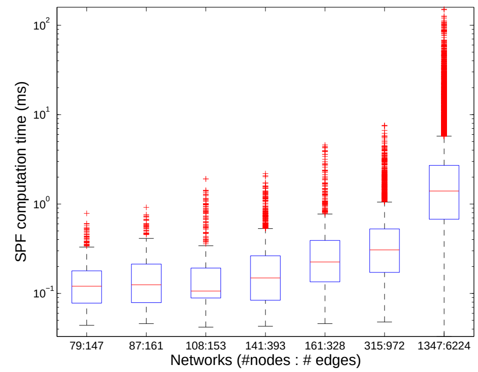

Figure 7: CPU time for computing incremental shortest paths for various Rocketfuel topologies in logarithmic scale. The box shows the lower quartile, upper quartile and median. The whiskers show the min and max data values, out to 1.5 times the interquartile range, and outliers are plotted as '+'s.

switch failure. The reason is that the large number of failed switches in regional failure reduces the amount of configuration updates Tesseract needs to send.

Link flapping: From the earliest days of routing in the Internet there has been concern that a rapidly flapping link could overload the control plane and cause a widespread outage worse than the failure of that single link. Using Emulab we conduct an experiment to show the effects of link flapping on the end-to-end behavior of Tesseract. On the emulated backbone network, we `ping` the Tokyo node from the Amsterdam node at an interval of 10 ms and measure the RTT. We start to flap the link between Santa Clara and Herndon 2 seconds into the experiment. The flapping link is up for 100 ms and then down for 2 seconds. As the link flaps, the route from Tokyo to Amsterdam oscillates between a 10-hop path traversing Santa Clara, Herndon, Weehawken, and London with an average RTT of 240 ms, and a 12-hop path through San Jose and Oak Brook with an average RTT of 246 ms, as shown in Figure 6.

This experiment demonstrates that a logically centralized system like Tesseract can handle continual network changes. It is also worth mentioning that the Tesseract decision plane makes it easy to plug in damping algorithms to handle this situation in a more intelligent way.

4.3 Scaling Properties

Another concern with a logically centralized system like Tesseract is can it scale to size of today's networks, which often contain more than 1,000 switches. Since Emulab experiments are limited in size to at most a few hundred switches, we perform several simulation experiments to evaluate Tesseract's scaling properties. This evaluation uses a DE running the same code and hardware as the previous evaluations, but its dissemination plane is connected to another machine that simulates the control plane of the network.

We evaluate Tesseract's scalability on a set of Rocketfuel topologies with varying sizes. For each topology, we independently fail each link in the graph and measure the time for the DE to compute new forwarding state and the size of the state updates.

DE Computation Time: Every time a failure occurs in the network, the decision element needs to recompute the forwarding tables for the switches based on the new state of the network. Figure 7 shows the results of DE path computation time. As shown in the figure, even in the largest network of 1347 nodes and 6244 edges, the worst case recomputation time is 151 ms and the 99th percentile is 40 ms.

Bandwidth Overhead of Control Packets: Each time the DE computes new forwarding state for a switch, it needs to push out the new state to the switch. Figure 8 plots the number of control bytes that the DE pushes out for independent link failures with different topologies. As shown in the figure, the worst case bandwidth overhead is 4.4MB in the largest network of 1347 nodes and 6244 edges. This is a scenario where 90% of the switches must be updated with new state.

Notice that the bandwidth overhead reported here includes only intra-domain routes. Even when a Tesseract network carries external BGP routes, the amount of forwarding state expected to change in response to an internal link failure will be roughly the same. Switches use two-level routing tables, so even if default-free BGP routing tables are in use, the BGP routes only change when the egress point for traffic changes — not when internal links fail. As has been pointed out by many [11, 7], Internet routing stability would improve if networks did not change egress points solely because the local cost changed, and Tesseract's framework for direct control makes it easier to implement this logic.

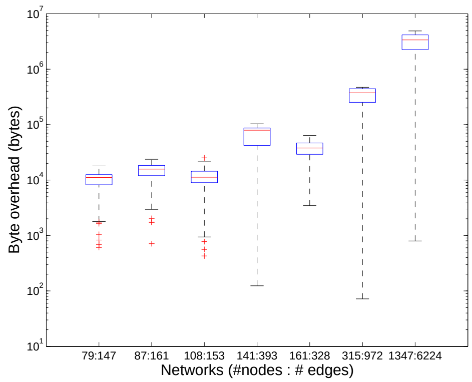

Figure 8: Switch configuration traffic sent out on a single link failure for various Rocketfuel topologies in logarithmic scale.

	Min	Mean	Max	SD
Backup DE takes over	130 ms	142 ms	155 ms	6 ms

Table 1: Minimum, mean, and maximum times, and standard deviation for DE failover in DE failure experiments on the backbone network.

4.4 Response to DE Failure and Partition

This section evaluates decision plane resiliency by measuring the *DE failover time*, defined as the time from when the master DE is partitioned to when a standby DE takes over and becomes the new master DE. We use the backbone network topology and perform 10 experiments in which the master and stand-by DEs are 50 ms apart.

DE failure: Failure of any DE but the master DE is harmless, since in Tesseract the other DEs are hot standbys. To evaluate the effect of the failure of the master DE, we abruptly shutdown the master DE. Table 1 shows the time required for a new DE to take control of the network after the master DE fails. As expected, the average failover time is approximately 140 ms, which can be derived from a simple equation that describes the expected failover time: $(DEDeadTime + PropagationDelay - HeartbeatInterval/2 = 100ms + 50ms - 10ms)$.

Network partition: We inject a large number of link failures into the backbone topology to create scenarios with multiple network partitions. In the partition with the original master DE, Tesseract responds in essentially the same manner as in the regional-failure scenarios examined in Section 4.2, since the original master DE sees the partition as a large number of link failures. In the partitions that do not contain the original master, the convergence time is the same as when the master DE fails.

Just as network designers can choose to build a topology that has the right level of resistance against network partition (e.g., a ring versus a complete graph), the designers can intelligently select locations for placing redundant DEs to defend against network partition.

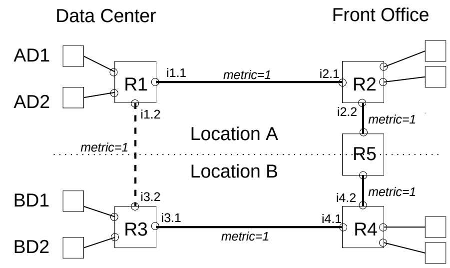

Figure 9: Enterprise network with two locations, each location with a front office and a data center. The dashed link is added as an upgrade.

5 Tesseract Applications

In this section, we demonstrate two applications that take advantage of Tesseract's direct control paradigm.

5.1 Joint Control of Routing and Filtering

Today, many enterprise networks configure packet filters to control which hosts and services can reach each other [9]. Unfortunately, errors in creating network configurations are rampant. The majority of disruptions in network services can be traced to mis-configurations [12, 13]. The situation with packet filters is especially painful, as routes are automatically updated by routing protocols to accommodate topology changes, while there is no mechanism to automatically adapt packet filter configurations.

The Tesseract approach makes joint routing and filtering easy. The decision logic takes as input a specification of the desired security policy, which lists the pairs of source and destination subnets that should or should not be allowed to exchange packets. Then, in addition to computing routes, for each source-destination subnet pair that is prohibited from communicating, the DE initially places a packet filter to drop that traffic on the interface closest to the destination. The decision logic then further optimizes filter placement by pulling the filters towards the source of forbidden traffic and combining them until further pulling would require duplicating the filters.

As a concrete example, consider the network in Figure 9. This company's network is spread across two locations, A and B. Each location has a number of front office computers used by sales agents (AF1-2 and BF1-2) and 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 routing metric for each link is shown in italics. Later, a dedicated link between the data centers (shown as a dashed line between R1 and R3) is added so the data centers can use each other as remote backup locations. The security policy is that front-office computers can communicate with the other location's front office computers and with 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.

We experimentally compared the Tesseract-based solution with a conventional solution that uses OSPF and manually placed packet filters. During the experiments we generate data traffic from AF1 to BF1 (which should be permitted) and from AF1 to BD1 (which should be forbidden) at 240 packets per second and monitor for any leaked or lost packets. In the OSPF network, the filter is manually placed on interface i3.1 to prevent A's front office traffic from reaching BD. After allowing the routing to stabilize, we add a new link between the data centers (dotted line in Figure 9). In the OSPF network, OSPF responds to the additional link by recomputing routes and redirects traffic from AF to BD over the new link, bypassing the packet filter on interface i3.1 and creating a security hole that will have to be patched by a human operator. In contrast, Tesseract computes both new routes *and new packet filter placements appropriate for those routes* and loads into the routers simultaneously, so no forbidden traffic is leaked. Most importantly, once the security policy is specified, it is automatically enforced with no human involvement required.

5.2 Link Cost Driven Ethernet Switching

Ethernet is a compelling layer-2 technology: large switched Ethernets are often used in enterprise, data center, and access networks. Its key features are: (1) a widely implemented frame format; (2) support for broadcasting frames, which makes writing LAN services like ARP, and DHCP significantly easier; and (3) its transparent address learning model, which means hosts can simply plug-and-play. Unfortunately, today's Ethernet control plane is primitive [14, 15, 16]. Based on routing frames along a spanning tree of the switches, it makes very inefficient use of the available links. Convergence time in response to failures can be long, as the IEEE 802.1D Rapid Spanning Tree Protocol (RSTP) is known to count to infinity in common topologies.

We have implemented a Tesseract control plane for Ethernet that preserves all three beneficial properties, avoids the pitfalls of a distributed spanning tree protocol, and improves performance. The DE first creates a spanning tree from the discovered network topology and generate default forwarding entries for the switches that follow the tree — this enables traditional tree-based broadcast. Additionally, when an end host sends its first frame to its first-hop switch, the switch notifies the DE of the newly discovered end host via the dissemination service. The DE then computes appropriate paths from all switches to that end host and adds the generated forwarding entries to the switches. From then on, all frames destined to the end host can be forwarded using the specific paths (e.g., shortest paths) instead of the spanning tree.

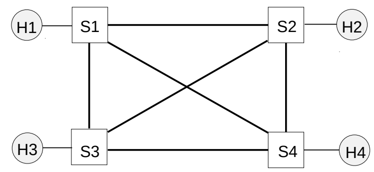

Figure 10: Full-mesh Ethernet topology.

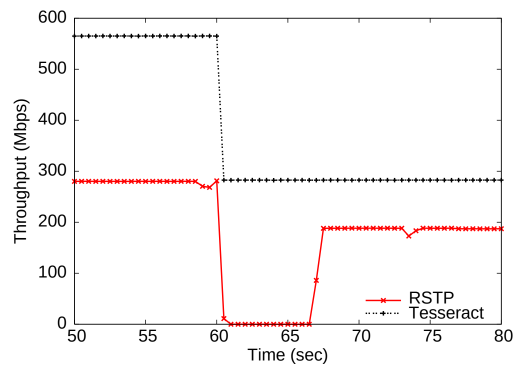

Figure 11: Aggregate network throughput, RSTP versus Tesseract. S1 fails at 60 second.

To experimentally illustrate the benefits of the Tesseract approach, we use the topology shown in Figure 10 on Emulab. The four switches are connected by 100 Mbps Ethernet links, and each end host is connected to one switch via a 1 Gbps Ethernet link. We run `iperf` [17] TCP servers on the four end hosts and simultaneously start six TCP flows. They are H1 to H2, H1 to H3, H1 to H4, H2 to H3, H2 to H4, and H3 to H4. In the first experiment, the network is controlled by Tesseract using shortest path as the routing policy. In the second experiment, the network is controlled by an implementation of IEEE 802.1D RSTP on Click.

Figure 11 shows the aggregated throughput of the network for both experiments. With the Tesseract control plane, all six TCP flows are routed along the shortest paths, and the aggregate throughput is 570 Mbps. At time 60 s, switch S1 fails and H1 is cut off. The Tesseract system reacts quickly and the aggregate throughput of the remaining 3 TCP flows stabilizes at 280 Mbps. In contrast, in a conventional RSTP Ethernet control plane,

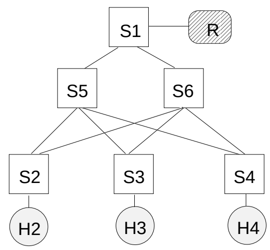

Figure 12: Typical Ethernet topology gadget.

forwarding is performed over a spanning tree with S1 as the root. This means the capacities of the S2-S3, S2-S4, and S3-S4 links are totally unused. As a result, the aggregate throughput of the RSTP controlled network is only 280 Mbps, a factor of two less than Tesseract. When switch S1 fails at time 60 s, RSTP tries to reconfigure the spanning tree to use S2 as the root and begins a count-to-infinity. The combination of frame loss when ports oscillate between forwarding/blocking state and TCP congestion control back-off means the throughput does not recover for many seconds. When RSTP has finally reconverged, the aggregate throughput is again substantially less than the Tesseract network.

As a second example of the value of being able to change the decision logic and the ease with which Tesseract makes this possible, consider Figure 12. This topology gadget is a typical building block found in Ethernet campus networks [18] which provides protection against any single link failure. Basic Ethernet cannot take advantage of the capacities of the redundant links since RSTP forms a spanning tree with S1 as the root, and the S2-S6, S3-S6, and S4-S6 links only provide backup paths and are not used for data forwarding. As a result, traffic flows from H2, H3, and H4 to R must share the capacity of link S1-S5. In contrast, when there exists two or more equal cost paths from a source to a destination, the Tesseract decision logic breaks the tie by randomly picking a path. By centralizing the route computations and using even such simple load-balancing heuristics, Tesseract is able to take advantage of the multiple paths and achieve a substantial increase in performance. In our example, the capacities of both link S1-S5 and S1-S6 are fully utilized for a factor of two improvement in aggregate throughput over RSTP.

The examples in this section illustrate the benefit of the direct control paradigm, where the only distributed functions to be implemented by network switches are those that discover the neighborhood status at each switch and those that enable the control communications between the DE and the switches. As a result, it becomes easy to design and change the decision logics that control the network. There is no need to design distributed protocols that attempt to achieve the desired control policies.

6 Related Work

Previous position papers [19, 3] have laid down the conceptual framework of 4D, and this paper provides the details of an implementation and measured performance. The Routing Control Platform (RCP) [7, 20] and the Secure Architecture for the Networked Enterprise (SANE) [21] are the most notable examples that share conceptual elements with 4D.

RCP is a solution for controlling inter-domain routing in IP networks. RCP computes the BGP routes for an Autonomous Systems (AS) at centralized servers to give the operators of transit networks more control over how BGP routing decisions are made. The RCP servers have a very similar role as the decision plane in 4D. For backward compatibility, the RCP servers use iBGP to communicate with routers. This iBGP communication channel has a role similar to the dissemination plane in 4D.

SANE is a solution for enforcing security policies in an enterprise network. In a SANE network, communications between hosts are disabled unless they are explicitly allowed by the domain controller. Switches only forward packets that have authentic secure source routes attached to them. For communications between switches and the domain controller, SANE constructs a spanning tree rooted at the domain controller. This spanning tree has a role similar to the dissemination plane in Tesseract.

Tempest [22] proposes an alternate framework for network control, where each switch is divided into switchlets and the functionality of each switch is exposed through a common interface called Ariel. Tempest allows multiple control planes to operate independently, each controlling their own virtual network composed of the switchlets, and the framework has been used on both MPLS and ATM data planes. Tesseract's dissemination plane provides a complete bootstrap solution, where Tempest's implementation assumed a pre-existing IP-over-ATM network for communication with remote switches. While both projects abstract switch functionality, Tesseract does not assume that switches can be fully virtualized into independent switchlets, and it leaves resource allocation to the decision logic.

FIRE [23] presents a framework to ease the implementation of distributed routing protocols by providing a secure flooding mechanism for link-state data, hooks to which route computation algorithms can be attached, and a separate FIB used for downloading code into the router. Tesseract eases the implementation of centralized network control algorithms by assembling a network-wide view, enabling direct control via a robust and self-bootstrapping dissemination plane, and providing redundancy through the election of DEs.

7 Summary

This paper presents the design and implementation of Tesseract, a network control plane that enables direct control. In designing Tesseract, we paid particular attention to the robustness of the decision plane and the dissemination plane. The security of Tesseract is enhanced by the mechanisms built into the dissemination service. The system is designed to be easily reusable and we demonstrated how Tesseract can be used to control both Ethernet and IP services. Finally, good performance is achieved by adopting efficient algorithms like incremental shortest path and delta encoding of switch configuration updates.

We find that Tesseract is sufficiently scalable to control intra-domain routing in networks of more than one thousand switches, and its reconvergence time after a failure is detected is on the order of one round-trip propagation delay across the network.

The most important benefit of Tesseract is that it enables direct control. Direct control means sophisticated control policies can be implemented in a centralized fashion, which may be much easier to understand and deploy than a distributed protocol. Direct control also means the software running on each switch is simplified, with potential benefits for operators and vendors. We strongly believe that the direct control paradigm is the right approach in the long run, as there is a clear trend towards ever more sophisticated network control policies.

Acknowledgments

Andy Myers contributed greatly to the early stages of this project. Khaled Elmeleegy generously provided an IEEE 802.1D RSTP Ethernet bridge implementation for our experiments. We would like to thank the anonymous reviewers and our shepherd Jon Crowcroft for helpful feedback on earlier versions of the paper. This research was sponsored by the NSF under ITR Awards ANI-0085920, ANI-0331653, and NeTS Grant CNS-0520187. 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 Microsoft, NSF, or the U.S. government.

References

[1] A. Shaikh and A. Greenberg, "Operations and management of IP networks: What researchers should know," August 2005. ACM SIGCOMM Tutorial.

[2] 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.

[3] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM Computer Communication Review*, October 2005.

[4] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," *J. ACM*, vol. 51, no. 6, pp. 968–992, 2004.

[5] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 1, pp. 612–613, 1979.

[6] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)." RFC 2048, November 1998.

[7] 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.

[8] N. Spring, R. Mahajan, and D. Wetheral, "Measuring ISP topologies with RocketFuel," in *Proc. ACM SIGCOMM*, August 2002.

[9] 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.

[10] "Quagga Software Routing Suite." `http://www.quagga.net`.

[11] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in IP networks," in *Proc. ACM SIGMETRICS*, June 2004.

[12] Z. Kerravala, "Configuration management delivers business resiliency." The Yankee Group, Nov 2002.

[13] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do internet services fail, and what can be done about it," in *Proc. USENIX Symposium on Internet Technologies and Systems*, 2003.

[14] A. Myers, T. S. E. Ng, and H. Zhang, "Rethinking the service model: Scaling Ethernet to a million nodes," in *Proc. HotNets*, November 2004.

[15] R. Perlman, "Rbridges: Transparent routing," in *Proc. IEEE INFOCOM*, March 2004.

[16] K. Elmeleegy, A. L. Cox, and T. S. E. Ng, "On count-to-infinity induced forwarding loops in ethernet networks," in *Proc. IEEE INFOCOM*, 2006.

[17] "Iperf – The TCP/UDP Bandwidth Measurement Tool." `http://dast.nlanr.net/Projects/Iperf`.

[18] "Gigabit campus network design – principles and architecture." Cisco White Paper.

[19] 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. HotNets*, pp. 59–64, November 2004.

[20] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and Jacobus van der Merwe, "Design and implementation of a Routing Control Platform," in *Proc. NSDI*, May 2005.

[21] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," in *Usenix Security*, August 2006.

[22] S. Rooney, J. van der Merwe, S. Crosby, and I. Leslie, "The Tempest: a framework for safe, resource assured, programmable networks," *IEEE Communications Magazine*, vol. 36, pp. 42–53, Oct 1998.

[23] C. Partridge, A. C. Snoeren, T. Strayer, B. Schwartz, M. Condell, and I. Castineyra, "FIRE: flexible intra-AS routing environment," in *Proc. ACM SIGCOMM*, 2000.