RICE UNIVERSITY

Designing Scalable Networks for Future Large Datacenters

by

Brent Stephens

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

Master of Science

Houston, Texas

May, 2012

ABSTRACT

Designing Scalable Networks for Future Large Datacenters

by

Brent Stephens

Modern datacenters require a network with high cross-section bandwidth, fine-grained security, support for virtualization, and simple management that can scale to hundreds of thousands of hosts at low cost. This thesis first presents the firmware for Rain Man, a novel datacenter network architecture that meets these requirements, and then performs a general scalability study of the design space.

The firmware for Rain Man, a scalable Software-Defined Networking architecture, employs novel algorithms and uses previously unused forwarding hardware. This allows Rain Man to scale at high performance to networks of forty thousand hosts on arbitrary network topologies.

In the general scalability study of the design space of SDN architectures, this thesis identifies three different architectural dimensions common among the networks: source versus hop-by-hop routing, the granularity at which flows are routed, and arbitrary versus restrictive routing and finds that a source-routed, host-pair granularity network with arbitrary routes is the most scalable.

Acknowledgments

Without the help of an inumberable number of people, this thesis would not have been possible. In this section, I will try to thank many of them. First, I would like to thank my committee. The unwavering support and direction provided by Dr. Alan L. Cox, Dr. Scott Rixner, Dr. T. S. Eugene Ng, and Dr. John Carter has been invaluable. Each of diverse perspectives that they provide have been necessary for the completion of this work.

Second, I would like to thank my collaborators at IBM Research, Austin. Working with Dr. John Carter and Wes Felter, as well as everybody else at the Austin research lab, has been a pleasure, and working on the Rain Man project provided the necessary real world grounding to this work.

Last, I need to acknowledge the blessings bestowed on me by the Almighty and thank the people who supported me so that I could complete this work. Without my family and friends, I never would have reached a position where I could undertake this work. Specifically, I need to thank my parents who were my first and most influential teachers, and I must acknowledge the sacrifices they made in order to enable me to pursue my education. Lastly, I need to thank my wife. Without her love, nourishment, and support, I would be lost. It is to her that I dedicate my successes.

Contents

Illustrations

Tables

CHAPTER 1

Introduction

1.1 Introduction

Modern datacenters require a network with high cross-section bandwidth, low latency, fine-grained security, support for virtualization, and simple management that can scale to hundreds of thousands of hosts at low cost [1, 2, 3, 4, 5]. Although Ethernet is the most commonly deployed Layer 2 (L2) datacenter network, traditional switched Ethernet cannot satisfy these requirements at a large scale. However, it is still desirable to use Ethernet for datacenter networks because existing datacenter hosts already have Ethernet hardware.

The current best practice solution for addressing the scalability limitations of traditional switched Ethernet is to use IP routers to interconnect multiple independent Ethernet networks. IP routers require that the independent Ethernet networks are configured as IP subnets, which are built by assigning hierarchical IP addresses to all the hosts in the network. IP routers then forward traffic by utilizing knowledge of the address hierarchy. Unfortunately, IP subnetting requires careful design, and IP routers are expensive devices that require complex configuration and maintenance. In some production networks, IP routers account for as much as 70% of the network operating costs [6]. If the individual Ethernet networks could be larger, the number of IP routers could be reduced, perhaps even to zero.

Because correct forwarding can be performed by flooding, the only strict limit on the size of an Ethernet network is the 48-bit name space. Unfortunately, traditional switched Ethernet's performance degradation limits the effective scale of Ethernet networks to around a thousand hosts, to the best of my knowledge. Given the scalability limitations of traditional switched Ethernet and the drawbacks of IP routers, there is a clear need to design an L2 network that can be a substitute for traditional Ethernet while maintaining the virtues of Ethernet, including automatic management and compatibility with existing hosts. I address this problem by studying the design of scalable L2 networks for the datacenter.

Before the scalability limitations of switched Ethernet can be addressed, the reasons why switched Ethernet does not scale must be considered. There are three principle causes: control bandwidth overhead, exhausting forwarding table state, and a lack of usable application bandwidth, which can be caused by the topology or by under-utilizing the links in the network. Ethernet switches, like other networking devices, use hardware forwarding tables to store information about how to forward traffic. The source of switched Ethernet's scalability limitations is that if an Ethernet switch has not yet learned the location of a host, it relies on flooding to forward the packet. A packet is flooded by duplicating the packet and sending a copy out all of the switch's active ports, except for the packet's ingress port. This guarantees that the packet will reach its destination, if the destination is attached to the network. Flooding wastes bandwidth, but it also requires the network topology to be a cycle-free tree so that flooded packets do not persist indefinitely in the network. This limits scalability because the switch at the root of the tree requires a disproportionate amount of forwarding table state. Even worse, cycle-free tree topologies limit application bandwidth and prevent traffic engineering, optimizing the distribution of traffic across links in the network, because there is only a single path between any two hosts.

Software-defined networking (SDN) is one promising class of network architectures that are suitable substitutes for traditional switched Ethernet. SDN lets an external controller manage the behavior of switches in a network. The global network view and control enabled by SDN allows for the fine-grained security and high performance required from a datacenter network. SDN is also a promising traditional switched Ethernet substitute because it is well suited for commodity-off-the-shelf (COTS) Ethernet switch hardware. This is because it can reduce the load on the control plane, which is typically under-provisioned.

All SDN architectures have three layers: the network controller, the switch firmware, and the switch hardware. This is in contrast to the architectures of traditional Ethernet switches and managed Ethernet switches. Traditional Ethernet switches only have one layer, the switch hardware, and managed Ethernet switches have two layers: the switch firmware and the switch hardware. The three layers of SDN are connected by two interfaces. The SDN protocol is the interface between the network controller and the switch firmware, and there is a proprietary interface between the hardware and software inside the switch. In essence, the SDN protocol provides an abstraction of the switching hardware to the network controller, and the switch firmware is responsible for mapping this abstraction to the hardware.

OpenFlow is a popular SDN protocol used to implement many SDN architectures. These architectures can eliminate Ethernet's broadcast overhead, topology restrictions, and routing restrictions that prevent switched Ethernet from scaling to large networks. Nonetheless, there is still a problem. Current OpenFlow implementations still do not scale to sufficiently large networks because the OpenFlow switch firmware does not make full and efficient use of the switch hardware.

There are other SDN networks besides OpenFlow, and even within OpenFlow there are different strategies that can be used for controlling the network. However, there have been no general scalability studies of the design space. The end result is that it is unclear what approaches for implementing an SDN network are the best and which SDN network, if any, is the most scalable Ethernet substitute. This thesis addresses these problems.

1.2 Contributions

In this thesis, I designed a new firmware architecture for Rain Man, a novel L2 SDN architecture based on OpenFlow. The new firmware improves the mapping from the OpenFlow protocol to the switching hardware to make more efficient use of the existing hardware. One major contribution of the Rain Man firmware is using the previously unused large, exact match L2 forwarding table and multipath capabilities of the hardware. Rain Man also contributes a novel algorithm for using the L2 forwarding table to forward packets toward the same destination along different paths and a novel packing algorithm that increases utilization of the multipath capabilities. On COTS Ethernet switch hardware, Rain Man scales efficiently at high performance up to one thousand switches and forty thousand hosts. Without the contributions of Rain Man, previous SDN designs achieved scalability only by restricting the topology. In contrast, Rain Man can support arbitrary topologies and is able to use a topology that has been shown to achieve 100% higher throughput than an equal bisection bandwidth instance of the topology required by previous designs [7].

Next, this thesis studies the scalability of different SDN architectures. Rain Man, the proposed SDN architecture, is constrained by the requirement of being implementable on commodity Ethernet switch hardware. Without this constraint, there are other points in the design space that are more scalable than Rain Man.

To evaluate this, I perform a scalability study of SDN network architectures, allowing for changes at all three layers of the SDN architecture. It would be infeasible to implement every SDN network, so first I identify three different architectural dimensions common among the networks: source versus hop-by-hop routing, arbitrary versus restrictive routing, and the granularity at which flows are routed. My work then compares the design space against the scalability limitations of Ethernet: broadcast and control bandwidth overhead, exhausting forwarding table state, and its ability to load-balance traffic across links in the network.

The results of the study show that source routing requires between two to five times less forwarding state than hop-by-hop routing. They also show that restricting routing to minimum length paths has little effect on performance. In contrast, restricting routing to a tree, either to use a lossless network or the L2 forwarding tables, has a much more significant performance penalty. Lastly, they demonstrate that routing at the TCP granularity is unnecessary for improving performance under the evaluated workloads.

1.3 Organization

This thesis is organized as follows. First, Chapter 2 introduces traditional switched Ethernet and discusses its scalability limitations. Next, Chapter 3 discusses related work. After that, Chapter 4 presents Rain Man. Next, Chapter 5 presents the SDN scalability study. Finally, Chapter 6 concludes.

CHAPTER 2

Ethernet

Ethernet is a popular network architecture because of its simplicity; Ethernet switches require little to no manual configuration because the switches rely on broadcasting and flooding to configure and manage the network. As an optimization, switches learn the locations of addresses as they forward traffic so that they can forward without flooding. While relying on broadcasting and flooding allows Ethernet to be simple, it is also the source of Ethernet's scalability limitations. They limit scalability by wasting bandwidth and forcing the active forwarding topology to be cycle-free. This distributes forwarding table state poorly across the network and limits the usable application bandwidth of the network. This chapter gives an overview of the operation of a traditional Ethernet network and discusses its scalability limitations, while Chapter 3 discusses substitutes for traditional Ethernet.

2.1 Operation

Ethernet is a ubiquitous network architecture primarily because the operation of Ethernet switches is simple and flexible. Ethernet switches require little to no manual configuration. Forwarding is performed based on globally unique identifiers assigned by device manufacturers. Switches learn how to perform forwarding automatically, and failure recovery is fully automatic with the RSTP protocol. Ethernet switches can be wired into any arbitrary topology, and upgrades to the standard retain backward compatibility with existing versions.

Switched Ethernet's ease of operation derives in large part from its ability to flood packets throughout the network. Flooding enables a packet to reach the destination host's interface without any configuration of that interface or the network, regardless of the interface's location in the network topology.

As an optimization to reduce packet flooding, Ethernet switches learn the correct egress port for hosts by remembering the ingress port a previous packet from that host used. If a switch has learned the ingress port for an address, any subsequent packets destined for the address will used the learned port. If a switch has not yet seen a packet from an address or the forwarding table of the switch is full, correct forwarding is achieved by simply flooding the packet.

Address resolution in Ethernet relies on broadcasting. Hosts broadcast ARP request packets to discover the Ethernet MAC address associated with a known destination IP address. The broadcast ARP request packet is guaranteed to reach the destination if it is attached to the network. The destination then sends a unicast ARP reply to the source MAC included in the ARP request packet. The reply is most likely not flooded because the switches have learned the egress port for the source.

Because Ethernet packets do not have a time-to-live (TTL) field, the network topology of an Ethernet network must not contain cycles. If cycles were allowed, flooded packets would persist in the network indefinitely as they are continually flooded over the cycle. Switched Ethernet does permit the existence of redundant links, but only to heal the network after a failure. Redundant links are never used to provide additional bandwidth. Ethernet switches employ a distributed algorithm, such as the Rapid Spanning Tree Protocol (RSTP), to reach agreement among the switches on a mapping from the physical topology to a cycle-free active topology. The Ethernet switches only use the active topology when forwarding packets to their destination. In effect, redundant network links are disabled. However, in the event of a network failure, the switches may selectively reactivate one or more disabled links and reconfigure the active topology to use those links, thereby healing the network.

2.2 Scalability Limitations

The scalability of Ethernet is limited by its control bandwidth overhead, which drastically increases when forwarding table state is exhausted, and a lack of usable application bandwidth caused by the topology.

The control bandwidth overhead of Ethernet is dominated by the overhead of broadcasting packets for network services and flooding packets to guarantee delivery. Although broadcasting and flooding packets to control the network allows Ethernet to be simple and flexible, these packets traverse unnecessary links in the network and waste bandwidth.

Broadcasting and flooding impose other limitations, as well. They restrict scalability by requiring the active forwarding topology to be a cycle-free tree so that packets do not persist indefinitely in the network. Because Ethernet allows cycles in the physical topology and disables them with RSTP, failures cause additional control overhead. During failures, RSTP consumes usable application layer bandwidth. The control overhead imposed by RSTP is not only incurred by switches local to the failure. RSTP requires communication between all of the switches in the network, even if most of the switches are unaffected by the failure.

The control overhead of Ethernet also depends on the forwarding table occupancy of the switches in the network. Forwarding tables need to be implemented with fast hardware to be able to transmit at line rate. For example, forwarding minimally sized packets on a 1Gbps and 10Gbps link at line rate requires that a new forwarding table lookup can commence every 608ns and 60.8ns per port, respectively. To meet this rate, forwarding tables are typically implemented with content addressable memories (CAMs) or hardware hash tables. These structures are of limited size, typically having between 4,096 and 32,768 entries for a CAM and up to a hundred thousand entries for hardware hash tables. When a switch exhausts is forwarding tables, the control overhead of Ethernet drastically increases because some flows traversing the switch must be forwarded by flooding rather than switching.

In Ethernet, forwarding along a tree also means that the switches near the root of the tree require a disproportionate amount of forwarding table state. The switches nearer the root of the tree exhaust the free entries in the forwarding table faster, causing thrashing. Having a sufficiently sized forwarding table is important for scalability because substantial bandwidth is wasted on flooding every time the forwarding working set of a switch is larger than the forwarding table.

Lastly, the topology restricts scalability by limiting usable application bandwidth. Even if there are redundant links in an Ethernet network, they are disabled by RSTP so that there is only a single path between any pair of hosts. This means that application bandwidth can only be increased by increasing the bandwidth of individual links, which cannot be increased arbitrarily due to physical constraints. Instead, if Ethernet did not restrict the topology to a tree, additional bandwidth could be added to the network by adding and using additional redundant links.

In summary, the scalability of Ethernet is limited by its control bandwidth overhead, which is exacerbated by exhausting forwarding table state, and a lack of usable application bandwidth caused by the topology.

CHAPTER 3

Related Work

This thesis both designs the firmware for Rain Man, a new L2 network suitable as a scalable substitute for Ethernet, and performs a scalability study of SDN Ethernet architectures. In order to compare Rain Man against other related L2 networks and perform the scalability study, I derived scalability metrics from the scalability limitations of Ethernet discussed in Section 2.2. The metrics are control overhead, forwarding table state, and usable application bandwidth. In Ethernet, forwarding table state is related to the control overhead of the network because correct forwarding may be performed by flooding. However, in general, forwarding table state may be its own independent scalability limitation. In some networks, flows must be denied access to the network if all of the forwarding table state is exhausted, regardless of the network utilization. Flow denial limits the size of the network independently of control overhead. The usable application bandwidth metric is determined by two distinct aspects. First, a scalable network must be able to support a topology with sufficient application bandwidth, but this alone is not sufficient. The network must also be able to load-balance traffic across the links in the network.

The scalability metrics described above are not the only possible network scalability metrics. Myers, *et al.* [8] address the scalability of an Ethernet network to a million nodes, and in their work, they identify the limitations of packet flooding and broadcasting. These limitations are only one of the three scalability factors identified by this work.

While Myers, *et al.* develop metrics that are a subset of those used in this thesis, Bondi [9] builds a general system scalability framework that encompasses the metrics used in this thesis, which are subcategories of the load and space categories presented by Bondi. Bondi also defines a structural category of scalability, but this is ignored by this thesis because the 48-bit name space of Ethernet does not hit address scalability limitations until networks are sized with trillions of hosts. Even after trillions of hosts, the structural scalability limitations can still be ignored because they are easily solved by increasing the number of bits used for addressing.

In the rest of this section, I will discuss related substitutes for traditional Ethernet and related scalability studies. First I will compare Rain Man against related Ethernet substitutes in Section 3.1, and then I will compare my evaluation against other scalability studies in Section 3.2. Some work falls into both categories. The parts of these works will be discussed in the relevant sections.

3.1 Ethernet Substitutes

Rain Man, the proposed architecture, is a new L2 network suitable as a scalable substitute for Ethernet. Rain Man addresses the control bandwidth overhead of the network by sending unicast traffic to a central network controller for all network configuration. To further reduce the control overhead of the network, the controller pre-installs default routes and uses sampling to gather network statistics. Rain Man increases the total usable forwarding table state at each switch by using large, previously unused forwarding tables. Rain Man addresses the usable application bandwidth problem by allowing for arbitrary topologies, installing random default routes, and using the controller to gather per-flow statistics and perform traffic engineering.

Other network architectures are suitable as substitutes for Ethernet. Some of these architectures were explicitly designed for scalability. These architectures address one or more of the scalability limitations of L2 networks.

Ethane [10] and OpenFlow [11] address Ethernet's lack of access control through software-defined networking (SDN), which uses simple switches and a central network controller to implement an access policy. Ethane was the first project to suggest using a central controller to manage an Ethernet network, and the OpenFlow protocol grew out of Ethane. As in Rain Man, the control overhead problems of Ethernet are solved in Ethane and can be solved in OpenFlow by integrating the services that require broadcasts, such as DHCP and ARP, into the controller. However, Ethane and OpenFlow do not explicitly address the forwarding state issues of Ethernet, and forwarding in Ethane and OpenFlow is performed on the granularity of the flow, which requires a TCAM so that the flow can be defined according to any fields in the packet header, including wildcards. Although forwarding at the flow granularity requires TCAM state, this, combined with allowing arbitrary topologies, allows Ethane and OpenFlow to perform traffic engineering, although this is not described in Ethane.

PortLand [1] is a proposed Ethernet replacement based on OpenFlow that has low control overhead, constant forwarding state, and randomized load-balancing. Unlike Rain Man, PortLand restricts the topology to the multipath fat tree topology to achieve these properties, which constrains the deployment of a PortLand network. PortLand, like Rain Man, also uses a controller to resolve ARP requests. In PortLand, this is the only source of control overhead during standard network operation. By taking advantage of the hierarchy of the topology, PortLand transparently rewrites host addresses to location specific addresses, which allows for wildcard forwarding table rules. Using wildcards in the forwarding tables limits the per-switch forwarding table state to being on the order of the number of ports per switch. Forwarding table state in Rain Man is still proportional to the size of the network, so PortLand requires less state than Rain Man, although the state reduction requires the topology to be restricted. PortLand, like Rain Man, load balances traffic in the network across the multipath fat tree topology with equal-cost multipath (ECMP) routing. The effect of ECMP routing is that a flow is randomly assigned one of the precomputed equal cost paths to the destination. The path is chosen by hashing the header fields of a packet so that the path assignment is consistent for a flow, which avoids TCP reordering issues.

Hedera [2] is a refinement of PortLand that uses network load statistics at the controller to dynamically route flows in the network, a technique that is also used by Rain Man. This design achieves near optimal traffic engineering for the workloads and topology that they evaluated. However, in Hedera, near optimal performance comes at the expense of higher control overhead and forwarding state. Dynamically routing flows in the network increases the control overhead of the network by requiring statistics collection. Dynamic routing also increases forwarding state proportionally with the number of dynamically routed flows.

DevoFlow [12] is an SDN architecture derived from OpenFlow that addresses the excessive overheads incurred by involving the switch's control-plane and OpenFlow controller too frequently in the network. DevoFlow notes that polling the forwarding tables for statistics, as in Hedera, quickly overloads the switch control plane and that involving the controller with every flow setup in the network quickly overloads the controller. This observation has also been validated by Rain Man. To address these issues, DevoFlow proposes hardware modifications. These modifications include support for rule cloning, multipathing, rapid re-routing, and statistics collection through sampling, triggers, and approximate counters. Most of these modifications reduce that control overhead of the network, which DevoFlow noted was a limiting factor in Hedera. Rule clone increases forwarding table state because every flow that was being routed by a single wildcard rule now has its own exact match rule. Traffic engineering is improved only by increasing the accuracy of the controller's network statistics. Like DevoFlow, Rain Man pre-installs default routes and uses sampling to collect network statistics in order to lessen the load on the switch control plane. However, Rain Man, unlike DevoFlow, is implementable on existing hardware because Rain Man uses existing packet sampling functionality and does not propose new hardware modifications.

Mahout [13] is another OpenFlow-based Ethernet replacement that addresses the issue of network statistics collection. The crucial difference is that Mahout modifies hosts to collect network load statistics, whereas the other networks rely on the switches to collect statistics. Modifying hosts causes Mahout to be incompatible with important network devices that are compatible with Rain Man, such as routers and middleboxes. However, by modifying hosts to collect statistics, Mahout can reduce the control overhead of statistics collection in the network. Mahout expects that hosts can detect elephant flows on their own and only communicate with the network controller when they have detected an elephant flow, where an elephant flow is any flow that requires dynamic routing. This has lower control overhead than sampling. Although Mahout does not address forwarding table state, Mahout can improve traffic engineering. Mahout is able to collect statistics with lower latency than Hedera or Rain Man, which increases the accuracy of the controller's network statistics. This can, in turn, improve the controller's ability to perform traffic engineering.

The Axon [14] is a new L2 hardware switch that transparently uses source routing to improve the scalability of Ethernet. The implementation of the Axons described in [14] uses a central controller to register services and install routes. This reduces the control overhead of the network by disallowing broadcasts. In source routing, all of the forwarding state for a host in source routing is located at the local switch, which reduces forwarding state. Using a central controller with Axons allows for traffic engineering, although Axons do not propose any specific load balancing mechanisms. Axon style source routing is compatible with Rain Man and can be used to reduce the forwarding state of a Rain Man network.

SPAIN [15] builds a more scalable network from commodity Ethernet switches. Although Rain Man and other OpenFlow networks can be implemented on commodity Ethernet hardware, SPAIN has the more strict requirement that the switches must run the unmodified traditional Ethernet firmware. SPAIN extends traditional Ethernet to exploit redundant topologies by using a controller to build multiple spanning trees, each on their own VLAN. The spanning trees are built considering network load, and SPAIN assumes that hosts can be modified to load balance traffic across across the multiple VLANs. SPAIN continues to rely on broadcasting and flooding for correct operation, and SPAIN increases the network control overhead because it requires that every host download the switch level topology and VLAN mappings. SPAIN also increases the forwarding state requirements of the network in proportion with the number of VLANs being used. The increase in control overhead and forwarding state is the cost paid for utilizing redundant links and improving the traffic distribution. In contrast, Rain Man can utilize redundant links without increasing the control overhead or forwarding state like SPAIN, and Rain Man does not restrict routes, which can further improve the traffic distribution over SPAIN.

NetLord [5] is follow-up work to SPAIN for virtualized environments that only addresses the forwarding table state requirements of SPAIN. The control overhead and traffic engineering are otherwise unaffected. In SPAIN, if there are k VLANs and h hosts, each switch requires $\mathrm{O}(kh)$ layer 2 forwarding table entries. NetLord reduces the state requirement to $\mathrm{O}(ks)$, where s is the number of switches in the network. NetLord accomplishes this state reduction by performing network address translation (NAT) in the hypervisor. Before sending a guest packet into the network, the hypervisor rewrites the packet headers to that it appears that the packet is coming from the hypervisor's local switch instead of the guest. This is possible because, in virtualized environments, the hypervisor is the first and last hop switch in the network.

MOOSE [16], similarly to NetLord, performs address translation to reduce forwarding table state. However, unlike NetLord, MOOSE performs address translation at the edge of the network in order to use location specific addresses within the network. This allows MOOSE to operate without needing to modify hosts. Other than renaming, MOOSE operates in the same way as traditional switched Ethernet, and thus suffers from the same scalability limitations.

The address translations performed by both NetLord and MOOSE are orthogonal and complementary to the operation of Rain Man. This means that using NetLord or MOOSE style address translation with Rain Man can further reduce the forwarding table state of the network.

HBR [17] merges flows into ensembles as they enter the network. HBR, like SPAIN, uses a multi-VLAN COTS Ethernet network. However, unlike SPAIN, HBR supports unmodified L2 hosts. Because HBR still uses traditional Ethernet, it does not change the control or broadcast overhead that Rain Man eliminates. In HBR, forwarding is then performed on the granularity of the ensemble rather than on the granularity of the flow. The forwarding table state in HBR is constant with the number of ensembles, which allows for tunable performance and forwarding state trade-offs. HBR assigns traffic to ensembles with a central controller, which is responsible for building the Ethernet VLANs and performing dynamic traffic engineering. Performing traffic engineering on the ensemble rather than on the flow as Rain Man can lead to suboptimal routing.

DCell [18] improves the scalability of Ethernet for the data center by using a multipath regular topology and integrating the servers into the network. In DCell, the servers, as well as the switches, perform forwarding, which means that existing servers that are supported by Rain Man are not compatible with DCell. Addressing and routing in DCell is based on positioning in the regular topology. This means that the control overhead is reduced because the network no longer relies on flooding for host discovery, although hosts continue to send broadcasts in the network as normal. Because routing is based on positioning in the regular topology, forwarding state is relative to the number of levels in the network, which is logarithmic with the size of the network and smaller than the forwarding state of Rain Man. DCell improves distribution of traffic in the network by using a multipath topology, although routing in DCell is deterministic, which prevents the traffic engineering performed by Rain Man.

BCube [19] improves upon DCell by having the servers attach source routes to the packets for forwarding. As in DCell, BCube uses both servers and switches to perform forwarding, and requires a regular topology. Using source routes improves upon DCell by allowing for routing in the network to utilize the multiple paths between any two hosts provided by the regular topology. This allows for load balancing traffic among the links in the network as is possible in Rain Man.

SecondNet [4] improves upon BCube by allowing for arbitrary topologies. Source routing in SecondNet is accomplished with static MPLS mappings in COTS Ethernet switches, while BCube requires special hardware. In a virtualized datacenter, SecondNet style source routing is also compatible with Rain Man. SecondNet uses a central controller for topology dissemination and route installation. SecondNet increases the control overhead of the network when compared with DCell and BCube because the central controller now needs to discover and disseminate routes, which were implicit processes in DCell and BCube, although this overhead is also present in Rain Man. SecondNet also increases the forwarding state requirements at the network edge because routing is no longer implicit, although this is not a scalability limitation because routes are stored at the hosts, where memory is plentiful and forwarding lookups typically occur in software. The forwarding state of the switches in the network is still constant. SecondNet performs load-balancing, but on bandwidth reservations made by tenants, not on the current network load. Because bandwidth reservations may not accurately reflect current network demand, it is possible for Rain Man to achieve higher aggregate network throughput.

Some Ethernet substitutes choose to use a distributed architecture instead of a centralized architecture like Rain Man. In general, the distributed architectures require more control overhead than Rain Man and do not perform traffic engineering like Rain Man.

In SEATTLE [20], the switches use a link-state protocol to learn the full switch-level topology and compute all pairs shortest paths. In SEATTLE, broadcasts are eliminated, and the control overhead is well distributed by performing location and address resolution through a lookup on a DHT maintained across all the switches. SEATTLE does not address forwarding table state, and SEATTLE restricts load balancing by requiring minimal length paths.

Virtual id routing [21], like SEATTLE, is a distributed architecture that proposes using a DHT for address resolution to eliminate broadcast overhead. However, while SEATTLE distributes the switch level topology to all the switches, routing in virtual id routing is restricted to a tree so that the forwarding state requirements are reduced from $O(N)$ to $O(\log N)$, where N is the number of switches in the network. This does not imply that the physical topology is restricted to a tree, however. Instead, virtual id routing also allows for arbitrary physical topologies. The reduction in forwarding state comes at the cost of usable application bandwidth and load balancing. Virtual id routing, like SEATTLE, cannot use arbitrary routes, but it further restricts routing by not always being able to use minimal length paths.

TRILL [22] is another distributed architecture. TRILL, like SEATTLE, allows for forwarding over arbitrary topologies by running a link-state routing protocol between the switches and introducing a TTL field. Unlike SEATTLE, TRILL does not eliminate broadcasts, and TRILL also incurs the control overhead of running a link-state routing protocol. TRILL also does not affect forwarding table state, but it does improve the distribution of traffic in the network by using redundant links, although it does restrict routing to minimum length paths.

Shortest-path bridging (SPB) [23] is a competing standard with TRILL that also replaces Ethernet's spanning tree protocol and adds multipath forwarding to Ethernet. SPB, like TRILL, does not eliminate broadcasts and relies on a link-state routing protocol. SPB uses the VLAN ID to indicate different paths, which gives the encapsulating switch better control over traffic engineering at the cost of increasing L2 table state by a factor of N. There are two flavors of SPB. SPB - MAC (SPBM), one of the flavors, reduces the forwarding table state by performing MAC-in-MAC encapsulation, so only switch addresses are used in the network.

Last, there is a new standard for Ethernet that may be used in conjunction with Rain Man. Converged Enhanced Ethernet (CEE) is a proposed IEEE 802.1 standard from the Data Center Bridging Task Group for improving Ethernet that allows for simultaneously supporting best-effort and multiple classes lossless communication over an Ethernet network [24], primarily for supporting Fibre Channel over Ethernet. CEE enhances Ethernet with mechanisms for assigning bandwidth to 802.1p priority classes and priority flow control. CEE does not explicitly address scalability. Instead, the mechanism used in the network for control and forwarding is orthogonal to enabling a lossless network. However, CEE can restrict traffic engineering because CEE requires that the routes used in the network be deadlock free. Deadlock freedom is the same as requiring that there be no cycle between any routes. Rain Man can control a network of CEE switches, which is necessary in cases where lossless Ethernet or bandwidth reservations are required.

3.2 Scalability Studies

The scalability study performed in this thesis compares three orthogonal architectural dimensions against three scalability metrics. The architectural dimensions are source versus hop-by-hop routing, arbitrary versus restrictive routing, and the granularity at which flows are routed. The metrics are control overhead, forwarding table state, and usable application bandwidth.

SEATTLE [20], like this thesis, looks at the effects of scale on forwarding table size and control overhead. However, the focus of SEATTLE is primarily on presenting a novel DHT-based architecture and evaluating it as a replacement for Ethernet, so only the scalability of a hop-by-hop, best-effort network with a single path between destinations is considered. This thesis is distinct from SEATTLE in that it evaluates the characteristics of the underlying switch architecture for many different architectures rather than simply evaluating a single architecture.

Shancho, *et al.* [25] study the scalability of restrictive routing. Specifically, they evaluate the decrease in path diversity associated with deadlock routing, which is also studied in this thesis. However, their work only compares the change in path diversity between different deadlock-free routing algorithms. They do not compare against unrestricted routing, as is done in this thesis. Also, their evaluation lacks any real-world traces, and the largest topology used only has 64 nodes, which is not enough for an evaluation of the algorithms at scale.

Two studies estimate the scalability of different architectures on forwarding table state. Ethane [10] estimates forwarding table sizing for hop-by-hop routing. Ethane does not perform any experiments at scale to determine forwarding table size but instead simply extrapolates from the state requirements of a 300-host network that a switch with a fanout of 64 on a university network requires approximately 8,192–16,384 forwarding table entries. Similarly, Shafer, *et al.* [14] also estimate the forwarding table size for source routing on the LBNL network by calculating reuse distance.

The forwarding table size estimates from previous work are not obtained from simulations, and they cannot be compared to each other. This thesis is unique in that it compares source and hop-by-hop routing directly against each other on realistic topologies and workloads.

CHAPTER 4

Rain Man: Scalable SDN Datacenter Switching

SDN is an emerging class of network architectures that enables fast network innovation by letting an external controller manage the behavior of switches in a network. Although the controller manages the switches, no SDN architectures let the external controller directly program the switching hardware. This is to reduce the control bandwidth between the controller and switches and to support vendor interoperability. Instead, SDN architectures have three layers: the network controller, the switch firmware, and the switch hardware. These three layers are connected by two interfaces. The SDN protocol is the interface between the network controller and the switch firmware, and there is a proprietary interface between the hardware and software. The layers of an SDN architecture are illustrated in Figure 4.1. The switch firmware exposes an abstraction of the hardware to the controller through the SDN protocol, and the switch control plane firmware is responsible for mapping this abstraction to the hardware.

OpenFlow, a popular SDN protocol, can be used to meet the requirements of a datacenter network. Unfortunately, current OpenFlow SDN architectures do not scale to large enough topologies because the OpenFlow switch firmware does not fully utilize the hardware.

In this chapter, I present the design of the firmware for Rain Man, a novel SDN

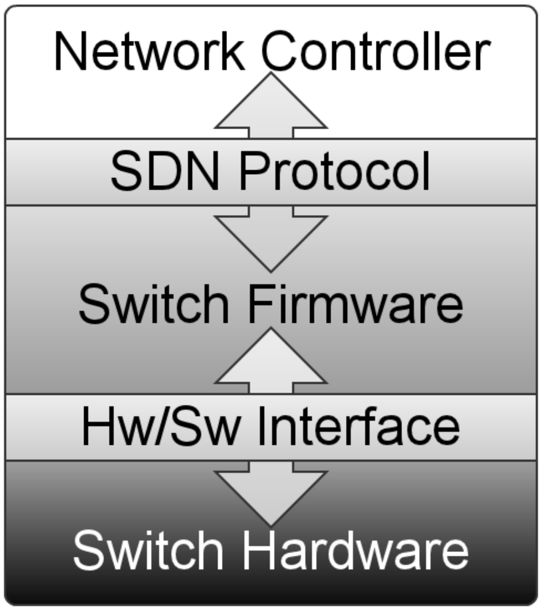

Figure 4.1 : Three Layers and Two Interfaces of SDN Architectures

architecture derived from OpenFlow designed for implementation on COTS hardware. Rain Man is an IBM Research project, and this chapter describes the extensions and analysis that I have performed for Rain Man. Rain Man has the same three layers as all other SDN architectures. In Rain Man, the controller is called Evil Genius, the firmware is named Minion, and the switching hardware is a commodity Ethernet switch. Rain Man uses OpenFlow as its SDN protocol, and is constrained to using the hardware/software interface defined by the hardware. A key insight of Rain Man is that commodity Ethernet switches already contain the hardware capabilities needed to implement high-performance, scalable datacenter networks, without requiring costly core switches or imposing topology restrictions.

One major contribution of Minion, the Rain Man firmware, is providing access to the switches' large L2 (Ethernet) forwarding tables, ECMP tables, and packet sampling functionality. The L2 forwarding table is orders of magnitude larger than a TCAM, so Rain Man can scale to tens of thousands of hosts on arbitrary network topologies. To provide good baseline performance at low state, Rain Man preinstalls default routes that perform ECMP forwarding over multiple minimum-hop-count paths between all host pairs. Sampling packets allows the controller to build global network statistics with less overhead than polling the forwarding tables.

Minion also contributes novel algorithms for improving the utilization of the newly exposed tables. Rain Man increases the number of simultaneous exception routes with a new algorithm for using the L2 forwarding table to route packets toward the same destination along different paths. Exception routes are used in Rain Man to explicitly perform traffic engineering and to support middlebox interposition. The Rain Man firmware also improves the utilization of the ECMP capabilities with a novel set packing algorithm.

Without the contributions of Minion, previous SDN designs achieved scalability only by restricting the topology. In contrast, Rain Man scales efficiently up to 1K switches and 40K hosts on arbitrary topologies. Because Rain Man supports arbitrary topologies, it can use a topology that achieves up to 100% higher throughput than an equal bisection bandwidth instance of the topology required by previous designs.

This chapter is organized as follows. Section 4.1 gives background information on the commodity datacenter Ethernet switch for which Rain Man is designed and discusses the SDN and OpenFlow. Section 4.2 discusses the design of the Rain Man architecture. Section 4.3 and 4.4 discuss my simulator and simulation results. Section 4.5 discusses Rain Man in the context of other network architectures. Finally, Section 4.6 concludes.

4.1 Background

In this section, I give an overview of typical switch hardware capabilities, including a more detailed overview of the switch used in my study. I also provide further details on SDN and OpenFlow networks.

4.1.1 Switch Hardware Overview

The Rain Man prototype is built on IBM BNT RackSwitch G8264 switches [26]. The G8264 is a 1U "top of rack" switch with 48 10GbE and 4 40GbE ports. It supports typical data center networking features including VLANs, port aggregation, Data Center Bridging (DCB)/lossless Ethernet, basic IPv4/v6 routing with RIP/OSPF/BGP, IGMP snooping, etc.

At the heart of the G8264 is a Broadcom BCM56846 ("Trident") switch chip [27] and a Freescale control plane processor. The Trident chip is one of three common commodity switch chips used to implement COTS Ethernet switch hardware, so Minion is applicable to a larger set of switches than simply the G8264. The control processor executes firmware called IBM Networking OS (NOS). In traditional switched Ethernet, the control plane processor is responsible for configuring the local Trident chip and executing any necessary control plane algorithms, such as spanning tree or OSPF.

There has been much discussion about "commodity" switch chips in the networking community, but little is publicly known about these chips. Vendors typically only publish short data sheets that contain the most basic information. Similarly, switch firmware is proprietary and often the key differentiating feature between different vendors' switches. This lack of transparency about both switch chip and switch firmware

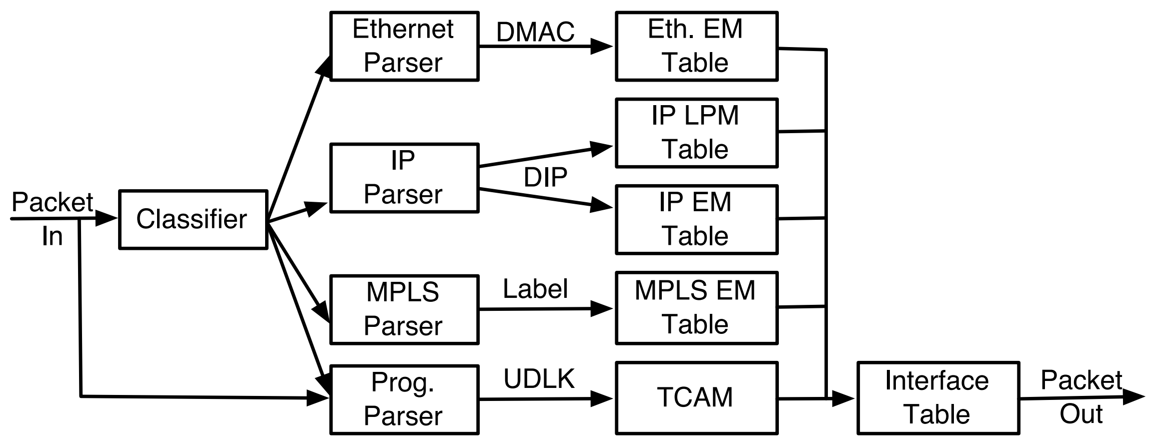

Figure 4.2 : Partial Broadcom Switch Pipeline

Table(s)	Entries
Rewrite TCAMs	2K
Forwarding TCAMs	2K
L2	100K
ECMP	1K

Table 4.1 : Switch Table Sizes

internals makes it difficult for networking researchers to know what kind of innovations are low-cost, *e.g.*, requiring only firmware changes, and which are high-cost, *e.g.*, requiring changes to the switch chip.

I am in the fortunate position of having access to detailed documentation on the Trident switch chip and the source code of the IBM NOS firmware. This level of documentation is necessary to be able to extend OpenFlow to support a richer set of hardware functions in Rain Man.

Figure 4.2 presents a high-level overview of the relevant portion of the Trident packet processing pipeline, and Table 4.1 presents the approximate number of entries in each of the forwarding tables shown in Figure 4.2. The table sizes presented in Table 4.1 are similar to the table sizes of comparable 10GbE switches. The Cisco Nexus 5000 Series Switches [28, 29] have a 16,000-32,000 entry L2 table and a 2,048 entry egress TCAM, and the HP 5900 Switch Series [30] has a larger L2 table that supports 128,000 entries. No data is published for the sizes of the other tables in the Cisco and HP switches, so no further comparisons can be made.

The Trident pipeline contains dozens of additional configurable tables, and in general Trident supports many features not exploited by Rain Man, *e.g.*, IP routing, DCB, and multicast. I limit my discussion below to those features that are relevant to Rain Man. The pertinent portions of the Trident packet processing pipeline operate as follows. Many details are elided due to the confidential nature of the source material.

The rewrite and forwarding TCAMs are flexible match tables that support wildcards and can match on most packet header fields (VLAN id, src/dst MAC address, src/dst IP, src/dst TCP port, etc.). The rewrite TCAMs are used to perform packet header modifications. The output of the forwarding TCAM is an output port or group, which is used in forwarding traffic. Output groups—similar to the same concept in OpenFlow 1.1—can be thought of as virtual ports used for multipathing and multicast functionality.

The L2 table performs an exact match lookup on only two fields, VLAN ID and destination MAC address. The output of the L2 table is also an output port or group. In traditional Ethernet, the L2 table is the workhorse of the switch.

If the output of the lookup stage is an ECMP group, the switch hashes certain configurable header fields (usually src/dst IP address and src/dst port number) to choose which port from the group the packet should be forwarded through. The switch chip supports around one thousand ECMP groups, and each group contains a list of output ports. Typically the ports in an ECMP group are chosen to be equally close to a particular destination; I describe detailed routing algorithms in the next section.

For statistics collection, each forwarding table entry and each physical port contains byte and packet counters. These counters can be used to gather per-flow and per-port statistics, but Curtis *et al.* [12] demonstrated that polling the switch for per-flow statistics quickly overloads the switch control plane, an observation I can corroborate.

As an alternative to frequent polling, the Trident chip supports sFlow [31] to gather statistics in a scalable fashion. When sFlow is enabled, the switch pushes periodic per-port statistics and sampled packet headers to a collector server. Packet sampling copies every N^{th} packet header, encapsulates it in an sFlow header, and sends it the collector. Because the sampling rate is configurable, the rate can be adjusted to trade-off statistics accuracy and switch load.

To provide an OpenFlow control plane performance baseline, I characterize the G8264's flow update performance using custom microbenchmarks build on the OFlops [32] framework.

The first test measures the maximum rate of flow mods per second by repeatedly sending a variable-length batch of OFPT_FLOW_MOD messages followed by one OFPT_BARRIER_REQUEST. Figure 4.3 shows the G8264's measured flow update performance for batch sizes ranging from one to twenty. When each flow update is sent and synchronized independently, the G8264 can sustain roughly 700 updates per second, which increases to over 1500/second for large batches.

The second test measures the rate at which the G8264 can generate maximum

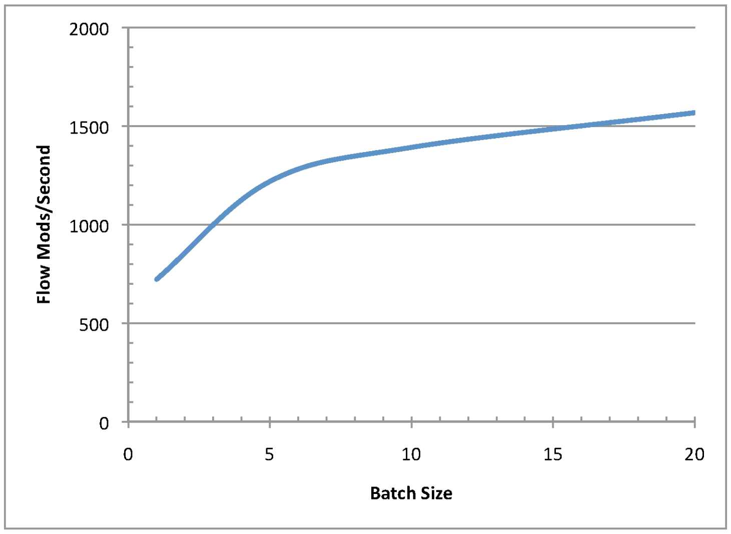

Figure 4.3 : G8264 Flow Update Performance

packet_in messages, which are used to forward broadcast ARP requests and packets that do not match any rule to the controller for handling. I found that the G8264 can generate 200 packet_in messages per second, which seems fairly low considering that the switch has 64 ports. This observation makes the case that SDN architectures should minimize the use of this feature (i.e., avoid reactive routing). I have not yet determined whether flow modifications and packet_ins are bottlenecked by the switch chip, the control plane processor, or the firmware.

4.1.2 SDN and OpenFlow

OpenFlow is an SDN architecture originally proposed as "a way for researchers to run experimental protocols in the networks they use every day" [11]. OpenFlow lets an external controller insert forwarding rules in a switch's match-action tables, rather than having the switch "learn" its own rules via a bridging or spanning-tree mechanism. By offloading most of the control plane to a (logically) centralized server, OpenFlow

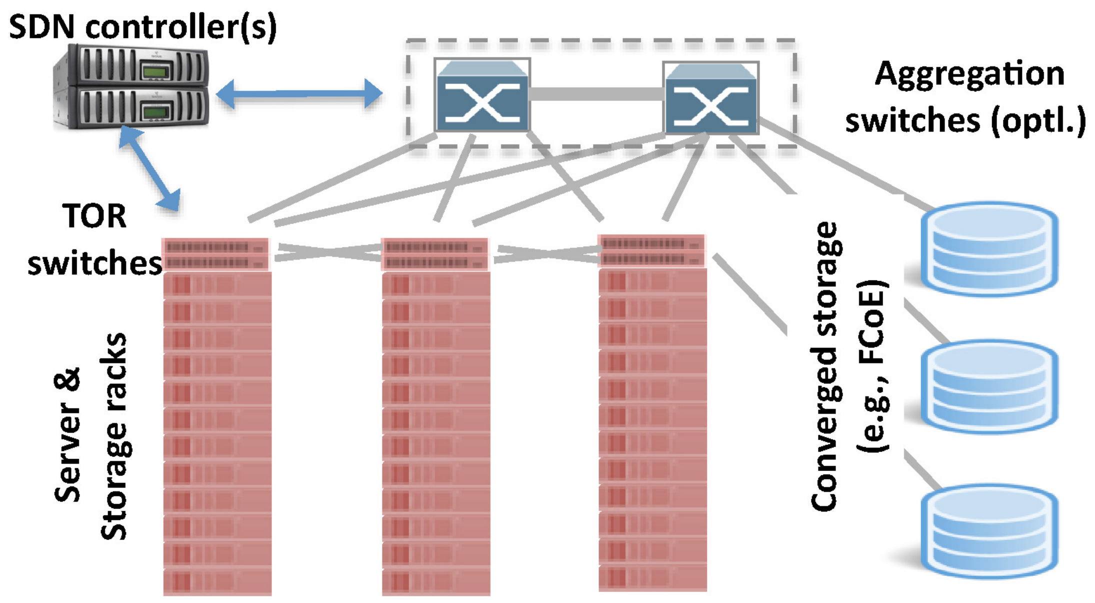

Figure 4.4 : Example SDN-based Datacenter Network

has the potential to simplify switch design, optimally route traffic, reduce forwarding state, eliminate route convergence time, consistently enforce QoS and security policies, simplify middlebox interposition, and better integrate virtual switches.

Although OpenFlow was not designed as an Ethernet substitute, the notion of entirely OpenFlow-controlled networks is gaining momentum [1, 2, 33, 34]. Figure 4.4 illustrates the major components of a data center network architecture implemented using SDN.

Within the datacenter network, OpenFlow is used exclusively to manage the network fabric, allowing innovation that is not constrained by compatibility with legacy protocols such as the Spanning Tree Protocol. Traffic destined for external hosts retains backwards compatibility with existing infrastructure by using standard networking protocols, such as Ethernet and IP. As an optimization, if external traffic is destined for another network that is also OpenFlow-enabled, then the two network controllers can coordinate by sharing information or by nominating one controller to be the master for inter-network traffic.

In this context, OpenFlow is not required to operate alongside traditional Ethernet, and all switch hardware features can be exploited by the OpenFlow-based network architecture. This is not how the hardware is used in typical OpenFlow switch firmware. Existing OpenFlow implementations use the few-thousand-entry TCAMs while ignoring the hundred-thousand-entry L2 table, among other tables.

In general, modern switches have many capabilities that should be made accessible through OpenFlow, starting with the large L2 exact-match forwarding table. Currently, important features like Data Center Bridging [24] (DCB), TRILL [22], AQM/ECN, and sFlow [31] must be configured outside of OpenFlow. These features are necessary for production networks, and their absence from the OpenFlow protocol standard makes it difficult to support pure OpenFlow networks. For example, the absence of support for DCB makes it difficult to support converged IP/storage networks and prevents an OpenFlow controller from dynamically adjusting bandwidth reservations.

Each layer of the network infrastructure contains potential bottlenecks. While controller scalability has largely been addressed with controllers achieving millions of flow modifications per second [35], switches remain a bottleneck.

The switch control plane is a potential bottleneck because the control plane processor is typically underprovisioned to reduce cost. The switch I evaluated can perform only 700-1,600 flow modifications per second even with no other load on the control plane. In practice, packet-in messages and statistics collection compete for the switch processor, which exacerbates the problem.

The switch hardware forwarding table capacity is another potential bottleneck.

Forwarding table state is an all or nothing resource. There is no downside to adding entries to a table until it becomes full, at which point the performance impact skyrockets. If a forwarding table is completely full when the controller wants to add a new entry, either the new entry and its corresponding flow will be denied or an old entry needs to be evicted. Both of these options impose extra latency, which is serious problem in a datacenter network [3].

4.2 Rain Man Architecture

While the contribution of this chapter is the design of Minion, the Rain Man switch firmware, I must discuss the whole Rain Man architecture to understand the design of Minion. Also, I must discuss the whole Rain Man architecture because changing the switch firmware in Rain Man also requires that Evil Genius be changed to use the new features exposed by Minion.

The Rain Man architecture consists of three main components: default flow routing, statistics collection, and exception flow routing. Default flow routes are preinstalled for low-latency communication without requiring a packet_in and controller action, and they can be used with or without security. Statistics collection is used to identify sub-optimal default routes, which can then be rerouted with exception routes to improve the traffic distribution. Exception routes are also necessary for logical middlebox interposition without requiring physical interposition. Rain Man is topology independent, and can be used to control a network with any arbitrary topology.

4.2.1 Default Routes

Rain Man preinstalls default routes between all pairs of hosts that are allowed to communicate. This is because requiring a flow-setup for each flow in the network does not scale. Reactively routing every flow at the controller overloads the switch control plane, and imposing a flow-setup latency for every flow in the network has an unacceptable performance penalty for datacenter networks [3, 12]. Packets to unknown destinations are dropped or sent to the controller, not flooded. Note that Rain Man does not rely on flooding to drive address learning. As soon as a host sends a single packet, the controller discovers its location and proactively installs routes for all legal destinations within the network.

In Rain Man, default routes can either be insecure or secure. For simplicity, default routing without security is described first, then adding security will be discussed.

Rain Man does not simply use a single path as the default route between any two hosts. Instead, Rain Man uses all of the minimum-hop-count paths between any given pair of hosts as the default routes. To install all of the minimum-cost paths, Minion uses the L2 and ECMP tables, and the specific route used by a given microflow is determined via ECMP.

Evil Genius first computes all of minimum-hop paths between all switch pairs. The first hop from each path in the set of paths from switch i to switch j is used to build a set of output ports, P_{ij}. If $|P_{ij}| > 1$, then P_{ij} is assigned a group id and sent to Minion on switch i, which then adds the group to the ECMP table.

Then, Evil Genius sends a rule for every host in the network to Minion on every switch in the network, which Minion adds to its L2 table. The L2 table matches on $< MAC, VLAN >$ tuples, so the entry added for host h is $< MAC_h, 0 >$, where

MAC_h is the MAC address of h and 0 is the default VLAN. At switch i, if host h is attached directly to the switch, the value stored in the L2 table entry for h is simply the port number on i where h is attached. Similarly, if h is attached to switch j and $|P_{ij}| = 1$, then the value stored in the L2 table entry for h is the single port number in P_{ij}. If h is attached to switch j and $|P_{ij}| > 1$, then the value stored in the L2 table at switch i is a pointer to the group id of $|P_{ij}|$ in the ECMP table. The proceeding lookup into the ECMP table will determine the exact output port used by a flow traversing through i towards h.

4.2.2 Statistics Collection

Rain Man uses a combination of sFlow [31] and per-port counters to build a global view of network utilization. sFlow is a simple protocol that requires switches to sample packet headers a configurable rate and send the selected packets' headers as sFlow datagrams to an sFlow collector, which is a central server running software to analyze the network traffic.

Minion exposes sFlow functionality, and Evil Genius simply uses the existing IBM AURORA network monitoring tool [36] to build per-flow statistics out of raw sFlow or NetFlow samples. AURORA calculates the top k flows and the controller periodically fetches them using a REST interface. sFlow statistics are inherently approximate, so congestion caused by a confluence of many small "mice" flows will likely not be detected by AURORA's "top k flows" mechanism. Thus, the Rain Man controller also reads per-port switch counters to improve the accuracy of congestion detection. Unlike per-flow counters, reading per-port counters imposes only a modest overhead. Assuming a 64-port switch and 88-byte per-port counters, sampling these counters multiple times per second requires less than 1 Mbps, which should not tax the switch control plane processor.

The per-port counters are not strictly necessary for the operation of Rain Man. Because the ECMP hashing algorithm is known, the path taken by any flow, default or not, is known by the controller. The sFlow statistics can be combined with flow path information to approximate per-port utilization. Since many small “mice” flows may never be detected by sFlow, per-port counters are used to improve the accuracy of the statistics.

4.2.3 Exception Routes

Exception routes in Rain Man cause a flow to traverse a different path than it would by using the default routes. These are used to both improve performance and allow virtual middlebox interposition. In Rain Man, exception routes can be used to route at any flow granularity, from destination-based to the full TCP header. However, for traffic engineering purposes, Rain Man only ever installs a single exception route between any two hosts.

Although ECMP can provide good load balancing, Al-fares *et al.* [2] showed that hash collisions between *elephant* flows can cause significant performance degradation. Elephant flows are defined as flows that carry a significant portion of the bytes in the network. Hash collisions can cause multiple elephants to traverse the same link(s), causing congestion and reducing throughput compared to optimal routing. Like Hedera, Rain Man uses network statistics to detect elephant flows and attempts to reduce congestion by re-routing them over less-congested links with exception routes.

Middleboxes are used in networks to provide services such as firewalls, network address translation, and intrusion detection. Currently, middleboxes are required to physically interpose themselves in the network between the source and destination.

With exception routes, middleboxes no longer need to physically interpose themselves in the network. Instead, exception routes can be used to logically interpose middleboxes.

When Evil Genius wants to build a new exception route, it sends a rule to every Minion along the chosen route. The exception route mechanisms is independent of whether the route is for performance or middlebox interposition, although there are two different mechanisms for implementing exception routes in Rain Man. Evil Genius has a choice to place exception routes either in the TCAM or L2 table, depending on table occupancy along the path.

TCAM exception routes are implemented as they would be with standard OpenFlow. Evil Genius defines the flow headers, and communicates the flow headers and output port to each Minion along the path. The Minions then simply add the rule to the forwarding TCAM, indicating that the exception rules are higher priority than the default rules in the L2 table.

L2 exception routes are able to route arbitrarily defined flows with the L2 forwarding table by taking advantage of the fact that there are unused VLANs. L2 exception routes require that the first hop tag the packet with a previously unused VLAN. After the tag has been added, the flow can be uniquely identified by its MAC address and VLAN. Subsequent hops can then use the L2 table to forward the flow. However, before the packet can leave the network, the VLAN tag must be removed. This is done at the second to last hop on the path to save forwarding state. Rain Man uses default routes, and all routes toward a given host, whether they are default or exception routes, use the same last hop, so it would waste forwarding state to duplicate the last hop forwarding. L2 exception routes require L2 forwarding table state at all the switches from the first to second-to-last hop. Because L2 exception routes also require VLAN tagging, they also require rewrite TCAM state at both the first and second-to-last hop.

L2 table exception routes use different forwarding tables than TCAM exception routes, so, in Rain Man, Evil Genius chooses between using TCAM and L2 exception routes based on the table occupancy of the switches. This increases the total number of exception routes supported by the network over using only TCAM or L2 exception routes.

4.2.4 Mobility

Rain Man is easily able to handle host mobility. When the status of a physical port changes, e.g., when a new physical host joins or leaves the network, the controller is notified. This allows the controller to update the appropriate forwarding tables. When virtual hosts move in the network, they send out a gratuitous ARP. This packet is sent to the controller, which can then update the appropriate forwarding tables. As an optimization, the controller may install a temporary exception route so that all traffic sent to the old location is forwarded to the new location while the forwarding tables are being updated.

Security

Default flow security in Rain Man is optional. Security requires additional forwarding table state, which can limit scalability in a network that does not require security, such as a single occupant datacenter.

Rain Man supports VLANs to isolate groups of hosts from each other. Each host (identified by MAC address) is assigned to a VLAN. Edge switches add VLAN tags as necessary to all packets entering the network and strip tags as packets leave the network; all packets on interior switch-to-switch links must be tagged. To support this capability, each host requires one rewrite TCAM entry per VLAN at both the ingress and egress switches. In addition, at each switch an L2 forwarding table entry is preinstalled for each valid host and VLAN combination.

If a host attempts to send to a MAC address on a different VLAN, the packet will be dropped at the ingress switch because there will not be an L2 forwarding table entry for the packet. Further, because the network controller intercepts broadcast ARP and ND packets, a host should be unable to discover the MAC address of any host that is not on the same VLAN, in the absence of a side channel attack.

4.2.5 Failure Handling

Failures and other network events like hosts joining and leaving also impact scalability and should be expected as regular events. Recent network traces [37] saw 36 million error events in a year across a network of hundreds of thousands of servers. Failures occur on every level of the infrastructure, with the most common causes being network misconfiguration, firmware bugs, and hardware failure. For a network to be scalable, it must be designed to minimize both the number of failures and the performance impact of failures.

In Rain Man, it is possible to implement devolved failure handling locally in each switch. If the controller specifies multiple output ports on a switch for a given destination, using either ECMP groups or multiple rules, then port failures can be handled gracefully by having the switch simply remove all rules that use the failed port. Doing so quickly redirects subsequent traffic intended for that destination to the remaining output port(s). The switch can also disable any exception flows passing through the failed port, allowing the traffic to flow over a suboptimal but working default route. The controller can be notified out-of-band so that it can update any exception flows that pass through the failed component and possibly add new rules (or change the existing ECMP entries) to re-establish the desired level of multipathing.

4.2.6 Virtualization

In virtualized data centers with virtual switches, the vSwitches should also support OpenFlow and should be controlled by the same Rain Man controller as the physical switches. In such networks, security and mobility would be handled completely by the vSwitches; vSwitch-to-pSwitch links would be considered interior links. If the network has more VMs than Rain Man can support, the vSwitches could implement an overlay technology such as NetLord [5], SecondNet [4], or VXLAN [38] while Rain Man would provide an efficient underlay network.

4.3 Methodology

Rain Man is designed to exploit the capabilities of IBM BNT RackSwitch G8264 switches. Rain Man was built on a 4-switch, 20-server testbed to validate the feasibility of the design. To evaluate issues that arise at scale and explore the scaling limits of different designs, a custom discrete event network simulator was used as the basis for the results presented below. The simulator can replay flow traces in open-loop mode or programmatically generate flows in closed-loop mode. For performance, the simulator models flows instead of individual packets, omitting the details of TCP dynamics and switch buffering. The bandwidth consumption of each flow is simulated by assuming that each TCP flow immediately receives its max-min fair share bandwidth of the most congested link that it traverses.

I then use the simulator to compare Rain Man against other SDN architectures

Algorithm 1 – Flow rate computation

```
Input: a set of flows $F$, a set of ports $P$, and a rate $p.rate()$ for each $p \in P$. Each $p \in P$ is
  a set of flows such that $f \in p$ iff flow $f$ traverses through port $p$.
Output: a rate $r(f)$ of each flow $f \in F$
  begin
  Initialize: $F_a = \emptyset; \forall f, r(f) = 0$
  Define: $p.used() = \sum_{f \in F_a \cap p} r(f)$
  Define: $p.unassigned\_flows() = p - (p \cap F_a)$
  Define: $p.flow\_rate() =$
    $(p.rate() - p.used())/|p.unassigned\_flows()|$
  while $P \neq \emptyset$ do
    $p = \arg\min_{p \in P} p.flow\_rate()$
    $P = P - p$
    $rate = p.flow\_rate()$
    for all $f \in p.unassigned\_flows()$ do
      $r(f) = rate$
      $F_a = F_a \cup f$
    end for
  end while
```

on three different datacenter workloads and three different datacenter topologies.

4.3.1 Network Simulator

The simulator uses Algorithm 1 to compute the rate of the TCP flows in the network. This algorithm was first used by DevoFlow [12] in their network simulator. The end result of this algorithm is that each flow receives its fair share of bandwidth of the most congested port it traverses. The flow rates assigned are most realistic of simulating a network that uses XCP [39] as its transport protocol because flows that use XCP converge very quickly to min-max fair bandwidth allocations.

There are many factors that affect the latency of the controller detecting an elephant flow and installing an exception route for the flow. These include the time for its packets to be sampled by sFlow, the time to compute a new exception route, and the time for the rule to be installed on all the switches in the network. In order to simplify the simulation, I simply assume that there is a 100ms latency between when a flow becomes an elephant to when the exception route is installed. Similarly, I assume that there is a 100ms latency between when a flow stops and when the exception route is removed.

The specific algorithm used by Evil Genius to re-route elephant flows is orthogonal to the design of Minion, but for simulation, I assume that Evil Genius is using a greedy worst fit algorithm. This effectively means that each elephant flow is routed independently when it is detected and is placed on the path with the most available bandwidth. While this algorithm is know to not be optimal, it is also not very computationally intensive when compared with other routing algorithms, such as simulated annealing. I chose this algorithm because I believe it may be good enough and that routes can be computed within the required latency.

Similarly, the Evil Genius can install exception routes for elephant flows at any flow granularity. For simulation, I assume that Evil Genius installs exception routes at the host pair granularity. This means that the exception route is installed to route all traffic between a given source and destination along the same path. I chose this algorithm because it is lower state than independently routing each TCP flow.

4.3.2 Workloads

To evaluate Rain Man, I use three different datacenter workloads. The first workload is generated synthetically from statistics published about traffic in a datacenter at Microsoft Research (MSR) by Kandula *et al.* [40]. The second workload is a closed-loop data shuffle based on the MapReduce/Hadoop workload, and the third workload is a mix of the first two.

Kandula *et al.* [40] instrumented a 1500-server production cluster at MSR for two months and characterized its network traffic. I generated synthetic traces based on these characteristics. Specifically, I sample from the number of correspondents, flow size, and flow inter arrival time distributions for intra-rack and entire cluster traffic. This workload includes few elephant flows, so traffic engineering is not much benefit.

Inspired by Hedera [2] and DevoFlow [12], I also examine a synthetic workload designed to model the shuffle phase of a map-reduce analytics workload. In the shuffle workload, each host transfers 128 MB to every other host, maintaining k simultaneous connections. In this workload, every flow becomes an elephant, which stresses the central allocator and causes this workload to benefit from traffic engineering.

Lastly, I also examine a mix of the two workloads - a shuffle with MSR-like traffic in the background.

4.3.3 Topologies

Four different topologies were used in the evaluation of Rain Man: the folded-Clos or fat tree topology, the HyperX topology, the random Jellyfish topology, and the optimal topology, which is simply a single, large non-blocking switch. While much of the literature focuses on full-bisection-bandwidth networks, I believe it is valuable to study oversubscribed networks as well. Given the order-of-magnitude difference in link speeds, in many datacenters 1 Gbps (or a few bonded links) per server is not sufficient but 10 Gbps is overkill.

The three non-optimal topologies are built with equal bisection bandwidth ratios so that they are comparable. Informally, the bisection bandwidth ratio of a graph is the ratio of the bandwidth of the links that cross a cut of the network to the bandwidth of the hosts on one side of the cut, for the worst case cut of the network. Formally, the bisection bandwidth ratio of a network $G = (V, E)$ is:

$$bisec(G) = \min_{S \subseteq V} \frac{\sum_{e \in \delta(S)} w(e)}{\min\{\sum_{v \in S} r(v), \sum_{v \in \bar{S}} r(v)\}}$$

where $\delta(S)$ is the set of edges with one endpoint in S and another in $\bar{S}$, $r(v)$ is the total bandwidth that vertex v can initiate or receive, and $w(e)$ is the bandwidth of edge e. This definition is based on the one given in REWIRE [41]. Bisection bandwidth is a useful metric for comparing topologies, because equal bisection bandwidth ratio topologies are bounded to the same worst case performance.

Folded-Clos (Fat Tree)

The folded-Clos (also called *fat tree*) topology has long been used in supercomputer interconnects, and in recent years it has been proposed for use in Ethernet-based commercial data center networks [42, 1, 43]. Clos networks have similar performance on all traffic patterns. This is because they effectively implement valiant load balancing by sending all traffic through intermediate core switches.

The size of a fully-provisioned Clos network with all hosts located on the same level is determined by the switch radix. Available sizes tend to be “lumpy”; e.g., with 64-port switches a two-tier network supports 2,048 hosts and a three-tier network supports 65,536 hosts. The lumpiness of fully-provisioned Clos networks can be solved with Extended Gerneralized Fat Trees (EGFTs) [44]. EGFTs for an arbitrary number of hosts can be built by allowing hosts to reside at different levels in the tree. Effectively, any switch or host in the fat tree can be replaced with another fat tree.

Every host in an L-level Clos network can be assigned an L-dimensional positional address. Switches can then perform longest-prefix matching (LPM) on this address for low state forwarding. Each switch needs a single LPM rule for each subtree below it in the fat tree. All remaining traffic can be routed towards an arbitrary core switch, from which all hosts can be reached. Single-path routing simply designates a single uplink port as the default path. Multipath routing can be done using hashing to choose between all available uplinks.

HyperX

The HyperX topology [45] and its less general form, the Flattened Butterfly topology, have also made the transition from supercomputing to Ethernet in recent papers [46]. HyperX topologies can support uniform traffic at a lower cost than Clos, because a 2:1 oversubscribed HyperX can forward uniformly distributed traffic from every host at full line-rate. This is in contrast with a 2:1 oversubscribed Clos topology, which can only forward traffic from every host at half line-rate [7]. Flattened Butterfly networks also suffer from similar lumpiness as Clos networks, but the more general HyperX topology solves the lumpiness problem in a manner similar to EGFTs.

Every host in a HyperX network can be assigned a positional address that encodes its coordinates. The diameter of a HyperX network is equal to the number of dimensions, and minimal routing proceeds by repeatedly choosing a dimension and forwarding the packet to the switch whose address in that dimension matches the destination address. A common single-path routing algorithm is dimension-order routing. Multipath routing can be done using hashing to choose the next dimension instead of a fixed order. I believe dimension-hashing would be simple to implement in hardware, but, to my knowledge, it is not available in any Ethernet switch chip.

Jellyfish

The recently-proposed Jellyfish topology [47] connects switches using a regular random graph. The properties of such graphs have been extensively studied, thus I do not have to start from scratch in analyzing its properties. The forte of the Jellyfish is its flexibility: switches in the network may have different numbers of attached hosts, which may be advantageous in more heterogeneous enterprise networks. A Jellyfish is also easily upgradeable; switches do not have to have the same radix, which allows multiple generations to coexist, and new switches can be added to the network with minimal re-cabling. Calculation of bisection bandwidth becomes complicated in heterogeneous Jellyfish networks, so I only simulate homogenous ones. Both upgrades and equipment failures preserve the Jellyfish's lack of structure. Jellyfish is very cost-effective since all switch ports are connected – either to hosts or to other switches; this maximizes use of expensive hardware.

The Jellyfish's lack of structure makes wildcard routing impossible unless one is willing to accept stretch, as in virtual id routing [21] and Ethernet on Air [48].

4.4 Evaluation

4.4.1 Throughput

Figure 4.5 shows the simulated aggregate throughput of the MSR+shuffle workload on all three topologies with three different oversubscription ratios. Each topology

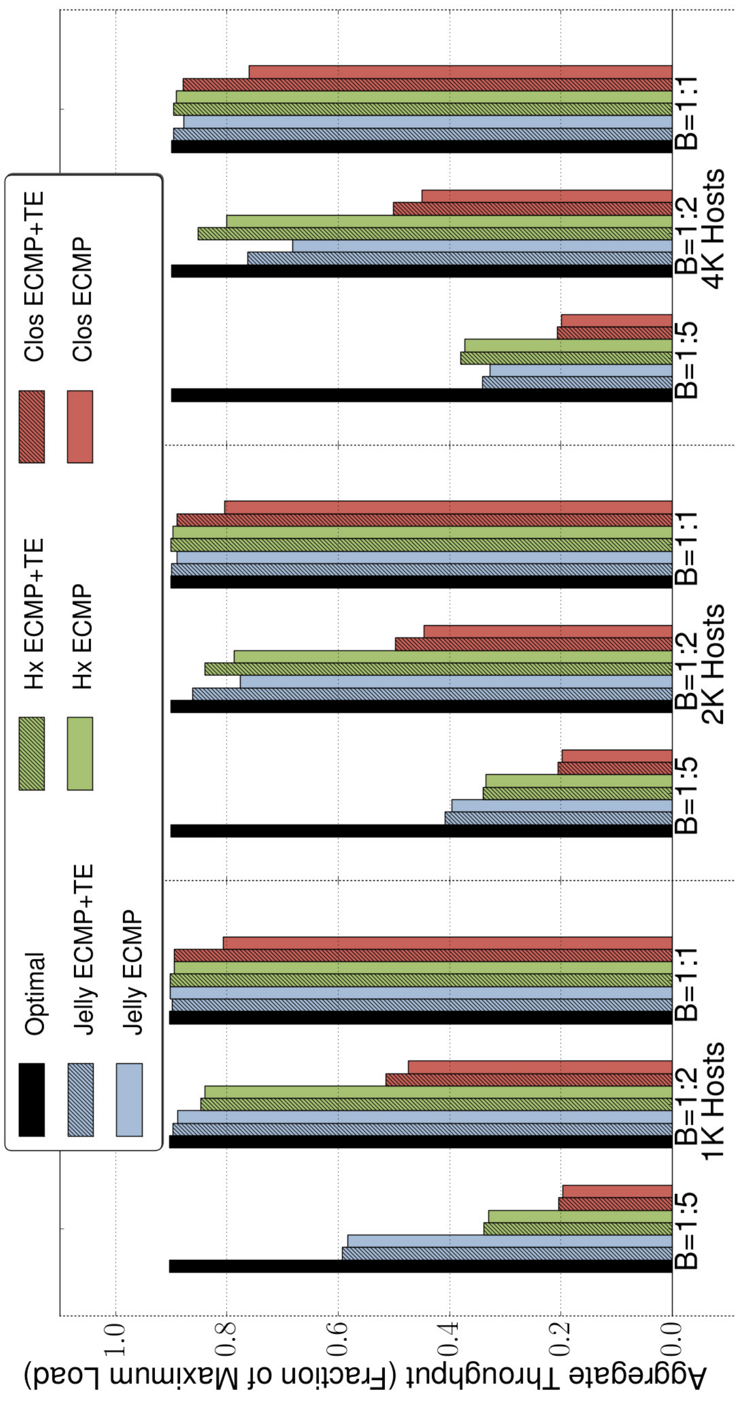

Figure 4.5 : MSR+Shuffle Workload Throughput - Equal Bisection Bandwidth

was simulated with just default multipath routing and multipath routing plus traffic engineering. *Jelly*, *Hx*, *Clos* refer to the Jellyfish, HyperX, and folded-Clos topologies, respectively. *ECMP* means that only multipath routing is used. Similarly, *ECMP+TE* refers to simulations that use both multipath routing and traffic engineering. *B=* is used to refer to the bisection bandwidth ratio of the topology. For example, *B=1:5* refers to a topology where the bisection bandwidth ratio is 1:5.

Figure 4.5 shows that the benefit of traffic engineering is marginal at low bisection bandwidth ratios, while it can improve performance by up to 10% as the bisection bandwidth ratio increases. This is because, at low bisection bandwidth ratios, flows are expected to collide, so hash collisions are of less importance. This is in contrast with the full bisection bandwidth topologies, which are nonblocking. In this case, any time there is a hash collision, there is another path that can be used for one of the colliding flows. As expected, traffic engineering is important to achieve near-optimal performance on high bisection bandwidth ratio topologies.

The throughput difference between the HyperX and folded-Close topologies as they are oversubscribed is dramatic. On the B=1:2 topologies, the folded-Clos topology achieves roughly half of the optimal throughput, whereas the HyperX topology is near optimal. This is because the folded-Clos topology effectively implements valiant load balancing. Choosing a random, intermediate core switch to forward through first performs well under adversarial workloads, but it also limits the network throughput to no greater than the worst case performance. In contrast, while bisection bandwidth limits worst case performance on a HyperX topology, the HyperX topology can have higher throughput under uniform workloads like the shuffle workload. This is because the HyperX topology does not have internal nodes. Under uniform traffic, half of the traffic does not traverse the worst case cut. These performance results agree with the data presented in the original flattened butterfly paper [7].

While the throughput of the Jellyfish topology at low hosts is significantly higher than both of the other topologies, the performance drops as the network size increases. This is because the Jellyfish topologies are sized according to asymptotic bisection bandwidth, although the topologies are still directly comparable because the Jellyfish uses fewer switches. On a Jellyfish topology, there is fixed ratio between uplink and downlink ports on each switch. As the network diameter grows, the switch-to-switch links must carry more traffic. At 4K hosts, the performance of the Jellyfish is already worse than that of the HyperX. Although the performance of the Jellyfish will never be worse than that of the folded-Clos because it too is bounded by its bisection bandwidth, the trend in the data implies that the performance under uniform bandwidth approaches the bisection bandwidth.

4.4.2 Forwarding State

Figure 4.6 shows the forwarding table utilization of multipath routing and multipath routing with traffic engineering for different state models on three topologies. The results are from the B=1:2 version of the topologies with 2K hosts with the MSR+shuffle workload. The total amount of forwarding table state is presented as the fraction of forwarding table state on the BNT G8264 that is used. The *OpenFlow* state model is where the TCP granularity ECMP flows and the exception routes are all routed independently using a TCAM. *OpenFlow L2* is where the ECMP flows use the TCAM, and the exception routes use the novel L2 exception route algorithm. *OpenFlow ECMP* is where the ECMP flows use the TCAM and there are no exception routes. The OpenFlow ECMP model is most similar to typical OpenFlow controllers. In the model, all TCP flows are routed independently along minimum hop-count

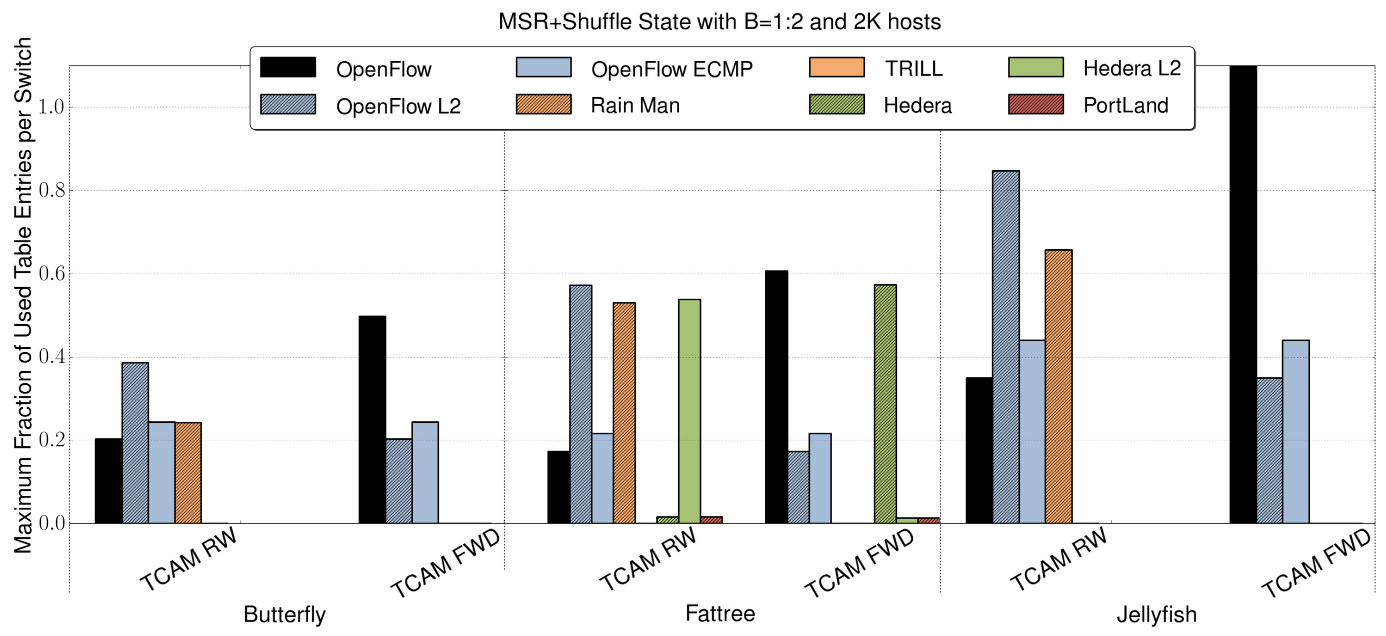

Figure 4.6 : MSR+Shuffle Workload Swtich State - Equal Bisection Bandwidth

paths. The default routing module for the Beacon OpenFlow controller [49] also installs routes in this way. *Rain Man* is meant to represent the state requirements of actually using Rain Man. The ECMP routes use the L2 and ECMP tables, while all of the exception routes use the novel L2 exception route algorithm. Note that in this model, no exception routes are placed in the forwarding TCAM because the rewrite TCAM was not yet full. This effectively means that there are 2K additional exception route entries per switch. In *TRILL*, ECMP routes use the L2 and ECMP tables, and there are no exception routes. This represents the state requirements of using TRILL with ECMP or Rain Man without exception routes. The *Hedera*, *Hedera L2*, and *PortLand* models all use a single wildcard rule in the TCAM along with the ECMP tables for ECMP routing. This is only possible on the folded-Clos topology, so the results are only presented for the folded-Clos topology. In *Hedera*, the exception routes use the TCAM. In *Hedera L2*, the exception routes use the novel L2 exception route algorithm, and there are no exception routes in *PortLand*.

The only state model that requires more state than is present in the BNT G8264 is the OpenFlow model on the Jellyfish topology. It does not do much better on the other topologies, where it uses at least half of the available state. Using the novel L2 exception route algorithm slightly reduces the state requirements. The OpenFlow ECMP model, which is shown in Figure 4.5 to reduce performance by 10%, requires half the state of the OpenFlow model, although at best the tables are already 20% full.

The model with the next highest table utilization is Rain Man on the Jellyfish topology at 67%. On the folded-Clos topology, Hedera, Hedera L2, and Rain Man all use a similar fraction of the forwarding table state for exception routes, Hedera in the forwarding TCAMs and Hedera L2 and Rain Man in the rewrite TCAMs.

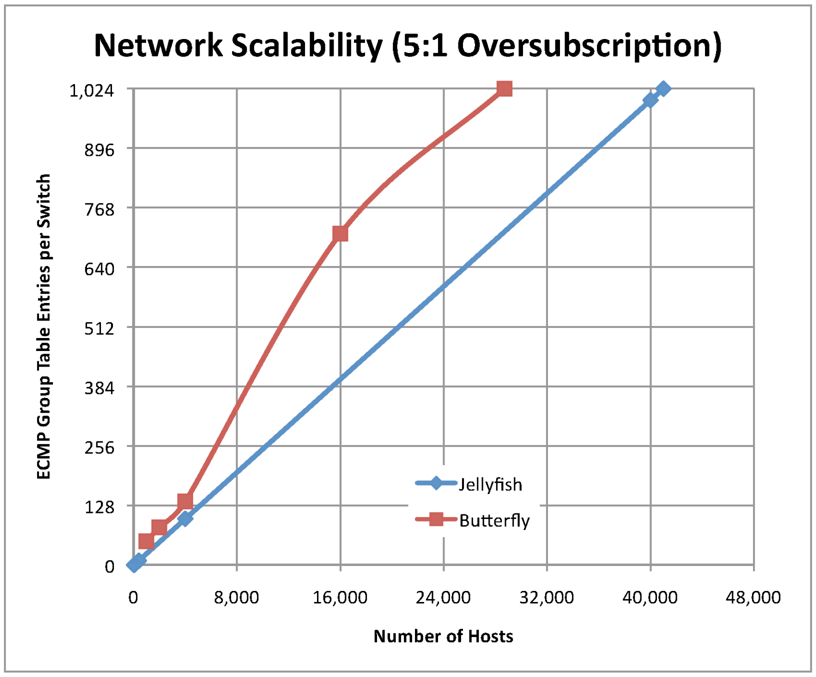

Figure 4.7 : Network Scalability

Expecting that larger topologies will use a proportional amount of forwarding table state, any of the algorithms that use exception routes will fill their tables by 10K hosts, even if both TCAM and L2 exception routes are used. Although, filling these tables is not critical because no traffic is denied from the network, even if they are full. Performance is lower bounded by the ECMP throughput.

Although ECMP in PortLand scales at constant state, ECMP with Rain Man on the Jellyfish and HyperX topologies also scale to large networks. Rain Man's ECMP scalability is limited by the sizes of the tables in the switch datapath. In particular, the 1K-entry ECMP group table is the first to fill up in butterfly or Jellyfish networks, limiting them to ∼1K switches. Depending on oversubscription ratio, this limits Rain Man networks to approx. 20K-40K hosts, as Figure 4.7 illustrates.

4.5 Discussion

While Rain Man is a new SDN architecture, it does share some similarities with existing network architectures. In this section, I will discuss the design of Rain Man in the context of related network architectures.

It is possible to configure TRILL for ECMP routing. TRILL would set up the L2 and ECMP tables in the same way as described in Section 4.2.1. The key difference between default routing in TRILL and Rain Man is that, in TRILL, the switch control planes use IS-IS to discover the network topology and each switch computes its own paths to every other switch. In contrast, Rain Man makes the network controller responsible for building and updating the L2 and ECMP tables for all switches. Using IS-IS in TRILL imposes significant computational overhead and can overload the control plane processor. Using IS-IS also means that TRILL does not support exception routes.

DevoFlow, like Rain Man, suggests using sFlow for statistics collection. However, DevoFlow, mentions the existence of the L2 table, but does not consider using it to reduce pressure on the other forwarding tables. Also, DevoFlow is not implementable on any commodity switch chip, whereas Rain Man is.

Without the constraint that Rain Man remain compatible with existing Ethernet hosts, it is possible to build a more scalable network. NetLord [5] is a network virtualization system that, like Rain Man, is concerned with reducing forwarding state. However, NetLord is orthogonal and complementary to this work. NetLord reduces state by performing NAT in the hypervisor. If NetLord style address virtualization is used in Rain Man, the L2 table state used by default routes at each switch will be reduced from the number of virtual hosts to the number of switches in the network.

Another point in the design space is to virtualize the VM addresses with that of the hypervisor, which reduces L2 table state to the number of physical hosts and does not require any network support outside of the hypervisor.

However, by placing all the forwarding state in the hosts it is possible to reduce switch forwarding state even more than with NetLord style address virtualization. SecondNet [4] demonstrated that by adding static MPLS tag to output port mappings to all the switches in the network, hosts can build source routes with MPLS tags. If this is the only way traffic is routed in the network, then the switch forwarding table state is constant and on the order of the number of ports per switch regardless of the network size.

Unfortunately, neither NetLord nor SecondNet are immediately suited for deployment. NetLord requires that traffic destined for external hosts be forwarded by tenant implemented software routers, which can become a bandwidth bottleneck. Similarly, SecondNet does not even support communicating with external hosts, and cannot support hardware IP routers, the unmodifiable devices typically used for communication with external hosts. However, because these designs are orthogonal to Rain Man, they can be used with Rain Man, which can lead to a more scalable network. Rain Man can provide support for default routes and exception routes for unmodifiable devices such as routers and middleboxes, while NetLord style NAT and SecondNet style MPLS source routing can be used for lower state default routes and exception routes for virtual hosts.

4.6 Conclusions

Rain Man demonstrates that it is possible to build a scalable SDN network for the datacenter that can be configured into arbitrary topologies. Rain Man can scale at near-optimal performance up to ten thousand hosts by introducing a new exception route algorithm that uses the L2 table. This effectively doubles the number of exception routes available to previous SDN designs, such as Hedera and OpenFlow. Beyond ten thousand hosts, the performance degradation is graceful - no more than 10% - out to as many as forty thousand hosts.

Hedera, a previous SDN architecture for the datacenter, can scale to larger networks than Rain Man. Hedera is only constrained by address space limitations for ECMP routing and is limited to the same number of exception routes as Rain Man. However, Rain Man is a better network architecture for networks with less than forty thousand hosts because Hedera requires the folded-Clos topology. Rain Man supports arbitrary network topologies, including the HyperX, which achieves 100% higher throughput than an equal bisection bandwidth version of a folded-Clos,

Although it is impossible to compare the Jellyfish topology against the HyperX and folded-Clos without a cost model, Mudigonda *et al.* [50] showed that equal bisection bandwidth instances of folded-Clos and HyperX topologies have similar cost. The significant difference between the performance of equal bisection bandwidth topologies of similar cost under a uniform workload implies that bisection bandwidth should not be the only performance metric used to differentiate topologies.

CHAPTER 5

Scalability Study of SDN Architectures

5.1 Introduction

There are many different SDN architectures that are suitable Ethernet substitutes, and it is unclear which SDN architecture, if any, is the most scalable. This chapter studies the scalability of the SDN design space. An early version of this work appeared in ANCS11 [51].

All SDN architectures have three layers: the network controller, the switch firmware, and the switch hardware. These three layers are connected by two interfaces. The SDN protocol is the interface between the network controller and the switch firmware, and there is a proprietary interface between the hardware and software. Rain Man, the novel SDN architecture presented in Chapter 4, is constrained by the requirement that it be implemented on COTS Ethernet switch hardware. This keeps the lowest layer of the SDN architecture fixed. By designing all three layers of the SDN architecture together, there are other points in the SDN design space that are more scalable than Rain Man.

This chapter explores the design space of SDN architectures and performs a study to determine what SDN architectures are the most scalable. It would be infeasible to implement every SDN network, so first I identify three different architectural dimensions common among the networks: source versus hop-by-hop routing, arbitrary versus restrictive routing, and the granularity at which flows are routed. My work then uses the simulator developed for Rain Man to compare the design space against the scalability limitations of Ethernet: broadcast and control bandwidth overhead, exhausting forwarding table state, and its ability to load-balance traffic across links in the network.

The architectural dimensions are chosen to be representative of common SDN networks. Source versus hop-by-hop routing is representative of comparing an SDN network implemented on the Axon hardware switch and traditional Ethernet switches. Arbitrary versus restrictive routing compares arbitrary routing versus restricting routes to trees and minimum hop count paths. Lossless Ethernet (DCB) and forwarding using the L2 table both restrict forwarding to different types of trees, and other network architectures for the datacenter, such as TRILL and SPB, restrict routes to minimum hop count paths. Flow granularity is representative of which hardware tables are used for forwarding. The controller can either use L2 forwarding tables or a combination of TCAMs.

The network simulations demonstrate that, in a realistic environment, source routing reduces the maximum state requirements of the network by 2-5x. They also show that restricting routing to a tree, either for deadlock-freedom for lossless networking or to use the L2 forwarding tables significantly affects network performance. In contrast, restricting routing to minimum length paths has little effect on performance. The last set of experiments demonstrates that, under the workloads I evaluated, routing at the TCP granularity is unnecessary for improving performance.

This chapter is organized as follows. Section 5.2 discusses the architectural dimensions that I define. Section 5.3 and 5.4 discuss the simulator and simulation results.

Architecture	Source vs. Hop-by-Hop	Arbitrary vs. Restrictive	Flow vs. Destination
BGP	Hop-by-Hop	Arbitrary	Destination (Aggregate)
OSPF	Hop-by-Hop	Restrictive	Destination (Aggregate)
MPLS	Hop-by-Hop	Arbitrary	Flow
Ethernet	Hop-by-Hop	Restrictive	Destination
MOOSE [16]	Hop-by-Hop	Restrictive	Destination
PortLand [1]	Hop-by-Hop	Arbitrary	Destination
Hedera [2]	Hop-by-Hop	Arbitrary	Flow
VL2 [37]	Hop-by-Hop	Restrictive	Destination
SEATTLE [20]	Hop-by-Hop	Restrictive	Destination
TRILL [22]	Hop-by-Hop	Restrictive	Destination
VIRO [21]	Hop-by-Hop	Restrictive	Destination
OpenFlow [11]	Hop-by-Hop	Arbitrary	Flow
SPAIN [15]	Hop-by-Hop	Restrictive	Flow
HBR [17]	Hop-by-Hop	Restrictive	Flow (Aggregate)
Axon [14]	Source	Arbitrary	Flow
DCell [18]	Hop-by-Hop	Restrictive	Destination
BCube [19]	Source	Arbitrary	Flow
SecondNet [4]	Source	Arbitrary	Flow
Infiniband [52]	Hop-by-Hop	Restrictive	Destination
Myrinet [53]	Source	Restrictive	Flow
Converged Enhanced Ethernet [24]	Hop-by-Hop	Restrictive	Destination

Table 5.1 : Architectural Dimensions

Finally, I conclude in Section 5.5.

5.2 Architectural Dimensions

From the many SDN architectures, I identify three distinct network architectural dimensions: source versus hop-by-hop routing, arbitrary versus restrictive routing, and the granularity at which flows are routed. A breakdown of the mechanisms of common network architectures can be found in Table 5.1. This table includes non-SDN architectures, which demonstrates that my taxonomy is applicable to more than SDN designs. However, my study is still restricted to studying these mechanisms in the context of SDN.

Next, I will discuss the three mechanisms that will be evaluated in this chapter in

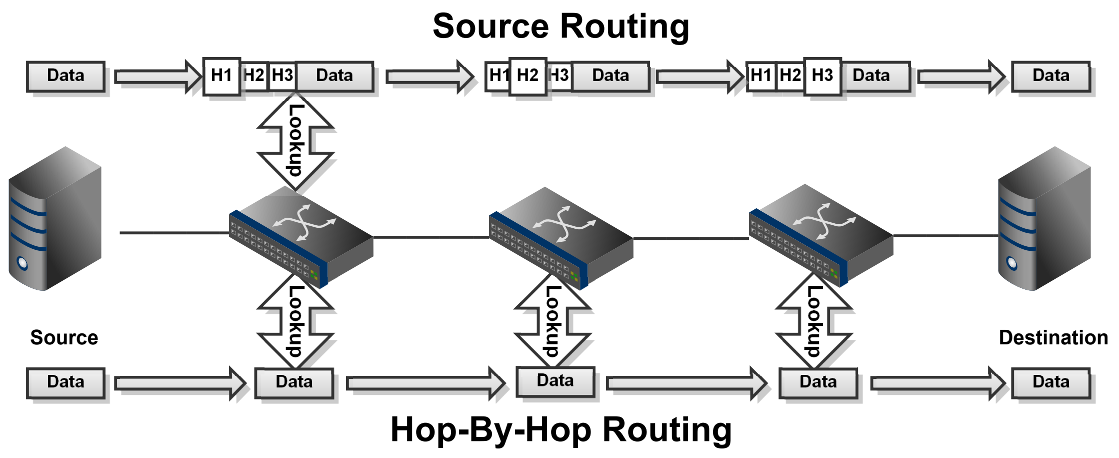

Figure 5.1 : Overview of Packet Forwarding on Source and Hop-By-Hop Networks

more depth.

5.2.1 Source vs Hop-By-Hop Routing

In this study, any architecture where all of the forwarding state for a flow is stored at the local switch is considered source routing. In practice, this is accomplished by the local switch attaching a header that contains the full route to the packet. Axons, BCube, and SecondNet all attach an ordered list of port numbers to the packet. Axons and BCube require custom forwarding hardware, while SecondNet repurposes MPLS support in commodity switches. PARIS [54] decorates packets with an ordered list of link IDs instead of port numbers. This study is equally applicable to this style of source routing as well.

Figure 5.1 gives an overview of packet forwarding for both source and hop-by-hop routed networks. In source routing, the packet contains all of the information necessary to route the packet after the first forwarding table lookup. In contrast, hop-by-hop routing must perform a forwarding table lookup at every hop along the path.

This study assumes that when a source or hop-by-hop route is installed at a switch, only one forwarding table entry is used. This is the case with hop-by-hop routing in OpenFlow. This is also the case with source routing on Axons, where the forwarding table lookup returns an address is SRAM where the full route is stored. Note that with source routing, routes are only installed at the switches local to the source and destination, whereas with hop-by-hop routing, routes are installed at every hop along the path.

Source routing can improve network scalability over hop-by-hop routing. All state needed for a host to communicate is stored at the *local* switch. Therefore, regardless of the size of the network, the state requirements of a single switch are proportional only to the demands of its locally connected hosts. In contrast, hop-by-hop routing requires a switch to store state for every packet that traverses the switch.

If traffic is routed at the flow granularity, it is not possible for hop-by-hop routing to have less forwarding table state than source routing; the local switches still need forwarding state as well. If traffic is routed only by destination or multiple flows are aggregated into an ensemble, then sharing can allow hop-by-hop routing to require less forwarding state than source routing.

However, the level of reduction in state is heavily dependent on workload and topology. For example, a destination routed network with an all-to-all communication pattern on a topology with no interior nodes requires exactly the same state on either a source-routed or a hop-by-hop network.

The traditional approach to improve the scalability of hop-by-hop routing is to aggregate multiple routes together in an ensemble. This is the approach taken by IP with CIDR, and recently, this approach has been adapted to existing Ethernet devices [17]. A negative consequence of aggregating lookups is that routing must be performed at the level of the ensemble, so traffic engineering becomes more difficult.

Ensemble routing is complementary to source routing, although it may not be necessary because of decreased state requirements, and it may not be as effective because ensembles are restricted to flows on the same source switch.

A disadvantage of source-routing is that it requires a larger packet header, which reduces the effective bandwidth available at the physical layer. On Axons [14], the bandwidth is reduced by 8 bits per hop. This is about 0.07% per hop for maximally sized Ethernet packets. SecondNet uses 20 bit MPLS headers per hop for source routing, which reduces bandwidth by about 0.175% for maximally sized packets.

The cost of installing routes in an SDN architecture is proportional to the forwarding table state. In a centralized architecture, the cost of installing routes includes the bandwidth used to send forwarding table state to the switches and the processing power used by the switch to install the state. SDN architectures can use either an out-of-band or in-band control network. In the case of an out-of-band control network, the control overhead is typically limited by the switch control plane, which has lower effective bandwidth than even 100 Mbps Ethernet. In the case of an in-band control network, the control bandwidth also wastes usable application bandwidth.

5.2.2 Arbitrary vs Restrictive Routing

Arbitrary routing can more flexibly perform traffic engineering, but some networks choose to restrict routing to either improve performance or reduce forwarding state. Networks such as TRILL, SPB, OSPF, and DCell restrict forwarding to minimum length paths to simplify path computation and forwarding. Routing using the L2 table also restricts routes to spanning trees. However, STP, which is required to be used with traditional switched Ethernet, is more restrictive than the spanning trees that can be used with SDN. With STP, a single spanning tree is used for forwarding all the traffic in the network or VLAN. With SDN, independent spanning trees can be used per host. Virtual id routing [21] restricts forwarding to a spanning tree in order to logarithmically reduce forwarding table state. Lossless Ethernet [24] requires that routes be deadlock-free in order to reduce TCP control overhead, guarantee low latency, and require minimal buffering.

A deadlock occurs when one or more packets get blocked forever because a set of resource requests (*e.g.*, packets traversing the network) form a cycle. The most popular deadlock-free routing algorithm is Up*/Down* [55]. Up*/Down* routing creates a spanning tree of the network and assigns directions to each link based on the spanning tree so that the directed links do not form loops. A valid route never uses a link in the up direction after using a link in the down direction, so there are never any routes that form a loop, and the routes are guaranteed to be deadlock-free.

Up*/Down* routing is less restrictive than the spanning tree imposed by STP, although is similar to the restrictions imposed by Virtual id routing. Up*/Down* routing allows for all working links to be used, although it does impose some routing restrictions. How the spanning tree is built can impact the number of routing restrictions imposed. The original description of Up*/Down* routing builds a BFS spanning tree [55]. Shancho, *et al.* [25] showed that using heuristics to build a DFS spanning tree can reduce the number of routing restrictions by 20%.

5.2.3 Flow Granularity

Flows can be defined at many different granularities, from destination only flows to IP pair to the 4-tuple of source and destination TCP port and IP address. Using a finer flow granularity allows for finer granularity traffic engineering and security policy at the expense of adding extra overhead. A finer granularity is guaranteed to increase routing state. Different flow granularities can also use different hardware forwarding tables, and typically, the finer the granularity, the smaller the table size. The control and bandwidth overhead of installing routes also increases proportionally to the flow granularity.

Although datacenter networks are typically contained in a single administrative domain, there are still security concerns. For example, there may be distinct user groups that should be isolated, such as tenants in a virtualized datacenter. Typically, this isolation is achieved with VLANs, but each VLAN requires its own buffer resources. Enforcing isolation by installing routes at the flow granularity allows a trade-off between device buffer size and forwarding table entries.

However, routing state per switch is finite and a limiting factor on the total size of a network. If too fine a flow granularity is used, the forwarding tables will become full and active flows will either have to be removed or requests for new flows will be rejected. Similarly, if too coarse a granularity is used, it is possible that performance will degrade due to flow collisions.

5.3 Methodology

The scalability study presented in this chapter uses the same methodology as described in Section 4.3. The same custom flow simulator is used to simulate the interactions between flows in the network. The same shuffle workload from Section 4.3 is used, and I introduce two new workloads. The first new workload is DCTCP, which synthetically generated from statistics published about a 6,000 server production cluster by Alizadeh *et al.* [3]. The second new workload is the UNIV workload from two sets of actual network traces collected from university datacenters from Benson, *et al.* [56]. The Univ workload is used because it has degrees of connections representative of real world datacenters. The Univ workload is not very demanding, so when I run simulations with the Univ workload, I scale down the network link rate to simulate a more congested network. The topology use in these simulations is the Jellyfish topology.

5.4 Network Simulations

In this section, the architectural mechanisms are evaluated on a realistic network. The experimental data presented demonstrates that source routing requires 2-5x less state than hop-by-hop routing. The simulations also show that restricting routing to minimum length paths has little effect on performance. In contrast, restricting routing to a tree, either to use a lossless network or the L2 forwarding tables, has a much more significant performance penalty. Lastly, the simulations demonstrate that, under the Univ workload, that routing at any granularity other than the destination granularity is unnecessary because the network throughput is unaffected and any finer granularity requires more forwarding table state in more expensive tables.

5.4.1 Source vs Hop-By-Hop Routing

The state requirements of hop-by-hop and source routing were computed from the flows installed during simulation. For a direct comparison of forwarding table requirements, I assume that all routes are installed in a TCAM in these simulations. Although flows are installed and removed during the experiment, for each switch, only the maximum state at any point during the trace is presented. This is because forwarding table state is an all or nothing resource. Overflowing the table is the only

Routing Type	Bisection Bandwidth					
	B=1:5		**B=1:2**		**B=1:1**	
	Univ1	**Univ2**	**Univ1**	**Univ2**	**Univ1**	**Univ2**
	Avg, Max	**Avg, Max**	**Avg, Max**	**Avg, Max**	**Avg, Max**	**Avg, Max**
Source DST	4.97, 14	44.67, 496	1.40, 10	8.63, 492	3.38, 14	23.90, 496
Source HPAIR	7.24, 21	67.17, 509	1.49, 13	9.76, 492	4.08, 19	31.42, 505
Source TCP	22.7059, 118	68.46, 511	3.75, 112	10.11, 494	11.64, 118	32.29, 507
Hop-By-Hop DST	7.47, 15	69.92, 504	3.31, 13	18.03, 493	5.12, 15	37.30, 497
Hop-By-Hop HPAIR	17.88, 140	180.79, 1071	5.05, 138	43.42, 1034	9.24, 140	92.02, 1048
Hop-By-Hop TCP	58.15, 313	184.23, 1078	13.84, 313	44.34, 1038	27.96, 313	94.82, 1053

Table 5.2 : Number of Forwarding Table Entries per Switch for Source and Hop-By-Hop Routing on the Univ Workload - 1 Mbps

Routing Type	Bisection Bandwidth					
	B=1:5		**B=1:2**		**B=1:1**	
	Univ1	**Univ2**	**Univ1**	**Univ2**	**Univ1**	**Univ2**
	Avg, Max	**Avg, Max**	**Avg, Max**	**Avg, Max**	**Avg, Max**	**Avg, Max**
Source DST	8.15, 29	65.83, 594	1.81, 22	12.05, 587	4.87, 29	34.89, 593
Source HPAIR	11.74, 47	111.13, 625	1.99, 36	14.26, 587	6.14, 45	50.42, 604
Source TCP	105.21, 789	113.08, 627	14.24, 788	14.88, 589	49.66, 788	51.71, 606
Hop-By-Hop DST	12.29, 34	106.38, 605	4.35, 25	26.61, 590	7.81, 30	56.92, 594
Hop-By-Hop HPAIR	30.15, 208	294.13, 1295	7.54, 199	65.70, 1199	14.65, 205	153.88, 1232
Hop-By-Hop TCP	308.59, 1414	300.81, 1299	71.34, 1388	67.36, 1203	163.30, 1404	156.78, 1232

Table 5.3 : Number of Forwarding Table Entries per Switch for Source and Hop-By-Hop Routing on the Univ Workload - 100 Kbps

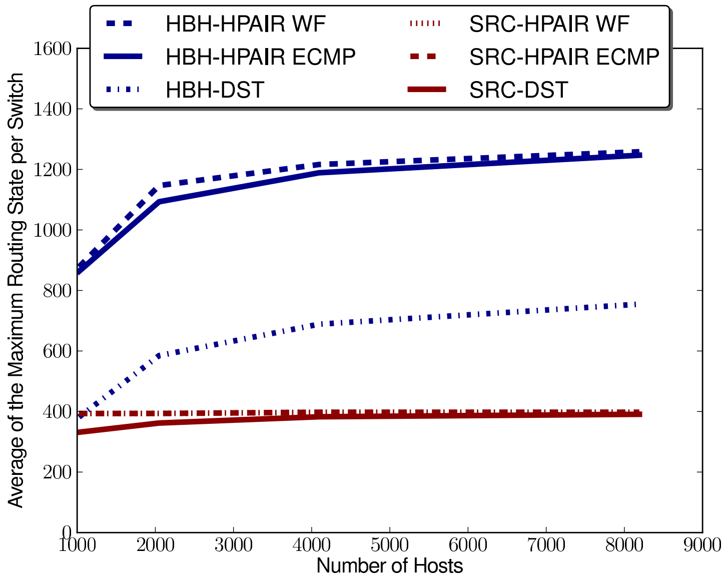

Figure 5.2 : B=1:5 Shuffle Average of the Maximum Routing State

way for forwarding table state to affect scalability.

The results of the computation for different routing schemes on different bisection bandwidth Jellyfish topologies under the shuffle workload are presented in Figures 5.2, 5.3, 5.4, and 5.5. These results are from B=1:5 bisection bandwidth ratio instances of the Jellyfish topology. The results for the other bisection bandwidth ratio topologies are omitted because of their similarity.

HBH-HPAIR is for hop-by-hop routing at the granularity of the host pair. This is similar to how OpenFlow [11] is commonly used. *WF* is for the greedy worst-fit algorithm, and *ECMP* is where each flow is randomly assigned a minimum hop count path. *HBH-DST* is for destination based hop-by-hop routing. Because multiple routes are installed for each destination, this is most similar to reactively using ECMP in the network. Similarly, *SRC-HPAIR* is for source routing at the IP pair granularity, and *SRC-DST* is for destination based source routing. Destination based source routing

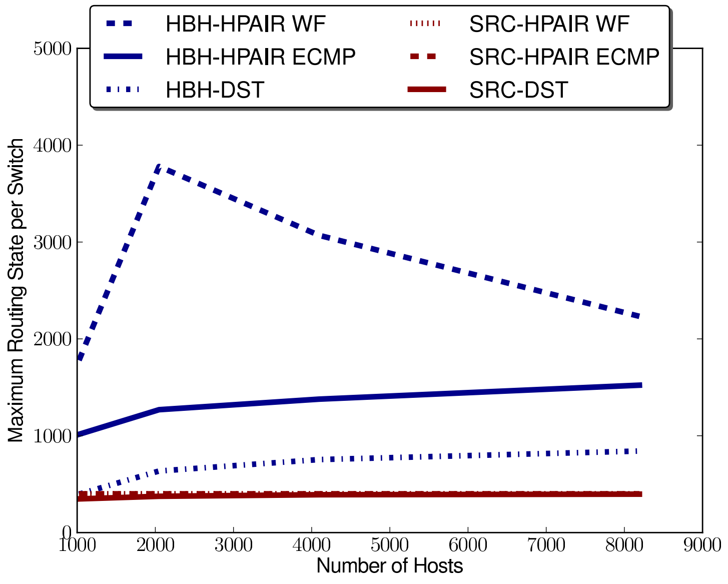

Figure 5.3 : B=1:5 Shuffle Maximum Routing State

is where all hosts on the same switch use the same route for a destination. The source routing data is similar to two different configurations of an Axon network [14].

The results from the shuffle experiment agree with the expected results. Under the shuffle workload, each host only ever has 10 open connections at a given time. For source routing, the state requirement of both the HPAIR routing algorithms is $10 * T$, where T is the number of terminals per switch. When the network size is small, it is probable that multiple hosts on the same switch will be talking to the same destination, which explains the state reduction of *SRC-DST*. As the network grows, this probability reduces, as expected.

The hop-by-hop results for the shuffle experiment also agree with the expected results. The state requirements scale with $10 * T * l_G$, where l_G is the average path length of the topology G. There is a greater chance for sharing routes in *HBH-DST* than *SRC-DST*, which is shown in the results. However, on topologies with $l_G > 2$,

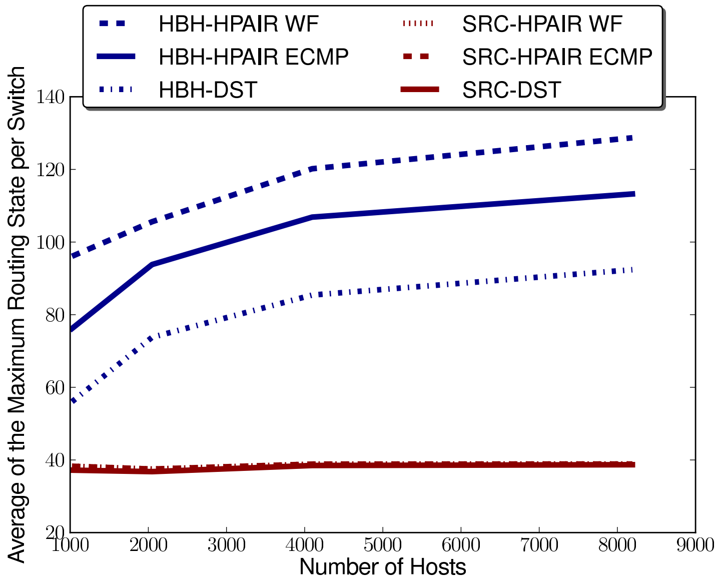

Figure 5.4 : B=1:5 DCTCP Average of the Maximum Routing State

the state requirements of *HBH-DST* are still double that of *SRC-DST*.

The maximum state requirements of hop-by-hop routing at two thousand hosts requires significantly more state than larger networks. This is because the greedy worst fit algorithm is not optimal, and in the evaluated topology, every switch connects to roughly half of the switches in the network. In the worst case switch, under the high degree of connectively presented by this topology, the worst fit algorithm does not perform well.

Tables 5.2 and 5.3 present the state requirements of different source and hop-by-hop routing schemes on 1 Mbps and 100 Kbps link rate instances of the Jellyfish topology, respectively. The results from the Univ workloads agree with the results from the shuffle workload.

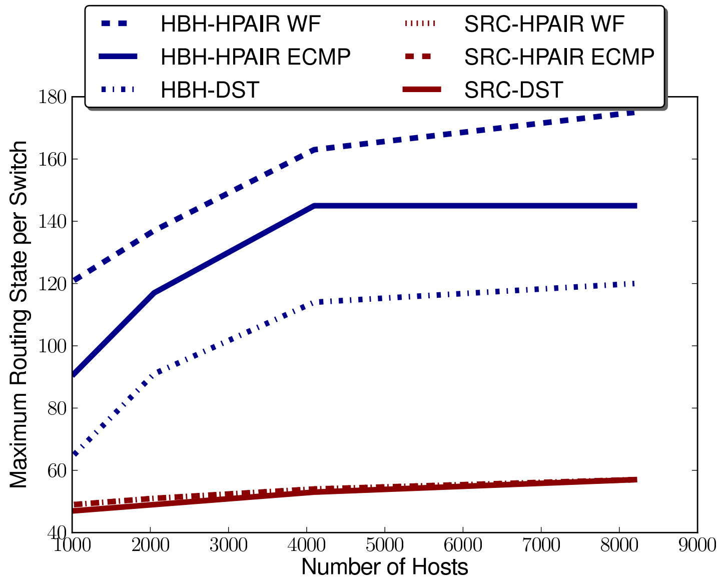

Figure 5.5 : B=1:5 DCTCP Maximum Routing State

5.4.2 Arbitrary vs Restrictive Routing

Figures 5.6 and 5.7 show the aggregate throughput of different routing schemes under the shuffle workload on two different B=1:5 bisection bandwidth ratio Jellyfish topologies. Figure 5.6 is built with 64-port switches, and Figure 5.7 is built with 16-port switches. The 16-port switch topology has significantly more switches than the 64-port switches, so it is more representative of a larger network because of its increased network diameter. In the figures, *ECMP* is used for ECMP routing. *Worst Shortest Fit* is a greedy worst fit algorithm that only chooses between shortest length paths. *Worst Fit* is the greedy worst fit algorithm. *UpDown DFS* and *UpDown BFS* stand for greedy worst fit routing on DFS and BFS Up*/Down* trees, respectively. Results for ECMP routing on DFS and BFS Up*/Down* trees are omitted for their similarity to the greedy worst fit versions. Lastly, *AHST* stands for the all hosts

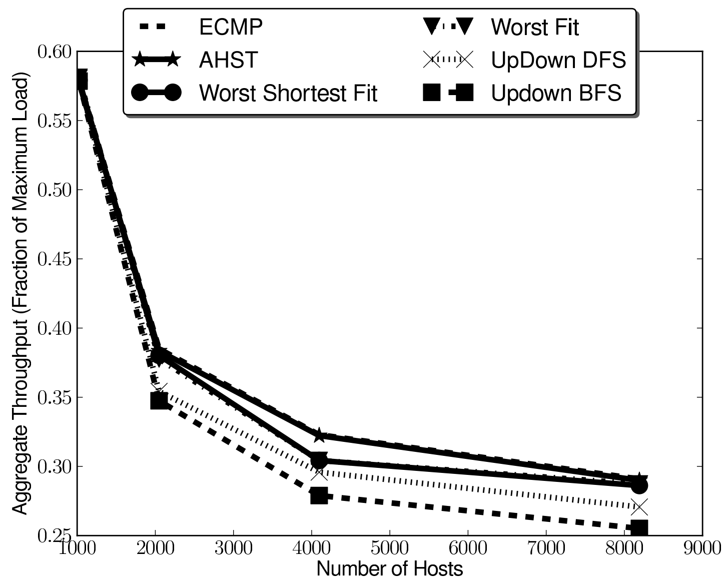

Figure 5.6 : Shuffle Aggregate Throughput - 64-Port Switches, B=1:5

spanning trees (AHST) algorithm. This is where a unique, random minimum spanning tree is built for each host. The spanning tree is built with a BFS algorithm. Routing with this algorithm is implementable with SDN, and all of the forwarding table state for this algorithm can fit into the L2 table. Routing with this algorithm on traditional switched Ethernet is not possible without modifying either the hosts or switches, similar to SPAIN [15].

The simulations show that the best performance comes from the worst fit algorithm and both of the shortest path algorithms, whose performance is very similar. The ECMP and AHST algorithms also have very similar performance. On low diameter networks, the ECMP and AHST algorithms perform the best, and both the worst fit and worst shortest fit algorithms perform near identically. As the diameter of the network increases, the worst fit algorithms perform better than ECMP, and the unrestricted worst fit algorithm performs slightly better than the shortest path

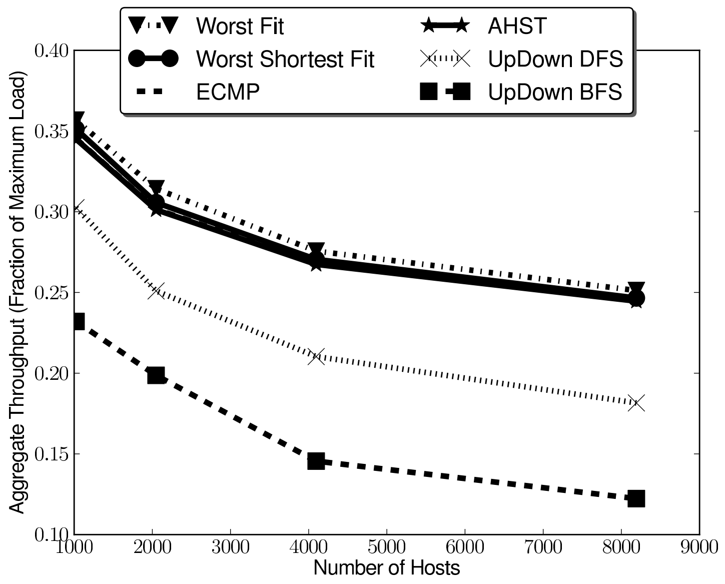

Figure 5.7 : Shuffle Aggregate Throughput - 16-Port Switches, B=1:5

worst fit algorithm.

At low network diameters, the both of the Up*/Down* algorithms perform well. At two thousand hosts on the topology with 64-ports, they both perform within 10% of the highest achieved throughput. This is because the algorithms allow all the links in the network to be used and impose few routing restrictions. However, as the network size grows, the performance drops off as more routes are restricted. At the highest diameter network, the DFS and BFS algorithms perform 28% and 52% worse than the best performing routing algorithm, respectively.

Figures 5.8 and 5.9 present the average of the maximum and the maximum state requirements, respectively, for the different routing algorithms on the 16-port Jellyfish topologies. In all of the routing schemes, I assume that state for the routes is installed reactively with zero latency, and that only a single forwarding table is used.

The state results for the different routing algorithms agree with the expected re-

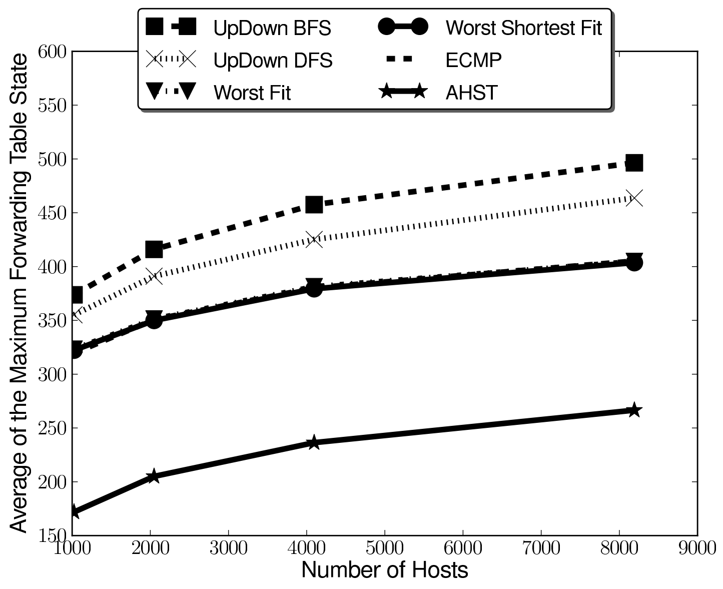

Figure 5.8 : Average of the Maximum Routing State - 16-Port Switches, B=1:5

sults. Both of the Up*/Down* algorithms restrict routes, which causes some switches to use more state. The DFS algorithm imposes less restrictions than the BFS algorithm and, as expected, requires less forwarding table state. At the worst case switch, which is the root of the tree, the state requirements of BFS routing are an order of magnitude larger than any other routing algorithm. The state requirements of the greedy worst fit, greedy worst shortest fit, and ECMP are all nearly identical. Because the AHST algorithm only ever uses a single L2 entry per switch, it uses roughly half the state of the other algorithms. It is worth noting that all of the other algorithms require TCAM state, whereas the AHST algorithm only requires L2 state.

5.4.3 Flow Granularity

In order to evaluate the impact of flow granularity on performance and forwarding table state, I simulated the Univ workload with different flow granularities. I chose

Routing Type	Bisection Bandwidth					
	B=1:5		B=1:2		B=1:1	
	Univ1	Univ2	Univ1	Univ2	Univ1	Univ2
	Avg, Max	Avg, Max	Avg, Max	Avg, Max	Avg, Max	Avg, Max
Source HPAIR	7.47, 21	84.23, 587	5.68, 20	58.07, 586	4.43, 19	40.61, 579
Source Limit 100 Kbps	62.21, 429	155.71, 836	48.52, 566	114.86, 841	33.57, 492	78.61, 854
Source Limit 1 Mbps	14.34, 64	91.71, 615	10.80, 75	63.92, 618	8.22, 64	44.59, 610
Source Limit 10 Mbps	7.62, 22	84.71, 587	5.82, 22	58.38, 586	4.54, 22	40.93, 579
Source TCP	30.53, 129	140.86, 2726	24.46, 129	96.62, 2726	18.11, 129	66.69, 2726
Hop-By-Hop DST	7.59, 15	85.90, 579	6.48, 15	59.94, 578	5.36, 15	46.36, 578
Hop-By-Hop HPAIR	15.06, 140	198.44, 1117	11.30, 140	136.59, 1128	8.86, 140	97.19, 1093
Hop-By-Hop Limit 100 Kbps	188.85, 1133	522, 1530	143.26, 1695	436.197, 1599	98.55, 1374	231.76, 1459
Hop-By-Hop Limit 1 Mbps	36.56, 163	233.71, 1121	27.02, 182	156.30, 1137	20.04, 169	109.96, 1099
Hop-By-Hop Limit 10 Mbps	15.97, 141	200, 1117	12.06, 140	137.817, 1130	9.24, 141	97.97, 1095
Hop-By-Hop TCP	62.97, 315	338.52, 2726	50.42, 318	210.59, 2726	36.82, 313	162.65, 2726

Table 5.4 : Number of Forwarding Table Entries per Switch for Differing Flow Granularities on the Univ Workload - 1 Mbps

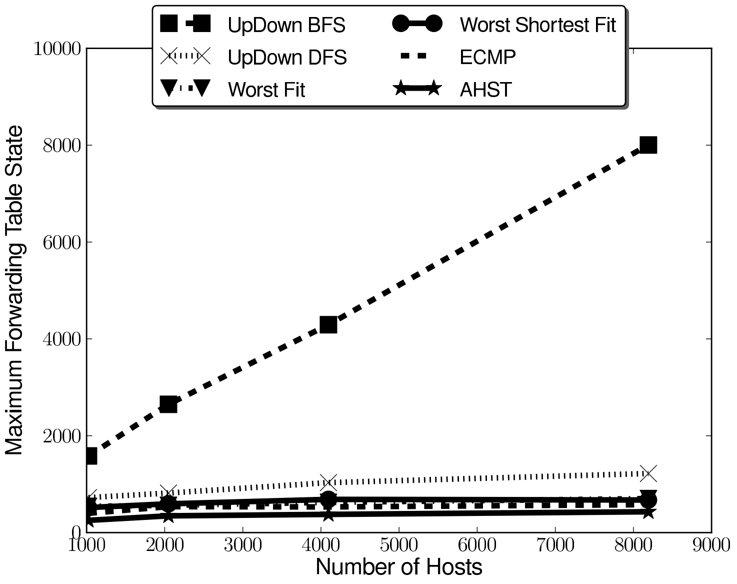

Figure 5.9 : Maximum Routing State - 16-Port Switches, B=1:5

the Univ workload because it is the only workload that I have that has a realistic number of TCP connections per host pair connections. The state requirements from the experiments are presented in Table 5.4. The routing types labeled with *100 Kbps*, *1 Mbps*, and *10 Mbps* refer to the bit count limit after which a flow is classified as an elephant flow and routed independently with a greedy worst fit algorithm. The throughput results are omitted because they are identical for all of the evaluated flow granularities.

The state results gathered for this section also agree with the results presented in Subsection 5.4.1. As expected, the finer the flow granularity, the larger the state requirements, with hop-by-hop routing at the TCP granularity requiring up to 6x the state of hop-by-hop routing at the destination granularity. An unexpected result is that using too low an elephant flow limit has drastic increase in the required state. This is because the greedy worst fit algorithm is not optimal and can lead to longer paths being used.

While these results show that routing at any granularity other than the destination granularity is unnecessary for improving performance, it is important to consider that these results are only applicable for the Univ workload, which is not as performance demanding as some other workloads. Recall that Subsection 5.4.2 demonstrated that routing at the destination granularity with the AHST algorithm has between a 10-58% reduction in performance when compared with the best performing routing algorithm. These results are from the shuffle workload, which is more demanding.

5.5 Conclusions

There are many recent proposals for SDN architectures for the datacenter. Unfortunately, there has been no concrete evaluation of the design space, and it is unclear from the proposals which techniques are best, and what architecture, if any, is best. The primary contributions of this chapter are the identification of three distinct mechanisms for addressing scalability and the real-world workload driven evaluation of these mechanisms in the context of scalability. The results of the evaluation show that source-routed networks are more scalable than hop-by-hop routed networks. The results also show that restricting routes to minimum length paths has a minimal effect on performance, where as restricting routing to a tree can have a significant impact on performance. Lastly, the results also show, that under some workloads, routing at the destination granularity is the most scalable flow granularity because it requires the least state and it does not affect performance.

Hop-by-hop routing requires on the order of the average path length more forwarding state than source routing at the same flow granularity. Because hop-by-hop routing requires more state, more switches need to be updated in order to respond to failure, while the difference is response time to failure between hop-by-hop and source routing is bounded to within a single RTT. This implies that future scalable SDN architectures should focus on using source routing to reduce forwarding table pressure.

On networks that can support routing on a regular network, forwarding table state is proportional to the number of ports on the switch. This is better than both source and hop-by-hop routing. However, exception flows are still necessary on a such networks, either for QOS, performance, or routing through middleware applications. Exception flows, by definition, cannot use the same forwarding table entries as the default routes. The results in this study imply that it is more scalable to use source routes instead of hop-by-hop routes, regardless of whether the routes are default or exception routes.

Restricting routing to minimum hop-count paths does not have a significant affect on throughput. Similarly, restricting routing to trees to use the L2 forwarding tables, such as with the AHST algorithm, also does not have a significant performance penalty. This implies that implementing default forwarding in a network with the L2 forwarding tables is scalable at reasonable performance. With this style routing, the scale of the network is only limited by the size of the L2 forwarding table, which can have over a hundred thousand entries. However, restricting routing to tree for deadlock-free routing does have a noticeable performance effect. The poor performance of the BFS Up*/Down* routing scheme implies that it should never be used. If DCB is necessary in the network for Fibre Channel support and the network topology does not have regularity that can be exploited to build deadlock-free routes, such as the folded-Clos and HyperX topologies, the DFS Up*/Down* algorithm should be used. However, deadlock freedom of any kind can still affect performance, so any flows that do not require lossless delivery should not use lossless delivery so that they are not restricted to deadlock-free routes.

Other work has shown that traffic engineering individual flows can achieve near-optimal throughput and outperforms static traffic engineering, such as ECMP [2]. This paper demonstrates that, on a university datacenter workload, routing all individual flows at the TCP granularity does not allow for better traffic engineering than routing at the host pair or destination granularity.

CHAPTER 6

Conclusions

Building a scalable network for datacenters is a challenging and important problem. This thesis contributes to solving this problem in three ways. First I build a framework of the scalability limitations of datacenter networks. Secondly, I design the firmware for Rain Man, a scalable SDN network for the datacenter that is implementable on commodity Ethernet switch hardware. Lastly, I perform a general scalability study of the SDN architecture design space.

Switched Ethernet is currently used to implement datacenter networks, but its inability to scale requires the use of IP routers, which are undesirable in the datacenter. There are three principle causes of Ethernet's inability to scale: control bandwidth overhead, exhausting forwarding table state, and a lack of usable application bandwidth, which can be caused by the topology or by under-utilizing the links in the network. I argue that any substitute for Ethernet in the datacenter will also suffer from the same scalability limitations.

Rain Man demonstrates that it is possible to build a scalable SDN network for the datacenter that can be configured into arbitrary topologies. Rain Man can scale at near-optimal performance up to ten thousand hosts by introducing a new exception route algorithm that uses the L2 table. This effectively doubles the number of exception routes available to previous SDN designs, such as Hedera and OpenFlow.

Beyond ten thousand hosts, the performance degradation of Rain Man is graceful - no more than 10% - out to as many as forty thousand hosts.

The Rain Man experiments also show that there is a need for another metric other than bisection bandwidth for comparing topologies. The results agree with previous work that has shown that two different equal bisection bandwidth topologies can have drastically different throughput. Under uniform workloads, the HyperX topology achieves 100% higher throughput than an equal bisection bandwidth folded-Clos topology.

The primary contributions of the SDN scalability study are the identification of three distinct mechanisms for addressing scalability and the real-world workload driven evaluation of these mechanisms in the context of scalability. The results of the evaluation show that source-routed networks are more scalable than hop-by-hop routed networks. The results also show that restricting routes to minimum length paths has a minimal effect on performance, where as restricting routing to a tree, either for deadlock-freedom or using the L2 table, can have a significant impact on performance. Lastly, the results also show, that under some workloads, routing at the destination granularity is the most scalable flow granularity because requires the least state and it does not affect performance.

Finally, the SDN scalability study shows that, as the network diameter increases, the performance penalty for routing using only the L2 table decreases. This implies that a large scale datacenter network with only a modest performance penalty can be implemented using only the L2 table. Considering that L2 tables can have hundreds of thousands of entries, this has important implications about building a large scale datacenter network.

6.1 Future Work

There are still plenty of questions left to answer regarding SDN network scalability. Currently, Rain Man has only been evaluated through simulations. Evaluations on a physical testbed are necessary to answer questions about Rain Man that cannot be answered with the current simulator. How does Rain Man affect the performance of real world applications? Are sampled network statistics alone sufficient for high performance, or are more exact per-port statistics necessary? If per-port statistics are necessary, how often do they need to be collected? What is the full system latency for Rain Man to detect a flow to be rerouted and calculate and install the new route, and how is performance affected as this latency increases?

As other future work, there are still significant questions to be answered by the scalability study about the performance of routing using the L2 forwarding table, multipath capabilities, and exception routes. This work will answer important questions about the performance of Rain Man beyond fourty thousand hosts, after the multipath hardware is completely utilized. This work also logically leads into designing a routing strategy for Rain Man that can still achieve high performance at a larger scale by strategically using the multipath capabilities of the switches and otherwise using the single path to each destination provided by the large L2 table.

Lastly, there are still significant questions to be answered about reducing the number of IP routers in a network. Can a network for a realistic autonomous system (AS), a network controlled by a single entity, be built entirely with L2 devices, without the need for IP routers? Although improving the scalability of the L2 network can reduce the number of IP routers in the network, current AS networks always require IP routers that speak the border gateway protocol (BGP) at the borders between ASs.

With SDN, this no longer needs to be necessary; the SDN controller can be used to provide full and transparent compatibility with existing BGP routers. This work will answer the following questions: What is the performance requirement placed on the SDN controller by replacing all of the border routers in the network? What are the forwarding state requirements of a border router, and does current COTS Ethernet switch hardware have enough forwarding state to replace a border router?

Bibliography

[1] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *SIGCOMM*, 2009.

[2] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *NSDI*, 2010.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM*, 2010.

[4] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: a data center network virtualization architecture with bandwidth guarantees," in *Co-NEXT*, 2010.

[5] J. Mudigonda, P. Yalagandula, J. C. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: a scalable multi-tenant network architecture for virtualized datacenters," in *SIGCOMM*, pp. 62–73, 2011.

[6] Z. Kerravala, "Configuration management delivers business resiliency," 2002.

[7] J. Kim and W. J. Dally, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *ISCA*, 2007.

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

[9] A. B. Bondi, “Characteristics of scalability and their impact on performance,” in *WOSP*, 2000.

[10] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane: Taking control of the enterprise,” in *SIGCOMM*, 2007.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[12] A. R. Curtis, J. C. Mogul, J. Tourrilhes, and P. Yalagandula, “DevoFlow: Scaling flow management for high-performance networks,” in *SIGCOMM*, 2011.

[13] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *INFOCOM*, pp. 1629–1637, IEEE, 2011.

[14] J. Shafer, B. Stephens, M. Foss, S. Rixner, and A. L. Cox, “Axon: A flexible substrate for source-routed Ethernet,” in *ANCS*, 2010.

[15] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, “SPAIN: COTS data-center Ethernet for multipathing over arbitrary topologies,” in *NSDI*, 2010.

[16] M. Scott, A. Moore, and J. Crowcroft, “Addressing the scalability of Ethernet with MOOSE,” in *Proceedings of DC CAVES Workshop*, 2009.

[17] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, “Ensemble routing for datacenter networks,” in *ANCS*, 2010.

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “DCell: A scalable and fault-tolerant network structure for data centers,” in *SIGCOMM*, 2008.

[19] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *SIGCOMM*, 2009.

[20] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: A scalable Ethernet architecture for large enterprises," in *Proceedings of ACM SIGCOMM*, 2008.

[21] G.-H. Lu, S. Jain, S. Chen, and Z.-L. Zhang, "Virtual id routing: A scalable routing framework with support for mobility and routing efficiency," in *MobiArch*, 2008.

[22] J. Touch and R. Perlman, "Transparent Interconnection of Lots of Links (TRILL): Problem and applicability statement." RFC 5556 (Informational), May 2009.

[23] D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, D. Fedyk, M. Ouellete, M. Seaman, and P. Unbehagen, "Shortest path bridging: efficient control of larger Ethernet networks," *Comm. Mag.*, vol. 48, pp. 128–135, October 2010.

[24] "IEEE 802.1 Data Center Bridging Task Group." http://www.ieee802.org/1/pages/dcbridges.html.

[25] J. Sancho and A. Robles, "Improving the Up*/Down* routing scheme for networks of workstations," in *Euro-Par 2000 Parallel Processing*, vol. 1900 of *Lecture Notes in Computer Science*, pp. 882–889, Springer Berlin / Heidelberg, 2000.

[26] "IBM BNT RackSwitch G8264." `http://www.redbooks.ibm.com/abstracts/tips0815.html`.

[27] "Broadcom BCM56846 StrataXGS 10/40 GbE Switch." `http://www.broadcom.com/products/features/BCM56846.php`.

[28] "Cisco Nexus 5010 and 5020 Switches." `http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/data_sheet_c78-461802.pdf`.

[29] "Cisco Nexus 5548 and 5596 Switches." `http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/data_sheet_c78-618603.pdf`.

[30] "HP 5900 Switch Series." `http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA3-8996ENW.pdf`.

[31] "sFlow." `http://sflow.org/about/index.php`.

[32] "OFlops." `http://www.openflow.org/wk/index.php/Oflops`.

[33] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *NSDI*, 2010.

[34] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *HotNets*, 2009.

[35] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable OpenFlow control," 2011.

[36] A. Kind, P. Hurley, and J. Massar, "A light-weight and scalable network profiling system," *ERCIM News*, pp. 67–68, January 2005.

[37] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *SIGCOMM*, 2009.

[38] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," Internet-Draft draft-mahalingam-dutt-dcops-vxlan-00.txt, IETF Secretariat, Jan. 2012.

[39] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *SIGCOMM*, 2002.

[40] S. Kandula, S. Sengupta, A. Greenberg, and P. Patel, "The nature of datacenter traffic: Measurements and analysis," in *IMC*, 2009.

[41] A. R. Curtis, T. Carpenter, M. Elsheikh, A. López-Ortiz, and S. Keshav, "REWIRE: An optimization-based framework for data center network design," in *INFOCOM*, 2012.

[42] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008.

[43] N. Farrington, E. Rubow, and A. Vahdat, "Data center switch architecture in the age of merchant silicon," in *Hot Interconnects*, 2009.

[44] S. Ohring, M. Ibel, S. Das, and M. Kumar, "On generalized fat trees," *Parallel Processing Symposium, International*, vol. 0, p. 37, 1995.

[45] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," *SC Conference*, 2009.

[46] D. Abts and J. Kim, *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities*. Morgan and Claypool, 2011.

[47] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, June 2011.

[48] D. Sampath, S. Agarwal, and J. Garcia-Luna-Aceves, "Ethernet on AIR: Scalable routing in very large Ethernet-based networks," in *ICDCS*, june 2010.

[49] "Beacon: a Java-based OpenFlow Control Platform." `http://www.beaconcontroller.net`.

[50] J. Mudigonda, P. Yalagandula, and J. C. Mogul, "Taming the flying cable monster: a topology design and optimization framework for data-center networks," in *USENIXATC*, 2011.

[51] B. Stephens, A. L. Cox, S. Rixner, and T. E. Ng, "A scalability study of enterprise network architectures," in *ANCS*, 2011.

[52] T. Hamada and N. Nakasato, "InfiniBand Trade Association, InfiniBand Architecture Specification, Volume 1, Release 1.0, http://www.infinibandta.com," in *International Conference on Field Programmable Logic and Applications, 2005*, pp. 366–373.

[53] Myricom, "Myri-10G NICs and software," Aug. 2008. Product brief.

[54] B. Awerbuch, I. Cidon, I. Gopal, M. Kaplan, and S. Kutten, "Distributed control for PARIS," in *PODC*, 1990.

[55] T. L. Rodeheffer and M. D. Schroeder, "Automatic reconfiguration in Autonet," in *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, 1991.

[56] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in *IMC*, 2010.