Stop Rerouting! Enabling ShareBackup for Failure Recovery in Data Center Networks

Yiting Xia Xin Sunny Huang T. S. Eugene Ng
Rice University

Abstract

This paper introduces sharable backup as a novel solution to failure recovery in data center networks. It allows the entire network to share a small pool of backup devices. This proposal is grounded in three key observations. First, the traditional rerouting-based failure recovery is ineffective, because bandwidth loss from failures degrades application performance drastically. Therefore, failed devices should be replaced to restore bandwidth. Second, failures in data centers are rare but destructive [11], so it is desirable to seek cost-effective backup options. Third, the emergence of configurable data center network architectures promises feasibility of bringing backup devices online dynamically. We design the ShareBackup prototype architecture to realize this idea. Compared to rerouting-based solutions, ShareBackup provides more bandwidth with short path length at low cost.

1 INTRODUCTION

Data center networks should be reliable to guarantee service performance. The mainstream solution to fault tolerance is rerouting: many data center network architectures provide redundant paths to increase bandwidth, and alternative paths can be used to reroute traffic around failures [4, 5, 12–14, 19, 21, 25, 26]. While rerouting maintains connectivity, bandwidth is nonetheless degraded under failures, which may jeopardize application performance drastically.

This concern has been validated by a measurement study: in a path-rich production data center, 10% less traffic is delivered for the median case of the analyzed failures, and 40% less for the worst 20% of failures [11]. Because data center traffic is expressed as sets of flows—known as coflows—to capture the application-level requirements [8], a small number of straggler flows influenced by failures can massively increase the Coflow Completion Time (CCT). The effect of failure is thus exacerbated on the coflow level. In our failure study (Section 2.2), a single failure can slow down CCT by several hundred times even with proper rerouting.

Another approach that can restore bandwidth immediately after failures happen is backup. Switches can keep a hot spare; hosts are multi-homed to the primary and the backup switches; and every link between two primary switches is duplicated by a mesh amongst them and their shadows. In this way, a switch can failover to its spare without bandwidth loss in the network. However, this 1:1 backup consumes a large number of backup switches and doubles the port requirements on hosts and switches. The prohibitive cost of extra hardware prevents the deployment of network-wide backup, so most data centers only backup a few crucial devices.

Two recent trends provide new opportunities for network-wide backup. First, commercial devices in data centers are increasingly more reliable. Despite the disastrous effect, the same measurement study shows failures are rare and transient: most devices have over 99.99% availability; and failures usually last for only a few minutes [11]. Therefore, 1:1 backup is unnecessary for occasional failure events. Second, configurable interconnects can facilitate physical-layer adaptation of the network topology, and novel architectures have been proposed to create paths on the fly according to traffic requirements [6, 7, 9, 10, 15, 16, 20, 22, 27, 29–31]. This configurability can be repurposed for efficient failure recovery.

In this paper, we introduce *sharable backup*, where a small pool of backup devices can repair failures on demand. This solution is both desirable and achievable from the above evidence. By connecting a group of switches and a few backup switches to highly reliable configurable interconnects, e.g. circuit switches, a backup switch can be brought online to replace any failed switch via simple circuit reconfiguration.

We realize this idea in ShareBackup, a prototype failure recoverable fat-tree network. We focus on fat-tree because fat-tree and its variants are widely adopted by many industrial data centers [3, 24]. The ShareBackup design faces many challenges. How to design the architecture to share a pool of backup switches? The per-port cost of small circuit switches are considerably lower than large ones [9, 23, 28]. How to leverage this cost benefit? For interface/link failures, instead of replacing switches on both sides of the link, how to diagnose the problem and only replace the faulty switch? Backup switches can replace regular switches on the physical layer, but how to impersonate a replaced switch on the control plane? Moreover, how to avoid extra delay from the impersonation process and let ShareBackup recover failures as fast as the highly responsive local rerouting? We address these

HotNets-XVI, November 30–December 1, 2017, Palo Alto, CA, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5569-8/17/11…$15.00
https://doi.org/10.1145/3152434.3152452

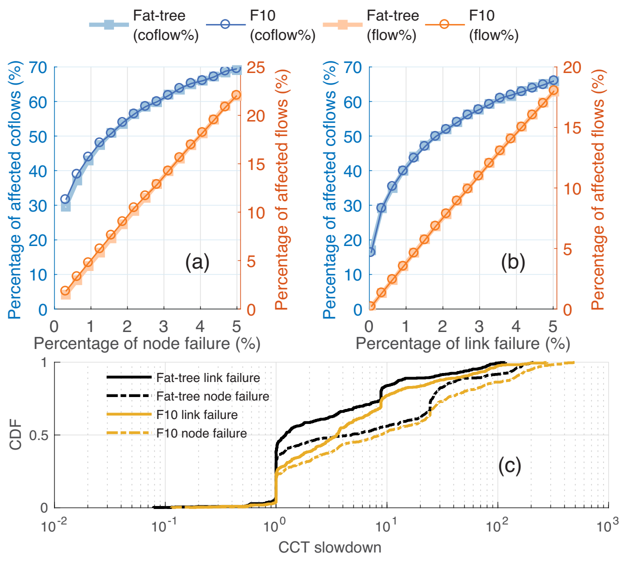

Figure 1: (a) and (b): impact of failures on flows and coflows; (c): coflow performance degradation with rerouting under a single failure

challenges and analyze the properties of our proposed architecture. Compared to rerouting-based solutions, ShareBackup provides more bandwidth with short path length at low cost.

2 MOTIVATION

2.1 Related Work

Many architectural solutions have been proposed to improve failure resiliency of data center networks. Fat-tree [5], DCell [14], BCube [13], VL2 [12], HyperX [4], and Jellyfish [25] build high-performance data center network architectures with redundant paths and provide customized rerouting schemes to bypass failures. PortLand enhances fault tolerance of fat-tree with a layer-2 routing and forwarding protocol [21]. F10 adjusts wiring of fat-tree to diversify alternative paths in the network structure [19]. It also improves responsiveness to failures by fast failure detection and local rerouting to longer paths. Aspen Tree adds redundancy to fat-tree to reduce failure convergence time, at the price of partitioning the network or introducing extra hardware [26].

The weaknesses of these rerouting-based solutions motivate the design of ShareBackup. First, rather than bypassing failures at compromised performance, we aim to replace failed devices completely to restore bandwidth. Second, redundancy may cause excessive hardware expenses [26], while we enable sharable backup via circuit switches to save cost. Third, alternative paths can have more hops [4, 14, 19, 25] and path re-computation may be expensive [25], so ShareBackup maintains original paths after failures to avoid rerouting overhead and path dilation. Fourth, some solutions experience slow failure propagation [5] or frequent state exchange [21], so ShareBackup employs a responsive and light-weighted control plane to achieve fast failure recovery.

Table 1: List of notations

Notation	Meaning
k	Fat-tree parameter: switch port count and # Pods [5]
n	# backup switches shared by $\frac{k}{2}$ switches per failure group
H_j	The jth host
$E_{i,j}$	The jth Edge switch in the ith Pod
$A_{i,j}$	The jth Aggregation switch in the ith Pod
C_j	The jth Core switch
$CS_{l,i,j}$	The jth Circuit Switch in the ith Pod on the lth layer
$FG_{l,u}$	The uth Failure Group on the lth layer
$BS_{l,u,v}$	The vth Backup Switch in $FG_{l,u}$

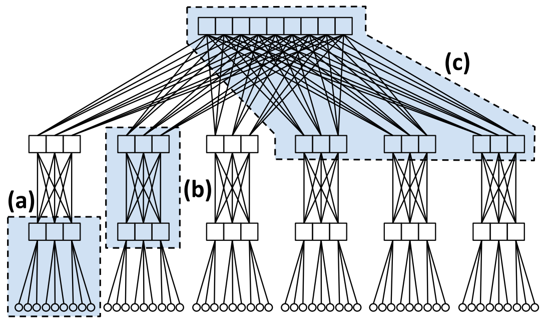

Figure 2: A k = 6 fat-tree [5]. To build a ShareBackup network from it, the blocks of devices like in the shaded areas should be replaced by the corresponding structures in Figure 3.

2.2 Failure study

We run the coflow trace of real data center traffic [1] on packet-level simulators of the fat-tree [5] and F10 [19] networks. The trace contains aggregated rack-level traffic from a 150-rack 10:1 oversubscribed network, so we map the traffic to similar-sized k = 16 fat-tree and F10 networks with the same oversubscription ratio at the edge switches. Fat-tree and F10 both use ECMP routing. Under failures, fat-tree uses global optimal rerouting, and F10 uses its local three-hop rerouting [19]. To study the effectiveness of rerouting, we simulate the final states after failures without the transient dynamics. We first measure the percentage of flows and coflows affected by failures with the variance of failure rate. A flow is considered affected if it traverses a failed node or link, and a coflow is affected if at least one flow in its set gets affected. Then we evaluate the effect of failures on Coflow Completion Time (CCT), which is the lifetime of the most long-lived flow in a coflow. Failures in data centers are rare, and most failures last for less than 5 minutes [11]. To simulate the real-world scenario, we create only one link or node failure at a time and run 5-minute partitions of the coflow trace. We measure the CCT slowdown, which is the CCT with failure divided by the CCT without failure.

We observe from Figure 1(a) and 1(b) that the impact of failures gets magnified significantly on the coflow level. The percentage of affected coflows is 3.3× to 90× that of individual flows. The coflow curves climb faster in the beginning, indicating a small number of failures have huge impact on applications. With only a single node and link failure, as many as 29.6% and 17% of coflows are affected respectively, and

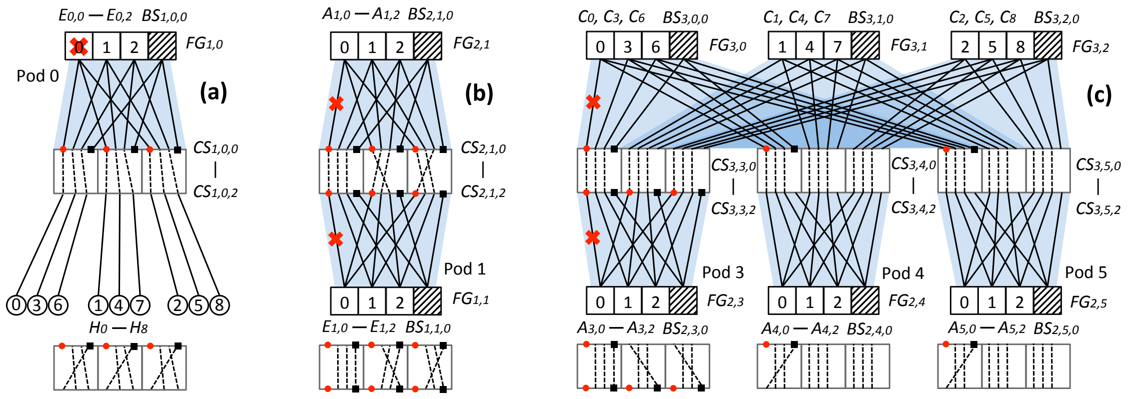
$E_{0,0}$ — $E_{0,2}$ $BS_{1,0,0}$
$FG_{1,0}$
Pod 0
(a)
$CS_{1,0,0}$
$CS_{1,0,2}$
H_0 — H_8
$A_{1,0}$ — $A_{1,2}$ $BS_{2,1,0}$
$FG_{2,1}$
(b)
$CS_{2,1,0}$
$CS_{2,1,2}$
Pod 1
$FG_{1,1}$
$E_{1,0}$ — $E_{1,2}$ $BS_{1,1,0}$
C_0, C_3, C_6 $BS_{3,0,0}$
C_1, C_4, C_7 $BS_{3,1,0}$
C_2, C_5, C_8 $BS_{3,2,0}$
$FG_{3,0}$
$FG_{3,1}$
$FG_{3,2}$
(c)
$CS_{3,3,0}$
$CS_{3,3,2}$
$CS_{3,4,0}$
$CS_{3,4,2}$
$CS_{3,5,0}$
$CS_{3,5,2}$
Pod 3
Pod 4
Pod 5
$FG_{2,3}$
$FG_{2,4}$
$FG_{2,5}$
$A_{3,0}$ — $A_{3,2}$ $BS_{2,3,0}$
$A_{4,0}$ — $A_{4,2}$ $BS_{2,4,0}$
$A_{5,0}$ — $A_{5,2}$ $BS_{2,5,0}$

Figure 3: Substructures of a ShareBackup network where k = 6 and n = 1. The subfigures correspond to the shaded areas in Figure 2. Devices are labeled according to the notations in Table 1. Edge and aggregation switches are marked by their in-Pod indices; core switches and hosts are marked by their global indices. Switches in the same failure group are packed together, which share a backup switch in stripe on the side. Circuit switches are inserted into adjacent layers of switches/hosts. The connectivity in shade is the basic building block for sharable backup. The crossed switch and connections represent example node and link failures. Switches involved in failures are each replaced by a backup switch with the new circuit switch configurations shown at the bottom, where connections regarding the original red round ports reconnect to the new black square ports.

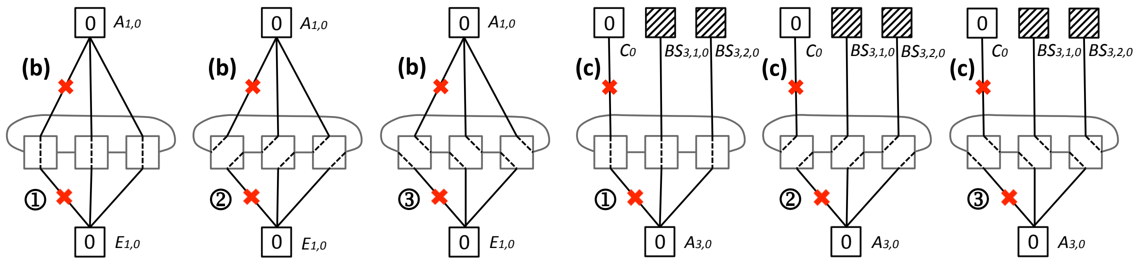

Figure 4: Circuit switch configurations for diagnosis of link failures shown by examples (b) and (c) in Figure 3. Circuit switches in a Pod are chained up using the side ports. Only "suspect switches" on both sides of the failed link and some related backup switches are shown. Through configurations ①, ②, and ③, the "suspect interface" on both "suspect switches" associated with the failure can connect to 3 different interfaces on one or multiple other switches.

these percentages translate to CCT increase by orders of magnitude as shown in Figure 1(c). There is bandwidth loss even with a single failure, and rerouting traffic to alternative paths may cause congestion. F10 has even worse CCT than fat-tree, because its local rerouting uses longer paths and thus causes heavier congestion.

An application can proceed only after an entire coflow has finished, so occasional failures have devastating harm to application performance, and rerouting is ineffective in mitigating the adverse effect. These insights validate our motivation to resolve failures by hardware replacement and to seek cost-effective sharable backup for a few failures.

3 NETWORK ARCHITECTURE

Fat-tree has $\frac{k}{2}$ edge and aggregation switches per Pod. To align with the architecture, we cluster $\frac{k}{2}$ switches into a failure group and allow them to share n backup switches. We leverage small-scale circuit switches to enable sharable backup at low cost. All switches in a failure group, including the backup switches, must connect to the same set of circuit switches with the same wiring pattern. In this way, a backup switch can be brought online at run time to replace a failed switch or failed links associated with it. Figure 2 and Figure 3 give intuitions of the architecture design, with notations listed in Table 1.

Figure 3(a) illustrates the basic building block for sharable backup. The edge switches in the same Pod form a failure group. We place $\frac{k}{2}$ units of $(\frac{k}{2}+n+2)$ by $(\frac{k}{2}+n+2)$ circuit switches between the edge switches and the hosts. Every switch, regular and backup switch alike, connects to these $\frac{k}{2}$ circuit switches each with a link. As shown in Figure 4, these $\frac{k}{2}$ switches are chained together via 2 side ports, which are omitted in Figure 3 for simplicity. Hosts connect to the edge switches via straight-through connections on the intermediate circuit switches. The ports to backup switches are unconnected internally. When a switch is down, the internal connections to it on all the circuit switches are reconfigured to connect to a backup switch, which thus replaces the failed switch completely. A switch whose links are down is replaced in the same manner so as to fix the link failures.

In Figure 3(b), the aggregation switches in the same Pod form a failure group. Edge and aggregation switches in their failure groups repeat the building block of connectivity in

Figure 3(a) to another set of $\frac{k}{2}$ circuit switches. In a fat-tree Pod, an edge/aggregation switch connects to each and every aggregation/edge switch, so we use a rotational wiring pattern in the circuit switches to achieve this shuffle connectivity, i.e. the different internal connections on $CS_{2,1,0}$ to $CS_{2,1,2}$.

Similarly, aggregation switches in each failure group shown in Figure 3(c) are connected upward to $\frac{k}{2}$ circuit switches with the wiring pattern in the building block. As Figure 2 shows, the connections from aggregation switches in each Pod iterate through all the core switches in consecutive order. Because the aggregation switches are already connected to the circuit switches, we wire up the core switches and the circuit switches to achieve the fat-tree connectivity. The core switches connect to $\frac{k}{2}$ circuit switches with a stride of $\frac{k}{2}$, and we set up straight-through connections in the circuit switches. Similar to the building block for sharable backup in Figure 3(a), only switches connected to the same set of circuit switches can be put into a failure group. As a result, core switches whose indices are in $\frac{k}{2}$ intervals form a failure group. We give each failure group n backup switches and connect them up in the same way as regular switches.

In fat-tree, edge and aggregation switches are packaged into Pods for ease of deployment. In each ShareBackup Pod, there are n additional edge and aggregation switches respectively as backup switches, and 3 sets of $\frac{k}{2}$ circuit switches between adjacent layers of switches and hosts. It is straightforward to package the backup switches and the circuit switches into the original fat-tree Pods with simple changes of wiring as shown in Figure 3. In practice, the core switches can be placed as in the original fat-tree, with the backup core switches added to the end. The reordering depicted in Figure 3 is unnecessary. By streamlining the connectors from within each Pod, we can maintain the original Pod-host and Pod-core wiring patterns in fat-tree.

4 CONTROL PLANE

4.1 Fast Failure Detection and Recovery

ShareBackup uses a logically centralized network controller for failure detection and recovery. Switches send keep-alive messages continuously to the controller. When the controller detects a node failure, it allocates an available backup switch to failover to and reconfigures the circuit switches associated with this failure group. As shown in Figure 3(a), in these circuit switches, original connections to the failed switch should reconnect to the backup switch.

We adopt the rapid failure detection mechanism in F10 [19] to detect link failures, where switches and hosts keep sending packets to each other to test the interface, data link, and forwarding engine. When a link is down, it takes time to determine which end has lost connectivity. For the purpose of fast recovery, the switches on both sides of the failed link are replaced. Both of the switches notify the network controller of the failure, and the controller reconfigures the circuit switches in the same way as they handle node failures. Figure 3(b) and 3(c) show examples of this approach.

4.2 Offline Failure Diagnosis

We run failure diagnosis in the background to find which "suspect interface" (and the "suspect switch" it belongs to) has caused the link failure. We chain up circuit switches in the same layer of a Pod as a ring through the side ports. Figure 4 shows the circuit switch configurations, through which the suspect interface on either end of the failed link can connect to 3 different interfaces, either on the same switch (as $A_{1,0}$, $E_{1,0}$, and C_0) or on different switches (as $A_{3,0}$).

The controller changes the circuit switch configurations and enforces the switches to exchange testing messages. A suspect interface that has connectivity in at least one configuration is redressed as healthy, so is the corresponding suspect switch. Failure diagnosis only involves suspect switches already taken offline and backup switches not in use, so it is completely independent of the functioning network.

Failure diagnosis requires both sides have at least one healthy interface, so that both suspect interfaces can be tested. If this condition is not met, both suspect switches are considered faulty. Since all hosts are actively in use, the offline failure diagnosis is not supported between hosts and edge switches. We assume switches are at fault for link failures to hosts. If the problem is not fixed after replacing the switch, we mark the switch as healthy and trouble-shoot the host.

After a failed switch is repaired or a suspect switch is exonerated, it is unnecessary to switch back to the original connectivity. Backup switches and regular switches are equal in functionality, so we keep the backup switch online and turn the replaced switch into a backup switch for future use. This design saves the reconfiguration overhead and avoids disruptions in the network. The network controller keeps track of the current backup switches in their failure groups.

4.3 Live Impersonation of Failed Switch

Traffic is redirected to the backup switch in the physical layer after a failed switch is replaced. The backup switch needs to impersonate the failed switch by using the same routing table. Fat-tree uses Two-Level Routing, where each switch has a pre-defined routing table [5]. To avoid the additional delay of inserting forwarding rules into the backup switch, we aim to preload the routing table and make the backup switch a hot standby. Regular switches recovered from failures can work as backup switches, so every switch needs to store the routing tables of all the switches in the failure group. The challenge is to resolve the conflicts between different routing tables.

In fat-tree, all the core switches and all the aggregation switches in the same Pod have the same routing table. Therefore, in the aggregation and core layers of our network, switches in a failure group only keep a common routing table. For in-bound traffic, edge switches in a Pod, also a failure group, have the same set of $\frac{k}{2}$ forwarding entries that match on the suffix of the end host addresses. For out-bound traffic, each of these edge switches has $\frac{k}{2}$ different entries. We use VLANs for differentiation. We first edit the original fat-tree routing tables by assigning every edge switch in the Pod a unique

Table 2: Cost of compared architectures, where the data center uses electrical (E-DC) and optical (O-DC) transmissions respectively.

Architecture	Cost
Fat-tree	$\frac{5}{4}k^3b + \frac{k^3}{2}c$
ShareBackup	$\frac{3}{2}k^2(\frac{k}{2} + n + 2)a + \frac{5}{2}k^2nb + \frac{5}{4}k^2nc +$ fat-tree cost
Aspen Tree	$\frac{k^3}{2}b + \frac{k^3}{4}c +$ fat-tree cost
1:1 Backup	$\frac{15}{4}k^3b + \frac{3}{2}k^3c +$ fat-tree cost

Variable	Meaning	Price	Notes
a	Per-port cost of circuit switches	\$3 E-DC \$10 O-DC	Electrical crosspoint switch [18] 2D MEMS optical switch [28]
b	Per-port cost of packet switches	\$60	\$3000 for a 48-port 10Gbps bare metal switch
c	Cost per link	\$81 E-DC \$40 O-DC	10m 10Gbps DAC [2] 10Gbps transceiver (\$16) $\times$ 2 + 10m 10Gbps optical fiber (\$8) [2]

VLAN ID and adding it to the out-bound routing table entries. The edited routing tables from all the edge switches are then combined together and stored in every switch in the failure group. A host knows which edge switch it should connect to, so it tags out-going packets with the VLAN ID of the edge switch. No matter what switches in the failure group are active, by matching the VLAN ID, packets can always refer to the correct routing table. This combined routing table from $\frac{k}{2}$ edge switches has $\frac{k}{2}$ in-bound entries and $\frac{k^2}{4}$ out-bound entries. This total number is within the TCAM capacity of commercial switches even for large-scale fat-tree networks. For instance, the table contains 1056 entries for a $k = 64$ fat-tree with over 65k hosts.

5 ARCHITECTURE PROPERTIES

5.1 Capacity to Handle Failures

Switch failures: In a failure group, n backup switches are shared by $\frac{k}{2}$ switches. Thus, ShareBackup can handle n concurrent switch failures per failure group. In data centers, failures are independent; most devices have over 99.99% availability; and failures usually last for only a few minutes [11]. As a result, a small n is sufficient for a large-scale data center. For instance, let $n = 1$ for a $k = 48$ fat-tree with over 27k hosts, the backup ratio is $n/\frac{k}{2} = 4.17\%$, which is more than 400× higher than the 0.01% switch failure rate.

Link failures: ShareBackup handles link failures as node failures. With failure diagnosis, we can identify the interface at fault, so we consume only one backup switch at the faulty end. For each failure group, ShareBackup can handle n independent link failures, which translates to up to kn link failures rooted at those n switches. Link failures are rare, and concurrent link failures are especially uncommon [11]. It is sufficient to target at a few link failures with a small n.

Circuit switch failures: Circuit switches are highly reliable. They are passive physical-layer devices with less than 10^{-12} bit-error rate [18, 28], and their bare-minimum control software for circuit reconfiguration receives infrequent requests only when switch and link failures happen. In the rare case that a circuit switch is down, switches connected to it will report link failures to the network controller. If the controller receives a large number of link failure reports associated with one circuit switch in a short period of time, i.e. over a pre-defined threshold, it will stop failure recovery and request for human intervention. A rebooted circuit switch can get up-to-date circuit configurations from the controller. Circuit switch port failures are sensed as link failures and handled by regular failure recovery.

Controller failures: The logically centralized network controller can be implemented as a small cluster of controller machines. The switches and hosts report status to them at the same time. A primary controller is elected to react to failures. When the primary controller fails, another controller will be elected to take its place.

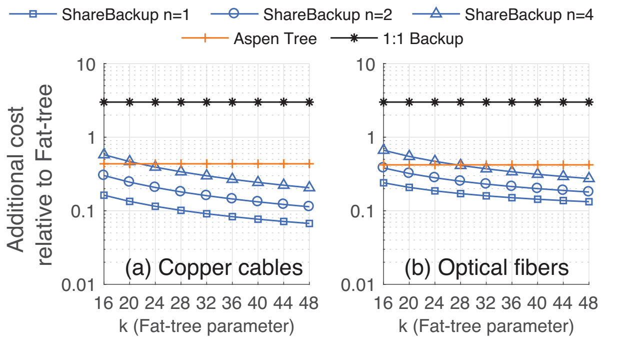

Figure 5: Additional cost of ShareBackup, Aspen Tree, and 1:1 Backup relative to fat-tree at different network scales using market prices in Table 2

5.2 Cost Analysis

To save cost, we choose the implementation technology of circuit switches based on the existing devices already deployed in the data center. If the data center has copper DAC cables, we can use electrical crosspoint switches whose per-port cost is only \$3 [18]. Although circuit switches split some cables into two parts, they generate no extra cabling cost. Cables connected to the passive circuit switches do not need active elements. With proper manufacturing, the cost of two cables each with one active element at the packet switch end is equivalent to the cost of the original cable.

We choose optical circuit switches for data centers with optical transceivers and fibers. Moderate-scale low-cost switching technologies, e.g. 2D MEMS, can be used. 2D MEMS costs only \$10 per port and scales to 32 ports [23, 28]. Its insertion loss can be overcome by the commercial transceivers already deloyed [2], so amplifiers are not needed. The cost of two shorter fibers split by the circuit switch is roughly the same as the original fiber, so the cabling cost stays the same.

We calculate ShareBackup's additional cost to fat-tree and compare to Aspen Tree [26] and 1:1 backup, which also add hardware to fat-tree to improve robustness. Table 2 lists the cost equations of these architectures and the market price of necessary devices. 1:1 backup requires twice as many switches as fat-tree, and the switch port count needs to be doubled. Assuming constant price of a switch port, the cost

Table 3: Performance characteristics of different architectures

Architecture	No bandwidth loss?	No path dilation?	No upstream repair?
ShareBackup	✓	✓	✓
Fat-tree	×	✓	×
F10	×	×	✓
Aspen Tree	×	✓	✓/×

of 1:1 backup is 4× that of fat-tree. Aspen Tree repurposes links between switches in adjacent layers. The lower-layer switches can disconnect half of the upper-layer switches to duplicate connections to the other half. One more layer of switches are needed to connect the partitioned network, so there are $\frac{k^2}{2}$ more switches and $\frac{k^3}{4}$ more cables.

ShareBackup has $\frac{5}{2}k$ failure groups, each with n backup switches, and each Pod contains 3 sets of $\frac{k}{2}$ circuit switches with $(\frac{k}{2}+n+2)$ by $(\frac{k}{2}+n+2)$ ports. Thus, ShareBackup has $\frac{5}{2}kn$ more switches, $\frac{5}{4}k^2n$ more cables, and $\frac{3}{2}k^2(\frac{k}{2}+n+2)$ circuit switch ports. ShareBackup is less costly than Aspen Tree and 1:1 backup because it uses sharable backup and cheap circuit switches. As shown in Table 2, a is much cheaper than b and c, and n is a small constant in practice. Compared to Aspen Tree and 1:1 backup, ShareBackup reduces the power of k in b- and c-related terms, and the factor a limits the additional term to a relatively small value.

Figure 5 shows ShareBackup is multi-folds less expensive than 1:1 backup and Aspen Tree. For a fixed n, the relative additional cost of ShareBackup decreases as the network scales up, because the backup switches can be shared by more switches in the failure group. As discussed in Section 5.1, n = 1 is sufficient for a k = 48 fat-tree network. In this case, the additional cost of ShareBackup is merely 6.7% and 13.3% of the cost of fat-tree with copper cables and optical fibers respectively, while Aspen Tree costs 6.5× and 3.2× as much. Even if n is increased to 4, which renders backup ratio as high as 16.7% for k = 48, ShareBackup is still cheaper than Aspen Tree. The cases where ShareBackup out-costs Aspen Tree show the flexibility of improving robustness by adding more backup switches.

5.3 Performance Characteristics

Scaling to large data centers with high robustness. The scalability of ShareBackup is determined by the port count of circuit switches, i.e. $(\frac{k}{2}+n+2)$. 256-port electrical crosspoint switches are common place today [18], and 32-port 2D MEMS optical switches can be realized [28]. Even with the 32-port limit, we have $\frac{k}{2}+n+2=32$, or $\frac{k}{2}+n=30$. When $n=1$, ShareBackup can support a $k=58$ fat-tree network with over 48k hosts. The backup ratio is $n/\frac{k}{2}=3.45\%$, which is significantly higher than the 0.01% switch failure rate. For a sizable $k=48$ fat-tree with 27k hosts, n can reach 6, leading to a backup ratio as high as 25%. These parameters relating to scalability and robustness can be tuned to meet practical needs of the data center.

Recovering failures as fast as state of the art. F10 and Aspen Tree reroute traffic locally as soon as a switch detects a failure [19, 26], so the recovery delay is the failure detector's probing interval plus the time of redirecting packets to a different NIC interface. Rerouting requires change of at least one routing table entry. Using SDN, it takes ∼1ms to modify a forwarding rule [17]. In ShareBackup, the controller probes switches for node failures, and switches probe each other for link failures and inform the controller. We assume the same probing interval as F10 and Aspen Tree. With the failure information, the controller sends requests to circuit switches to reset circuits. The circuit reconfiguration delay is negligible, which is only 70ns for crosspoint switches [18] and 40μs for 2D MEMS [28]. The communication channels are actively on for probing. With efficient controller implementation, e.g. as a kernel module, the delay of switch-to-controller and controller-to-circuit-switch communications can be reduced to sub-ms level. Therefore, failure recovery in ShareBackup is as fast as that in F10 and Aspen Tree.

No bandwidth loss, no path dilation, and no upstream repair. Table 3 compares key features of failure-resilient architectures. Because ShareBackup replaces failed hardware completely, it does not have bandwidth loss. All other rerouting-based solutions have to cope with the remaining bandwidth resource, and we have demonstrated in Section 2.2 that a single link or node failure can be disastrous to application performance. Fat-tree requires failure announcements to propagate multiple hops so rerouting can be performed upstream. To improve responsiveness, F10 reroutes traffic locally through longer paths, and Aspen Tree creates duplicate paths with extra hardware. As Figure 5 shows, the additional cost of Aspen Tree is high, so it provides the option of partial duplication that requires upstream repair. With sharable backup, our proposal replaces failed devices locally without path dilation at minimal additional cost to the network.

6 CONCLUSION

The concept of sharable backup goes beyond this work. Like fat-tree, most data center network architectures have symmetric structures [4, 12–14]. Sharable backup is thus readily applicable to these networks, with different plans for partitioning failure groups. Non-uniform failure groups should also be explored, so that this idea can be extended to unstructured networks, such as Jellyfish [25], and we can have more backup on critical devices and less backup on unimportant ones. Several questions are worth further studies. First, the efficiency of the control plane depends on the implementation of the network controller, so the real-world performance is yet to be tested. Second, it is desirable to use distributed network controllers to share the burden and reduce the latency of failure detection and recovery. The placement and coordination of controllers remain open questions. Third, when backup switches are idle, they can be activated to add bandwidth to the network. How to make better use of backup switches to improve performance with guaranteed fault tolerance is an interesting research topic.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their thoughtful feedback. This research was sponsored by the NSF under CNS-1422925, CNS-1305379 and CNS-1718980.

REFERENCES

[1] Coflow-Benchmark, https://github.com/coflow/coflow-benchmark/.

[2] FS.COM, http://www.fs.com/.

[3] Introducing data center fabric, the next-generation Facebook data center network, https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/.

[4] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks. In *SC '09*, pages 41:1–41:11, Portland, Oregon, USA, November 2009.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM '08*, pages 63–74, Seattle, Washington, USA, August 2008.

[6] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *NSDI '12*, San Joes, CA, April 2012.

[7] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, and Q. Dong. WaveCube: A Scalable, Fault-tolerant, High-performance Optical Data Center Architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1903–1911, April 2015.

[8] M. Chowdhury and I. Stoica. Coflow: A Networking Abstraction for Cluster Applications. In *HotNets-XI*, pages 31–36, Redmond, WA, 2012.

[9] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *SIGCOMM '10*, pages 339–350, New Delhi, India, August 2010.

[10] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, pages 216–229, Florianopolis, Brazil, August 2016.

[11] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 350–361, New York, NY, USA, 2011. ACM.

[12] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[13] 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 '09*, pages 63–74, Barcelona, Spain, August 2009.

[14] 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 '08*, pages 75–86, Seattle, Washington, USA, August 2008.

[15] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 38–49, Toronto, Ontario, Canada, August 2011.

[16] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. FireFly: A Reconfigurable Wireless Data Center Fabric Using Free-space Optics. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 319–330, Chicago, Illinois, USA, August 2014.

[17] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan. Measuring Control Plane Latency in SDN-enabled Switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 25:1–25:6, Santa Clara, California, 2015.

[18] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao. XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 15–29, Santa Clara, CA, 2016. USENIX Association.

[19] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A Fault-Tolerant Engineered Network. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 399–412, Lombard, IL, 2013. USENIX.

[20] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav. Quartz: A New Design Element for Low-latency DCNs. In *SIGCOMM '14*, pages 283–294, Chicago, Illinois, USA, August 2014.

[21] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[22] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat. Integrating Microsecond Circuit Switching into the Data Center. In *SIGCOMM '13*, pages 447–458, Hong Kong, China, August 2013.

[23] M. Schlansker, M. Tan, J. Tourrilhes, J. R. Santos, and S.-Y. Wang. Configurable optical interconnects for scalable datacenters. In *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013*, pages 1–3. IEEE, 2013.

[24] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM '15*, pages 183–197, London, United Kingdom, August 2015. ACM.

[25] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking Data Centers Randomly. In *NSDI '12*, pages 1–14, San Jose, California, USA, April 2012.

[26] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo. Aspen Trees: Balancing Data Center Fault Tolerance, Scalability and Cost. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 85–96, New York, NY, USA, 2013. ACM.

[27] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan. c-Through: Part-time Optics in Data Centers. In *SIGCOMM '10*, pages 327–338, New Delhi, India, August 2010.

[28] M. C. Wu, O. Solgaard, and J. E. Ford. Optical MEMS for Lightwave Communication. *Journal of Lightwave Technology*, 24(12):4433–4454, December 2006.

[29] Y. Xia and T. S. E. Ng. Flat-tree: A Convertible Data Center Network Architecture from Clos to Random Graph. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 71–77, Atlanta, GA, November 2016.

[30] Y. Xia, X. S. Sun, S. Dzinamarira, D. Wu, X. S. Huang, and T. S. E. Ng. A tale of two topologies: Exploring convertible data center network architectures with flat-tree. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 295–308, New York, NY, USA, 2017. ACM.

[31] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng. Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 443–454, Helsinki, Finland, August 2012.