Abstract—Failure recovery in IP networks is critical to high quality service provisioning. One of the challenges is how to achieve fast recovery without introducing high complexity and resource usage. Today’s main approaches are route recalculation and lower layer protection, where recalculation could take a long time to complete; while protection usually requires considerable bandwidth redundancy.

IP fast rerouting achieves ultra fast failure recovery by calculating alternate paths in advance. When a failure is detected, the affected packets are immediately forwarded through alternate paths to shorten the service disruption. We present an algorithm called Efficient SCan for Alternate Paths (ESCAP) to achieve fast rerouting. The algorithm guarantees 100% recovery of single-link and single-node failures. In particular, it supports generic multi-path routing (where a router maintains multiple paths to a single destination) and does not require the paths to have equal cost. The implementation of ESCAP has low complexity and does not introduce explicit signaling between routers. Simulations show that our scheme yields comparable performance to shortest path route recalculation. This work illuminates the possibility of using pure IP layer solutions to enhance the Internet survivability.

I. INTRODUCTION

The Internet has evolved to a global information platform that supports numerous applications. For such a critical infrastructure, survivability is a stringent requirement in that services interrupted by network failures must be recovered as quickly as possible [1]. Typically, a recovery time of tens of milliseconds satisfies most requirements (e.g., SDH/SONET automatic protection switching (APS) is completed within 50 ms [2]). It is also expected that failure recovery schemes have low complexity and do not reserve redundant bandwidth. Network failures can be caused by a variety of reasons such as fiber cut, interface malfunctioning, software bugs, misconfiguration and attacks. Despite continuous technological advances, failures cannot be completely avoided even in well-maintained networks [3].

The fundamental issue of failure recovery is how to set up a new path to replace a damaged one. The main approaches used by today’s IP networks are route recalculation and lower layer protection [4], [5]. All routing protocols (such as open shortest path first (OSPF) [6]) are designed to perform failure advertising, route recalculation and routing table update to recover from failures. Although these mechanisms are capable of dealing with any type of failures, the process can easily reach seconds [7], which is unacceptable to critical applications such as stock trading systems. On the other hand, lower layer protection achieves fast recovery by establishing backup connections in advance (e.g., a time slot channel) and use them to replace damaged connections. In this case, the IP layer is protected from failures without any modifications on the routing tables. However, this approach reserves considerable redundant bandwidth for the backup connections. More importantly, relying on lower layer protection means the IP layer is not independent in term of survivability. From this point of view, the original objective of packet switching — to design a highly survivable network where packet forwarding in each router is adaptive to the network status — still needs extensive work [8].

This paper focuses on IP fast rerouting (IPFRR) [7]. The basic idea is to let a router maintain a backup port for each destination and use it to forward packets when the primary port fails. Since the backup ports are calculated in advance and do not occupy redundant bandwidth, IPFRR achieves fast failure recovery with great efficiency.

We present an algorithm called Efficient SCan for Alternate Paths (ESCAP) to achieve IP fast rerouting under single-link and single-node failures. In particular, ESCAP also works for networks that perform multi-path routing where a router may maintain multiple paths to a single destination [9], [10]. A special case is equal-cost multi-path (ECMP), which has been implemented in practical networks. Our algorithm supports generic multi-path routing in that it does not require the paths to have equal cost. To the best of our knowledge, this generic scenario has not been thoroughly investigated in literature. We show that ESCAP guarantees 100% failure recovery and is feasible to be implemented in practical networks. It is worth noting that an important issue related to failure recovery is failure detection [11], [12], which, however, is beyond the scope of this paper.

This paper is organized as follows. The next section gives an overview of the issue and related work. Section III describes the algorithm. Section IV discusses the implementation issue. Section V presents the simulation results Section VI concludes the work and identifies future research.

II. OVERVIEW

A. IP Fast Rerouting

Each IP router normally maintains a primary forwarding port for a destination(prefix). When a failure occurs, some of the primary ports could point to the damaged link/node and become unusable. The idea of IPFRR is to proactively calculate backup ports that can be used to replace primary ports temporarily until the subsequent route recalculation is completed. Figure 1 shows an example with node 1 as the
route recalculation is performed in parallel. The key points are:

- How to find backup ports? This is non-trivial because inconsistency between backup ports may create routing loops. In Figure 1, pointing the backup port of node 4 to node 3 would create a loop.
- How to perform failure recovery? The answer helps routers to determine when to use primary/backup ports. To shorten service disruption, it is required to make the decision without waiting for failure advertisement.
- How to realize distributed implementation? The implementation requires modifying existing routers. Therefore, the complexity and the compatibility to existing routing protocols must be considered.

B. Related Work

A simple scheme related to IPFRR is equal cost multi-paths (ECMP), where a number of paths with the same cost are calculated for each source/destination pair [13]. The failure on a particular path can be handled by sending packets along an alternate path. This approach has been implemented in practical networks. However, an equal cost path may not exist in certain situations (such as in a ring), thus ECMP cannot guarantee 100% failure recovery [7].

A scheme to find loop-free alternate paths is presented in [14]. Consider the route from $S$ to $D$. If $S$ has a neighbor $X$ that satisfies $d(X, D) < d(X, S) + d(S, D)$, where $d(i, j)$ is the cost from $i$ to $j$, it can send packets to $X$ as an alternate path. The condition ensures that packets do not loop back to $S$. Similar to ECMP, this scheme does not guarantee 100% failure recovery since a node may not have such a neighbor.

In [15], a scheme is proposed to set up a tunnel from node $S$ to node $Y$ that is multiple hops away. The alternate path to a destination $D$ is from $S$ to $Y$ then to $D$. This guarantees 100% failure coverage. The extra cost is the maintenance of many tunnels and potential fragmentation when the IP packet after encapsulation is longer than the maximum transmission unit (MTU) [16].

A scheme called failure insensitive routing (FIR) is presented in [17] for single-link failures. Given a primary path $S \rightarrow D$, FIR identifies a number of key links such that removing any of these links forces the packets to go back to $S$. Therefore, the failure of any key links can be inferred by $S$ at a deflected packet. To provide an alternate path, FIR removes the key links and runs shortest path routing from $S$ to $D$. FIR is extended to cover single-node failures in [18]. FIR also supports ECMP. Our scheme and FIR share similar ideas. The difference is: we develop a different algorithm that does not have any assumptions on the primary paths (e.g., the primary paths can be either shortest or non-shortest); and our algorithm supports generic multi-path routing where the paths could have different costs.

An algorithm called multiple routing configuration (MRC) is presented in [19]. The scheme lets each router maintain multiple routing tables (configurations). After a failure is detected, the routers search for a configuration that is able to bypass the failure. After that, the index of the selected configuration is inserted into packet headers to notify each router which table to use. MRC achieves 100% failure coverage. The overhead of MRC is maintaining multiple routing tables and adding an extra index to packet headers.

Recently, an inspiring work is done on path diversity, which discusses how to find multiple paths between source/destination pairs using routing deflection [20]. The authors derive three neat conditions that achieve generic path diversity. Although the scheme is not designed for a specific application, it is shown to be promising for failure recovery. In this stage, directly using the scheme cannot guarantee 100% failure coverage.

III. EFFICIENT SCAN FOR ALTERNATE PATHS

A. Algorithm Description

Without loss of generality, we select node 1 as the destination. When there is no multi-path routing, the primary paths form a spanning tree with node 1 as the root. We denote the sub-tree rooted at node $n$ as $T(n)$ and use the following algorithm to find the backup port of each router. These backup ports are used to form alternate paths for fast rerouting in case of link/node failures.

1) Init: Set the backup port of each node to null, i.e., $b_n = 0$ $(n = 2, \ldots, N)$.

2) Explore the primary tree $T(1)$ in depth-first order. For each node $n$, do the following:
   a) If node $n$ is not a child of the root node, jump to step 2b. Otherwise, we assume its primary port fails, dye $T(n)$ black and $T(1)\setminus T(n)$ white. We use the following steps to find an alternate path connecting the black sub-tree to the white part.
      i) Scan $T(n)$ in breadth-first order to find the first node $j$ that has a white neighbor $w$, set $b_j = w$, which we call an exit.
      ii) Follow the links in $T(n)$, find the path from $n$ to $j$, which is the recovery path. Set the backup ports of the nodes on the recovery path. Go back to step 2.
   b) If node $n$ has $m_n$ children, denote them as $c_1, \ldots, c_{m_n}$. Assume node $n$ fails, dye $T(n)$ black and $T(1)\setminus T(n)$ white.
Therefore, we set \( \text{exit} \). Using the same method, we find \( 8 \rightarrow 5 \rightarrow 9 \) as the exit.

**Definition 2:** Sub-Graph \( \mathcal{G}(n) \): Defined as the directed graph consisting of node \( n \) and all the nodes/links with paths traversing node \( n \). For example, \( \mathcal{G}(6) \) in Figure 4 consists of nodes 6, 8, 9, 10 and the solid arrows.

**Definition 3:** Breadth-First Search: Given a node \( n \), explore all its children before going to its grandchildren. For example, a breadth-first explore of \( \mathcal{G}(2) \) in Figure 4 yields \( 2-4-5-7-8-10-10 \), where node 10 is hit twice because it has two paths to node 2.

**Definition 4:** Depth-First Search: Given a node \( n \), explore as far as possible along each branch before backtracking. For example, a depth-first explore of \( \mathcal{G}(2) \) in Figure 4 yields \( 2-4-7-10-8-10-5 \), where node 10 is also hit twice.

**Solution**

We still use the same algorithm in Section III-A to find backup ports. The modifications include:

- \( \mathcal{T}(n) \) is replaced with \( \mathcal{G}(n) \); child, parent, breadth-first search and depth-first search are based on the above definitions.
- When a node has multiple primary ports, a backup port is found for each of them.
- When marking a backup port, we bind it to the primary port that overlaps with the recovery path. In Figure 4(b), when link 2–1 fails, we find backup port 8–6. This backup port is bound to primary port 8–4 because 8–4 overlaps with the recovery path. On the other hand, Figure 4(c) shows that considering failure 3–1 finds backup port 8–5, which is bound to forwarding port 8–6.
- In step 2c of the algorithm, we check the backup port corresponding to the primary port that is assumed to be failed. For example, after the search in Figure 4(b), if we assume node 4 fails and check node 8, we only check its backup port bound to primary port 8–4. In this case, the backup port is already found and the subsequent scan of \( \mathcal{G}(8) \) is skipped.

**Failure Coverage:** Consider a particular destination, any single-link/node failure separate a black part from the primary routing tree (or directed-graph in case of multi-path routing).
ESCAP guarantees the existence of an alternate path to reconnect each possible black part to the white part. Therefore, it achieves 100% failure coverage.

**Free of Dead Loops:** Given a black sub-tree/sub-graph, ESCAP guarantees that the alternate path starts from its root and goes through the exit link following the reverse direction of the primary paths. Since the primary path is loop-free, the alternate path inside each black sub-tree/sub-graph is also loop-free. Each alternate path does not re-enter any sub-tree/sub-graph it has traversed and it uses the primary path after entering the white part. Therefore, any alternate path is free of dead loops.

**Complexity:** ESCAP has low complexity because the scan is based on linear search and the computation at each step is efficient. For an $N$-node tree/graph, the complexity is $O(N^2)$ to find the backup port of each node for the destination. The discussions in the next section shows that further simplification is achieved in the distributed implementation.

**Sequence of Search:** Although we use depth-first search in step 2, breadth-first search works as well. This is because the backup port of a node could be affected only by its parent or indirect parent. Therefore, the only requirement for the sequence of search is to find the backup ports from the top to the bottom of a primary tree.

IV. DISTRIBUTED IMPLEMENTATION

ESCAP requires that each router has the knowledge of the overall topology, therefore, the implementation is design for networks using link-state routing protocols, such as OSPF.

A. When to Switch to a Backup Port

When a failure occurs, only a subset of routers need to switch to their backup ports. Therefore, it is critical for a router to determine how to switch between its primary and backup ports. ESCAP enables a packet forwarding policy that does not require explicit failure notification. The policy is illustrated in Figure 5, which is explained as follows.

- If a failure is detected on the primary port, the backup port is certainly chosen for packet forwarding. In case of multi-path routing, the primary port is the one selected for the incoming packet (e.g., using hashing).
- If a packet comes in from the primary port, it implies a failure on the primary forwarding path. Therefore, the backup port is used to forward this packet.
- Otherwise, the primary port is used.

For example, when node 2 in Figure 3 fails, packets from node 5 follow the path $5\rightarrow 10\rightarrow 6\rightarrow 10\rightarrow 7\rightarrow 3\rightarrow 1$. Node 5 and 6 always use their backup ports because failures are detected on their primary ports. Node 10 uses its primary port when packets arrive from node 5 and selects the backup port when packets are deflected back from node 6. All the other nodes stick to their primary ports.

B. Routing Table Extension

Each IP router maintains a routing table where an entry has the structure of Figure IV-B. To enable efficient distributed processing, the routing information may be downloaded to each line card to construct a forwarding table [21]. Upon the arrival of an IP packet, the link card performs longest prefix matching and table look-up to retrieve the appropriate next_port and port, which identify the output port to send the packet to. To support IPFRR, each entry is extended by adding the backup port information: bk_next_port and bk_port, as illustrated in Figure IV-B. The backup ports are stored in different memory banks and the addresses are aligned with the primary ports. Therefore, each read/write operation accesses the primary and backup ports in parallel, thus achieving high speed table look-up.

C. Backup Port Calculation

ESCAP explores the whole tree to find backup ports. In the distributed implementation, the computation is simplified by letting each router explore the tree partially. Suppose the primary path from router $k$ to 1 is $k \rightarrow m_1 \rightarrow \ldots \rightarrow m_1 \rightarrow 1$, router $k$ scans failures on link $m_1$, node $m_1$, ..., $m_L$ to get its backup port. This is because failures on the other links/nodes does not affect its primary path. Similarly, when a router performs multi-path routing, it only explores the link/node failures on its forwarding paths. For example, in Figure 3, node 10 considers link 2-1, node 2 and 6 sequentially. In Figure 4, node 8 explores two paths: 1-2-4-8 and 1-3-6-8, respectively.

D. Discussions

The above implementation has several advantages:

- The switch-over of each router is fast, adaptive and does not require explicit failure notification.
- The additional memory requirement for the routing table extension is bounded. Only two fields are added to each entry, which can be achieved with minor cost increase.
- The speed of the routing table look-up is not affected because a primary port and its backup port are accessed in a single read operation.
- The complexity of the backup port calculation for each destination is bounded by the number of nodes in the network. The algorithm consumes little computation resources.

V. PERFORMANCE EVALUATION

We use computer simulations to study the rerouting path lengths and traffic distribution using ESCAP, which have significant impact on router-to-router delay, congestion, and network efficiency. We compare ESCAP with shortest path route
sets of data: age of long and short path. For each topology, we obtain three scenarios: Normal, IPFRR, and route recalculation. In the Normal scenario, we explore all possible failures and use ESCAP to find the alternate paths. In the IPFRR scenario, we explore all possible failures and use ESCAP to find the alternate paths. In the route recalculation scenario, we explore all possible failures and use route recalculation to see the difference. The topologies adopted in our evaluation include several practical and random networks. The results show that ESCAP has consistent performance in various networks. Due to space limit, we only present the results of single-node failures in COST239 (Figure 7(a)), NSFNet (Figure 7(b)) and random topologies.

A. Rerouting Path Lengths

We study the distribution of path lengths to see the percentage of long and short path. For each topology, we obtain three sets of data:

- Normal: There is no failure in the network, all paths are minimum hop paths.
- IPFRR: We explore all possible failures and use ESCAP to find the alternate paths.
- Recalculation: We explore all possible failures and recalculate the minimum hop paths.

The results in Figure 8 show comparable performance between IPFRR using ESCAP and route recalculation. Compared to the shortest paths obtained using route recalculation, ESCAP creates more long paths, which is an expected price paid for the short recovery interval. Nonetheless, the percentage of such long paths are quite small. The results also show more long paths in NSFNet than in COST239. This is caused by the intrinsic characteristics of the topologies: the connections in NSFNet are not as dense as those in COST239. Nonetheless, the results show consistent performance of ESCAP.

B. Link Load

We study the volume of traffic on each link to identify hot spots in networks. Hot spots often have negative influence on network performance since congestion tends to occur with high probability on links with heavy load. We explore all possible single-node failures and measure the average and worst traffic load on each directed link. The comparison among normal, IPFRR and route recalculation is shown in Figure 9. We assume the traffic demand between any two nodes is 1 Mb/s. For clarity, the links are sorted based on their normal load.

The results show that IPFRR and route recalculation generate similar average load on each link. However, the worst case load generated by IPFRR is often heavier than that generated by route recalculation. In other words, under certain failures, IPFRR is more likely to cause a congested link than route recalculation. There are two solutions to handle this issue in practical networks. First, the capacity of each link can be carefully dimensioned to accommodate such traffic increase when a failure occurs. Second, packets can be prioritized so that the delivery of critical traffic is guaranteed at the cost of degraded service to best-effort traffic. Figure 9(a) shows that some links have lighter load when a failure occurs. This is because a failed node does not generate traffic, which reduces the overall traffic demand.

The last two directed links in Figure 9(a) connect the same node pair but have different IPFRR load. This is because ESCAP is not specifically designed to make the rerouting paths symmetric. If symmetry is necessary, minor modifications can be introduced to satisfy the requirement.

C. Overall Traffic Volume

Denote the traffic load on link \( i \rightarrow j \) as \( u_{i,j} \), the overall traffic volume of the network is defined as

\[
U = \sum_{i,j} u_{i,j}.
\]

Given a traffic demand, \( U \) is determined by the routing scheme and reflects the efficiency of the network. The smaller the overall traffic volume is, the higher efficiency the network has. The comparison between ESCAP and route recalculation under each single failure is shown in Figure 10. For clarity, the data are sorted by the recalculation volume measurements. IPFRR usually generates higher overall traffic volume than route recalculation does since the rerouting paths are often longer than the shortest paths. Nonetheless, the difference is acceptable. The results also shows that COST239 is less sensitive to the location of failures than NSFNet is in that the...
show that ESCAP achieves almost the same efficiency as route recalculation. The results from 4 to 14. In each topology, we assume the traffic demand of ESCAP to deal with multiple failures. Second, combining research directions. First, it is interesting to study the extension of the scheme for path-vector routing so as to enhance the survivability of inter-domain routing.

VI. CONCLUSIONS AND FUTURE WORK

We propose an IP fast rerouting algorithm called ESCAP to achieve failure recovery. Our algorithm guarantees 100% coverage of single-link and single-node failures. In particular, it provides good support of generic multi-path routing, which has been adopted in practical networks but has not received thorough study in existing IP fast rerouting algorithms. ESCAP has low complexity and can be easily applied to practical networks to substantially enhance the survivability. We verify the performance of our algorithm in a variety of practical and random networks and show that the price paid for the survivability enhancement is insignificant. The path lengths, link load and network overall traffic volume using ESCAP are comparable to those using shortest path route recalculation.

IP fast rerouting illuminates the possibility of building a highly survivable Internet without employing complicated solutions. Based on our work, there are several promising research directions. First, it is interesting to study the extension of ESCAP to deal with multiple failures. Second, combining ESCAP with load balancing could further improve the quality of service during failure recovery. Third, it is interesting to bring shared risk link group (SRLG) into consideration, where multiple links sharing the same fiber are vulnerable to a single physical link failure [23], [24]. Finally, ESCAP is designed for link-state routing protocols, it is interesting to study the extension of the scheme for path-vector routing so as to enhance the survivability of inter-domain routing.

ACKNOWLEDGEMENTS

The authors would like to thank Mike Shand for his constructive suggestions on this research.

REFERENCES